EVOLVABLE BINARY ARTIFICIAL NEURAL NETWORK FOR DATA CLASSIFICATION

Janusz Starzyk Jing Pang
School of Electrical and Computer Science
Ohio University
Athens, OH 45701, U. S. A.
(740) 593-1580

ABSTRACT

This paper describes a new evolvable hardware organization and its learning algorithm to generate binary logic artificial neural networks based on mutual information and statistical analysis. First, thresholds to convert analog signals of the training data to digital signals are established. In order to extract feature function for multidimensional data classification, conditional entropy is calculated to obtain maximum information in each subspace. Next, dynamic shrinking and expansion rules are developed to build the feed forward neural networks. At last, hardware mapping of learning patterns and on-board testing are implemented on Xilinx FPGA board.

KEYWORDS

Conditional entropy, binary logic artificial neural network, dynamic function generation

1. INTRODUCTION

Artificial neural networks (ANN) have attracted the attention of many researchers in different areas, such as neuroscience, mathematics, physics, electrical and computer engineering, and psychology. Generally, such systems consist of a large number of simple neuron processing units performing computation by a dense mesh of nodes and connections. In addition, they have the very attractive properties of adaptiveness, self-organization, nonlinear network processing and parallel processing. A lot of efforts have been put on applications involving classification, association, decision-making and reasoning.

Recently, evolutionary algorithms have been suggested by many researchers to find well performing architectures for artificial neural networks, which employ the evolutionary dynamics of reproduction, mutation, competition, and selection [1][2]. Evolutionary algorithms, like genetic algorithms, evolutionary programming, and evolution strategies are well suited to the task of evolving ANN architectures [3]. Much work has been done on combining the evolutionary computation techniques and neural network [4][5][6][7][8][9][10]. The most challenging part is how to determine the link between structure and functionality of ANN so that optimization of the composition of a network can be implemented using evolutionary methods. Popular methods like genetic algorithms, which use parse trees to construct neural network, greatly depend on the qualities of their approaches to evolve the interconnection weight [11]. In this paper, an efficient method to construct dynamically evolvable artificial neural network is proposed. It is based on mutual information calculation to threshold the input data, organize logic blocks and their interconnections, and to select logic blocks, which are most informative about data distribution. In addition, expansion and shrinking algorithms are formulated to link the multiple layers of the binary feed forward neural network.

With the rapid advance of VLSI microelectronics technology, large computational resources on a single chip become available. Reconfigurable computing and FPGA technology make fast massively parallel computations more affordable and neural networks with architecture adapted to a specific task can be easily designed.

Since most of the signals in the real world are analog, many researchers have developed analog neural networks [12][13][14]. But in such systems, matching between the simulated analog neuron model and the hardware

implementation is critical. Other problems like noise, crosstalk, temperature variation, and power supply instability also limit system performance. Moreover, the programmability is also hard to achieve for analog neural network design. As a result, many people turn to digital logic for an alternative solution. The flexibility, and accuracy plus mature commercial CAD tools for digital system design greatly save the time and effort of design engineers. Moreover, the recent fast development of FPGA technology has created a revolution in logic design. With the dramatic advances in device performance and density combined with development tools, programmable logic provides a completely new way of designing and developing systems. On the other hand, a hardware description language VHDL becomes more and more popular for logic design and uses the complete tools for VHDL code compiling, functional verification, synthesis, place and route, timing verification and bit file downloading. These greatly facilitate the FPGA logic system design and also make it possible to design artificial neural network on board.

In our design, Matlab program generates logic structures based on the learning procedure. Then VHDL codes are written and simulated to map the evolved structures to programmable hardware. At last, the results of learning are implemented on the FPGA board. Highly parallel structure is developed to achieve the fast speed neural network for large data set classification. The class characteristic codes generated during learning procedure are saved on board, so that comparators can judge the success or failure of test data easily. Moreover, the finite state machine structure facilitates the selection and the display of different class test results. Real time testing of the implemented structures on Xilinx board is successful with high correct classification rate.

In this paper, section 2 covers the theoretical justification and the major algorithm, which describes our algorithm and hardware mapping procedure of evolvable multiplayer neural network for data classification. A thresholding rule to construct digital representations for analog signal characteristics was developed based on mutual information, so that we can utilize the digital logic to build a binary logic artificial neural network. Then FPGA structures are evolved using entropy measures, statistical analysis and dynamic interconnections between logic gates. One complete design example is demonstrated in section 3. This includes data generation, threshold establishment, dynamic neural network generation, and hardware implementation of evolvable binary neural network. At last, the conclusion and reference are put at the end of this paper.

2. LEARNING STRATEGY

Multidimensional data sets can represent many real world problems from astronomy, electrical, engineering, remote sensing or medicine. The classification and clustering of these data sets are meaningful. To test our approach, we generated random data class sets for the learning and training procedures to simulate the real world problems. Each real type data represents one analog signal. Then, we construct threshold surface to separate data sets in multidimensional space and also to obtain binary codes for all signals. The further division of space into subspaces to clarify the classification of these binary codes is a core of our developed learning algorithms. In order to keep maximum information at one subspace, the selection of input functions to a layer of perceptrons is performed dynamically based on conditional entropy analysis. At the same time, the structure of one layer obeys the expansion and shrinking rule. Many layers can then be cascaded, with outputs of one layer connected to the inputs of the next layer, to form a feed-forward type network. The decision of making division of each space can be represented in a table, and each row of the table corresponds to the feature code classifying each class data set.

2.1 Data Preparation

In order to simulate the real world signals, we generate multi-dimensional random variables, chosen from the normal distributions. The mean value and covariance matrix is different for each class data set. Half of the data sets in each class are selected for learning procedure, and another half are used for training. These data sets projected onto two dimensions can overlap. Moreover, each class can have different ellipse shape with major axis in different directions.

2.2 Mutual Information and Thresholding

We set up a threshold in each space so that the original analog signal values in this space, which are higher than the corresponding threshold, are treated as logic true, and others are logic false. In this way, a threshold plane roughly separates data sets in one-dimensional space and binary data values are used to define learning functions.

Let $X = \{0, 1\}$ be a binary random variable with $P\{X = 1\} = p$ and $P\{X = 0\} = 1 - p$, then

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p) \qquad (1)$$

where H(p) is called Binary Entropy Function.

For a class problem, the mutual information I(X; Y) satisfies the following equations:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \quad (2)$$

$$H(X) = \sum_{i \in function} -p_i \log_2 p_i - (1-p_i)\log_2(1-p_i) \quad (3)$$

$$H(Y) = \sum_{j \in class} -p_j \log_2 p_j - (1-p_j)\log_2(1-p_j) \quad (4)$$

$$H(X,Y) = -\sum_{x \in function} \sum_{y \in class} \{p(x,y)\log_2 p(x,y) + [1-p(x,y)]\log_2[1-p(x,y)]\ \} \quad (5)$$

Where H(X) is called function entropy, H(Y) is called class entropy, and H(X,Y) is joint entropy. Since when we go from one layer to another layer of our neural network, the class distribution does not change, we only need to calculate the difference between the function entropy H(X) and the joint entropy H(X,Y) to measure how much mutual information we gain by dividing one space into subspaces. Furthermore, because H(X) is less than H(X,Y) in each space, the minimum difference of H(X,Y) and H(X) corresponds to the maximum mutual information I(X;Y). We call this procedure of obtaining logic signal from original data the thresholding rule.

2.3 Expansion and Shrinking Rules

Suppose in a multidimensional space, there are n logic variables $x_1, x_2, \mathrm{L}\ x_n$. Each two Boolean variables from these variables can be combined to make four Boolean functions $F(x_1, x_2),\ F(\bar{x}_1, x_2),\ F(x_1, \bar{x}_2),\ F(\bar{x}_1, \bar{x}_2)$.

In this way, for n logic variables, there are $(4 \cdot C_n^2)$ new combinational logic functions, and the original n logic variables can be considered as additional functions. Now we have totally $(4 \cdot C_n^2 + n)$ logic functions. We call this procedure for generating new combinational logic functions the expansion rule.

In order to get rid of the redundant parts in the expanded $(4 \cdot C_n^2 + n)$ logic functions, in this paper, we choose n logic functions, which correspond to the n maximum mutual information values $I_1(X;Y) \sim I_n(X;Y)$. The calculation of mutual information is similar with what we described in part 2.2. But here we don't need to establish thresholds any more since the binary logic codes have been prepared. As a result, the learning rate is greatly improved with a good numerical performance. We call this procedure the shrinking rule.

2.4 Layer Generation

One major point of learning is to generate layer structures for the binary neural network. The following steps describe the layer generation procedure:

Step 1. After we set up thresholds for n dimensional analog data sets, we obtain binary signals, which are the n input logic variables $x_1, x_2, \mathrm{L}\ x_n$. Suppose we have m classes of data sets. We apply the expansion and shrinking rules until there is no more improvement of mutual information. As a result, the function that corresponds to the final maximum mutual information is the selected characteristic function f_1 for the first layer of our binary neural network. The logic vector selections made during the expansion and shrinking procedure correspond to the logic gates, which are the basic units in each layer structure. We use the following structure in the Matlab program to represent these logic gates:

sub_stamp_order =

[li sub_num_layer subi subj
li sub_num_layer subi -subj
li sub_num_layer -subi subj
li sub_num_layer -subi -subj]

where $1 \le i \le n \ and \ 1 \le j \le n$, 'li' records the top layer number, the 'sub_num_layer' records the sub layers generated between the current top layer and the next top layer. Only when the neural network starts to evolve its next top layer, the new characteristic function will be generated, and the decision of dividing the new subspace will be made. 'subi', 'subj' record the branch order of previous sub layer, and negative sign corresponds to logic '0', otherwise logic '1' is assumed for the function value in the generated subspace. Since the 'sub_stamp_order' structure is arranged in decreasing order, the selection made for each sub layer can be recorded in another variable called 'sub_layer_orders' and can easily facilitate tracing back the sub layer structures in the designed hardware.

Step 2. Since further division is always bounded by the previous space partition, we should go through the following expansion procedure:

First, start with the original n input variables $x_1, x_2, \mathrm{L}\ x_n$ at the beginning of new space division. Then pick up function f_1 and one of the logic variables x_i, and generate the expansion functions:

$F(f_1, x_i),\ F(\bar{f}_1, x_i),\ F(f_1, \bar{x}_i),\ F(\bar{f}_1, \bar{x}_i),$ where $1 \le i \le n$.

So there are totally 4n new combinational logic functions plus n input variables. Next the new n logic variables should come up from 5n logic functions that correspond to the maximum mutual information sets $I_1(X;Y) \sim I_n(X;Y)$. If there is an improvement in the maximum mutual information, we will update the previous n logic variables with the new logic variables, and apply expansion and shrinking rule again to generate the new sub layers. Otherwise, we stop and start building the next layer and make decisions on how to divide the new subspace. As a result, function f_2 is generated, which corresponds to the maximum mutual information.

Step 3. Repeat procedures described in step 2 combining one of the logic variables x_i with all functions $f_1 f_2 \text{ L } f_j$ (j is the number of top layers) to make (j+1)-tuple logic pairs to apply the expansion procedure. The layer generation will come to the end when m-1 division of subspace is made. At last, we will generate m-1 function codes $f_1\ f_2 \text{ L } f_{m-1}$.

2.5 Class Characteristic Code Table

The division of the data space can be represented by a binary code. It starts with a binary value of the selected logic function f_1 and partitions the space into two parts. The decision on whether to divide subspace 1(0) or subspace 1(1) depends on the maximum mutual information calculation. In order to make quick selection, we decide that if we divide subspace 1(1) into another two parts according to logic value of f_2, we don't care what is the logic value of f_2 in subspace 1(0). This decision of selective partition is represented in a table by logic 0, logic 1, or –1 (do not care). Suppose that we have five class data sets. Four divisions of space can be summarized and shown in the class characteristic code table 1.

Table 1 describes the subspace division structure. First, in the original space 1, the subspace corresponding to logic 1 of function f_1 is divided into two subspaces, which define space 2. Actually, now data space is divided into three parts, corresponding to 'do not care', logic 1, and logic 0 value of function f_2.

Table 1. Class characteristic code table for space division

Space1	Space2	Space3	Space4
0	-1(do not care)	0	-1
0	-1	1	-1
1	0	-1	-1
1	1	-1	1
1	1	-1	0

In the next step, we can see from the above table that 'do not care' part of subspace 2 is further divided into two subspaces by logic values of function f_3 creating space 3. Similarly, the part of subspace, which corresponds to logic 1 of function f_1, logic 1 of function f_2, and 'do not care' of function f_3, is divided into two subspaces creating space 4.

2.6 Learning Decision

The final learning decision is based on comparing the logic function values in each row of class characteristic code table with class function values $f_1\ f_2 \text{ L } f_{m-1}$ for data from each class. Suppose the function codes of class 2 has the maximum probability for the function values specified in the second row of Table 1, then (0 –1 1 –1) will be the characteristic code for class 2. Actually, the learning decision is to relate each class with the corresponding row in the class characteristic code table.

Since layer generation procedure builds up a binary neural network, and learning decision identifies each class with its characteristic class code, we have evolved a binary neural network for classification.

Notice that no prior assumption was made regarding the organization of the neural network like the number of neurons, the number of layers or over the layer structure. All the computations necessary to evolve the organization of the neural network, type of logic elements used and interconnection made can be performed locally on a large number of the processing elements inside the programmable chip structure. Having finished the design of neural network, we are ready for the testing procedure.

2.7 Testing

Testing data should first be translated into binary codes by applying the same thresholds as those evolved in the learning procedure. Then the binary testing codes are fed into the evolved binary neural network. They will generate a set of function codes $f_1'\ f_2'\text{ L }f_{m-1}'$ for each class. At last, comparing these function codes with the characteristic code for each class, we can tell whether the

test is successful or not. The success rate can tell us how well our design classifies data.

2.8 Hardware Design Consideration

Because each class logic data set will flow through the same neural network structure, the highly parallel structure can be built up in hardware to implement quick on board training. The characteristic class code can be saved on board, so that comparators can judge the success or failure of testing data easily. Moreover, the finite state machine structure can facilitate the selection and display of different class results. On board testing is performed on generated test data applied to the final design implementation. We wrote VHDL codes and used Active VHDL tool to compile our source codes, and make functional verification. Finally, we use Xilinx foundation tools to do synthesis, place and route, timing verification and bit file generation. The bit file was downloaded to a Xilinx board with XL4010 chip. Both the evolved logic structures and test data were stored inside the FPGA chip for fast and reliable verification.

3. DESIGN EXAMPLE

We generated six dimensional data sets for six classes, and with $(300 \quad 600 \quad 300 \quad 300 \quad 300 \quad 300)$ samples in the successive classes. Half of the data sets in each class are used for learning, and another half are used for testing. The distribution of learning and testing data is illustrated in the figure 2. Setting up the threshold values is based on the maximum mutual information calculation simplified to the minimum difference between joint entropy and function entropy. This is illustrated in figures 3, and 4.

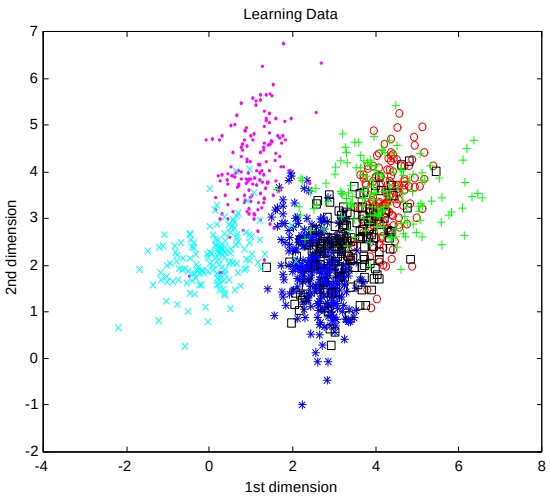

Fig. 2. Learning data distribution in the 1st and 2nd dimension

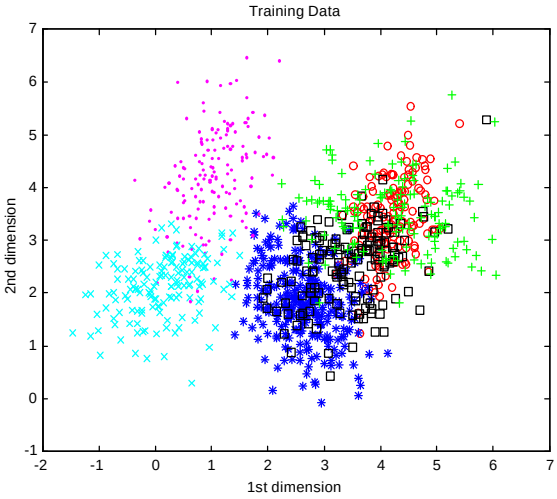

Fig. 3. PART1-Testing data distribution in the 1st and 2nd dimension

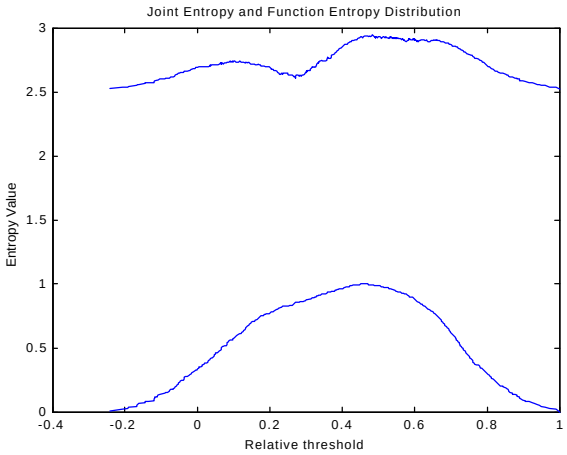

Fig. 3. PART2- Joint entropy and function entropy distribution

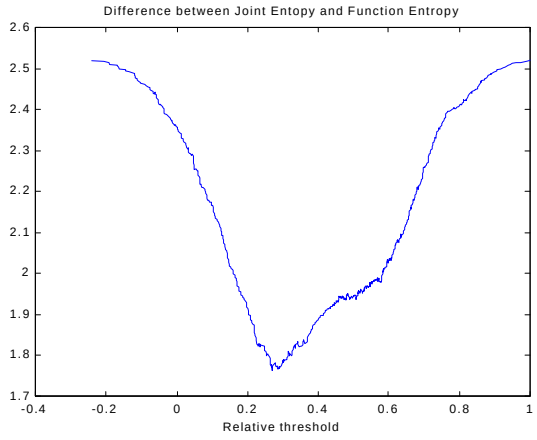

Fig. 4. Difference between joint entropy and function entropy distribution for threshold set up

The major learning results are represented in three tables: Binary neural network layer structure table (Table 2), class characteristic code table(Table 3) and learning decision table(Table 4).

Table 2. Binary neural network layer structure table

Column a → top layer no. Column b → sub layer no.

Column c, d→ branch 1 and 2 input of combinational logic gate

Minus sign → inverted input

a	b	c	d	a	b	c	d	a	b	c	d
1	1	1	0	3	1	-2	-6	4	1	-2	4
1	1	4	0	3	1	4	-6	4	1	4	-5
1	1	3	0	3	1	6	0	4	1	-4	-5
1	1	2	0	3	1	1	6	4	1	4	0
1	1	6	0	3	1	1	-6	4	1	-1	4
1	1	5	0	3	1	-3	6	4	1	-1	-4
2	1	-3	-5	3	2	-1	-2	4	2	1	2
2	1	-2	-6	3	2	1	0	4	2	1	0
2	1	2	5	3	2	1	-3	4	2	1	-3
2	1	2	3	3	2	1	-4	4	2	1	4
2	1	-5	6	3	2	1	5	4	2	1	5
2	1	1	-5	3	2	1	-6	4	2	1	-6
2	2	-3	-4					5	1	3	0
2	2	-4	6					5	1	1	3
2	2	-1	-2					5	1	1	-3
2	2	-2	-4					5	1	2	3
2	2	1	-2					5	1	2	-3
2	2	1	0					5	1	-3	5

Table 3. Class characteristic code table

0	-1	-1	0	-1
0	-1	-1	1	-1
1	0	-1	-1	0
1	0	-1	-1	1
1	1	0	-1	-1
1	1	1	-1	-1

Table 4. Learning decision table

Column a: class id. Column b: row number of Table 3

a	b
1	5
2	6
3	2
4	4
5	3
6	1

Table 3 not only represents the class characteristic code, it also describes how the space is divided into subspaces. Table 4 shows association of the class id number with the row number of Table 3. According to tables 2, 3 and 4, we can easily build up the binary neural network.

The hardware implementation graphs are demonstrated in figures 5 and 6. Figure 6 gives the logic gates configuration inside the layer structures in figure 5. According to Table 2, each layer has its typical logic connections.

The final training results are described in Table 5, where columns correspond to correct classification rate for different classes obtained with 100% declaration rate for test data.

Table 5. Training success ratio measurement

class no.	1	2	3	4	5	6
success ratio	0.93	0.93	0.77	0.83	0.77	0.95

4. CONCLUSION:

We presented a new schema and self-organizing procedure for evolvable logic neural network for pattern classification. This procedure can be implemented in the programmable hardware allowing design of sophisticated neural networks without the computational burden of the off-line, supervised learning. Classification of the learning data results from this self-organizing structure. A demonstration project was implemented using Xilinx technology and was successfully tested on a set of randomly generated data. Further work in this area carry a fascinating promise of hardware design, which will organize itself depending on the type of problem that is supposed to solve. Although this approach is different from genetically motivated evolutionary algorithms, it achieves similar objectives within simple structures of logic components. The next stage of our research will be directed on designing unique programmable architectures with computing elements built to locally estimate quality of information produced by logic components.

References

[1]. Machine Learning: ECML-93, Proc. European Conf. on Machine Learning, P.B. Brazil (ed), [ECML93]: published by Springer, New York, NY, USA, 1993

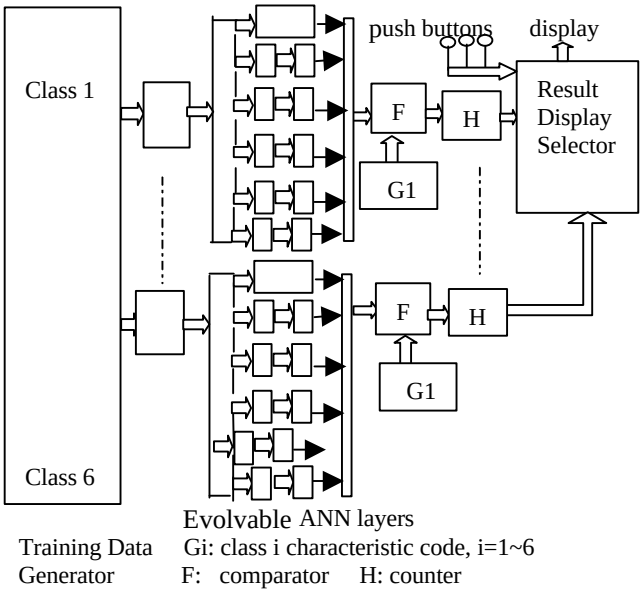

Fig. 5. Hardware block diagram of the implemented design

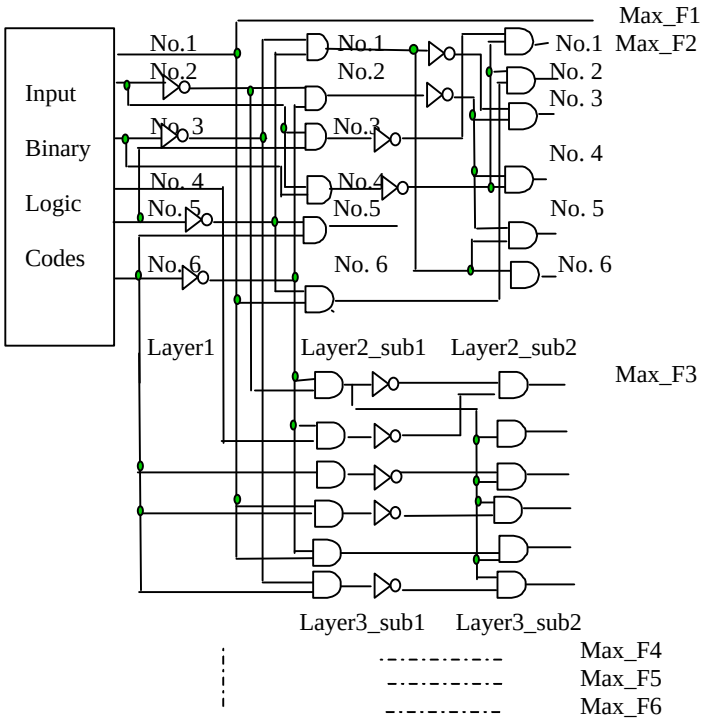

Fig. 6. Basic logic units in each layer

[2]. Th. B™ck and H. –P. Schwefel, "An overview of evolutionary algorithms for parameter optimization", Evolutionary Computation, 1(1): pp. 1-23, 1993

[3]. V. W. Porto, D.B. Fogel, L.J. Fogel, "Alternative neural network training methods", IEEE expert 10, pp. 16-22, 1995

[4]. L. Bull, "On model-based evolutionary computation", Soft Computing, Volume: 3, Issue: 2, September 23, 1999, pp. 76-82

[5]. Collins R, Jefferson D., "An artificial neural network representation for artificial organisms", In: Schwefel HP, Manner R(Eds). Parallel Problem Solving from Nature, Springer, Berlin, pp 249-263, 1991

[6]. Whitley D, Dominic S. Das R., "Genetic reinforcement learning with multiplayer neural networks", In: Belew BK., Booker LB(Eds). Proc 4th Int. Conf. on Genetic Algorithms, Los Allos, CA: Morgan Kaufmann, pp 562-569, 1991

[7]. Eberhart RC., "The role of generic algorithms in neural network query-based learning and explanation facilities", In: Whitley LD, Schaffer JD(Eds). COGANN '92: International Workshop on Combinations of Generic Algorithms and Neural Networks, IEEE, pp 169-183, 1992

[8]. D.W. Opitz and J. W. Shavlik, "Actively searching for an effective neural network ensemble", Connection Science, 8(3&4): 337-354, 1996

[9]. X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural network", IEEE Tran. On Systems, Man and Cybernetics, 28B(2), 1998

[10]. C. M. Friedrich, "Ensembles of evolutionary created artificial neural networks and nearest neighbor classifiers", Advances in Soft Computing, Proceedings of the 3rd On-line Conference on Soft Computing in Engineering Design and Manufacturing (WSC3), Springer , June 1998.

[11]. Pujol, João Carlos Figueira, Poli, Riccardo, "Evolving the Topology and the Weights of Neural Networks Using a Dual Representation", Applied Intelligence, Volume: 8, Issue: 1, January 1, 1998, pp. 73-84

[12]. M. Valle, D.D. Caviglia, and G.M. Bisio, "An experimental analog VLSI neural network with on-chip back-propagation learning", Analog Integrated Circuits &

Signals Processing, Kluwer Academic Publishers, Boston, vol. 9, pp. 25-40, 1996.

[13]. M. Valle, H. Chiblé, D. Baratta, and D.D. Caviglia, "Evaluation of synaptic multipliers behaviour in the back-propagation analog hardware implementation", In Proceedings of ICANN'95 Fifth International Conference on Artificial Neural Networks, vol. 2, pp 357-362, Paris 9-13 October, 1995.

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