Control
and Cybernetics
VOL. 20 (1991) No. 2

Logic operations with neural networks

by

Bohdan Macukow

Warsaw University of Technology
Institute of Mathematics
00–661 Warsaw, POLAND

Henri H. Arsenault

Centre d'Optique, Photonique et Laser,
Université Laval
Québec, P.Q., CANADA G1K 7P4

Most publications on neural networks focus on pattern recognition and associative memories. This paper is designed to present a new but promising area for neural nets application – logic operations. It is shown that a multilayer system composed of simple identical elements can perform any boolean function of two, three or more variables. Moreover, the system combining with boxes and CAM networks can perform any complex boolean operation.

1. Introduction

There are many similarities between an electronic computer and a neural network. Both systems are composed of a large number of simple processing units

– gates or neurons. On both systems these elements perform simple operations but the whole network can perform very complicated tasks.

The possibility to realize the equivalent of electronic gates using neuronlike elements can be an important step toward a creation of a computer based on neural networks.

There is a big difference in speed. A single neuron transmits about 1000 pulses per second, while silicon chip can handle a billion or more. The high degree of connectivity in neural nets (each neuron is connected to thousands of others, while electrical gates usually have only a few interconnections), allows neural nets to work as a parallel processor which can perform a huge number of simple operations per second.

Some recent papers dealt with the application of optical structures to realize logic operations of a few variables [1, 2, 3]. Neural nets with their simple elements and massive parallelism can perform most logic operations with high speed and great accuracy. Logic gates based on this approach could be a step towards a creation of a general – purpose optical computer.

One of the interesting and promising applications of multilayer neural nets is their ability to perform logical operations. In our previous papers [4, 5] a few simple neural nets able to perform the **XOR** logical operation were introduced. A universal three – layer network model which can perform eight logic operations was also shown, but the network contained one element with an artificially increased threshold value.

In the section 2 of this paper we first briefly recall some general ideas of a presentation of the logical functions of n variables. We also present a new, improved version of previous module (see Fig. 2), a four–layer network with all elements identical as well as a universal five–layer module for three – element logical operations (Fig. 3).

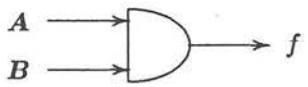
Two other models able to perform any logical operation of n variables are shown in section 3. The first model is composed of the five layers (for $n > 4$): the input layer, the output layer and three intermediate (hidden) layers. For $n = 2$ and $n = 3$ there are necessary only 1 and 2 intermediate layers respectively. The special structure of the second model allows to realize the same logical functions with only one intermediate layer. Moreover, if the network is devoted to perform only 2^n elementary logical operations of n variables – the output layer can be removed. Concluding remarks in the section 4 suggest some possible further generalizations and improvements of the models.

a)

A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

b) $f = AB$

c)



d)

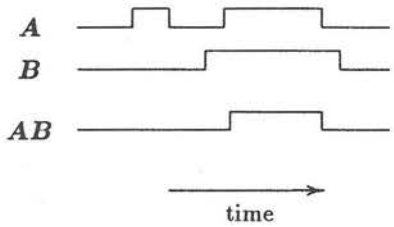


Figure 1. The four presentations of the **AND** function. a) a function table; b) an algebraic expression; c) a block diagram; d) a timing chart.

2. Logic operations

There are four method of displaying a Boolean function which a logician uses to characterize a net of switching elements [6]. They are

1. a function or truth table
2. an algebraic expression
3. a block or logic diagram
4. a timing chart (wave forms).

Figure 1 shows an **AND** function of two variables in each of its four presentations.

The number of significant Boolean functions is of interest. There are 2^{2^n} functions of n variables, and this number goes up dramatically with the number of variables n. Values up to four variables are given in Table 1.

Any logical function can be written in a canonical form. An expression is said to be in a canonical *sum-of-products* form when variables are logically ANDed into groups (called minterms) that are logically ORed to form a function. Every variable appears in every minterm once in the canonical sum-of products form. All 2^n minterms of n variables can be generated in a network of $n+1$ levels, and the minterms can be combined into arbitrary functions in an additional level.

number of variables n	number of functions of n variables
1	4
2	16
3	256
4	65 536

Table 1. Table of numbers of function of n variables.

2.1. Functions of two variables

The canonical form for a function of two variables is

$$f = \overline{AB} f_0 + \overline{A}B f_1 + A\overline{B} f_2 + AB f_3 \tag{1}$$

Each term of this canonical form (minterm form) corresponds to one row of a function table.

Any of the required operations may be achieved by a multilayered universal logic module shown in Fig. 2. In the network we distinguish input neurons (input layer) to which excitatory input signals are applied, assuming the values 0 or 1. There are a two intermediate slabs and one output element. The lines with arrowheads between the input and the first intermediate slab and between the intermediate slabs represent interconnections having weights equal to +1. The lines without arrowheads represent inhibitory interconnections with weights equal to −1. The neurons of the network are identical binary elements with threshold equal to zero. The left element in the input layer (shown white), is always activated by the application of an input signal equal to +1. This is necessary to perform operations for which global zero input ($A = 0$ and $B = 0$) leads to a non–zero output.

With the input vector IN

$$IN = [1, A, B] \tag{2}$$

the total input to the first intermediate slab is equal to

$$X = IN \star W^1 \tag{3}$$

where $\star$ indicates a matrix product and

$$W^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

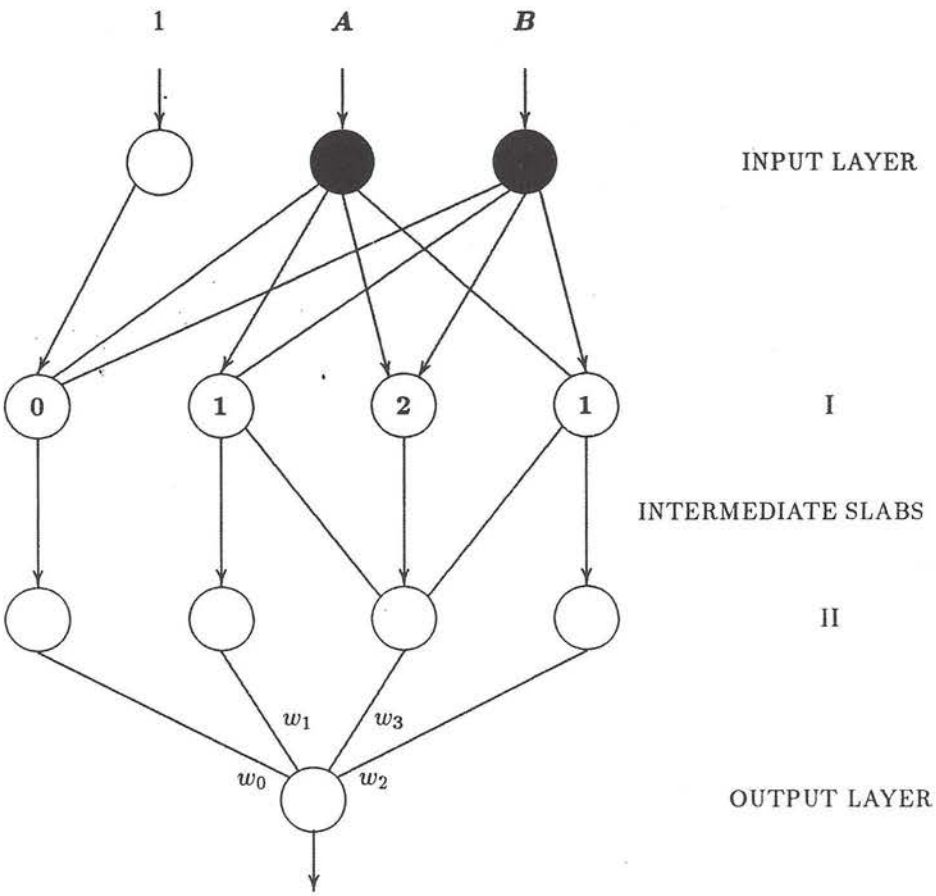


Figure 2. A universal logic module programmable for any of the 16 two – element logic operations.

is the matrix of connection weights between the input layer and the first intermediate slab. The threshold binary function Φ gives the layer output

$$\hat{\boldsymbol{X}} = \Phi(\boldsymbol{X}) = \begin{cases} 1 & \text{for } x_i > 0 \\ 0 & \text{otherwise, } i = 1,2,3,4. \end{cases} \tag{4}$$

The total input to the second intermediate slab is defined by

$$\boldsymbol{Y} = \hat{\boldsymbol{X}} \star \boldsymbol{W}^2 \tag{5}$$

where

$$\boldsymbol{W}^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

is the matrix of connection weights between the slabs. After thresholding, the output of the second intermediate slab

$$\hat{\boldsymbol{Y}} = \Phi(\boldsymbol{Y}) \tag{6}$$

multiplied by the matrix $\boldsymbol{W}^3$ yields the network output

$$\boldsymbol{OUT} = \Phi(\hat{\boldsymbol{Y}} \star \boldsymbol{W}^3) \tag{7}$$

where

$$(\boldsymbol{W}^3)^T = [w_0, w_1, w_2, w_3]$$

For the network shown in Fig. 2

$$\begin{aligned} \boldsymbol{OUT} &= \Phi\{\Phi(1-\boldsymbol{A}-\boldsymbol{B})w_0 + \Phi(\boldsymbol{A}-\boldsymbol{B})w_1 + \Phi(\boldsymbol{B}-\boldsymbol{A})w_2 \\ &\quad + \Phi[\Phi(\boldsymbol{A}+\boldsymbol{B}) - \Phi(\boldsymbol{A}-\boldsymbol{B}) - \Phi(\boldsymbol{B}-\boldsymbol{A})]w_3\} \end{aligned} \tag{8}$$

By setting values of weight w_0, w_1, w_2, and w_3 to zero or plus one the network can perform each of the 16 possible two–element logical operations such as **OR**, **XOR**, **AND**, and so on.

It should be pointed out that each of the 16 operations can be expressed in the form of Eq. (1), and the values of the coefficients f_0, f_1, f_2, and f_3 (equal to zero or plus one) are identical with the values of the weight w_0, w_1, w_2, and w_3 from Fig. 2.

Each term in Eq. (1) can be called the elementary term (minterm), and each of those four elementary logical operations can be obtained with the network shown in Fig. 2 with only one weight equal to +1. Any other operation can be expressed as a linear combination of those four elementary terms (operations 1, 2, 3, and 4 in Tab. 2). The values of weight for each operation are shown in Table 2.

The well–known operation **OR** $(\boldsymbol{A}+\boldsymbol{B})$, using logical theorems (expansion, distributive, commutative, De Morgan's etc.) can be rewritten into a canonical form

$$\begin{aligned} \boldsymbol{A}+\boldsymbol{B} &= \boldsymbol{A}(\boldsymbol{B}+\overline{\boldsymbol{B}}) + \boldsymbol{B}(\boldsymbol{A}+\overline{\boldsymbol{A}}) = \boldsymbol{AB} + \boldsymbol{A}\overline{\boldsymbol{B}} + \boldsymbol{BA} + \boldsymbol{B}\overline{\boldsymbol{A}} \\ &= \boldsymbol{AB} + \boldsymbol{A}\overline{\boldsymbol{B}} + \boldsymbol{A}\overline{\boldsymbol{B}} \end{aligned} \tag{9}$$

#	Operation		weights			
			w_0	w_1	w_2	w_3
1	$\overline{A}\overline{B}$	NOR	1	0	0	0
2	$A\overline{B}$		0	1	0	0
3	$\overline{A}B$		0	0	1	0
4	AB	AND	0	0	0	1
5	$\overline{A}\overline{B}+A\overline{B}$	$\overline{B}$	1	1	0	0
6	$\overline{A}\overline{B}+\overline{A}B$	$\overline{A}$	1	0	1	0
7	$A\overline{B}+AB$	A	0	1	0	1
8	$\overline{A}B+AB$	B	0	0	1	1
9	$A\overline{B}+\overline{A}B$	XOR	0	1	1	0
10	$A\overline{B}+\overline{A}B+AB$	OR	0	1	1	1
11	$\overline{A}\overline{B}+AB$	$\overline{\text{XOR}}$	1	0	0	1
12	$\overline{A}\overline{B}+A\overline{B}+AB$	$A+\overline{B}$	1	1	0	1
13	$\overline{A}\overline{B}+A\overline{B}+\overline{A}B$	NAND	1	1	1	0
14	$\overline{A}\overline{B}+\overline{A}B+AB$	$\overline{A}+B$	1	0	1	1
15	OUTPUT ALWAYS 0	FALSE	0	0	0	0
16	OUTPUT ALWAYS 1	TRUE	1	1	1	1

Table 2. The 16 possible two – element logical operations.

The network from Fig. 2 can perform this operation by setting the values of weights

$$w_0 = 0, \quad w_1 = 1, \quad w_2 = 1, \quad w_3 = 1 \tag{10}$$

which gives the required result.

2.2. Functions of three variables

A similar method can be applied to three–element logic. The network is shown in Fig. 3 with the same rules of connections as for the network shown in Fig. 2. The canonical form of any function (any among 256 possible operations) is described by the expression

$$\begin{aligned} f \;=\; & \overline{A}\overline{B}\overline{C}f_0 + A\overline{B}\overline{C}f_1 + \overline{A}B\overline{C}f_2 + \overline{A}\overline{B}Cf_3 \\ & + ABCf_4 + AB\overline{C}f_5 + A\overline{B}Cf_6 + \overline{A}BCf_7 \end{aligned} \tag{11}$$

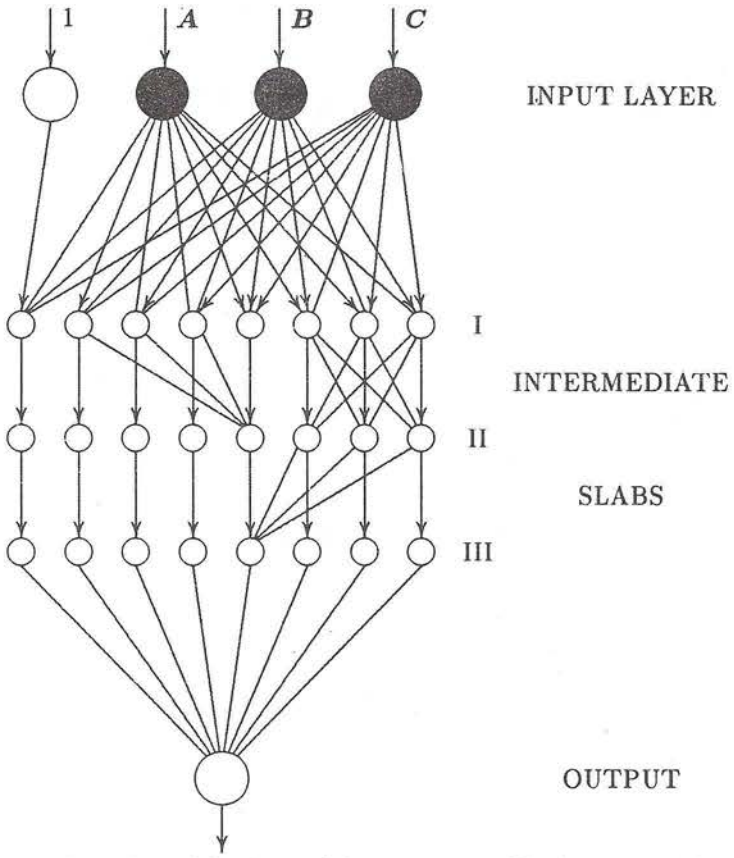


Figure 3. A universal logic module programmable for any of the 256 three–element logic operations.

Each of those eight elementary terms can be performed by the network shown in Fig. 3 with only one weight w_i equal to plus one. Any other from all 248 operations (the total number according to the Tab. 1 is equal to 256, including the 8 elementary), can be obtained by setting at least two of the values of coefficients f_i $(i = 0, 1, \ldots, 7)$ to plus one and the rest to zero.
The zero or plus one values of the coefficients $f_0, \ldots, f_7$ are identical with the values of weights w_i $(i = 0, 1, \ldots, 7)$ shown in Fig. 3.
To achieve the three–element logical operation **OR** $(A + B + C)$, the transformation to the canonical form is

$$A + B + C = A(B + \overline{B})(C + \overline{C}) + B(A + \overline{A})(C + \overline{C}) + C(A + \overline{A})(B + \overline{B})$$

$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + \overline{A}BC + \overline{A}B\overline{C} + \overline{A}\overline{B}C \quad (12)$$

which means that the f_i $(i = 1, 2, \ldots, 7)$, coefficients are equal to plus one as are the weights w_i $(i = 1, 2, \ldots, 7)$ also.
For **NAND** operation, the canonical form is

$$\begin{aligned}\overline{ABC} &= \overline{A} + \overline{B} + \overline{C} = \\ &= \overline{A}(B + \overline{B})(C + \overline{C}) + \overline{B}(A + \overline{A})(C + \overline{C}) + \overline{C}(A + \overline{A})(B + \overline{B}) \\ &= \overline{A}BC + \overline{A}B\overline{C} + \overline{A}\overline{B}C + A\overline{B}C + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C} \quad (13)\end{aligned}$$

which means that the f_i $(i = 0, 1, \ldots, 7)$ coefficients and the w_i $(i = 0, 1, \ldots 7)$ weights are equal to plus one.

For the network shown in Fig. 3 the following algebraic description (similar to Eqs (2) – (7)) can be obtained

$$IN = [1, A, B, C] \quad (14)$$

$$X = IN \star W^1 \quad (15)$$

where

$$W^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

Now

$$\hat{X} = \Phi(X) \quad (16)$$

$$Y = \hat{X} \star W^2 \quad (17)$$

where

$$W^2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix}$$

and

$$\hat{\boldsymbol{Y}} = \Phi(\boldsymbol{Y}) \tag{18}$$

$$\boldsymbol{Z} = \hat{\boldsymbol{Y}} \star \boldsymbol{W}^3 \tag{19}$$

where

$$\boldsymbol{W}^3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

$$\hat{\boldsymbol{Z}} = \Phi(\boldsymbol{Z}) \tag{20}$$

and finally

$$\boldsymbol{OUT} = \Phi(\boldsymbol{Z} \star \boldsymbol{W}^4) \tag{21}$$

where

$$(\boldsymbol{W}^4)^T = [w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7]$$

2.3. A multi–output network

As mentioned before another realization of the network functions can be obtained by replacing a single output element by a layer of the 2^{2^n} outputs elements. In this case the interconnections to the output layer are fixed, and each output element corresponds to one logical function. Each element of the second intermediate slab reacts to only one term of the canonical form (Eq. (1) and Eq. (11)) of a logical function.

For the networks shown in Fig. 2 and Fig. 3 the maximum number of elements in the output layer is equal to 16 and 256 respectively.

Of course the output value $\boldsymbol{OUT}$ will now be the output vector and the vector $\boldsymbol{W}^3$ in Eq. (7) and the vector $\boldsymbol{W}^4$ in Eq. (21) must be replaced by a matrix of connections between the second intermediate slab and the output layer. Equation (8) will give the value of i–th element of the vector $\boldsymbol{OUT}$ and the weights w_j, $(j = 1, 2, 3)$, have to be replaced by the elements of i–th column of a matrix $\boldsymbol{W}^3$, $(w^3(j, i), \quad j = 1, 2, 3)$.

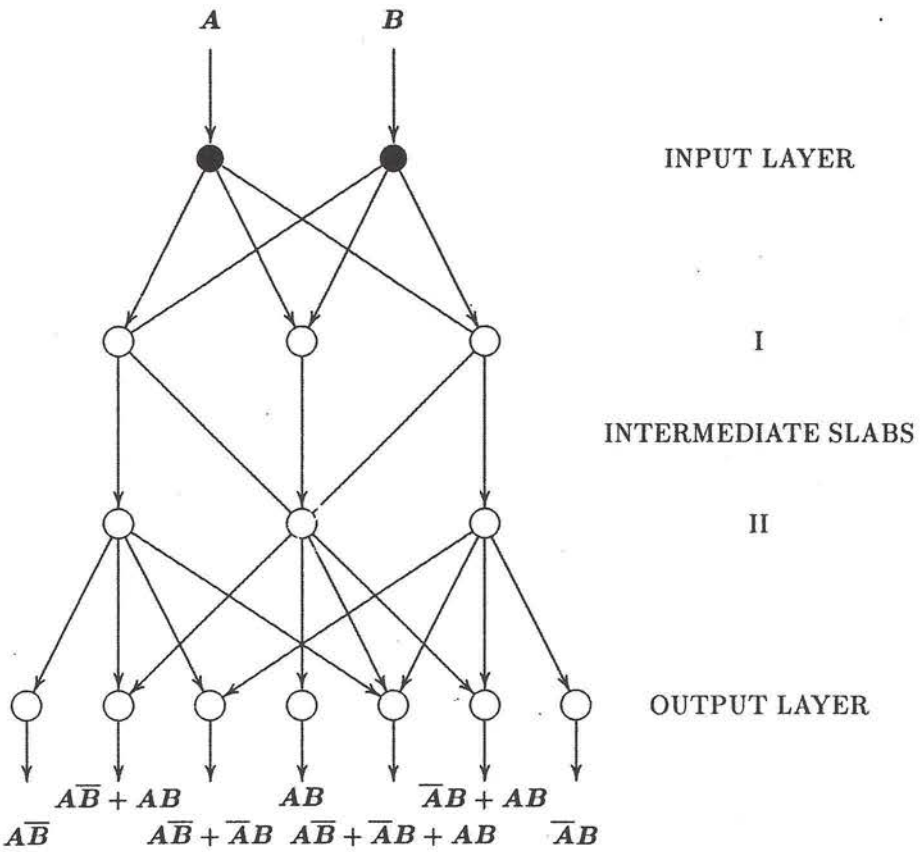


Figure 4. A universal logic module with separate outputs for any of seven two – element logic operations.

A simplified version of the network is shown in Fig. 4, and this network can perform eight of all 16 two–element logical operations, neglecting only those for which a total zero input lead to a non–zero output.

The output layer has seven elements and each element realizes the function indicated at the bottom of Fig. 4. For example, the symbol

$$f = A\overline{B} + AB$$

means that the output of the element is equal to one if the network input is

$$(A = 1 \text{ and } B = 0) \quad \text{or} \quad (A = 1 \text{ and } B = 1)$$

3. Models for n – variables

In Figure 5 the simple model for four–element logic with additional negative connections (weights are equal to -1) from the fixed input element (shown white) is presented. These inhibitory connections are applied to each element of the first intermediate layer (layer I) except the elements with only one positive and three negative inputs (for n – variables: one positive and $n-1$ negative). The remaining connections between the input layer and the layer I are the same as for the networks shown in Fig. 2 and Fig. 3. For clearance and legibility of the drawing only the connections between the input layer and one element of each kind in the layer I are shown. In general, the element marked with "m" inside a circle, has m – positive and $(n-m)$ – negative inputs from the elements (shown black) in the input layer.

The rules of the connections between the first and second intermediate layers (layers I and II) and the second and the third intermediate layers (layers II and III) are determined as follows (for n – variables):

1. The connection between every element (of layer I or II) and the element placed exactly below (in layer II or III) is always positive with the weight of $+1$ (the lines with the arrowheads). The remaining connections (the lines without the arrowheads) are negative with the weight of -1.

2. For $n > 3$ the output of each element in layer I with the three positive and $n-3$ negative inputs from the black elements in the input layer (these elements are marked with the number 3 inside a circle), is connected to the similar elements in the layer II (with the rules of p. 1).

3. For $n > 4$, selected elements of the layer I (marked with 3), are connected with each element (marked with 4) in the layer II. It means, that each element of the layer II (marked with 4) obtains the inhibitory signals from the elements (marked with 3 in the layer I) having 2 positive inputs among 4 positive inputs coming to the element placed exactly above it (marked with 4).

4. For $n > 5$, selected elements of the layer I (marked with 4) are connected with each element (marked with 5) in the layer II. Similarly as in p. 3, these elements marked with 4 have 3 positive inputs among 5 positive inputs coming to the element places exactly above the element marked with 5.

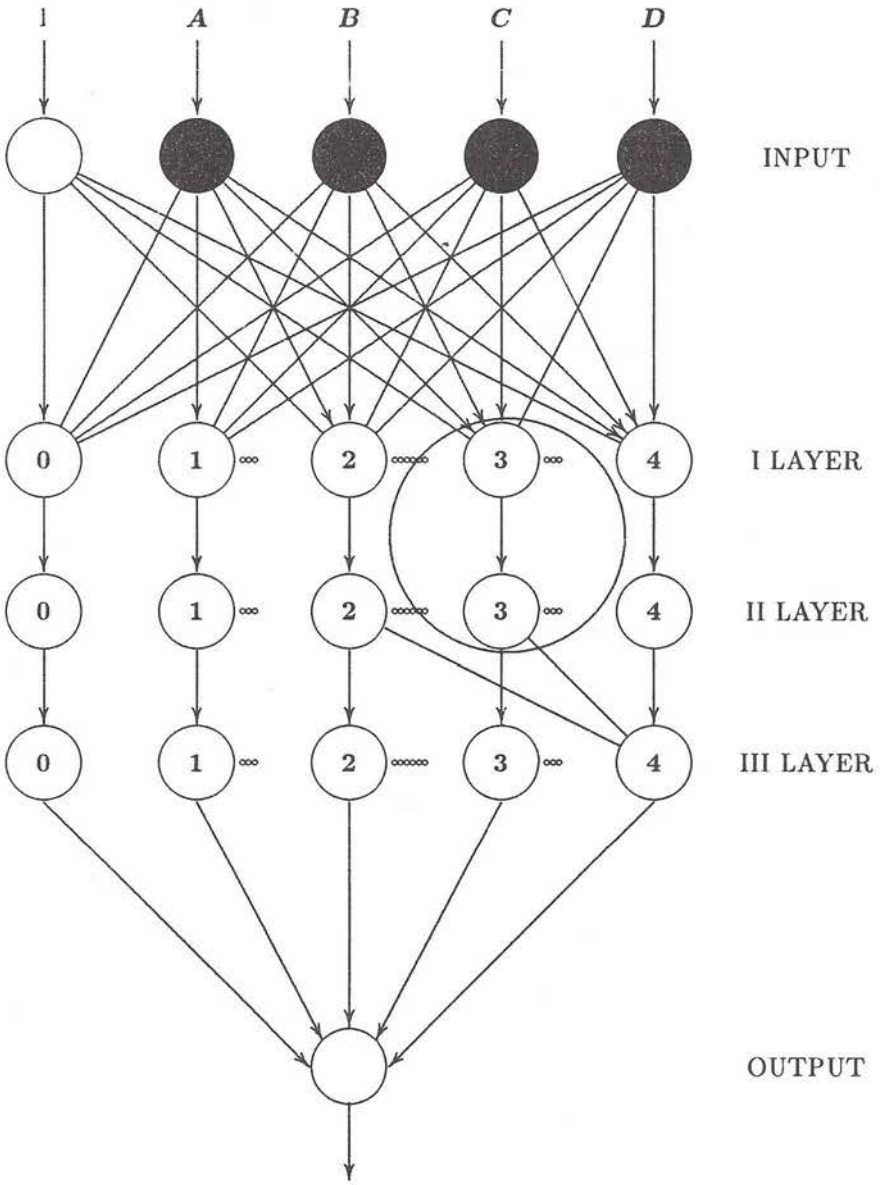


Figure 5. (a) The five-layer logic module programmable for any of the 65536 four - element logic operations.

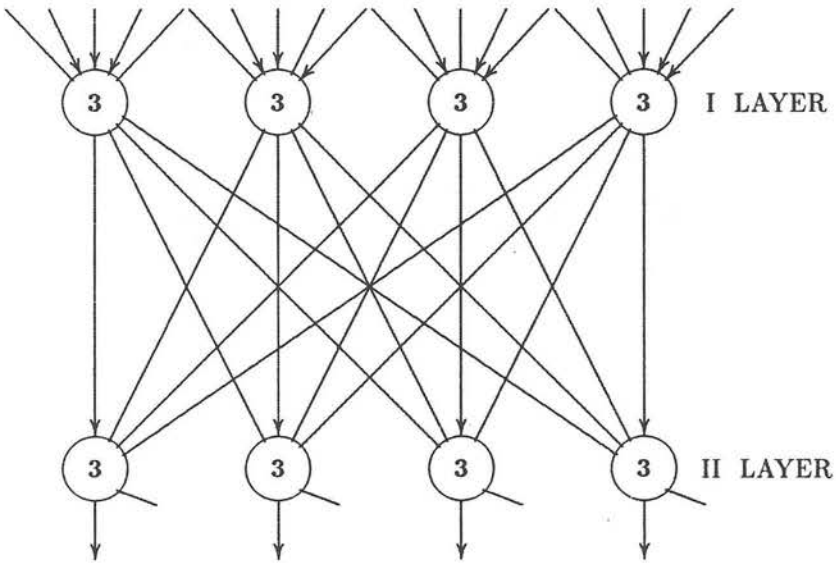


Figure 5. (b) The magnified part of the network shown in Fig. 5a – marked with the big circle.

5. For $n > 6$, selected elements of the layer I (marked with 4) are connected with each element (marked with 6) in the layer II, etc.

6. The output of each element in the layer II marked with m ($m = 4, 5, ..., n-1$) is connected to the similar elements in the layer III (with the rules of p. 1).

7. Every element of the layer II, marked with $2, 3, \ldots, n-1$ is connected to the element marked with n in the layer III.

In the output layer can be either one element, and then the output function is realized setting the proper values of the weights (zero or plus one) for the connections between the layer III and this single output, or the output layer can consists the 2^{2^n} elements with fixed connections from the layer III, and then every output element corresponds to one logical function (see sec. 2 p. 3).

This model needs a lot of elements (54 for $n = 4$, 103 for $n = 5$, 200 for $n = 6$ etc., in general $3 \cdot 2^n + n + 2$ for $n \geq 4$) and also many connections. The total number of connections can be reduced replacing most of the connections (described in p. 7) by the additional connections between Ist and IInd layer. For instance, connecting the elements marked with 3 in the layer I with the

element marked with n in the layer II removes the connections between the elements marked with 2,3 and 4 in the layer II and element marked with n in the layer III.

The alternation of the negative connections from the fixed input element (shown white in Fig. 5 and Fig. 6) yields the further simplification of the model.

For the model shown in Fig. 6, the weights of these connections depend of the number of positive inputs to the individual elements of the singular intermediate layer. The inhibitory connection between fixed input and element marked with "m" has the weight equal to "$m-1$" (m – is the number of the positive inputs to the element, and for $m = 1$ there is no inhibitory connection). That rule allows to reduce the number of intermediate layers to only one as well as the number of elements and connections.

For n – variables the model contains:

- $2^n + n + 2$ elements,
- $(n+1)2^n - n$ connections between the input and the intermediate layer,
- 2^n connections between the intermediate layer and the output element.

If the network is devoted to perform only the elementary functions (defined by a single term in the canonical form), the output element is unnecessary because each element of the intermediate layer reacts to only one term (minterm) of the canonical form. However, if the network is devoted to perform any of 2^{2^n} different logical functions of n variables, the output layer has to be composed of 2^{2^n} elements with the 2^{2^n+n-1} connections between the intermediate layer and the output layer.

For the network of this type with 3 variables we obtain the following algebraic description

$$\boldsymbol{IN} = [1, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}] \tag{22}$$

$$\boldsymbol{W}^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & -1 & -1 & -2 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 \end{bmatrix} \tag{23}$$

$$(\boldsymbol{W}^2)^T = [w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7] \tag{24}$$

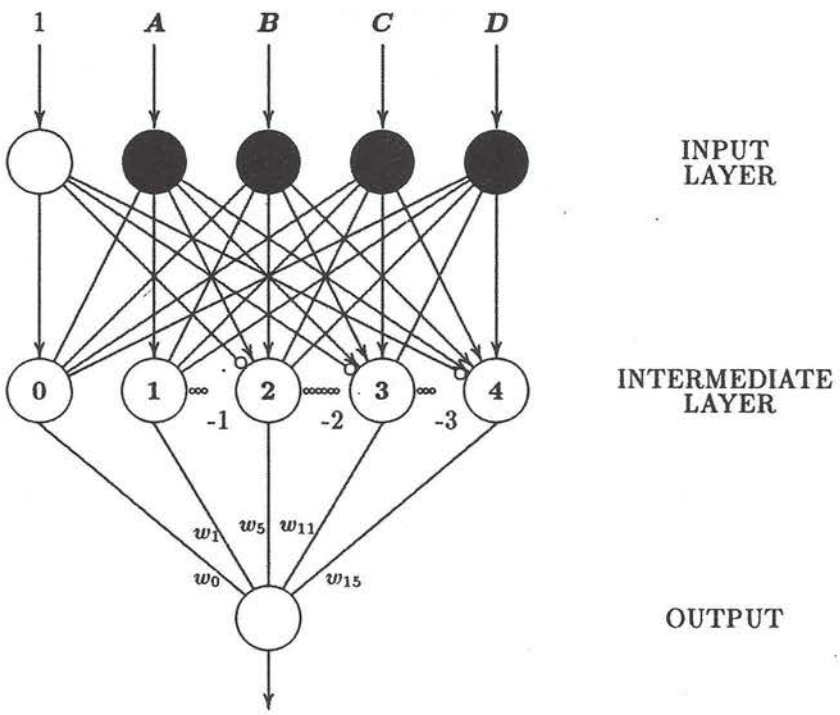


Figure 6. The three – layer logic module with variable inhibitions for four – element logic operations.

where $\boldsymbol{W}^1$ is the matrix of connections between the input layer and the intermediate layer,

$\boldsymbol{W}^2$ is the vector of connection weights between the intermediate layer and the output element.

$$\boldsymbol{Y} = \Phi(\boldsymbol{IN} \star \boldsymbol{W}^1) \tag{25}$$

and finally

$$\boldsymbol{OUT} = \Phi\left(\Phi(\boldsymbol{IN} \star \boldsymbol{W}^1) \star \boldsymbol{W}^2\right) \tag{26}$$

4. Conclusion

Neural networks are finding many areas of applications. Although they are particularly well–suited for applications related to associative recall such as content – addressable memories, neural net can be used in many other applications ranging from logic operations to the solution of optimization problems [4].

In our previous papers [4, 5] we described how the basic models can be trained to accomplish logical operations on input data, and in this paper we generalized these results to obtain a universal logic module able to accomplish any logical operations.

It is well known that with two basic logical operations **AND** and **NOT** (or simply a two–input **NAND** gate) any Boolean function can be simulated. It can be exploited to create complex logical nets composed of neuronlike elements to perform any complex boolean operation.

If the activation is multi–level instead of binary, and if the weights w_i in Figs 2, 3, and 4 are not confined to zeros and ones, the networks may perform intermediate logic functions. Of course the basic neuron element characteristic (type of threshold, function etc.) must also be changed.

The interesting aspect of doing logic operations on neural networks is the ability to accomplish precise logical operations on highly degraded data.

One may ask whether it is worthwhile to do logic by means of neural nets when cheap electronic devices are available. This is an open question, but there are at least five advantages to using neural nets:

1. The logic modules can be integrated into an all – neuron network using the same kinds of elements (neurons) as for the rest of the network, thus avoiding the mixing of different types of elements;

2. The logic operations can be learned, which means that the same modules can be used to do different logical operations;

3. The universal logic module can perform any logical operation depending only of the unit wiring;

4. The outputs of the logic operations can be used as the inputs to other neural system in order to carry out more complex tasks;

5. The basic modules can be combined with neural network associative memories to accomplish complex combinations of data and logic operations on corrupted data [4,7].

References

[1] HASSOUN M.H., ARRATHOON R., Logical signal processing with optically connected threshold gates, *Opt. Eng.* **25**, (1986), 056-068

[2] MURDOCA M.J., HUANG A., JAHNS J., STREIBL N., Optical design of programmable logic arrays, *Appl. Opt.*, **27**, (1988), 1651-1660.

[3] GUILFOYLE P.S., WILEY W.J., Combinational logic based digital optical computing architectures, *Appl. Opt.*, **27**, (1988), 1661-1673.

[4] ARSENAULT H.H., MACUKOW B., Beyond pattern recognition with neural nets, *Proc. SPIE*, **960**, (1988), 206-216.

[5] MACUKOW B., ARSENAULT H.H., Neural Networks for Logic Operations, *Proc. SPIE*, **1134**, (1989), 40-43.

[6] MALEY G.A., EARLE J., The Logic Design of Transistor Digital Computers, Englewood Clifts, New Jork, 1963.

[7] ARSENAULT H.H., MACUKOW B., Neural Networks Model for Fast Learning and Retrieval, *Opt. Eng.* **28**, (1989), 506-512.

Operacje logiczne wykonywane za pomocą sieci neuronowych

Większość publokacji związanych z sieciami neuronowymi skupia się na zagadnieniach rozpoznawania obrazów i pamięciach asocjacyjnych. Ten artykuł ma na celu przedstawienie nowego, lecz obiecującego zastosowania sieci neuronowych w operacjach logicznych. Wykazano, że wielowarstwowe systemy złożone z prostych jednakowych elementów mogą wykonać dowolną prostą operację boolowską o dwóch, trzech lub większej liczbie argumentów. Ponadto system złożony z i sieci CAM może wykonać dowolną złożoną operację boolowską.

Логические операции выполняемые с помощью нейронных сетей

Большинство публикаций связанных с нейронными сетями сосредоточено на вопросах разпознавания образов и ассоциационой памяти. Данная

статья преследует цель представления нового, но обещающего применения нейронных сетей в логических операциях. Показано, что многослойные системы, состоящие из простых одинаковых элементов могут выполнять произвольную простую операцию Буля с двумя, тремя и более аргументами. Кроме этого, система состоящая из блоков сети АСУ может реализовать произвольную сложную операцию Буля.