Decomposition of Boolean Function Sets for Boolean Neural Networks

Roman Kohut, Bernd Steinbach
Freiberg University of Mining and Technology
Institute of Computer Science
D-09596 Freiberg, Germany
email: {kohut, steinb}@informatik.tu-freiberg.de

Abstract
This paper presents a new type of neuron, called Boolean neuron. We suggest algorithms for decomposition of Boolean functions sets based on Boolean neural networks that include only Boolean neurons. The advantages of these neural networks consist in the reduction of memory space and computation time in comparison to the representation of Boolean functions by usual neural networks. The Boolean neural network can be mapped to a FPGA so that our new approach substitutes classical design methods of these circuits. We show as example the AND-decomposition of a Boolean function set into unique basic functions and their mapping to the general Boolean neural network.

1 Introduction
The decomposition of Boolean functions is an important approach in circuit design. It leads to multi-level logic circuits of digital systems. Such circuits can be mapped easily into field programmable gate arrays (FPGAs) [1]. FPGAs are widely used, especially in small and medium companies, because they are cost-effective for small and medium series of products, and very well suited for re-configurable computing, rapid prototyping and many other applications that require both high speed and reprogramming [2].

There are many approaches to use the neural networks technique for representing single Boolean functions or sets of Boolean functions [3-10, 13]. It was proven that all types of functional dependences can be represented using neural networks. A Boolean function is one of them. There are many approaches suggested for Boolean neural networks. A summary of their advantages and disadvantages can be found in papers by Wang and Chaudhari [3, 4], Lauria, Sette and Visco [6], Deolalikar [13], and our previously suggested approaches [5, 10]. It is well known that a first type of Boolean neural networks was introduced by McCulloch and Pitts in their seminal work [7]. In this paper we return to the root and apply neural network techniques to the compact representation and fast computation of Boolean functions. The main aim is the use of neural networks for decomposing a set of Boolean functions. We do not emphasize an efficient method for the representation of Boolean functions by neural networks, even though one is proposed.

This paper extends our previous research [5], where we have presented a Boolean neuron and the general structure of a neural network that includes only Boolean neurons. We propose the training and the use of algorithms for Boolean neural networks that substitute classical design methods of FPGAs. These algorithms use a special type of OR- and AND-decomposition of a set of Boolean functions. Moreover, these Boolean neural networks reduce memory space and computation time in comparison to the representation of Boolean functions by usual neural networks.

The remainder of this paper is organized as follows. Section 2 motivates and describes a new type of neural elements, called Boolean neuron that solve the problem of unacceptably large sizes of operating memory required for neural networks for Boolean data. Our new algorithms for training and using Boolean neural networks are described in section 3. Section 4 shows how a set of Boolean functions can be implemented as Boolean neural network using the AND-decomposition. Finally, section 5 summarizes the paper.
2 Boolean Neuron

The input signals and weighting coefficients of the neurons in modern neural networks are real or integer numbers. The output signal is determined by an activation function, and can be also a real or integer number, less often a Boolean value. It is well known that the representation of real and integer data in a computer requires sets of bits which is clearly inefficient for Boolean data because of unnecessary memory expenses. Since Boolean data need only one bit for each value we propose to use a neuron that operates directly with Boolean values and uses only Boolean operations. Such a type of neural elements is called a Boolean neural element (or Boolean neuron) [5]. Corresponding to the mathematical description of a usual neuron [12, 14], the Boolean neuron defines a relation between the vector of input signals \( \text{Inp}_B \) and output signal \( \text{Out}_B \):

\[
\text{Out}_B = f_B(\text{Inp}_B, w_B),
\]

where the index \( B \) indicates Boolean values and

\[
\text{Inp}_B = \{x_1, x_2, \ldots, x_N\}, \quad x_i \in \{0,1\},
\]

\[
w_B = \{w_1, w_2, \ldots, w_N\}, \quad w_i \in \{0,1\},
\]

\( f_B \) - Boolean dependence such as a Boolean transfer function,

\[
\text{Out}_B \in \{0,1\}.
\]

In the following we omit the index \( B \) because only Boolean neurons are considered.

Boolean neurons give us two advantages. First, the restriction to Boolean operations reduces significantly the time for converting the input vector into the output signal. An additional advantage of the Boolean neuron consists in the reduction of the necessary memory size. The structure of a Boolean neuron is shown in Figure 1.

![General structure of Boolean neuron](image)

Since the structure of such a Boolean neuron is unchanged in comparison to regular neurons, it is possible to design neural networks on the base of Boolean neurons. Moreover, the structure and the general computational process of such neural networks are unchanged, too [10 and 14].

In our previous work [5] the neural network model, called “Functional on the tabular functions set” (FTFS), was chosen as the basic structure of the neural network. The reason for this selection were the higher precision and shorter time for learning. For the same reason we will use FTFS-based neural networks.

A new type of neural networks, called Boolean neural networks (BNN), was created by substituting normal neural elements by Boolean neurons. Boolean neural networks belong, like FTFS networks, to the class of feed-forward neural networks. The training algorithm of BNN is typically a sequential training algorithm that adds hidden layers and hidden neurons during the training process. In our case the Boolean neural network has one input layer, one hidden layer and one output layer. The Boolean neurons on the hidden layer have different transfer functions. Each Boolean neuron on the hidden layer possesses a Boolean transfer function that is defined on the whole set or subset of input variables and is different from the Boolean transfer function of all other neurons in this layer. These Boolean functions are determined in the training process.

The mathematical description of the neuron with number \( z \) on the hidden layer is

\[
\text{Out}^{[z]} = f^{[z]}(\text{Inp}_z, w^{[z]})
\]
where 
\[
\begin{align*}
Out^{[j]} &= \text{output signal of the neuron with number } z, \\
f^{[j]} &= \text{transfer function of the neuron with number } z, \\
z &= \text{index } z = 1, \ldots, Z_N, \\
Z_N &= \text{number of neurons on the hidden layer}, \\
\land &= \text{Boolean operation „AND”,} \\
&\forall i \neq j; \ i, j \in [1, Z_N].
\end{align*}
\]

All neurons of the output layer have a fixed Boolean transfer function. This Boolean function connects the weighted inputs by one Boolean operation, such as “AND”, “OR”, “equivalence” or “exclusive-OR” (EXOR). The training process of Boolean neural network assumes a given Boolean operation for all neurons of the output layer. This Boolean operation specifies the type of decomposition of a set of Boolean functions, and we call it the basic operation. The expressions in the brackets of (3) and (4) determine that in the case of weight \( w^{[j]} = 1 \) the inputs signal \( \text{Inp}_i \) is an argument of the function \( f \), otherwise \( \text{Inp}_i \) is unimportant.

\[
Out^{[j]} = \sum_{i=1}^{Z_N} f(\text{Inp}_i \land w^{[j]} \lor \overline{w^{[j]}}) 
\]

where

\[
f \in \{ \text{AND, “equivalence”}\},
\]

\( j \) – number of neurons on the output layer of the Boolean neural network.

\[
Out^{[j]} = \sum_{i=1}^{Z_N} f(\text{Inp}_i \land w^{[j]}) 
\]

where

\[
f \in \{ \text{OR, EXOR}\}.
\]

Since the algorithms of training and the use of the Boolean neural network for the basic operation “XOR” (exclusive-OR) was described in our previous work [5], we will introduce in the following training and application procedures of the BNN with the other three basic operations. Additionally we describe the synthesis of the structure of Boolean neural networks. The main difference between BNN and common FTFS network will be shown in the description of these algorithms. Boolean neural networks need, like all feed-forward neural networks, two basic procedures, training and use. As shown in [5], the training algorithm of a Boolean neural network describes a special type of decomposition of a set of Boolean functions.

3 Decomposition algorithms

3.1 Equivalence-decomposition

The Equivalence-decomposition can be done similar to the EXOR-decomposition [5]. Applying the law of de Morgan, an EXOR expression can be transformed into an equivalence expression. Thus we have to start with the set of negated output functions, use the known algorithm for EXOR-decomposition [5] and negate finally the calculated functions of the hidden layer.

3.2 AND-decomposition

The aim of the AND-decomposition is a Boolean neural network that uses AND-operations as transfer function on the output layer. The function table of the set of Boolean functions is the base of all training algorithms of the suggested Boolean neural network. This table is also called implementation matrix \( A \). The matrix \( A \) has \( 2^N_x \) rows and \( N_y \) columns, where \( N_x \) is the number of Boolean variables, and \( N_y \) is the number of Boolean functions. The input signals of the network are argument values. These variables are omitted in (5) because we do not need the associated values in the training algorithm. Output signals are the values of the Boolean functions [5]. The rows of \( A \) are ordered by the decimal code of the input variables.
The aim of the training algorithm of the Boolean neural net is the representation of the set of Boolean functions by finite AND-polynomials of simpler shared Boolean base functions. The AND-operations of these polynomials are realized in the output layer of BNN. The base functions must be created by the Boolean neurons of the hidden layer.

Consider the training algorithm of the Boolean neural network. Each row of matrix $A$ has a coefficient $m_i$, and each column has a coefficient $n_j$. These weights $m$ and $n$ are computed by (6) and (7).

$$m_i = \sum_{j=1}^{N_v} a_{i,j}$$  \hspace{1cm} (6)

$$n_j = \sum_{i=1}^{N_v} a_{i,j}$$  \hspace{1cm} (7)

If all coefficients $m_i$ are equal to $N_v$, then the algorithm stops. Otherwise, we must determine the index of base row $I$ of matrix $A$. In order to do this, we define the two functions. The function min($v, N_v$) returns the minimum element of the vector $v$. The arguments of the function are initially a vector $v$ and its length $N_v$. The second function is numb($v_i$) that returns the index of the element $v_i$ in the vector $v$.

$$I = \text{numb(min}(m, 2^{N_v}))$$  \hspace{1cm} (8)

If there are more than one minimal values of the vector $m$ (assume there are $N_f$ such elements) then calculate the auxiliary vector $s$ by

$$s_i = \sum_{j=1}^{N_v} (n_j + a_{i,j})$$  \hspace{1cm} (9)

and the index of the base row $I$ by

$$I = \text{numb(max}(s, N_f)),$$  \hspace{1cm} (10)

where analogously to the function min, the function max($v, N_v$) has two arguments, the vector $v$ and its length $N_v$, but returns a maximal element of the vector $v$.

The base row $v$ of Matrix $A$ is defined by (11).

$$v_j = a_{I,j}$$  \hspace{1cm} (11)

We choose a column-vector $k$ from the columns of the matrix $A$ with number $j$ that corresponds to the condition (12), where $N_v$ is the set of column numbers $j$ with $v_j = 1$.

$$v_j = 1, \quad n_j = \text{max}(n, N_v)$$  \hspace{1cm} (12)

The subset $N_f \subseteq N_v$ includes all column numbers of the matrix $A$ which solve (12). If some of the columns in the subset $N_f$ are changed in previous cycles of the training procedure, then these columns can be replaced in the current matrix $A$ by columns of the primary matrix $A$ with the same number $j$. This substitution aims on a minimum weight of vector $k$, where the weight of vector $k$ is the number of its values 1.

$$k_i = \sum_{j \in N_f} a_{i,j}$$  \hspace{1cm} (13)

Subsequently, we define the vector of synaptic weights for neurons of the output layer of BNN by (14).
where $N_k$ is the set of row numbers $i$ with $k_i = 0$. Next we calculate a new matrix $A$ by (15).

\[
a_{i,j}^{(z+1)} = \begin{cases} a_{i,j}^{(z)}, & w_j = 0 \\ a_{i,j}^{(z)} \oplus k_i, & w_j = 1 \end{cases}
\]

(15)

where $\oplus$ is the Boolean operation exclusive-OR. We continue with the next cycle by calculating formula (6) and (7).

### 3.3 OR-decomposition

The aim of the OR-decomposition is a Boolean neural network that uses OR-operation as transfer function on the output layer.

The weights of rows $m$ and weights of columns $n$ of matrix $A$ are computed by (6), (7). The algorithm stops in the case of the OR-decomposition if all coefficients $m_i$ are equal to zero. Otherwise, we must determine the index of base row $I$ of matrix $A$. In order to do this, we find the maximum value of $m$ and determine the index of base row $I$ by (16).

\[
I = \text{numb}(\max(m, 2^N))
\]

(16)

If there are more than one maximal values of the vector $m$ (assume there are $N_I$ such elements) then we calculate the auxiliary vector $s$

\[
s_i = \sum_{j=1}^{N_I} a_{i,j} a_{i,j}
\]

(17)

and the index of base row $I$.

\[
I = \text{numb}(\min(s, N_I)).
\]

(18)

The base row $v$ of matrix $A$ is defined by (19).

\[
v_j = a_{i,j}
\]

(19)

We choose a column vector $k$ among the columns of the matrix $A$ with number $j$ that corresponds to condition (20), where $N_v$ is the set of column numbers $j$ with $v_j = 1$.

\[
v_j = 1, \quad n_j = \min(n, N_v).
\]

(20)

The subset $N_I \subseteq N_v$ includes all column numbers of matrix $A$ that solve (20). If some of the columns in the subset $N_I$ are changed in previous cycles of the training procedure, then these columns can be replaced in the current matrix $A$ by columns of the primary matrix $A$ with the same number $j$.

The aim of this substitution is a maximum weight of vector $k$.

\[
k_i = \bigwedge_{j \in N_I} a_{i,j}
\]

(21)

The synaptic weights for neurons of the output layer of BNN are defined by (22).

\[
w_j = v_j \wedge \bigwedge_{i \in N_k} a_{i,j}
\]

(22)

where $N_k$ is the set of row numbers $i$ with $k_i = 1$. Next we calculate a new matrix $A$ by (23).

\[
a_{i,j}^{(z+1)} = \begin{cases} a_{i,j}^{(z)}, & w_j = 0 \\ a_{i,j}^{(z)} \oplus k_i, & w_j = 1 \end{cases}
\]

(23)

where $\oplus$ is the Boolean operation exclusive-OR. We continue with the next cycle by calculating (6) and (7).
3.4 Using the Boolean neural network

As result of the training algorithm we get the structure of a Boolean neural network and its parameters, i.e. the transfer functions of hidden neurons and the weight coefficients of output neurons. Previously FTFS algorithms design the three-layer architecture: input layer, output layer and one hidden layer. The structure of BNN has $N_i$ inputs (input neurons) and $N_y$ output neurons. The number of neurons on the hidden layer $Z_N$ is defined during the training algorithm and is equal to the number of training cycles [5]. The unique Boolean base functions $k_z, \forall z \in [1, Z_N]$ are used as transfer function of neurons on the hidden layer of BNN and depend on input signals $x$.

$$k_z = f(x). \quad (24)$$

Based on (3), (4) and the structure of BNN, it follows that the Boolean function depends on the input signals $x$ and can be represented by (25) and (26).

$$y(x) = \bigwedge_{z=1}^{Z_N} (w_{z,j} \land k_z(x) \lor \overline{w_{z,j}})$$

$$y(x) = \bigvee_{z=1}^{Z_N} (w_{z,j} \land k_z(x))$$

Equation (25) describes the procedure of using the Boolean neural network for AND-decomposition, and equation (26) for OR-decomposition, respectively.

4 Example

In this section we illustrate the synthesis of structure during the training process of BNN and verify the obtained result by using algorithms of Boolean neural networks. We consider the AND-decomposition of a set of Boolean functions $y_1, y_2, \ldots, y_{10}, y_j = f(x_1, x_2, x_3)$. The function table for these ten Boolean functions has $2^{N_i} = 2^3 = 8$ rows and $N_y = 10$ columns (see Table 1). The weights $m$ are computed for each row of matrix $A$ by (6). Because not all of these weights are equal to $N_y = 10$, the algorithm does not stop and the coefficients $n$ are computed by (7). The vector $m$ includes the smallest value 1 only once. Thus, the index $I$ of the base row is 5, and the vector $v$ is calculated by (11) as the complement of this base row. The largest coefficient $n_{j1} = 5$ occurs only once in the last column in combination with a value 1 in the vector $v$. Since the subset $N_j = \{10\}$ includes only the element 10, the last column of $A$ is selected as vector $k_1$ (13). The vector $k_1$ is equal to zero for the elements $N_{k0} = \{2,5,7\}$. Note, the top element has the index 0. According to (14) the first vector $w_1$ of synaptic weights is calculated; values one in the rows of $N_{k0}$ cause additional zeros in $w_1$ (see Table 1).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
<th>$y_6$</th>
<th>$y_7$</th>
<th>$y_8$</th>
<th>$y_9$</th>
<th>$y_{10}$</th>
<th>$m$</th>
<th>$k_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$n$</th>
<th>4</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>2</th>
<th>2</th>
<th>5</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$w_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Using formula (15) and the vectors $k_1$ and $w_1$, we calculate the new matrix $A$ of Table 2. The values of the third and the last row of the matrix became ones. In the next cycles, the same steps are repeated.
Table 2. Matrix $A$ after the first cycle of training

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
<th>$y_6$</th>
<th>$y_7$</th>
<th>$y_8$</th>
<th>$y_9$</th>
<th>$y_{10}$</th>
<th>$m$</th>
<th>$k_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

$n$  4 3 6 3 7 5 2 5 5 8
$v$  1 1 1 1 0 1 1 1 1 0
$w_2$ 0 0 1 1 0 0 1 1 1 0

Table 3. Matrix $A$ after the second cycle of training

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
<th>$y_6$</th>
<th>$y_7$</th>
<th>$y_8$</th>
<th>$y_9$</th>
<th>$y_{10}$</th>
<th>$m$</th>
<th>$k_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

$n$  4 3 8 5 7 5 4 7 7 8
$v$  0 1 0 0 1 0 1 1 1 0
$w_3$ 0 1 0 0 1 0 1 1 1 0

Table 4. Matrix $A$ after the third cycle of training

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
<th>$y_6$</th>
<th>$y_7$</th>
<th>$y_8$</th>
<th>$y_9$</th>
<th>$y_{10}$</th>
<th>$m$</th>
<th>$k_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

$n$  4 4 8 4 8 4 4 8 8 8
$v$  1 1 0 1 0 1 1 0 0 0
$w_4$ 1 1 0 1 0 1 1 0 0 0

Table 5. Matrix $A$ after the training

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
<th>$y_6$</th>
<th>$y_7$</th>
<th>$y_8$</th>
<th>$y_9$</th>
<th>$y_{10}$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

$y_1$ 1 1 1 1 1 1 1 1 1 1
$y_2$ 1 1 1 1 1 1 1 1 1 1
$y_3$ 1 1 1 1 1 1 1 1 1 1
$y_4$ 1 1 1 1 1 1 1 1 1 1
$y_5$ 1 1 1 1 1 1 1 1 1 1
$y_6$ 1 1 1 1 1 1 1 1 1 1
$y_7$ 1 1 1 1 1 1 1 1 1 1
$y_8$ 1 1 1 1 1 1 1 1 1 1
$y_9$ 1 1 1 1 1 1 1 1 1 1
$y_{10}$ 1 1 1 1 1 1 1 1 1 1
At the end of the fourth cycle all the elements of vector \( m \) are equal to \( N_y = 10 \). Therefore the training process stops. The artificial neural network has been learned now and can be used. In other words, the set of ten Boolean functions \( y_1, y_2, \ldots, y_{10} \), \( y_j = f(x_1, x_2, x_3) \) has been decomposed into four unique Boolean functions \( k_1, k_2, \ldots, k_4 \) by AND operation.

The architecture of the associated Boolean neural network is shown in Figure 2. The number of neurons on the hidden layer is equal to the number of executed cycles in the training algorithm. On the output layer we have 10 neurons. Each of them represents one Boolean function of the set.

![Figure 2. Structure of a Boolean neural network to represent a set of Boolean functions](image)

As a result of the learning procedure, we obtain the transfer functions \( k_1, k_2, \ldots, k_4 \) for the four neurons of the hidden layer and weights of ten output neurons where each of them has 4 inputs.

The upper right part of Table 6 is created from the synaptic weights \( w \) in the upper left part and indicates the reuse of the found unique basic functions. In order to verify the results of the AND-decomposition, we apply formula (25). Table 6 shows that the decomposition process was executed correctly and the Boolean neural network is now ready to work.

<table>
<thead>
<tr>
<th>Out:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( w_2 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( w_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( w_4 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \begin{array}{cccc}
  x_1 & x_2 & x_3 & k_1 & k_2 & k_3 & k_4 \\
  0 & 0 & 0 & 1 & 0 & 1 & 1 \\
  0 & 0 & 1 & 1 & 0 & 1 & 0 \\
  0 & 1 & 0 & 0 & 1 & 1 & 1 \\
  0 & 1 & 1 & 1 & 1 & 0 & 0 \\
  1 & 0 & 0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 1 & 0 & 1 & 1 & 0 \\
  1 & 1 & 0 & 1 & 1 & 0 & 1 \\
  1 & 1 & 1 & 1 & 0 & 1 & 1 \\
\end{array} \]

\[ \begin{array}{ccccccccccc}
  y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} \\
  1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
  0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
  1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

Table 6. Reconstruction of the set of Boolean functions

5 Conclusion and Future Work

Mostly neural networks are used to solve tasks on real or sometimes integer numbers. But the technique of neural networks can be used very efficiently for solving Boolean tasks, too. An important difference between these two classes of tasks is that even the smallest computing error is illegal for the tasks based on Boolean logic. The inadmissible size of memory makes a direct use of the typical types of the neural networks for the representation of Boolean data inefficient.

To solve these problems a new type of neural elements, the Boolean neuron, was suggested. Additionally we show the application of Boolean neurons in Boolean neural networks (BNN). The
Boolean neuron allows to minimize the necessary memory space and to decrease the calculation time for both training and using the BNN. A further advantage is the possibility to map easily the BNN into FPGAs. The proposed Boolean neural networks can be used for compact presentation and fast calculation of Boolean functions.

The implicit representation of Boolean function sets extends the possible approaches in circuit design significantly. As shown in this paper, using the Boolean neural networks for representation of Boolean function allows to determine a special decomposition of Boolean functions sets into a small number of unique basic functions. It must be emphasized that the learning procedure of a BNN solves the very difficult decomposition task for a set of Boolean functions by adapting the FTFS approach of classical neural networks.

We show as example how a set of Boolean functions can be mapped into the structure of a Boolean neural network where the output layer realizes AND operations. In the future we try to optimize the BNN using mixed types of output neurons. Based on the results of this paper we will develop a program that uses Boolean neural networks for compact presentation, decomposition and fast calculation of Boolean functions.

References


