Binary Higher Order Neural Networks for Realizing Boolean Functions

Chao Zhang, Jie Yang, and Wei Wu

Abstract—In order to more efficiently realize Boolean functions by using neural networks, we propose a binary product-unit neural network (BPUNN) and a binary pi-sigma neural network (BPSNN). The network weights can be determined by one-step training. It is shown that the addition “+”, the multiplication “×”, and two kinds of special weighting operations in BPUNN and BPSNN can implement the logical operators “∨”, “∧”, and “¬” on Boolean algebra (\(Z_2, ∨, ∧, ¬, 0, 1\)) (\(Z_2 = \{0, 1\}\)), respectively. The proposed two neural networks enjoy the following advantages over the existing networks: 1) for a complete truth table of \(N\) variables with both truth and false assignments, the corresponding Boolean function can be realized by accordingly choosing a BPUNN or a BPSNN such that at most \(2^N\) hidden nodes are needed, while \(O(2^N)\), precisely \(2^N\) or at most \(2^N\), hidden nodes are needed by existing networks; 2) a new network BPUPS based on a collaboration of BPUNN and BPSNN can be defined to deal with incomplete truth tables, while the existing networks can only deal with complete truth tables; and 3) the values of the weights are all simply 1 or 1, while the weights of all the existing networks are real numbers. Supporting numerical experiments are provided as well. Finally, we present the risk bounds of BPUNN, BPSNN, and BPUPS, and then analyze their probably approximately correct learnability.

Index Terms—Binary pi-sigma neural network, binary product-unit neural network, Boolean function, principle conjunctive normal form, principle disjunctive normal form.

I. INTRODUCTION

Neural networks possess simple parallel structure, computational robustness, and ease of hardware implementation. These advantages make them good candidates for the effective realization of Boolean functions [1]–[3]. Various kinds of neural networks have been proposed and studied to this end, such as McCulloch–Pitts neurons [4], linear threshold neurons [5], [6], pi-sigma neural networks (PSNNs) [7], spiking neurons [8], radial basis function neurons [9], sigma-pi neurons [10], [11], and cellular neural networks [12]. Anthony [13] analyzes the performances of linear threshold neurons, spiking neurons, and sigma-pi neurons used as classifiers to compute Boolean functions. Xiong et al. [14] consider the approximation to Boolean functions by the neural networks with summation units and show that a Boolean function of \(N\) variables can be approximated by a three-layer neural network with \(2^N\) hidden nodes. Computation and approximation to Boolean functions are mentioned in [15]. By using an algebraic form, Chen [16] converted the logic-based input-state dynamics of Boolean networks into an algebraic discrete-time dynamic system, and then investigated the structure of Boolean networks (also see [17]). Chen et al. [18] studied the realization of linearly separable Boolean functions and parity Boolean functions by using universal perceptrons with a deoxyribonucleic acid (DNA)-like learning algorithm. Subsequently, based on the fact that a linearly nonseparable Boolean function can be decomposed into logic XOR operations of a sequence of linearly separable Boolean functions, Chen et al. [19] proposed a DNA-like learning and decomposing algorithm for implementing linearly nonseparable Boolean functions.

Any Boolean function of Boolean algebra \((Z_2, ∨, ∧, ¬, 0, 1)\) can be uniquely represented by either a principle disjunctive normal form (PDNF) or a principle conjunctive normal form (PCNF) [20]. Hence, a certain kind of networks can realize arbitrary Boolean functions if the networks can implement arbitrary PDNFs or PCNFs. In order to construct such networks, we concentrate our attention in this paper to the implementation of the logical operators “∨”, “∧”, and “¬”, and of PDNFs and PCNFs.

In order to enhance the nonlinear mapping ability of the traditional feed-forward neural networks with summation units, various kinds of higher order neural networks (HONNs) have been developed, such as sigma-pi neural networks (SPNNs) [21]–[23], PSNNs [24]–[26], and product-unit neural networks (PUNNs) [27]–[31]. There are also some important works on HONNs proposed in [32]–[36]. The main difference between HONNs and the traditional feed-forward networks is that HONNs have the multiplication operation “II” which can provide more nonlinearity. It is proved in [10] and [11] that fully connected SPNNs can realize arbitrary Boolean functions (see [23] for the concept of “fully connected”), and two one-step methods are respectively proposed to compute the weights. In [7], it is shown that any conjunctive normal form can be realized by PSNN, and the elementary algebra operations “Σ” and “II” are taken as the logical disjunction “∨” and the logical conjunction “∧”, respectively. The logical negation “¬” is implemented in [7] by a combination of inputs and threshold values. The implementation of the logical operators “∨”, “∧”, and “¬” by elementary algebra operations is also mentioned.

Manuscript received December 23, 2008; revised February 1, 2011; accepted February 1, 2011. This work was supported in part by the National Natural Science Foundation of China under Grant 10871220.

Chao Zhang was with the School of Mathematical Sciences, Dalian University of Technology, Dalian 116023, China. He is now with the School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: zhangchao1015@gmail.com). Jie Yang and Wei Wu are with the School of Mathematical Sciences, Dalian University of Technology, Dalian 116023, China (e-mail: yangjiee@dlut.edu.cn; wuweiw@dlut.edu.cn).
in [37], where an extracting rule from the trained networks is developed. In the domain $Z_2$, the logical conjunction “∧” is equal to the multiplication “II.” Because of the existence of the multiplication “II,” if the addition “Σ” and some weighting operation can respectively implement logical operators “∧” and “¬” in some kinds of HONNs, Boolean functions can be realized by the HONNs.

The concept of functional completeness shows a sufficient condition for a model to realize arbitrary Boolean functions (see [20]).

Definition 1: A set of logical operators $\{g_1, g_2, \ldots, g_k\}$ is said to be functionally complete if all possible Boolean functions can be implemented by using only the operators $g_1, g_2, \ldots, g_k$.

Our model will be based on the set of logical operators $\{\lor, \land, \neg\}$. The reason for doing so is as follows.

1) Each of the sets $\{\lor, \land, \neg\}$, $\{\land, \neg\}$, and $\{\lor, \neg\}$ is functionally complete, while the set $\{\lor, \land\}$ is not (see [20]).

2) The set $\{\land, \neg\}$ or $\{\lor, \neg\}$ contains fewer operations. However, since $x \lor y = \neg(\neg x \land \neg y)$ and $x \land y = \neg(\neg x \lor \neg y)$ (see [20]), models based on $\{\land, \neg\}$ or $\{\lor, \neg\}$ need more number of computations than models based on $\{\lor, \land, \neg\}$.

3) Models based on $\{\lor, \land, \neg\}$ directly correspond to Boolean expressions on the Boolean algebra $(Z_2, \lor, \land, \neg, 0, 1)$.

Our aim in this paper is to construct neural networks that are based on $\{\lor, \land, \neg\}$ and can more efficiently realize Boolean functions than other existing networks. In particular, we propose a binary PUNN (BPUNN) and a binary PSNN (BPSNN) derived from PUNN and PSNN, respectively. We prove that BPUNN (resp. BPSNN) can implement arbitrary PDNFs (resp. PCNFs) and hence arbitrary Boolean functions with both truth and false assignments. In both BPUNN and BPSNN, as usual, the addition “Σ” and the multiplication “II” stand for the logical disjunction “∨” and the logical conjunction “∧”, respectively. An important trick in our approach is to implement the unary logical operator “¬” by two special input-to-hidden weighting operations, respectively. The two neural networks enjoy the following advantages compared to the existing networks.

1) At most $2^{N-1}$ hidden nodes are needed to realize a Boolean function with a complete truth table of $N$ variables by suitably choosing a BPUNN or a BPSNN, with the exception that $2^N$ hidden nodes are needed to realize two special Boolean functions with $2^N$ truth assignments or $2^N$ false assignments respectively, while $O(2^N)$, precisely $2^N$ and at most $2^N$, hidden nodes are needed for multilayer perceptron (MLP) in [13], SPNN in [10] and [11], and PSNN in [7], respectively.

2) A Boolean function with an incomplete truth table can also be realized by a new network BPUPS based on a collaboration of BPUNN and BPSNN, such that it gives the right response to an input that is in the truth table and recognizes an input that is not in the truth table, while this issue is not considered for the existing networks.

3) The values of the weights are simply $-1$ or $1$, which is convenient for storage and hardware realization ([11]), while the weights of the existing networks are real numbers.

4) The network weights can be determined by one-step training, while they are obtained through iterative training procedures for the DNA-like algorithm in [18] and [19], MLP in [13], and PSNN in [7]. The network weights of SPNN in [10] and [11] also enjoy this advantage.

For the further investigation of the properties of BPUNN and BPSNN, we consider a more general scenario which allows that the training samples can be evaluated with certain probability. We give the Vapnik–Chervonenkis (VC) dimension of a function class composed of BPUNN (resp. BPSNN). Afterwards, we present the risk bound of BPUNN (resp. BPSNN) based on the VC dimension, and then analyze the learnability of the two networks by using the resulting risk bounds. Finally, we discuss the learnability of BPUPS.

The rest of this paper is arranged as follows. The next section gives a brief introduction on Boolean functions. BPUNN and BPSNN are respectively defined in Sections III and IV. The main theorems and their proofs are presented in Section V. Section VI gives a comparison with other kinds of neural networks and provides the corresponding numerical experiments. In Section VII, we propose a network to deal with incomplete truth tables. The learnability of BPUNN, BPSNN, and BPUPS is analyzed in Section VIII and some conclusions are drawn in the last section. The relevant proofs are given in the Appendixes.

II. PRELIMINARIES ON BOOLEAN FUNCTIONS

In this section, we present some preliminaries on Boolean functions, and refer to [20] for the further details.

A. PDNF

Definition 2: A minterm in the variables $x_1, x_2, \ldots, x_N$ is a Boolean expression of the form

$$\bar{x}_1 \land \bar{x}_2 \land \cdots \land \bar{x}_N$$

where $\bar{x}_n$ is either $x_n$ or $\neg x_n$.

For a minterm $\bar{x}_1 \land \bar{x}_2 \land \cdots \land \bar{x}_N$, a vector $a = (a_1, a_2, \ldots, a_N) \in Z_2^N$ is generated as follows. For $1 \leq n \leq N$, we set

$$a_n = \begin{cases} 1, & \bar{x}_n = x_n ; \\ 0, & \bar{x}_n = \neg x_n . \end{cases}$$

The resulting vector $a$ is the unique truth assignment for the minterm $\bar{x}_1 \land \bar{x}_2 \land \cdots \land \bar{x}_N$. Thus, for $x = (x_1, x_2, \ldots, x_N)$, the minterm can be represented as

$$m_a(x) = \bar{x}_1 \land \bar{x}_2 \land \cdots \land \bar{x}_N$$

and

$$m_a(x) = \begin{cases} 1, & x = a ; \\ 0, & x \neq a . \end{cases}$$

The next theorem and its proof can be found in [20].

Theorem 1: If $f : Z_2^N \to Z_2$, then $f$ is a Boolean function. If $f$ is not identically zero, let $a_1, a_2, \ldots, a_J \in Z_2^N$
be the truth assignments for Boolean function \( f \). For each 
\[ a_j = (a_{j,1}, a_{j,2}, \ldots, a_{j,n}) \quad (1 \leq j \leq J), \]
set
\[ m_{a_j}(x) = \bar{x}_1 \land \bar{x}_2 \land \cdots \land \bar{x}_N \]
where \( x = (x_1, x_2, \ldots, x_N) \in Z_2^N \), and
\[ \bar{x}_n = \begin{cases} x_n, & a_{j,n} = 1; \\ \lnot x_n, & a_{j,n} = 0. \end{cases} \]

Then
\[ f(x) = m_{a_1}(x) \lor m_{a_2}(x) \lor \cdots \lor m_{a_J}(x). \quad (5) \]

**Definition 3:** The representation (5) of a Boolean function \( f : Z_2^N \rightarrow Z_2 \) is called the PDNF.

**B. PCNF**

**Definition 4:** A maxterm in the variables \( x_1, x_2, \ldots, x_N \) is a Boolean expression of the form 
\[ \bar{x}_1 \lor \bar{x}_2 \lor \cdots \lor \bar{x}_N \]
where \( \bar{x}_n \) is either \( x_n \) or \( \lnot x_n \).

For a maxterm \( \bar{x}_1 \lor \bar{x}_2 \lor \cdots \lor \bar{x}_N \), we generate a vector 
\[ b = (b_1, b_2, \ldots, b_N) \in Z_2^N \] as follows. For \( 1 \leq n \leq N \), we set 
\[ b_n = \begin{cases} 1, & \bar{x}_n = \lnot x_n; \\ 0, & \bar{x}_n = x_n. \end{cases} \quad (7) \]

The resulting vector \( b \) is the unique false assignment for the maxterm \( \bar{x}_1 \lor \bar{x}_2 \lor \cdots \lor \bar{x}_N \). Thus, for \( x = (x_1, x_2, \ldots, x_N) \), the maxterm can be represented as 
\[ M_b(x) = \bar{x}_1 \lor \bar{x}_2 \lor \cdots \lor \bar{x}_N \]
and
\[ M_b(x) = \begin{cases} 1, & x \neq b; \\ 0, & x = b. \end{cases} \quad (9) \]

Combining the dual principle and Theorem 1, we have the following result, which can be found in [20].

**Theorem 2:** If \( f : Z_2^N \rightarrow Z_2 \), then \( f \) is a Boolean function. If \( f \) is not identically zero, let \( b_1, b_2, \ldots, b_I \in Z_2^N \) be the false assignments for Boolean function \( f \). For each 
\[ b_i = (b_{i,1}, b_{i,2}, \ldots, b_{i,N}) \quad (1 \leq i \leq I), \]
set
\[ M_{b_i}(x) = \bar{x}_1 \lor \bar{x}_2 \lor \cdots \lor \bar{x}_N, \quad 1 \leq i \leq I \]
where \( x = (x_1, x_2, \ldots, x_N) \in Z_2^N \), and
\[ \bar{x}_n = \begin{cases} x_n, & b_{i,n} = 0; \\ \lnot x_n, & b_{i,n} = 1. \end{cases} \]

Then
\[ f(x) = M_{b_1}(x) \land M_{b_2}(x) \land \cdots \land M_{b_I}(x). \quad (10) \]

**Definition 5:** The representation (10) of a Boolean function \( f : Z_2^N \rightarrow Z_2 \) is called the PCNF.

---

**III. BPUNN**

**A. BPU**

The form of PUs is \( \prod_{n=1}^{N} x_n^{w_n} \). Compared to the form of traditional summation units \( \sum_{n=1}^{N} x_n w_n \), PUs have more powerful nonlinearity due to their special exponential form. It is shown in [27] that the information capacity of a single PU (as measured by its capacity for learning random Boolean patterns) is approximately \( 3N \), compared to \( 2N \) for a single summation unit [38], where \( N \) is the number of inputs to the units. Therefore, PUNNs with a hidden layer composed of PUs can handle many complicated cases [31]. For example, as demonstrated in [39], a PUNN with only one PU in its hidden layer is sufficient to solve difficult symmetry problems and parity problems.

Based on PU, we try to define a BPU with inputs and outputs in \( Z_2 \), as shown in Fig. 1.

Write \( w = (w_1, \ldots, w_N) \in \{-1,1\}^N \) as the weight vector. For an input vector \( x = (x_1, \ldots, x_N) \in Z_2^N \), let \( H(w, x) \) be the output of BPU

\[ H(w, x) = \prod_{n=1}^{N} h_{\lnot(x_n)}(w_n) \quad (11) \]

where \( h(t) : Z_2 \rightarrow \{1/2, 2\} \) is defined by
\[ h(t) = \begin{cases} \frac{2}{1-t}, & t = 1; \\ \frac{1}{2}, & t = 0 \end{cases} \quad (12) \]
and \( h^{-1}(t) \) is the inverse function of \( h(t) \)
\[ h^{-1}(t) = \begin{cases} 1, & t = 2; \\ 0, & t = \frac{1}{2}. \end{cases} \quad (13) \]

The operation of \( h(t) \) is to switch the input domain \( Z_2 \) into \( \{2, 1/2\} \). (Actually, we may use \( \{(1/a, a) \} \) in place of \( \{1/2, 2\} \) for any \( a > 1 \).) The switch operation will be used in the weighting operation of BPU to implement logical negation “\( \lnot \).”

**B. BPUNN**

Fig. 2 shows a BPUNN with one output and \( J \) hidden nodes (BPUs). The weight vector between inputs and the \( j \)th hidden node is denoted by \( w_j = (w_{j,1}, w_{j,2}, \ldots, w_{j,N}) \in \{-1,1\}^N \quad (1 \leq j \leq J) \), and the hidden-to-output weights are
Let \( v \) be the weight vector. For a binary input vector \( x = (x_1, \ldots, x_N) \in \mathbb{Z}_2^N \), the output of the hidden node is

\[
K(v, x) = g \left( \sum_{n=1}^{N} k^{-1}(v_n k(x_n)) \right)
\]  

where \( g(t) \) is given in (15), \( k(t) : \mathbb{Z}_2 \rightarrow \{-1, 1\} \) is defined by

\[
k(t) = \begin{cases} 
1, & t = 1; \\
1, & t = 0 
\end{cases}
\]

and \( k^{-1}(t) \) is the inverse function of \( k(t) \)

\[
k^{-1}(t) = \begin{cases} 
1, & t = 1; \\
0, & t = -1.
\end{cases}
\]

A BPSNN with \( I \) hidden nodes and one output is illustrated in Fig. 4, where \( x = (x_1, x_2, \ldots, x_N) \in \mathbb{Z}_2^N \) stands for the input vector and \( v_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,N}) \in \{-1, 1\}^N \quad (1 \leq i \leq I) \) is the weight vector between the input layer and the \( i \)th hidden node. The output of the \( i \)th hidden node is denoted by \( K(v_i, x) \) (16). The hidden-to-output weights are fixed to 1. According to (16) and (17), the final output of the BPSNN is

\[
y = \prod_{i=1}^{I} K(v_i, x) = \prod_{i=1}^{I} g \left( \sum_{n=1}^{N} k^{-1}(v_{i,n} k(x_n)) \right)
\]

where \( g(t), k(t), \) and \( k^{-1}(t) \) are defined by (15), (17), and (18), respectively.

## IV. BPSNN

The PSNN was proposed by Shin in 1991 [24]. Incorporating product neurons with polynomials of inputs, PSNN has powerful capability of nonlinear mapping and avoids the dimensional explosion that might appear in SPNN [24]. Based on PSNN, we propose BPSNN with both inputs and outputs domains being \( \mathbb{Z}_2 \).

### A. Hidden Node of BPSNN

The structure of a hidden node of BPSNN is shown in Fig. 3. Let \( v = (v_1, \ldots, v_N) \in \{-1, 1\}^N \) be the weight vector. For a binary input vector \( x = (x_1, \ldots, x_N) \in \mathbb{Z}_2^N \), the output of the hidden node is

\[
K(v, x) = g \left( \sum_{n=1}^{N} k^{-1}(v_n k(x_n)) \right)
\]
Furthermore, for $1 \leq n \leq N$, we have
\[
h^{-1}((h(x))^{w}) = \begin{cases} x, & w = 1; \\ \neg x, & w = -1. \end{cases} \tag{20}
\]
and
\[
k^{-1}(\upsilon k(x)) = \begin{cases} x, & \upsilon = 1; \\ \neg x, & \upsilon = -1. \end{cases} \tag{21}
\]

**Proof:** Using (12), (13), (15), (17), and (18), the above formulas can be directly obtained by the principles of logical operators \(\lor, \land\), and \(\neg\).

In (20), \(h(t)\) switches the input domain \(Z_2\) into \((1/2, 2)\) so that the exponential weight of input can implement the logical negation \(\neg\) in (21).

**B. Implementation of Boolean Functions by BPUNN**

First, we consider the implementation of minterms. Let \(x = (x_1, x_2, \ldots, x_N) \in Z_2^N\). A given minterm has the form (1)
\[
m_a(x) = \overline{x}_1 \land \overline{x}_2 \land \cdots \land \overline{x}_N
\]
where \(\overline{x}_n\) is either \(x_n\) or \(\neg x_n\). Let the vector \(a = (a_1, a_2, \ldots, a_N) \in Z_2^N\) be the unique truth assignment for the minterm (2). We define a function \(\psi(a) : Z_2 \rightarrow \{-1, 1\}\) as follows:
\[
\psi(a) \triangleq \begin{cases} 1, & a = 1; \\ -1, & a = 0. \end{cases}
\]
and then define a mapping \(\Psi(a) : Z_2^N \rightarrow \{-1, 1\}^N\)
\[
\Psi(a) \triangleq (\psi(a_1), \psi(a_2), \ldots, \psi(a_N)). \tag{22}
\]
We define the weight vector \(w = (w_1, w_2, \ldots, w_N) \in \{-1, 1\}^N\) as
\[
w = \Psi(a). \tag{23}
\]
By (3), (4), (11), (12), (20), and (24), the output of BPU satisfies
\[
H(w, x) = m_a(x) = \begin{cases} 1, & x = a; \\ 0, & x \neq a. \end{cases} \tag{24}
\]
The above discussion leads to the following theorem.

**Theorem 4:** For any given minterm of \(N\) variables shown in (1), if the weight vector of a BPU with \(N\) inputs (11) is determined by (24), then the BPU can implement the minterm.

Since any minterm has a unique truth assignment, for a given minterm of \(N\) variables, the corresponding BPU can be obtained by (24). Since arbitrary Boolean functions can be written as PDNFs—the disjunction of minterms, if a kind of networks can implement PDNFs, then the networks can realize arbitrary Boolean functions as well.

Let \(f : Z_2^N \rightarrow Z_2\) be a given Boolean function with \(J\) truth assignments \(a_1, a_2, \ldots, a_J \in Z_2^N\) \((1 \leq J \leq 2^N)\). By (3) and (4), the vector \(\overline{a}_j = (a_{j,1}, a_{j,2}, \ldots, a_{j,N})\) is the truth assignment for the minterm \(m_a\) \((1 \leq j \leq J)\). Then, according to Theorem 1, we have
\[
f(x) = m_{a_1}(x) \lor m_{a_2}(x) \lor \cdots \lor m_{a_J}(x) \tag{25}
\]
where \(x = (x_1, x_2, \ldots, x_N) \in Z_2^N\).

As shown in Fig. 2, we consider a BPUNN with \(J\) hidden nodes and let the \(j\)th hidden node correspond to the minterm \(m_{a_j}(x)\) \((1 \leq j \leq J)\). By (24) and Theorem 4, the output of the \(j\)th hidden node is
\[
H(w_j, x) = m_{a_j}(x) = \begin{cases} 1, & x = a_j; \\ 0, & x \neq a_j. \end{cases} \tag{26}
\]
where \(w_j = \Psi(a_j)\). Then, combining (14), (26), and (27), we have
\[
g \left( \sum_{j=1}^{J} H(w_j, x) \right) = m_{a_1}(x) \lor m_{a_2}(x) \lor \cdots \lor m_{a_J}(x)
\]
where \(g(t)\) is defined by (15). As a consequence, we have the following theorem.

**Theorem 5:** Let \(f : Z_2^N \rightarrow Z_2\) be a given Boolean function with \(J\) \((1 \leq J \leq 2^N)\) truth assignments. Let a BPUNN be given with \(N\) inputs, \(J\) hidden nodes, and one output as shown in Fig. 2, and with the hidden nodes being determined by (27). Then the Boolean function \(f\) can be realized by the resulting BPUNN.

Note that if \(J = 0\), the outputs of \(f\) are all zeros and \(f\) has no truth assignment. Therefore, according to Theorem 4, BPUNN cannot realize such a function \(f\). However, we will show that BPSNN can achieve it in the following discussion.

**C. Implementation of Boolean Functions by BPSNN**

Let \(x = (x_1, x_2, \ldots, x_N) \in Z_2^N\). A given maxterm has the form (see (6))
\[
M_b(x) = \overline{x}_1 \lor \overline{x}_2 \lor \cdots \lor \overline{x}_N
\]
where \(\overline{x}_n\) is either \(x_n\) or \(\neg x_n\). Let \(b = (b_1, b_2, \ldots, b_N) \in Z_2^N\) be the unique false assignment for the maxterm defined in (7).

We define a function \(\phi(b) : Z_2 \rightarrow \{-1, 1\}\) as follows:
\[
\phi(b) \triangleq \begin{cases} 1, & b = 0; \\ -1, & b = 1. \end{cases} \tag{27}
\]
and then define a mapping \(\Phi(b) : Z_2^N \rightarrow \{-1, 1\}^N\)
\[
\Phi(b) \triangleq (\phi(b_1), \phi(b_2), \ldots, \phi(b_N)). \tag{28}
\]
We define the weight vector \(v = (v_1, v_2, \ldots, v_N) \in \{-1, 1\}^N\) as
\[
v = \Phi(b). \tag{29}
\]
For each maxterm \(M_b(x)\), we accordingly define a hidden node of BPSNN with the output
\[
K(v, x) = M_b(x) = \begin{cases} 1, & x \neq b; \\ 0, & x = b. \end{cases} \tag{30}
\]
Consequently, we have the following theorem.

**Theorem 6:** For any given maxterm of \(N\) variables shown in (6), if the weight vector of a hidden node of BPSNN with \(N\) inputs [see (16)] is determined by (31), then the hidden node can implement the maxterm.

Because any Boolean function can be written as PCNF—a conjunction of maxterms, if a network can implement PCNF, then the network can realize the Boolean function as well.
For a given Boolean function \( f : \mathbb{Z}_2^N \rightarrow \mathbb{Z}_2 \) with \( I \) false assignments \( \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_I \in \mathbb{Z}_2^N \), \( 1 \leq I \leq 2^N \). By (8) and (9), the vector \( \mathbf{b}_i = (b_{i1}, b_{i2}, \ldots, b_{iN}) \) is the false assignment for the maxterm \( M_{b_i} \), \( 1 \leq i \leq I \). According to Theorem 2, we have

\[
 f(x) = M_{b_1}(x) \land M_{b_2}(x) \land \cdots \land M_{b_I}(x) \tag{33}
\]

where \( x = (x_1, x_2, \ldots, x_N) \in \mathbb{Z}_2^N \).

We consider a BPSNN with \( I \) hidden nodes. Let the \( i \)th hidden node correspond to the maxterm \( M_{b_i}(x) \), \( 1 \leq i \leq I \), and define its output as

\[
 K(v_i, x) = M_{b_i}(x) = \begin{cases} 
 1, & x \notin b_i; \\
 0, & x = b_i
\end{cases} \tag{34}
\]

where \( v_i = \Phi(b_i) \).

Then, combining (19), (33), and (34), we have

\[
 \prod_{i=1}^{I} K(v_i, x) = M_{b_1}(x) \land M_{b_2}(x) \land \cdots \land M_{b_I}(x) = \begin{cases} 
 1, & x \notin \{b_1, b_2, \ldots, b_I\}; \\
 0, & x \in \{b_1, b_2, \ldots, b_I\};
\end{cases} \tag{35}
\]

Subsequently, the following theorem holds.

**Theorem 7:** Let \( f : \mathbb{Z}_2^N \rightarrow \mathbb{Z}_2 \) be a given Boolean function with \( I \) (\( 1 \leq I \leq 2^N \)) false assignments. Let a BPSNN be given with \( N \) inputs, \( I \) hidden nodes, and one output as shown in Fig. 4, with the hidden nodes being determined by (34). Then the Boolean function \( f \) can be realized by the resulting BPSNN.

Note that, if \( I = 0 \), the outputs of \( f \) are all 1’s and the \( f \) has no false assignment. Therefore, according to Theorem 6, BPSNN cannot realize such a function \( f \), but BPUNN can just do it (see Theorem 5).

**D. Choose BPUNN or BPSNN Such That At Most \( 2^{N-1} \) Hidden Nodes Are Needed**

According to Theorems 5 and 7, when BPUNN (resp. BPSNN) is used to implement a Boolean function of \( N \) variables with both truth and false assignments, the hidden-node number is equal to the number of truth (resp. false) assignments for the Boolean function. For a given complete truth table, we can choose a network with smaller size from BPUNN and BPSNN. For example, for a truth table of \( N \) variables with both truth and false assignments, if the number of truth assignments is less than that of false assignments in the truth table, we should choose BPUNN to implement the Boolean function with at most \( 2^{N-1} \) hidden nodes and every hidden node corresponds to a truth assignment. Contrarily, BPSNN is a better choice when the number of false assignments is less than that of the truth ones, and then every hidden node corresponds to a false assignment. In this strategy, for arbitrary Boolean functions of \( N \) variables with both truth and false assignments, the resulting networks have at most \( 2^{N-1} \) hidden nodes.

For completeness, we remark that, as pointed out in the ends of Sections V-B and C, for the two special Boolean functions with no truth or no false assignment, respectively, \( 2^N \) rather than \( 2^{N-1} \) hidden nodes are needed.

**VI. NUMERICAL EXPERIMENTS**

This section compares BPUNN and BPSNN with MLP, SPNN, PSNN, and PUNN for realizing arbitrary Boolean functions. The comparison focuses on the following two respects: 1) the hidden-node number of a network to realize a Boolean function, and 2) the feasibility to train such a neural network.

The number of all the Boolean functions is \( 2^N \) for given \( N \) variables, which becomes very large as \( N \) increases. Therefore, for the sake of computational cost, we only consider the case \( N = 3 \). All the networks have one hidden layer.

BP algorithm is applied to train MLP, SPNN, PSNN, and PUNN for realizing arbitrary Boolean functions of three variables. We set the hidden-node number of MLP, PSNN, and PUNN from 1 to 10. Zhang et al. discussed the divisibility of SPNN in [23]. To compare with the existing works [10], [11], we adopt the same SPNN. The hidden-node number of the presented SPNN is set from 4 to 8. We are mainly concerned with how the order of SPNN influences the performance of SPNN for realizing Boolean functions. Therefore, we remove the higher order hidden nodes from the fully connected SPNN one after another to form new SPNNs with 4–7 hidden nodes, respectively. In detail, the fully connected SPNN with three inputs has eight hidden nodes [23, Fig. 1]. First, we remove the third-order node to form a SPNN with seven hidden nodes, which has three second-order hidden nodes. We then remove a second-order node from the resulting SPNN to form a SPNN with six hidden nodes. In this manner, the resulting SPNN with four hidden nodes is actually equivalent to a perceptron network [23].

All initial weights are randomly selected from the interval \([-0.5, 0.5]\). For any Boolean function and any given number of hidden nodes, we run repeatedly 30 training procedures to train MLP, SPNN, PSNN, and PUNN. A training procedure is stopped when the error is smaller than the error bound 0.05 (deemed as “success”), or when it reaches the maximum iterative epoch 10000 (deemed as “failure”). We record the number of successes in the 30 training procedures and then compute the success rate.

According to Theorems 5 and 7, BPUNN and BPSNN can be trained by one-step methods for realizing arbitrary Boolean functions. In the experiment, the numbers of successfully realized Boolean functions are recorded for BPUNN and BPSNN. Moreover, the cooperation of BPUNN and BPSNN, denoted by “BPUNN&BPSNN,” is considered and the corresponding results are provided as well.

The experimental results are shown in Table I. For example, for MLP with two hidden nodes \( (N_H = 2) \), four Boolean functions \( (N_B = 4) \) fail to be realized \( (R_s = 0\%) \), five (resp. 41) Boolean functions are realized in the success rate between 40 and 83.3\% (resp. 86.7–96.7\%) in all the 30 training procedures, i.e., \( R_s \in [40\%, 83.3\%] \) (resp. \( R_s \in [86.7\%, 96.7\%] \)), and 206 Boolean functions are realized successfully in all the 30 training procedures \( (R_s = 100\%) \).

Table I shows that MLP cannot successfully realize all the Boolean functions when the number of hidden nodes is less than 10. PSNN behaves better in that it can realize
all Boolean functions with three and four hidden nodes by performing up to 30 training procedures in our experiment, but it is not guaranteed to be successful in each training procedure. In fact, this is a trouble that iterative network training procedures often face. It is guaranteed that there exists a target somewhere, but it is not guaranteed for a particular training procedure to finally reach the target. We also observe another interesting phenomenon that more hidden nodes do not necessarily bring about better convergence for MLP and PSNN. This is related to another well-known fact, when approximating a given mapping by using neural networks, too many hidden nodes may give more trouble than help, just like the case to approximate a given function by using polynomials of too high orders.

From Table I, we can find that PUNN and SPNN have better performance than MLP and PSNN. This is because PUNN and SPNN have the nonlinearity strong enough to afford to realize Boolean functions of three variables. Moreover, the structures of the presented PUNNs and SPNNs are not complicated and thus the BP algorithm can be used to train them in a high success rate. PUNN can use relatively few hidden nodes to realize most Boolean functions and the success rate increases as the hidden-node number increases. However, it is actually very difficult to train a PUNN to realize high-dimensional Boolean functions because of its extremely mountainous error surface ([31]). As shown in Table I, if a Boolean function can be realized by a SPNN, every training is successful. When the hidden-node number is 4, the SPNN

<table>
<thead>
<tr>
<th>$N_H$</th>
<th>$N_B$</th>
<th>$R_S$ (%)</th>
<th>$N_H$</th>
<th>$N_B$</th>
<th>$R_S$ (%)</th>
<th>$N_H$</th>
<th>$N_B$</th>
<th>$R_S$ (%)</th>
<th>$N_H$</th>
<th>$N_B$</th>
<th>$R_S$ (%)</th>
<th>$N_H$</th>
<th>$N_B$</th>
<th>$R_S$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>152</td>
<td>0</td>
<td>1</td>
<td>152</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>23</td>
<td>70</td>
<td>25</td>
<td>93.3 ~ 96.7</td>
<td>8</td>
<td>100</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>254</td>
<td>100</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>76.7</td>
<td>19</td>
<td>90 ~ 96.7</td>
<td>36</td>
<td>100</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>247</td>
<td>86.7 ~ 96.7</td>
<td>2</td>
<td>0</td>
<td>9</td>
<td>90</td>
<td>245</td>
<td>100</td>
<td>92</td>
<td>100</td>
<td>92</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>13</td>
<td>13</td>
<td>40 ~ 73.3</td>
<td>2</td>
<td>0</td>
<td>152</td>
<td>0</td>
<td>162</td>
<td>100</td>
<td>104</td>
<td>100</td>
<td>162</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>3 ~ 16.7</td>
<td>2</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>218</td>
<td>100</td>
<td>218</td>
<td>100</td>
<td>218</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>16</td>
<td>49</td>
<td>3   ~ 10</td>
<td>2</td>
<td>0</td>
<td>60</td>
<td>196</td>
<td>246</td>
<td>100</td>
<td>246</td>
<td>100</td>
<td>246</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>0</td>
<td>16</td>
<td>9</td>
<td>3.3</td>
<td>2</td>
<td>0</td>
<td>254</td>
<td>100</td>
<td>254</td>
<td>100</td>
<td>254</td>
<td>100</td>
<td>254</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>0</td>
<td>20</td>
<td>256</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>254</td>
<td>100</td>
<td>256</td>
<td>100</td>
<td>256</td>
<td>100</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>0</td>
<td>20</td>
<td>256</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>254</td>
<td>100</td>
<td>256</td>
<td>100</td>
<td>256</td>
<td>100</td>
<td>256</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>0</td>
<td>27</td>
<td>256</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>254</td>
<td>100</td>
<td>256</td>
<td>100</td>
<td>256</td>
<td>100</td>
<td>256</td>
</tr>
</tbody>
</table>

$N_H$: number of hidden nodes  
$N_B$: number of Boolean functions  
$R_S$: success rate  
a ~ b: between a and b
is equivalent to a perceptron network that is also the MLP with one hidden node and their experimental results are the same. If a SPNN has more number of higher order hidden nodes, more Boolean functions can be realized by the SPNN. Especially, the SPNN with eight hidden nodes can realize all of 256 Boolean functions, which is in accordance with the theoretical results given in [10] and [11].

According to Theorems 5 and 7, for a given Boolean function with truth (resp. false) assignments, the hidden-node number of the corresponding BPUNN (resp. BPSNN) is equal to the number of the truth (resp. false) assignments of the Boolean function. Therefore, as shown in Table I, the number of Boolean functions that can be realized by BPUNN (resp. BPSNN) increases to 255 with the exception of the Boolean function with no truth (resp. false) assignment, as the hidden-node number increases to 8. Since BPUNN (resp. BPSNN) is trained by one-step methods given in (24) [resp. (31)], every training is successful when the hidden-node number matches the number of the truth (resp. false) assignments, and hence its success rate is 100%.

As predicted by our theoretical results in the last section, it is shown in the last column of Table I that we only need four hidden nodes to realize all the Boolean functions with both truth and false assignments. The two exceptional Boolean functions with no false or no truth assignment can only be realized by BPUNN or BPSNN, respectively, by using eight hidden nodes.

As a comparison, although SPNN can also realize arbitrary Boolean functions of \( N \) variables by one-step training, it always needs \( 2^N \) hidden nodes (see [10], [11]) to realize any Boolean function.

### VII. Network for Implementing Incomplete Truth Tables

All the inputs and corresponding outputs of a given Boolean function form a complete truth table. But, sometimes, there might be some assignments missing from a truth table. This truth table is then called an incomplete truth table. The implementation to an incomplete truth table should give right response to an input that is in the truth table and recognize an input that is not in it. In this section, we integrate BPUNN and BPSNN into a new network—BPUPS, which meets the requirements.

For a given incomplete truth table of \( N \) variables with \( J \) truth assignments and \( I \) false assignments \((I + J < 2^N)\), we select all the truth (resp. false) assignments to form a set of input vectors, and accordingly construct a BPUNN (resp. BPSNN) as described in Section V. Furthermore, employing the resulting BPUNN and BPSNN, we construct a BPUPS with \( N \) inputs, \( I + J \) hidden nodes, and two outputs as shown in Fig. 5. The output vector of BPUPS is denoted by \( (y_1, y_2) \), where \( y_1 \) and \( y_2 \) are, respectively, the outputs of BPUNN and BPSNN. For different kinds of inputs, BPUPS will generate different output vectors as follows.

1) When a truth assignment is inputted into the network, the output vector \((y_1, y_2)\) is \((1, 1)\).
2) If a false assignment is inputted, the output vector \((y_1, y_2)\) is \((0, 0)\).
3) For a missing assignment, the output vector \((y_1, y_2)\) is \((0, 1)\).

Let us give a simple example to illustrate the details. An incomplete truth table is given in the first two columns of Table II with three truth assignments \(\{a_1, a_2, a_3\}\), three false assignments \(\{b_1, b_2, b_3\}\) and two missing assignments \(\{c_1, c_2\}\).

As described in Section V, we use the truth assignments \(\{a_1, a_2, a_3\}\) to construct a BPUNN with three inputs, three hidden nodes and one output (Fig. 2). By (24), the input-to-hidden weight vectors are specified as follows:

\[
\begin{align*}
  w_1 &= \Psi((1, 1, 1)) = (1, 1, 1) \\
  w_2 &= \Psi((1, 1, 0)) = (1, 1, -1) \\
  w_3 &= \Psi((0, 1, 1)) = (-1, 1, 1).
\end{align*}
\]

Similarly, we use the false assignments \(\{b_1, b_2, b_3\}\) to construct a BPSNN with three inputs, three hidden nodes, and one output (Fig. 4). According to (31), we compute the input-to-hidden weights as follows:

\[
\begin{align*}
  v_1 &= \Phi((1, 0, 1)) = (-1, 1, -1) \\
  v_2 &= \Phi((1, 0, 0)) = (-1, 1, 1) \\
  v_3 &= \Phi((0, 1, 0)) = (1, -1, 1).
\end{align*}
\]

Thus, the BPUNN and the BPSNN are obtained.

Then, we integrate the resulting BPUNN and BPSNN into a BPUPS to implement the incomplete truth table (Fig. 5). For \(\{a_1, a_2, a_3\}\), \(\{b_1, b_2, b_3\}\), and \(\{c_1, c_2\}\), the BPUPS gives different responses to different kinds of inputs as shown in

**Table II**

<table>
<thead>
<tr>
<th>(x = (x_1, x_2, x_3))</th>
<th>(f(x))</th>
<th>((y_1, y_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1 = (1, 1, 1))</td>
<td>1</td>
<td>((1, 1))</td>
</tr>
<tr>
<td>(a_2 = (1, 1, 0))</td>
<td>1</td>
<td>((1, 1))</td>
</tr>
<tr>
<td>(a_3 = (0, 1, 1))</td>
<td>1</td>
<td>((1, 1))</td>
</tr>
<tr>
<td>(b_1 = (1, 0, 1))</td>
<td>0</td>
<td>((0, 0))</td>
</tr>
<tr>
<td>(b_2 = (1, 0, 0))</td>
<td>0</td>
<td>((0, 0))</td>
</tr>
<tr>
<td>(b_3 = (0, 1, 0))</td>
<td>0</td>
<td>((0, 0))</td>
</tr>
<tr>
<td>(c_1 = (0, 0, 1))</td>
<td>-</td>
<td>((0, 1))</td>
</tr>
<tr>
<td>(c_2 = (0, 0, 0))</td>
<td>-</td>
<td>((0, 1))</td>
</tr>
</tbody>
</table>
the last column of Table II. Hence, the BPUPS meets the requirements that a right response is given to an input that is in the truth table, and that an input that is not in can be recognized.

VIII. LEARNABILITY OF BINARY HONNS

The discussion in the former sections is based on a deterministic scenario, where the target Boolean function can be evaluated deterministically. This section is concerned with the behavior of our models from a probabilistic point of view when the target Boolean function cannot be evaluated deterministically. In particular, the risk bounds and the learnability of BPUNN and BPSNN are investigated.

A. Risk Bounds of BPUNN and BPSNN

Let us reframe our models BPUNN and BPSNN in the learning theory framework. Define $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is an input space and $\mathcal{Y}$ is its corresponding output space. Since this paper is restricted to binary issues, we choose $\mathcal{X} \subseteq \mathbb{Z}_2^M$ and $\mathcal{Y} \subseteq \mathbb{Z}_2$. Let the pair $z = (x, y) \in \mathcal{Z}$ be a random variable distributed according to a certain probability distribution $P(z)$.

Given a function class $\Omega$, it is expected to find a function $T \in \Omega : \mathcal{X} \rightarrow \mathcal{Y}$ so as to predict, for any given input $x$, the corresponding output $y$. A natural criterion to choose this function $T$ is the low probability of the error $\Pr(T(x) \neq y)$. Thus, we define the expected risk of a function $\psi \in \Omega$ as

$$E_P(\psi) := \int 1_{\psi(x) \neq y} dP(z)$$

(36)

where

$$1_{T(x) \neq y} = \begin{cases} 1, & \text{if } T(x) \neq y; \\ 0, & \text{otherwise}. \end{cases}$$

Then, the desired function $T$ satisfies

$$E_P(T) = \min_{\psi \in \Omega} E_P(\psi).$$

(38)

Theorem 5 (resp. Theorem 7) provides a one-step method to obtain a BPUNN (resp. BPSNN) based on a given truth table. The resulting networks minimize the expected risk (36) in a special deterministic case, where each input $x$ corresponds deterministically to an output value $y$.

Generally, if the distribution $P(z)$ is unknown, the target function $T$ cannot be directly obtained by minimizing (36). Instead, we can utilize the empirical risk minimization to handle this issue. Given a function class $\Omega$ and an independently and identically distributed (i.i.d.) sample set $S^M = \{z_m\}_{m=1}^M$ drawn from $\mathcal{Z}$ with $z_m = (x_m, y_m)$, we define the empirical risk of $\psi \in \Omega$ by

$$\hat{E}_M(\psi) := \frac{1}{M} \sum_{m=1}^M 1_{\psi(x_m) \neq y_m}$$

(39)

which is an approximation to the expected risk (36). We then choose a $\psi_M$ to minimize the empirical risk over $\Omega$ and deem $\psi_M$ as an estimation to $T$ with respect to the sample set $S^M$. Therefore, we have to consider the asymptotic behavior of $E_P(\psi_M) - E_P(T)$ when the sample number $M$ goes to the infinity (see [40], [41]). Since $\hat{E}_M(T) - \hat{E}_M(\psi_M) \geq 0$, we have

$$E_P(\psi_M) = E_P(\psi_M) - E_P(T) + E_P(T)$$

$$\leq \hat{E}_M(T) - \hat{E}_M(\psi_M) + E_P(\psi_M)$$

$$- E_P(T) + E_P(T)$$

$$\leq 2 \sup_{\psi \in \Omega} |E_P(\psi) - \hat{E}_M(\psi)| + E_P(T).$$

Thus, we have

$$0 \leq E_P(\psi_M) - E_P(T) \leq 2 \sup_{\psi \in \Omega} |E_P(\psi) - \hat{E}_M(\psi)|.$$ (40)

Therefore, the upper bound

$$\sup_{\psi \in \Omega} |E_P(\psi) - \hat{E}_M(\psi)|$$

becomes a major concern in statistical learning theory, and is called the risk bound.

The risk bound measures the probability that a function produced by an algorithm has a sufficiently small error and is usually used to analyze the learnability of function classes. It is well known that risk bounds can be obtained by incorporating complexity measures of function classes [40]. Since this paper concerns binary issues, we only adopt the complexity measures for function classes with the range $[0, 1]$.

Let us define

$$\hat{\psi}_S := (\psi(z_1), \psi(z_2), \ldots, \psi(z_M))$$

(41)

and

$$\Lambda(\Omega, S^M) := \left\{ \hat{\psi}_S \mid \psi \in \Omega \right\}.$$ (42)

Then, the growth function is defined by

$$G(\Omega, M) := \max_{S^M \in \mathcal{Z}^M} |\Lambda(\Omega, S^M)|,$$

(43)

where $|S|$ stands for the cardinality of a set $S$. Since the range of $\Omega$ is $[0, 1]$, we have $G(\Omega, M) \leq 2^M$. Therefore, the VC dimension of the function class $\Omega$ can be defined as follows [41].

Definition 6: Let $\Omega$ be a function class with the range $[0, 1]$. Then, the VC dimension of $\Omega$ is defined as

$$VCDim(\Omega) := \max \left\{ M > 0 \mid G(\Omega, M) = 2^M \right\}.$$ (44)

In this paper, we consider the function class $\Omega_J (J \geq 1)$ composed of BPUNNs (resp. BPSNNs) with $J$ hidden nodes defined in (14) [resp. (19)]. According to (42), (43), and Definition (44), we can get the following theorem on the complexity of the function class $\Omega_J$. The proof of this theorem is presented in Appendix A.

Theorem 8: Let $\Omega_J (J \geq 1)$ be the function set of BPUNNs or BPSNNs with $J$ hidden nodes. Then, we have

$$\Lambda(\Omega_J, S^M) \leq G(\Omega_J, M) \leq \sum_{j=0}^J C_M^j$$

(45)

and

$$VCDim(\Omega) \leq J$$

(46)

where $C_M^j$ stands for a binomial coefficient.
In the next theorem, the risk bounds of BPUNN and BPSNN are given in terms of the VC dimension. The proof of this theorem is postponed to Appendix B.

**Theorem 9:** Let $\Omega_j$ $(j \geq 1)$ be a set composed of BPUNNs or BPSNNs with $J$ hidden nodes, and $S^M = \{z_m\}_{m=1}^M$ be an i.i.d. sample set drawn from $Z$. Then, for any $\xi > 0$ such that $M\xi^2 \geq 2$, we have

$$\Pr \left\{ \sup_{\psi \in \Omega_j} \left| E_P(\psi) - \hat{E}_M(\psi) \right| > \xi \right\} \leq 2 \exp \left( -M\xi^2 \frac{2}{8} \right).$$  \hspace{1cm} (47)

**B. Learnability of BPUNN and BPSNN**

We now study the learnability of BPUNN and BPSNN in the probably approximately correct (PAC) learning framework [42]. In this framework, under the assumption that the samples are drawn from an arbitrary distribution, the main concern is the existence of a learning algorithm that leads to an estimation to the target function related to the distribution with a high probability. Anthony [13] applied this framework to neural networks and analyzed the PAC-learnability of some traditional neural networks.

Let $\Omega$ be a function class and $S^M = \{z_m\}_{m=1}^M$ be an i.i.d. sample set drawn from an arbitrary probability distribution on $Z$. Let us regard a learning algorithm as a function $L : S^* \rightarrow \Omega$, where $S^*$ is the set composed of all the possible sample sets, i.e., of all the subsets of the input space. Then, for instance, $L(S^M)$ denotes the learning result with respect to the sample set $S^M$. Define the smallest error over $\Omega$ as

$$E_P^*(\Omega) := \inf \{ E_P(\psi) : \psi \in \Omega \}.$$  \hspace{1cm}

Then, the PAC-learnability of the function class $\Omega$ can be formalized as follows.

**Definition 7:** Let $\Omega$ be a function class with the range \{0, 1\}. We say that $\Omega$ is PAC-learnable if there is a learning algorithm $L$ such that, for any $\delta, \xi > 0$, there exists a number $M(\delta, \xi)$ satisfying

$$\Pr \left\{ \sup_{S^M \in Z^M} \left| E_P(\left( L(S^M) \right)) - E_P^*(\Omega) \right| > \xi \right\} \leq \delta \hspace{1cm} (48)$$

for any $M > M(\delta, \xi)$ and any probability distribution $P$ on $Z$.

An equivalent expression of (48) is that, for any probability distribution $P$ on $Z$

$$\lim_{M \to +\infty} \Pr \left\{ \sup_{S^M \in Z^M} \left| E_P(\left( L(S^M) \right)) - E_P^*(\Omega) \right| > \xi \right\} = 0.$$

The next theorem confirms the PAC-learnability of BPUNN and BPSNN. Its proof is given in Appendix C.

**Theorem 10:** Let $\Omega_j$ $(j \geq 1)$ be a set composed of BPUNNs or BPSNNs with $J$ hidden nodes. Then, $\Omega_j$ is PAC-learnable.

**C. Learnability of BPUPS**

At the end of this section, we discuss the learnability of BPUPS that is proposed to implement incomplete truth tables in Section VII. For convenience, we reform the output $(y_1, y_2)$ of BPUPS (Fig. 5) as follows:

$$y = y_1 + y_2. \hspace{1cm} (50)$$

It is clear that $y \in \{0, 1, 2\}$. By combining Fig. 5 and (50), for different kinds of inputs, the reformed BPUPS will generate the following outputs.

1) If a truth assignment is inputted into the network, the output $y$ equals to 2.

2) If a false assignment is inputted, the output $y$ equals to 0.

3) For a missing assignment, the output $y$ equals to 1.

Therefore, the reformed BPUPS is equivalent to the original BPUPS from a functional perspective. Based on the reformed BPUPS, we study the risk bound of BPUPS and its learnability.

In the rest of this section, the reformed BPUPS is called BPUPS if no confusion arises.

We consider a function class $\Omega_{I,J}$ $(I, J \geq 1)$ composed of BPUPS with $I + J$ hidden nodes shown in Fig. 5. Similar to Theorem 8, we have the following result on the complexity of $\Omega_{I,J}$.

**Theorem 11:** Let $\Omega_{I,J}$ $(I, J \geq 1)$ be the function set of BPUPS with $I + J$ hidden nodes shown in Fig. 5. Then, we have

$$\Lambda \left( \Omega_{I,J}, S^M \right) \leq G(\Omega_{I,J}, M) \leq \sum_{i=0}^{I+J} C_M^i. \hspace{1cm} (51)$$

Subsequently, following the style of Theorem 9, we can obtain the risk bound of BPUPS as follows.

**Theorem 12:** Assume that $\Omega_{I,J}$ $(I, J \geq 1)$ is a function set of BPUPS with $I + J$ hidden nodes shown in Fig. 5. Let $S^M = \{z_m\}_{m=1}^M$ be an i.i.d. sample set drawn from $Z$. Then, for any $\xi > 0$ such that $M\xi^2 \geq 8$, we have

$$\Pr \left\{ \sup_{\psi \in \Omega_{I,J}} \left| E_P(\psi) - \hat{E}_M(\psi) \right| > \xi \right\} \leq 2 \exp \left( -\sum_{i=0}^{I+J} C_M^i \frac{M\xi^2}{32} \right). \hspace{1cm} (52)$$

The above theorem can be proved in the same way as Theorem 9 and thus we omit it. According to Theorem 12, we have the following result on the PAC-learnability of BPUPS.

**Theorem 13:** Let $\Omega_{I,J}$ $(I, J \geq 1)$ is a function set of BPUPS with $I + J$ hidden nodes shown in Fig. 5. Then, $\Omega_{I,J}$ is PAC-learnable.

We omit the proof of Theorem 13, because it is similar to that of Theorem 10.

**IX. CONCLUSION**

In this paper, we proposed two binary HONNs: BPUNN and BPSNN. They both correspond to a functionally complete set $\{\lor, \land, \neg\}$ that provides a sufficient condition for realizing arbitrary Boolean functions (see Definition 1). Based on this
point, we theoretically proved that BPUNN and BPSNN can realize arbitrary Boolean functions. Numerical experiments were provided to show the excellent performance of BPUNN and BPSNN for realizing Boolean functions compared with other neural networks.

The structures of BPUNN and BPSNN were respectively derived from those of PUNN and PSNN. The original network structures were made so as to make them correspond to the functionally complete set \( \{ \vee, \wedge, \neg \} \). In particular, the logical disjunction \( \lor \) and the logical conjunction \( \land \) were implemented by the addition \( + \) and the multiplication \( \times \), respectively. A key point of our approach was to implement the unary logical operator \( \neg \) by using a special input-to-hidden weighting operation (Figs. 1 and 3). By using such structures, all weights of BPUNN and BPSNN can be chosen simply as either \( 1 \) or \( -1 \) by applying a one-step training.

Our approach could deal with both complete truth table and incomplete truth table. Given a complete truth table of \( N \) variables with both truth and false assignments, the corresponding Boolean function could be realized by accordingly choosing a BPUNN or a BPSNN such that at most \( 2^N-1 \) hidden nodes were needed. The two exceptional Boolean functions with \( 2^N \) truth assignments but no false assignment, or with \( 2^N \) false assignments but no truth assignment, could be realized by BPUNN or BPSNN, respectively, with \( 2^N \) hidden nodes. On the other hand, a new network BPUPS based on a collaboration of BPUNN and BPSNN was proposed to deal with incomplete truth tables. Numerical experiments were also provided to support our theoretical results.

The above conclusion was based on a deterministic scenario, where the target Boolean function is deterministic. We also considered a more general scenario where the samples could be evaluated with certain probability. We showed that the VC dimension of the function class composed of BPUNN or BPSNN with \( J \) hidden nodes is at most \( J \), and then obtained the risk bounds of BPUNN and BPSNN based on the VC dimension. The PAC-learnability of BPUNN and BPSNN was confirmed by using the risk bounds. Finally, we presented the risk bound of BPUPS and studied the learnability of BPUPS. It should be pointed out that these results are preliminary and further discussions remain to be done for BPUPS in future works.

### APPENDIX A

**PROOF OF THEOREM 8**

We only prove the theorem for BPUNN. The proof for BPSNN can be done analogously.

**Proof of Theorem 8:** According to (42) and (43), it is clear that \( \Lambda(\Omega_j, S^M) \leq G(\Omega_j, M) \).

Next, assume that the function class \( \Omega_j' \) contains the BPUNNs with all possible weights. By Theorem 5, \( \Omega_j' \) is equivalent to a set of all Boolean functions with at most \( J \) truth assignments. We then separate \( \Omega_j' \) into \( J + 1 \) disjoint subset \( \Omega_j' \) \( (0 \leq j \leq J) \) such that

\[
\Omega_j' = \bigcup_{j=0}^{J} \Omega_j' \quad (53)
\]

where \( \Omega_j' \) is composed of the \( J \)-hidden-node BPUNNs corresponding to the Boolean functions with \( j \) truth assignments. According to (41)–(43), and (53), for any \( 0 \leq j \leq J \), we have

\[
G(\Omega_j', M) = C_j
\]

and then

\[
G(\Omega_j', M) = G \left( \bigcup_{j=0}^{J} \Omega_j', M \right) = \sum_{j=0}^{J} G(\Omega_j', M) = \sum_{j=0}^{J} C_j
\]

(55)

Noting \( \Omega_j \subseteq \Omega_j', G(\Omega_j, M) \leq G(\Omega_j', M) \), we get (45).

Finally, (46) results from Definition 6 and (45). This completes the proof.

### APPENDIX B

**PROOF OF THEOREM 9**

In order to prove the theorem, we need the following lemmas.

**Lemma 1** [40]: Assume that \( \Omega \) is a function class with the range \( \{0, 1\} \), and let \( S^M = (z_m)_{m=1}^{2^M} \) be two i.i.d. samples both drawn from \( \mathcal{Z} \). Then, for any \( \xi > 0 \) such that \( M\xi^2 \geq 2 \)

\[
\Pr \left\{ \sup_{\psi \in \Omega} |E_p(\psi) - \hat{E}_M(\psi)| > \xi \right\} \leq 2 \Pr \left\{ \sup_{\psi \in \Omega} |\hat{E}'_M(\psi) - \hat{E}_M(\psi)| > \frac{\xi}{2} \right\}.
\]

(56)

Based on Lemma 1, we can obtain the following result.

**Lemma 2**: Let \( \Omega \) be an indicator function class with the range \( \{0, 1\} \) and \( S^{2M} = (z_m)_{m=1}^{2M} \) be a set of \( 2M \) i.i.d. samples drawn from \( \mathcal{Z} \). Then, for any \( \xi > 0 \) such that \( M\xi^2 \geq 2 \), we have

\[
\Pr \left\{ \sup_{\psi \in \Omega} |E_p(\psi) - \hat{E}_M(\psi)| > \xi \right\} \leq 2E \left\{ |\Lambda(\Omega, S^{2M})| \right\} \max_{\psi \in \Omega} \Pr \left\{ |E_p(\psi) - \hat{E}_M(\psi)| > \frac{\xi}{2} \right\}.
\]

(57)

**Proof**: According to Lemma 1, for any \( \xi > 0 \) such that \( M\xi^2 \geq 2 \), we have

\[
\Pr \left\{ \sup_{\psi \in \Omega} |E_p(\psi) - \hat{E}_M(\psi)| > \xi \right\} \leq 2 \Pr \left\{ \sup_{\psi \in \Omega} |\hat{E}'_M(\psi) - \hat{E}_M(\psi)| > \frac{\xi}{2} \right\}
\]

(58)
Since $S^M$ and $S^M$ are both independently drawn from an identical distribution, according to (58), we have
\[ \Pr \left\{ \sup_{\psi \in \Omega} \left| E_P(\psi) - \hat{E}_M(\psi) \right| > \xi \right\} \leq 2E \left| \Lambda(\Omega, S^2M) \right| \max_{\psi \in \Omega} \left\{ \left| E_P(\psi) - \hat{E}_M(\psi) \right| > \frac{\xi}{4} \right\}. \] (59)

This completes the proof.

Next, we introduce a concentration inequality for the i.i.d. learning process.

Lemma 3 [40]: Let $z_1, \ldots, z_M$ be $M$ i.i.d. random variables with $\psi(z) \in [a, b]$. Then, for all $\xi > 0$, we have
\[ \Pr \left( \left| \hat{E}_M(\psi) - E_P(\psi) \right| > \xi \right) \leq 2 \exp \left( -\frac{2M\xi^2}{(b-a)^2} \right). \] (60)

Now, we are ready to prove the theorem.

Proof of Theorem 9: According to Theorem 8 and Lemmas 2 and 3, we can obtain (47). This completes the proof.

APPENDIX C

PROOF OF THEOREM 10

In order to prove the theorem, we need the following lemma.

Lemma 4: For any $M \geq J$
\[ \sum_{j=0}^{J} C_M^j \leq \left( \frac{M}{J} \right)^J. \] (61)

Proof: Since $M \geq J$, we have
\[ \sum_{j=0}^{J} C_M^j \leq \left( \frac{M}{J} \right) \sum_{j=0}^{J} C_M^j \left( \frac{J}{M} \right)^j \leq \left( \frac{M}{J} \right) \left( 1 + \frac{J}{M} \right)^M \leq \left( \frac{M}{J} \right)^J \exp J. \]

Next, we begin to prove Theorem 10.

Proof of Theorem 10: By combining (47), Theorem 8, and Lemma 4, we can obtain, if $M > J/2$
\[ \Pr \left\{ \sup_{\psi \in \Omega} \left| E_P(\psi) - \hat{E}_M(\psi) \right| > \xi \right\} \leq 2 \exp \left( \frac{J^2}{(\ln 2M - \ln J) - \frac{M\xi^2}{8}} \right). \] (62)

Let $\psi^* \in \Omega$ be the function satisfying $E(\psi^*) = E^*_P(\Omega)$. For any i.i.d. sample set $S^M$ drawn from $Z$, we have $E_M(\psi^*) - \hat{E}_M(L(S^M)) \geq 0$, and then
\[ E_P(L(S^M)) = E_P(L(S^M)) - E^*_P(\Omega) + E^*_P(\Omega) \leq E_M(\psi^*) - \hat{E}_M(L(S^M)) + E_P(L(S^M)) - E_P(\psi^*) + E_P(\psi^*) \leq 2 \sup_{\psi \in \Omega} \left| \hat{E}_M(\psi) - E_P(\psi) \right| + E_P(\psi^*). \]

Namely, for any i.i.d. sample set $S^M$, there holds
\[ E_P \left( L \left( S^M \right) \right) - E_P(\psi^*) \leq 2 \sup_{\psi \in \Omega} \left| \hat{E}_M(\psi) - E_P(\psi) \right|. \] (63)

Therefore, according to (62) and (63), we have, if $M > J/2$
\[ \Pr \left\{ \sup_{\psi \in \Omega} \left| E_P(L(S^M)) - E^*_P(\Omega) \right| > \xi \right\} \leq \Pr \left\{ \sup_{\psi \in \Omega} \left\{ \left| E_P(L(S^M)) - E_P(\psi) \right| \right\} > \xi \right\} \leq 2 \exp \left( \frac{J^2}{(\ln 2M - \ln J) - \frac{M\xi^2}{32}} \right) \leq 2 \exp \left( \frac{\left( \ln(2e) + \ln(M/J) \right)}{M(J)} - \frac{\xi^2}{32} \right). \] (64)

We see clearly that, for a fixed $J$, the right-hand side of the above equation goes to 0 when $M \to \infty$. This completes the proof.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers and the editors for their valuable comments and suggestions.

REFERENCES


EDITOR QUERY

EQ:1 = Please provide the accepted date for this article, since this is not available in the provided “metadata.xml” file.

AUTHOR QUERY

AQ:1 = Please provide the expansion for BPUPS.