Received September 23, 2020, accepted September 26, 2020, date of publication October 8, 2020, date of current version October 27, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3029576*

# Training Hardware for Binarized Convolutional Neural Network Based on CMOS Invertible Logic

**DUCKGYU SHIN[1], NAOYA ONIZAWA[1], (Member, IEEE), WARREN J. GROSS[2], (Senior Member, IEEE), AND TAKAHIRO HANYU[1], (Senior Member, IEEE)**
[1]Research Institute of Electrical Communication, Tohoku University, Sendai 980-8577, Japan
[2]Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 0E9, Canada

Corresponding author: Duckgyu Shin (duckgyu.shin.p4@dc.tohoku.ac.jp)

This work was supported in part by JST PRESTO Grant JPMJPR18M5, in part by the MEXT Brainware LSI Project, and in part by the WISE Program for AI Electronics, Tohoku University.

**ABSTRACT** In this article, we implement fast and power-efficient training hardware for convolutional neural networks (CNNs) based on CMOS invertible logic. The backpropagation algorithm is generally hard to implement in hardware because it requires high-precision floating-point arithmetic. Even though parameters of CNNs can be represented by fixed points or even binary during inference, it is still represented by floating points during training. Our hardware uses low-precision data representation for both inference and training. For hardware implementation, we exploit CMOS invertible logic for training. The use of invertible logic enables logic circuits to compute probabilistic bidirectional operation (forward and backward modes) and can be implemented by stochastic computing. The proposed hardware obtains parameters of neural networks such as weights directly from given data (an input feature map and a true label) without backpropagation. For performance evaluation, the proposed hardware is implemented on an FPGA and trains a binarized 2-layer convolutional neural network model using a modified MNIST dataset. This implementation shows an energy efficiency improvement of approximately 134x compared to that of a CPU implementation that executes the training of the same model as that used in the proposed hardware. Training on the proposed hardware is approximately 40x faster than training on the CPU using the backpropagation algorithm while maintaining almost the same cognition accuracy.

**INDEX TERMS** CMOS invertible logic, neural network, stochastic computing.

## I. INTRODUCTION

Advances in the development of deep neural networks (DNNs) [1] have led to the explosive growth of their use in AI applications such as image cognition [2] and speech cognition [3]. The growth of in the use of DNNs has led to a demand for deep-learning (DL) hardware accelerators for inference and training that are two main processes of DNNs. Hardware accelerators for inference have been implemented using field-programmable gate arrays (FPGAs) [4] or application specific integrated circuits (ASICs). In addition, the process of inference can be simplified through low-precision computing such as fixed-point arithmetic [5] or binary precision [6], [7]. For example, inference of binarized neural networks (BNNs) can be implemented by XNOR gates and bit counters because of binarized activations and weights [6].

However, the hardware for training has less focus than that for inference [8]. A general algorithm of training DNNs is backpropagation [9], which requires floating-point computations. Even BNNs that use binarized weights during inference require floating-point computations for their training [6]. Several studies on training hardware using the backpropagation algorithm have recently been reported [10]–[12]. Training using a hardware accelerator [10], [11] shows insufficient improvement in terms of latency compared to the improvement obtained for inference carried out on the same accelerator. Moreover, these works require floating-point arithmetic during training even though they represent low-precision data during inference.

In traditional training, parameters of the DNNs, such as weights, are updated by error propagated backward, which

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

is represented by floating-point arithmetic. Backpropagation performs many floating-point computations, and thus, it is difficult to implement the training hardware with backpropagation. Usually, training processes are accelerated by graphical processing units (GPUs) based on floating-point processing-oriented architectures. However, the GPU-based acceleration is still time-consuming (e.g., several days) when training a large number of data and has high power consumption [13].

In this article, we implement energy-efficient training hardware using CMOS invertible logic. Invertible logic provides a capability for probabilistic bidirectional operations (forward and backward modes) [14] using a nanomagnetic device model [15] and a Boltzmann machine [16]. The device model can be implemented by a magnetic tunnel junctions [17] or approximated by stochastic computing that represents CMOS invertible logic [18]. In our previous study, the hardware implementation of CMOS invertible logic was limited to training hardware for a single binarized perceptron [19]. In this work, the application of the training hardware is expanded to binarized convolutional neural networks (BCNNs). Invertible circuits compute inputs corresponding to a given output on a backward operation. Through the backward operation, the proposed training hardware can directly obtain the weights of DNNs without backpropagation. Furthermore, invertible logic circuits can operate similar to the typical logic gates at the forward mode; thus, the proposed training hardware can perform inference if the input data and weights are given. This means that inference and training can be run on the same hardware with the same precision.

For evaluation, the proposed hardware trains a 2-layer BCNN model that contains 2 binarized convolutional layers. This network model contains only convolutional layers and has binarized weights of filters without biases. A simplified and binarized Modified National Institute of Standard and Technology (MNIST) dataset [20] is used to train and test the model. The proposed hardware trains the modified dataset and obtains binarized weights for the model. It is implemented on a Xilinx Kintex-7 FPGA under a clock frequency of 12.5 MHz. Training using the proposed hardware achieves approximately 40x faster latency than that of traditional training on a CPU while maintaining almost the same cognition accuracy. In addition, the training hardware demonstrates a power efficiency improvement of approximately 134x compared to that of the CPU used to perform traditional training.

The contribution of this work is to extend the application of the invertible hardware to the training of multilayer BCNNs without backpropagation. In realizing the training hardware for BCNNs using CMOS invertible logic, a main issue is converting convolution functions to a Hamiltonian that serves as the basis of the CMOS invertible circuit. The conversion method that is introduced in this article is a solution to the floating-point arithmetic-free training that can be implemented on the CMOS invertible hardware. The proposed

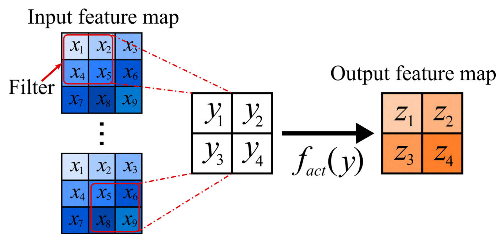

**FIGURE 1. The computation processes of the 2D convolution function. The filter is multiplied while shifting over the input feature map, and $f_{act}$ is the activation function.**

hardware can perform training with the same precision on the same hardware as performing inference because it uses the bidirectional operation on CMOS invertible logic.

The rest of this article is organized as follows. Section II reviews binarized neural networks and backpropagation. Section III introduces CMOS invertible logic. Section IV designs the Hamiltonian corresponding to the the BCNN model. Section V implements the proposed training hardware using the Hamiltonian designed in Section IV. Section VI introduces the test environment including the BCNN model and the dataset. Section VII evaluates the performance of the proposed hardware and compares this hardware implementation to the CPU and GPU implementations. Section VIII concludes the paper.

## II. PRELIMINARIES

### A. INFERENCE OF THE BCNN

Many recent models of DNNs consist of CNNs composed of multiple convolution functions (convolution layers) shown in Fig. 1. The convolution function multiplies the weights of a shifting filter over the input data called the input feature map and applies an accumulator on the results of the multiplication. After calculating $y$, the activation function $f_{act}$ is applied on $y$ in order to construct a much deeper NN model [21]. The output of the convolution function is an output feature map $z$, which is the input data for the next convolution layer. To reduce memory usage or power consumption, DNNs with low-precision data representation such as binarized convolutional neural networks (BCNNs) have been investigated.

BCNNs represent weights and activations using binary precision (+1, −1) during forward propagation [6]. The weights and activations are binarized using two different functions: deterministic and stochastic binarization. The deterministic binarization function is the sign function:

$$x^b = Sign(x) = \begin{cases} +1 & x \geq 0, \\ -1 & x < 0, \end{cases} \tag{1}$$

where $x^b$ are the binarized weights or activations. The second binarization function uses the hard sigmoid function:

$$\sigma = max(0, min(1, \frac{x+1}{2})), \tag{2}$$

and is defined as:

$$x^b = \begin{cases} +1 & p = \eta(x), \\ -1 & 1-p. \end{cases} \tag{3}$$

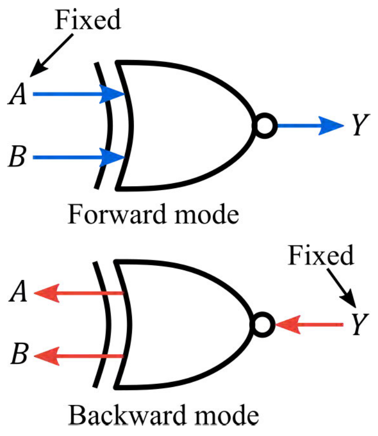

**FIGURE 2.** **Concept of bidirectional operation (forward and backward) based on invertible logic. The output is obtained from the fixed input at the forward mode. In contrast, the output is fixed, and the input is obtained at the backward mode.**

The multiplication of binarized weights and activation can be implemented using XNOR gates and bit counters. For example, the result of multiplication when $x^b = 1$ and $W^b = -1$ is the same as the output of the XNOR gate when its inputs are 0 and 1. Because the results of multiplication also have binary representation, summation is realized using the bit counter.

### *B. TRAINING OF THE BCNN*

Backpropagation is a training algorithm that updates the parameters of DNNs with a gradient of an output error [9]. The error is calculated by an error function, $E$, that computes the differences between the output obtained from forward propagation, and the actual output given as a true label. After the error is calculated, the gradients of the error obtained by the chain rule are backpropagated through the entire network model. Parameters are updated by these back-propagated gradients according to the following equation: $W' = W - \eta \cdot \partial E/\partial W$, where $\eta$ is a learning rate. Back-propagation realizes a high cognition accuracy with DNNs, but it requires high-precision floating-point computations to maintain this accuracy. The floating-point arithmetic causes inefficiency with regard to energy consumption and leads to a low training speed. The weights and activations of the BCNNs are binarized during forward propagation. Despite the binarized parameters, the BCNNs update real-valued parameters using gradients that are calculated from binarized parameters. Therefore, BCNNs require floating-point arithmetic to precisely represent the gradients.

## III. CMOS INVERTIBLE LOGIC

### *A. HAMILTONIAN DESIGN*

Invertible logic [14] realizes probabilistic bidirectional operations (forward and backward modes) shown in Fig. 2. At the forward mode with fixed inputs, invertible logic circuits operate as typical logic circuits. In contrast, inputs are obtained corresponding to the fixed output in the backward mode. This unique feature of invertible logic is derived from a Boltzmann machine [16] and a nanomagnetic device model [14].

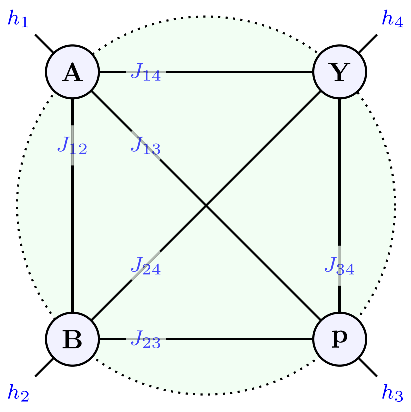

**FIGURE 3.** **A Hamiltonian configuration of an invertible XNOR gate. *A* and *B* are inputs of the XNOR gate, *Y* is the output, and *p* is the arbitrary node. *h* and *J* are obtained from the ground-state spin logic.**

The invertible logic circuit is a network of nodes with a bias ($h$) and interaction weights ($J$). These $h$ and $J$ are determined by a Hamiltonian ($H$) that represents an energy of the network and $H$ is given by

$$H = -\sum_i h_i m_i - \sum_{i<j} J_{ij} m_i m_j, \tag{4}$$

where $m_i = [-1, +1]$ means an output of a node. Fig. 3 shows a configuration of an invertible XNOR gate obtained by the $H$. The invertible XNOR gate has 2 input nodes (A and B), an output node (Y), and an arbitrary node (p) that does not correspond to the inputs and the output of the gate. The $h$ and $J$ for basic logic gates such as the XNOR gate can be obtained by ground-state spin logic [22], [23]. In invertible logic, the $h$ and $J$ are determined so that the Hamiltonian (energy) is minimum in the valid state, and in contrast, the Hamiltonian is greater than the minimum in the invalid state according to the truth table. For example, the truth table of the XNOR gate and states of nodes are shown in Table 1 including the energy at each states. When [A, B, p, Y] is [0, 0, 0, 1], the state is valid as the XNOR gate; therefore, the energy takes the minimal value $-4$. The correct state of the arbitrary node is regarded as an OR gate output of two inputs. As shown in the Table 1, the $h$ and $J$ for the minimum energy to be $-4$ are given by:

$$h_i = \begin{bmatrix} -1 & -1 & +2 & +1 \end{bmatrix}, \tag{5a}$$

$$J_{ij} = \begin{bmatrix} 0 & -1 & +2 & +1 \\ -1 & 0 & +2 & +1 \\ +2 & +2 & 0 & -2 \\ +1 & +1 & -2 & 0 \end{bmatrix}. \tag{5b}$$

The states of the input and output nodes are calculated by the following equations:

$$m_i(t+\tau) = sgn(rnd(-1,+1) + tanh(I_i(t+\tau))), \tag{6a}$$

$$I_i(t+\tau) = I_0(h_i + \sum J_{ij} m_j(t)), \tag{6b}$$

where $I_0$ is a scaling factor, $sgn$ is the sign function, and $rnd(-1, +1)$ is a random value between $-1$ and $+1$.

**TABLE 1. Energy of the invertible XNOR gate. It has 4 nodes; thus, the combinations of the state are $2^4$ in total. When every state is correct as the XNOR gate, the *h* and *J* are determined so that the energy is minimal. −1 means logical value '0' and +1 means '1'. The minimal energy is −4 in case of the XNOR gate.**

| Truth table | | | | State of nodes | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | p | Y | A | B | p | Y | $E$ | Validity |
| 0 | 0 | 0 | 0 | -1 | -1 | -1 | -1 | -2 | Invalid |
| 1 | 0 | 0 | 0 | +1 | -1 | -1 | -1 | 4 | Invalid |
| 0 | 1 | 0 | 0 | -1 | +1 | -1 | -1 | 4 | Invalid |
| 0 | 0 | 1 | 0 | -1 | -1 | +1 | -1 | -2 | Invalid |
| 0 | 0 | 0 | 1 | -1 | -1 | -1 | +1 | -4 | Valid |
| 1 | 1 | 0 | 0 | +1 | +1 | -1 | -1 | 14 | Invalid |
| 1 | 0 | 1 | 0 | +1 | -1 | +1 | -1 | -4 | Valid |
| 1 | 0 | 0 | 1 | +1 | -1 | -1 | +1 | -2 | Invalid |
| 0 | 1 | 1 | 0 | -1 | +1 | +1 | -1 | -4 | Valid |
| 0 | 1 | 0 | 1 | -1 | +1 | -1 | +1 | -2 | Invalid |
| 0 | 0 | 1 | 1 | -1 | -1 | +1 | +1 | 4 | Invalid |
| 1 | 1 | 1 | 0 | +1 | +1 | +1 | -1 | -2 | Invalid |
| 1 | 1 | 0 | 1 | +1 | +1 | -1 | +1 | 4 | Invalid |
| 1 | 0 | 1 | 1 | +1 | -1 | +1 | +1 | -2 | Invalid |
| 0 | 1 | 1 | 1 | -1 | +1 | +1 | +1 | -2 | Invalid |
| 1 | 1 | 1 | 1 | +1 | +1 | +1 | +1 | -4 | Valid |

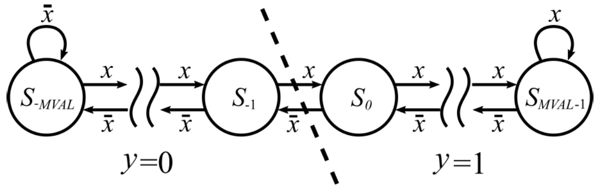

**FIGURE 4. FSM of approximated *tanh* function using stochastic computing. The number of states in FSM is determined by *MVAL*.**

By calculating Eq. 6a and Eq. 6b, the energy decreases to the minimum so that all nodes lead to the valid state. The invertible circuit realizes the bidirectional operation of a function required for the circuit by minimizing the energy.

### B. CIRCUIT DESIGN USING STOCHASTIC COMPUTING

An operation of nodes can be approximated by CMOS devices for implementation on FPGAs or ASICs with stochastic computing [18]. Stochastic computing represents real values as frequencies of '1' in stochastic bit streams [24]. It can be categorized into binary stochastic computing and integer stochastic computing according to the range of the real value represented by the stochastic bit stream. Let us denote by $P_x$ a probability of appearance of '1' in the stochastic bit stream $x \in \{0, 1\}$. In binary stochastic computing, the real value $X \in [-1 : 1]$ is represented by $2 \cdot P_x - 1$. In contrast, integer stochastic computing represents the real value, $X \in [-r : r]$, where $r \in \{1, 2, \ldots\}$, using one or more bit streams $x \in [-r : r]$.

Some functions such as multiplication or *tanh* are implemented in the hardware in an area-efficient manner using stochastic computing [25], [26]. A *tanh* function in Eq. 6a is approximated in a finite state machine (FSM) shown in Fig. 4, and *Stanh* (stochastic *tanh* function) is given by:

$$Stanh(MVAL, x) \approx tanh(x \cdot MVAL/2), \tag{7}$$

where *MVAL* is the number of states. *MVAL* changes the slope of the *Stanh* function; for example, if *MVAL* is larger, the slope of *Stanh* increases [25].

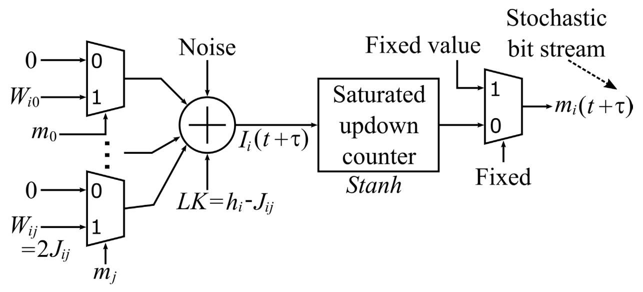

**FIGURE 5. Block diagram of a spin gate that composes a node block. Eqs. (8a) and (8b) are implemented in the spin gate circuit. $m_i$ is the output of the spin gate that is represented in a binary stochastic bit stream.**

In Eq. 6a, the output bit stream of *Stanh* must be summed with a random value, *rnd*, requiring stochastic addition. As an alternative, the weighted noise source with the corresponding magnitude denoted as $n_{rnd}$ is summed alongside input signals, and then, Eqs. (6a) and (6b) are approximated as

$$m_i(t+\tau) \approx sgn(tanh(I_i(t+\tau))), \tag{8a}$$

$$I_i(t+\tau) \approx h_i + \sum J_{ij}m_j(t) + n_{rnd} \cdot sgn(rnd(-1, +1)), \tag{8b}$$

where $I_0$ can be included in $h$ and $J$. The output, $m_i$, is represented by binary stochastic computing so that "$m_i = -1$" means a logical value of '0' and "$m_i = +1$" means '1'. For computation of nodes, Eqs. (8a) and (8b) are implemented in a spin gate circuit as shown in Fig. 5 using binary and integer stochastic computing. The input and output of the spin gate are represented in a binary stochastic bit stream as $M_i = (m_i + 1)/2$ ($M_i \in 0, 1$), where $M_i$ is the output of the spin gate. The approximated *Stanh* function is implemented using a saturated bit counter described in Fig. 5. The spin gate computes the output based on Eq. 8a and (8b) when it is unfixed, and in contrast, when the spin gate is fixed, it just delivers the fixed input value.

In this article, the proposed training hardware for BCNNs is designed based on CMOS invertible logic. To implement the invertible training hardware, a function used in the training process of DNNs needs to be converted to the Hamiltonian. The conversion method is explained in the following section.

## IV. DESIGN OF THE HAMILTONIAN TO TRAIN THE BCNN

### A. HAMILTONIAN DESIGN OF A CONVOLUTION FUNCTION

A convolutional function generates output feature maps while shifting filters on input feature maps, so its process is represented by sequential logic. To implement the CMOS invertible logic circuit that computes the convolutional layer, the convolutional function should be converted to the Hamiltonian; however, only a function that is represented by combinational logic can be converted to the Hamiltonian. For example, functions implemented in the invertible logic circuit can be represented by combinational logic in [18] and [19]. To convert the convolutional function to the Hamiltonian, the sequential process should be represented by combinational logic. The convolutional function can be mapped to

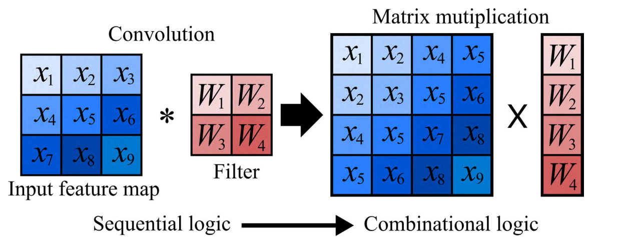

**FIGURE 6.** **Mapping convolution functions to the matrix multiplication for the conversion of the Hamiltonian. The convolution function is represented by sequential logic. The input feature map and the filter can be stretched to the matrix so that the convolution function becomes matrix multiplication represented by combinational logic.**

a matrix multiplication for high-speed computation and parallelization even though this results in inefficient memory storage [27].

Fig. 6 shows that the convolution function is mapped to the matrix multiplication. This mapping technique can be used to convert the convolution in the Hamiltonian. A local region filtered by the moving weight feature map is stretched in a row while duplicating pixels that are calculated repeatedly. The input feature map and the filter are expressed as the matrix; thus, the convolution is realized as the matrix multiplication. Each multiplication of rows, $x$, and columns, $W$ is implemented in a multiplier and an adder that are represented by combinational logic so that it can be converted to the Hamiltonian.

## B. HAMILTONIAN DESIGN OF A 2-LAYER BCNN

The convolutional function is converted to the Hamiltonian by representing the sequential process as combinational logic. In [6], the binarized matrix multiplication is computed by XNOR gates and a bit counter, as well as an activation function that is defined as Eq. 1. For example, the convolution in Fig. 6 is converted to the Hamiltonian through the process described in Fig. 7. The binary multiplication is implemented to a computation block that consists of four XNOR gates, a 4-input bit counter and a threshold function. The Hamiltonian of this computation block is composed of a combination of small Hamiltonians that are obtained from components in the computation block such as the XNOR gate, and the mechanism of combining $H$ is described in [18]. In the case of Fig. 7, there are four computation blocks for parallel multiplication, and each block is converted to Hamiltonians consisting of 4 input nodes, 4 weight nodes, an output node, and several arbitrary nodes.

When the computation blocks are converted to the Hamiltonians, the input nodes and the weight nodes in each Hamiltonian are duplicated because the convolutional function is mapped to the matrix multiplication. Duplicated nodes such as $x_2$ or $W_1$ have the same input value and need to generate the same output value. It is difficult to synchronize these stochastic nodes, and a hardware area becomes inefficient as the number of nodes increases. By overlapping duplicated nodes, the convolutional layer can be converted to the Hamiltonian possessing the same number of input and weight nodes as those of the convolutional layer. Two $x_2$ nodes in Fig. 7 are

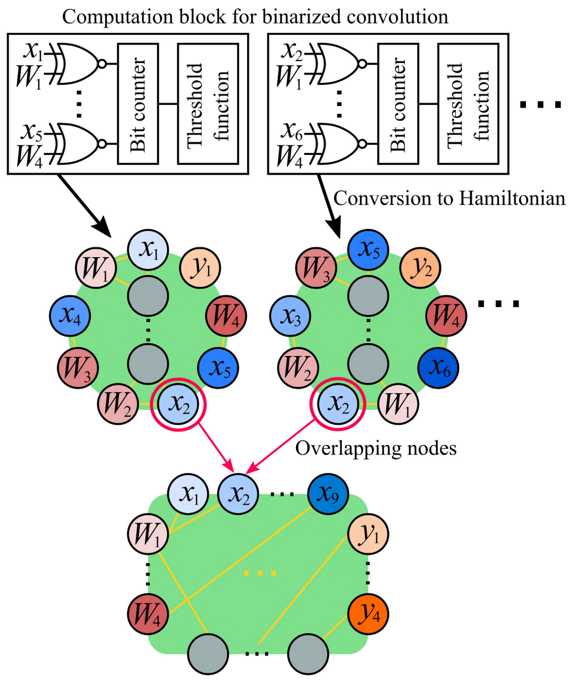
Computation block for binarized convolution
Bit counter
Threshold function
Conversion to Hamiltonian
Overlapping nodes

**FIGURE 7.** **Conversion of binarized convolution functions to the Hamiltonian. The binary matrix multiplication that is mapped from the binarized convolution is implemented to the computation block, which consists of XNOR gates, the bit counter, and the threshold. Computation blocks are converted to the Hamiltonians, and the Hamiltonians are combined while overlapping duplicated nodes.**

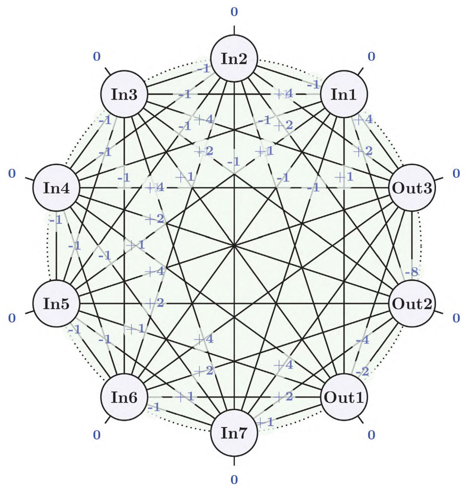

**FIGURE 8.** **Hamiltonian configuration of a 7-input bit counter. The bit counter counts '1' of the input, and its output is a 3-bit wise integer ranging from 0 to 7.**

overlapped into a single node, and the mechanism of nodes overlapping is same as that of combining small Hamiltonians. The number of the nodes in the Hamiltonian is restrained so that the training hardware based on this Hamiltonian can be implemented in an area-efficient manner.

In this article, a 2-layer BCNN model is converted to the Hamiltonian. To convert the BCNN model, the Hamiltonians of an XNOR gate, saturated bit counter, threshold function, and ripple carry adder (RCA) are required. Fig. 8 is a Hamiltonian configuration of the 7-input bit counter, and Fig. 9 is a threshold function. The RCA is used to accumulate the

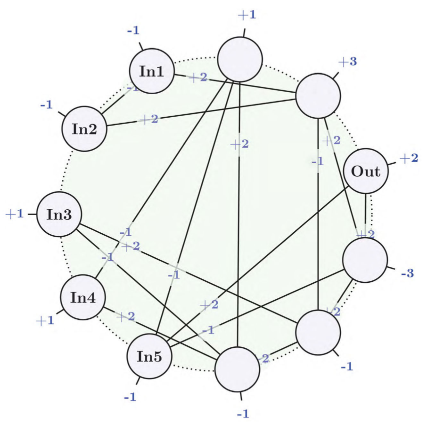

**FIGURE 9.** Hamiltonian configuration of a threshold function. The Hamiltonian of the threshold function has five input nodes, representing a 5-bit integer, an output node, and five arbitrary nodes. The threshold function operates in the same manner as a sign function, and its threshold value is 13 because each input node is represented by '1' and '0.'

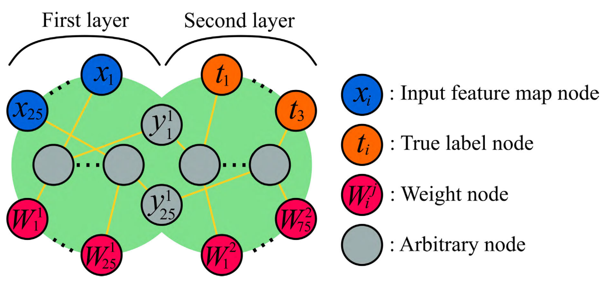

**FIGURE 10.** The Hamiltonian of the proposed training hardware for the 2-layer BCNN modes consists of 2645 nodes, of which 25 are input feature map nodes ($x$), 25 are first-layer weight nodes ($W^1$), 75 are second-layer weight nodes ($W^2$), 3 are true label nodes ($t$), and the remaining nodes are arbitrary nodes.

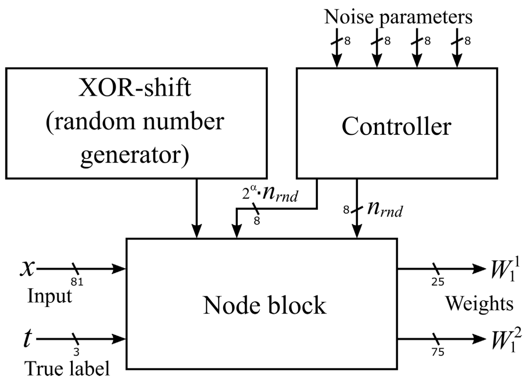

**FIGURE 11.** Structure of the proposed training hardware. The XOR-shift generates the random number $sgn(rnd(-1, +1))$. The controller manages the noise signal $n_{rnd}$ using several parameters. The node block is composed of nodes based on the Hamiltonian.

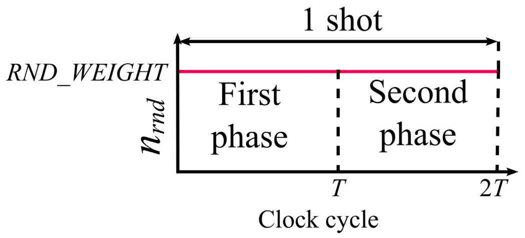

**FIGURE 12.** Parameterized noise signal, $n_{rnd}$, controlled by the noise parameters. The noise signal is characterized by four parameters, namely, *RNDWEIGHT*, *T*, *MVAL*, and $\alpha$. The magnitude of the noise maintains the same value during all phases. In the second phase, the output of the nodes is determined.

output of a bit counter because a 7-input bit counter does not have a sufficient number of inputs for a $5 \times 5$ filter. If a bit counter that has a greater number of inputs is used, the RCA can be removed, which decreases the number of nodes in the Hamiltonian. However, the Hamiltonian of the bit counter that has more than 7 inputs is difficult to obtain by ground-state spin logic because the number of nodes is too large. Combining these Hamiltonians, entire convolutional layers are converted to a single Hamiltonian that can be implemented to the hardware. Fig. 10 demonstrates the simplified Hamiltonian configuration of the proposed training hardware. It contains 2,645 nodes, of which 25 are input feature map nodes ($x$), 25 are first-layer weight nodes ($W^1$), 75 are second-layer weight nodes ($W^2$), 3 are true label nodes ($t$), and the remaining nodes are arbitrary nodes.

## V. HARDWARE IMPLEMENTATION

### A. HARDWARE ARCHITECTURE

Fig. 11 shows a structure of the proposed training hardware that is composed of a nodes block, a controller, and a random number generator. The proposed hardware trains the weights of the 2-layer BCNN that contains the inputs, $x$, weights of a first layer, $W^1$, weights of a second layer, $W^2$, and outputs, $t$. Each element of $x$ and $W$ is represented by binary precision because the proposed hardware uses the same precision that is used in the inference. The XOR-shift random number generator [28] generates a noise signal, $rnd(-1, +1)$, in Eq. 8b. The magnitude of the noise signal, $n_{rnd}$ is controlled by the controller through several parameters. The noise, $n_{rnd}$, ensures that the states of the nodes achieve the correct value and that the Hamiltonian converges to the global minimum. Similar to simulated annealing [29], the states of the nodes of the CMOS invertible circuits are fluctuated by the noise signals. The proposed hardware uses the parameterized flat noise, $n_{rnd}$, shown in Fig. 12, unlike the gradually decreasing noise signals that are usually used in simulated annealing. *RND_WEIGHT* is the magnitude of the noise signal, and $T$ is the number of the clock cycle in which the noise signal affects the node block. The input and first-layer weight nodes have a large number of interconnections than other nodes because they are overlapped nodes. In these nodes, for a correct effect at the nodes, the noise signals are scaled as follows:

$$n'_{rnd} = 2^{\alpha} \cdot n_{rnd}, \tag{9}$$

where $\alpha$ is a scaling factor, and $n_{rnd}$ is a noise signal shown in Fig. 12.

A structure of the node block is illustrated in Fig. 13. The nodes of the nodes block are categorized according to the Hamiltonian that is obtained from the 2-layer BCNN, and the details of the 2-layer network model are introduced in Section VI. When the proposed hardware performs training, the input and label nodes are fixed. The outputs of other nodes

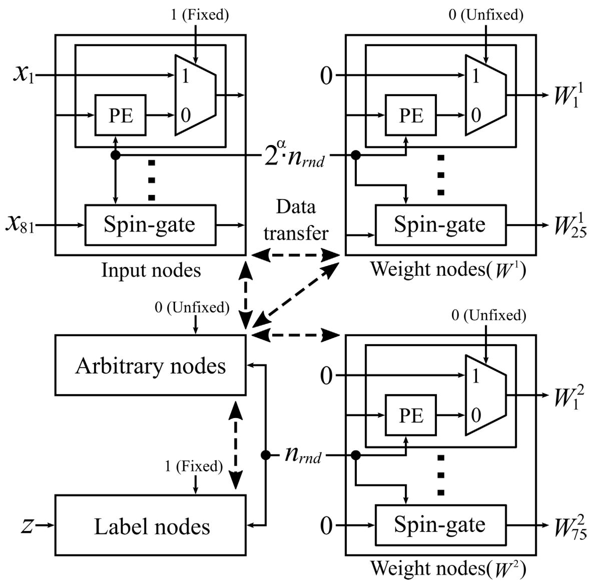

**FIGURE 13.** Structure of the node block that consists of spin gates. The nodes in the node block are categorized according to the Hamiltonian. The input and label nodes are fixed, and the weight nodes are fluctuated by the noise signal. The noise of the input and the first layer weight nodes is scaled using $\alpha$ because they have a larger number of connections than other nodes.

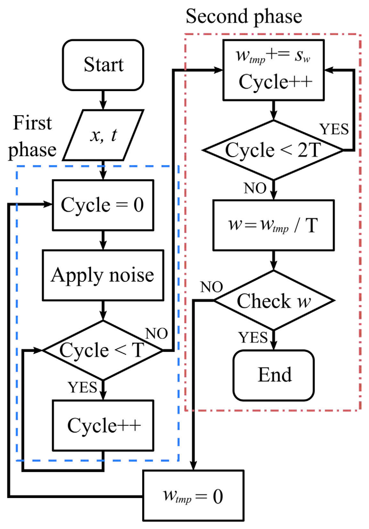

**FIGURE 14.** The invertible training algorithm of the proposed hardware. A flow chart describes the process of the training algorithm until cycle $2T$. $s_w$ is the outputs of weight nodes. $x$ and $t$ in this flow chart represent the individual training data and true label, respectively, rather than the entire dataset.

such as weight and arbitrary nodes are not fixed, so they are fluctuated by the $n_{rnd}$, and their states and the states of other nodes are governed by Eqs. (8a) and (8b). The nodes are implemented by spin gates described by Eqs. (8a) and (8b) , and the spin gate circuit is described in Fig. 5.

### B. INVERTIBLE TRAINING ALGORITHM

A training algorithm of the proposed hardware is designed based on an invertible binarized perceptron developed in our previous work [19]. The proposed hardware trains each datum in parallel and obtains the temporal weights, $w$, corresponding to each trained datum. Fig. 14 shows the process of obtaining each $w$ during a 1-shot ($2T$ cycles). While training, an operation of the nodes is divided into two phases, and the first and second phases are shown in Fig. 12. The nodes are fluctuated by noise signals, and the output of the nodes is not determined during the first phase. During the second phase, the proposed hardware accumulates $w_{tmp}$ using outputs of the weight nodes, $s_w$, and then, $w$ is determined as $w_{tmp}/T$. The obtained $w$ is checked for validity using its corresponding training data. The proposed hardware counts this $2T$ cycles process as one shot. The number of shots depends on the training data because the training of each datum finishes at different number of shots $N_{data}$. The total number of clock cycles, $N_{cycle}$, during which the hardware trains all of the dataset determines the training time of the proposed hardware and is defined as

$$N_{cycle} = 2T \cdot N_{data}. \tag{10}$$

The proposed hardware performs this process for the entire dataset and obtains $w$ equivalent to the number of data. The proposed hardware determines the weights of the trained BCNN model, $W$, by averaging all $w$ of the values. After obtaining $W$, $W$ is evaluated for the accuracy of the trained model using the test data.

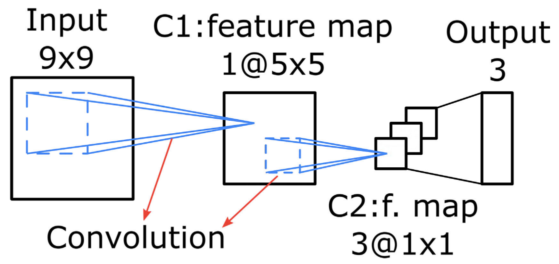

**FIGURE 15.** Description of the 2-layer BCNN that the proposed hardware trains. It contains 2 binarized convolutional layers, C1 and C2, which have 5 × 5-pixel filters.

## VI. EXPERIMENT SETUP

### A. DATASET AND NETWORK

Fig. 15 illustrates the 2-layer neural network that the proposed hardware trains. It contains 2 binarized convolutional layers without biases, a threshold function as an activation, and an output layer. A C1 layer takes 9 × 9-pixel images of the input feature maps and calculates the convolution using a 5 × 5-pixel filter; thus, its output feature map is 5 × 5 pixels. A C2 layer has three 5 × 5-pixel filters and generates a classification label.

The training dataset is a modified MNIST dataset of 9×9-pixel binary images shown in Fig. 16. The simplification has 3 steps, and each step is described as follows. First, images of '0', '1', and '5' are extracted from the original MNIST, and true labels are attached. Second, these images are shrunk to 9 × 9-pixel images. Finally, the images are binarized using adaptive thresholding with a Gaussian filter. The dataset includes 10,800 training images and 3600 testing images. Images of '0', '1', and '5' are used in the training 2-layer BCNN because they are easy to distinguish from each other after the binarization.

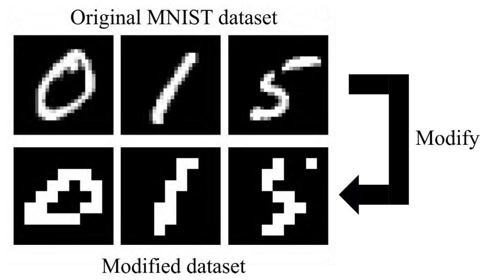

**FIGURE 16.** Dataset modification from the MNIST dataset. The modified dataset consists of images of '0,' '1,' and '5' that are shrunk to 9 × 9 pixels and binarized.

**TABLE 2.** Resource utilization for the proposed training hardware on FPGA. The FPGA board is a Digilent Genesys2 (Kintex-7) The proposed training hardware consumes 83.57% of LUT and 10.54% of FF.

| Look up table (LUT) | Flip Flops (FF) |
|---|---|
| 170317 / 203800 (83.57%) | 42951 / 407600 (10.54%) |

### B. TEST ENVIRONMENT

The proposed training hardware is implemented on an FPGA. The FPGA board is a Digilent Genesys2 powered by an Xilinx Kintex-7. The hardware is designed using SystemVerilog, and it is synthesized by Xilinx Vivado 2018.3. The power dissipation is 0.913 W with a clock frequency of 12.5 MHz, and the utilization of the FPGA resources is summarized in Table 2. The implemented hardware is controlled by the PC using Python 3.6. Training data and labels are delivered by a universal asynchronous receiver/transmitter (UART), including noise parameters. After the hardware trains the given input data, it outputs the temporal weights to the PC. The PC receives weights and tests them with the network that the hardware trains by Python.

The training on the proposed hardware is compared with the conventional training with backpropagation on a CPU and GPU. Conventional training executed on a CPU with Python running on an Intel Core i7 7800X @ 4.4 GHz, and the power dissipation of the CPU is 122.0 W. The conventional method also is implemented on a GPU, which is an Nvidia Geforce GTX 1060 with 6 GB memory.

## VII. EVALUATION

### A. NOISE OPTIMIZATION

To achieve high cognition accuracy, an appropriate combination of noise parameters is required. Parameterized noise signals that are used in the proposed hardware have four parameters, namely, $RND\_WEIGHT$, $MVAL$, $T$, and $\alpha$. To optimize the parameters, all four noise parameters are swept, the proposed hardware trains 100 data at each combination, and the cognition accuracy is verified using test data after training. Table 3 is a list of parameters combinations that achieve the top-4 cognition accuracy, and Fig. 17 shows the accuracy of each combination while training 100 data. The proposed hardware achieves 87.97% accuracy and trains 100 data in 66.85 ms for Prop1.

### B. PERFORMANCE COMPARISONS

Before comparing the performance, the hyperparameters of the conventional training algorithm must be optimized. Fig. 18 shows the optimization of the learning rate ranging from 0.0001 to 0.1 when the minibatch size is 100 and the max epoch is 10. Since the accuracy is highest when the learning rate is 0.01, the results of this case are compared with the proposed hardware. The 2-layer BCNN model has a simple structure, and all of the weights and dataset are represented in binary form. Because of this simple structure, the learning rate did not significantly affect the training accuracy and latency. The randomly selected data in a minibatch or the initial values of the weights affected the training result.

The training on the proposed hardware is compared with the conventional training that uses the same 2-layer BCNN. Table 4 summarizes the performance comparison of the proposed training hardware with the conventional method implemented on a CPU and GPU. The conventional training algorithm is the well-known backpropagation, with a learning rate $\eta$ of 0.01 and a minibatch size of 100. This algorithm requires training the entire dataset 10 times (max epoch = 10) to achieve the maximum cognition accuracy of 90.89% and requires 2.68 s. The proposed hardware achieves an approximately 40x faster training time than the conventional training on a CPU while maintaining almost the same cognition accuracy of 87.97%. The conventional training achieves 87.64% accuracy with a max epoch number of 5, which is the same as the maximum accuracy of the proposed hardware, and its training time is 1.30 s. The proposed hardware is 19x faster than the conventional training and even achieves the same accuracy. The implemented training hardware in the FPGA consumes 0.913 W, which is approximately 134x higher energy efficiency than the power dissipation of the CPU.

The implementation on the GPU using the same model with the same hyperparameters as the training on the CPU obtains a training time of 5.88 s. The training time of the proposed hardware is approximately 88x lower than that for the GPU. Generally, the training process on the GPU is much faster than that carried out on the CPU. However, the BCNN model trained in this article consists of only two binary convolution layers; thus, it is not a complicated structure compared to other NN models such as LeNet-5. Moreover, the dataset

**TABLE 3.** Results of searching for combinations of noise parameters that achieve the top-4 cognition accuracy. The accuracy and the latency of the proposed hardware depends on the noise parameters. The maximum accuracy is achieved for Prop1.

| Index | $RND\_WEIGHT$ | $MVAL$ | $T$ | $\alpha$ | Accuracy [%] | $N_{cycle}$ | Latency [ms] |
|---|---|---|---|---|---|---|---|
| Prop1 | 17 | 18 | 53 | 2 | 87.97 | 7,883 | 66.85 |
| Prop2 | 20 | 21 | 53 | 2 | 87.77 | 5,818 | 48.43 |
| Prop3 | 16 | 15 | 50 | 2 | 87.42 | 8,547 | 69.23 |
| Prop4 | 18 | 30 | 36 | 2 | 87.33 | 3,004 | 17.30 |

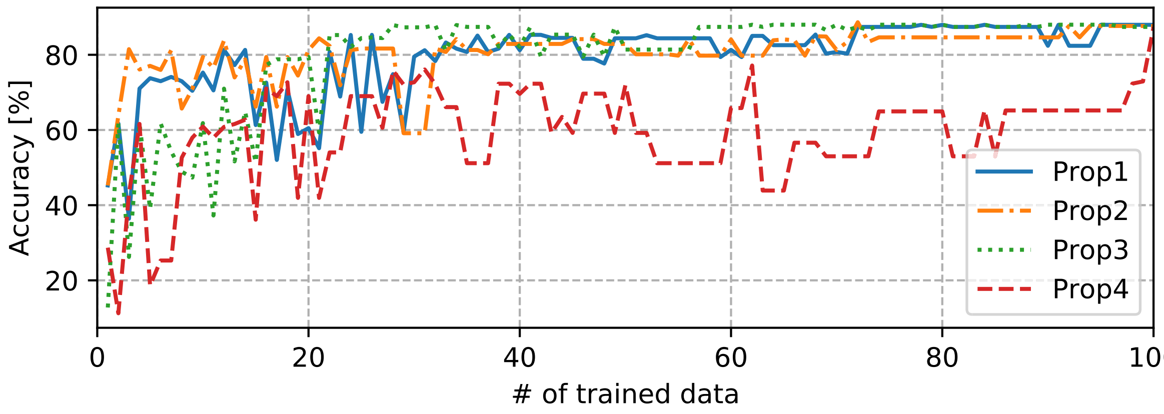

**FIGURE 17.** Cognition accuracy comparison between different combinations of the noise parameters. The noise parameters have a considerable effect on the training process.

**TABLE 4.** Performance comparisons between the proposed and conventional training. The proposed hardware trains 100 data in 66.85 ms and achieves a maximum cognition accuracy of 87.97%, which is 3% less than the maximum accuracy of the conventional training.

| | Proposed | Conventional on a CPU | | Conventional on a GPU |
|---|---|---|---|---|
| Core | Xilinx Kintex-7 | Intel Core I7 7800X | | Nvidia GTX 1060 |
| Clock frequency | 12.5 MHz | 4,400 MHz | | 1,544 MHz |
| Accuracy | 87.97% | 87.64% | 90.89% | 90.89% |
| Training time | 66.85 ms | 1.30 s | 2.68 s | 5.88 s |
| Power dissipation | 0.913 W | 122 W | | 29.2 W |
| Energy dissipation | 0.061 J | 158.60 J | 326.96 J | 171.70 J |
| $N_{cycle}$ | 7883 | - | | - |
| # of trained data | 100 | 10,800 | | 10,800 |
| Precision of training | Binary | Floating-point | | |
| Precision of inference | Binary | Binary | | |
| Algorithm | Invertible logic | Backpropagation | | |

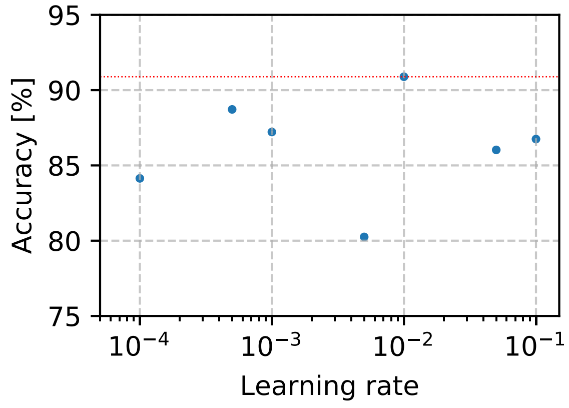

**FIGURE 18.** Results of learning rate optimization. The trained BCNN model has only two convolution layers, and its parameters are binary precision. Therefore, the learning rate did not significantly affect the training accuracy.

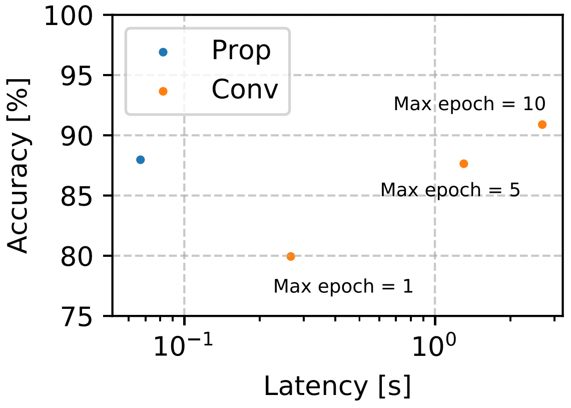

**FIGURE 19.** A plot of accuracy versus latency comparing the proposed training with the conventional training with different max epoch numbers. The proposed hardware achieves the maximum accuracy of 87.97% in 66.85 ms, which is faster than the latency when the conventional training reaches the maximum accuracy.

used for training contains 9 × 9-pixel binary data, so training on the CPU and GPU may not show much difference from each other. A minibatch that includes 100 data is sent to the GPU at each training step while training on the GPU. Therefore, this data transfer becomes a bottleneck in GPU training, causing high training latency. In terms of power dissipation, the GPU consumes 29.2 W during training, which is approximately 32x more than that consumed by the proposed hardware.

Fig. 19 shows accuracy versus latency by conventional training under different max epoch numbers and the proposed training. The learning rate of the conventional training is 0.01, the minibatch size is 100, and the max epoch number of each case is 1, 5, and 10. The conventional training reaches the maximum accuracy of 90.89% after training the entire dataset 10 times for 2.68 s. In contrast, the proposed hardware achieves the maximum accuracy of 87.64% even though it trains only 100 data in 66.85 ms.

Table 5 summarizes the comparison of the speedup over a CPU implementation in the proposed hardware and other hardware-based training implementation on an FPGA. The F-CNN [30] presents an FPGA implementation using backpropagation as the training algorithm. It trains the LeNet-5

**TABLE 5. Comparison between hardware-based training methods using FPGAs. Although it is difficult to compare the proposed training hardware with F-CNN because of the difference between the trained dataset and NN model, our work improves the training latency by a factor of 40x, which is much greater than the improvement obtained by F-CNN.**

| Work | This work | F-CNN [30] |
|---|---|---|
| Network | 2-layer BCNN | LeNet-5 [31] |
| Algorithm | CMOS invertible logic | Backpropagation |
| Frequency | 12.5 MHz | 150 MHz |
| Speedup over CPU | 40x | 4.3x |
| Baseline | i7 @ 4.4 GHz | Xeon 2.5 GHz |
| Precision | Binary | 32-bit floating point |

[31] with the MNIST dataset [20] and achieves 4.3x lower training latency than the CPU implementation. While the result of [30] is difficult to compare precisely with our training hardware because it trains a different dataset, the performance of the proposed hardware is compared with that in [30] because there are few works reported in the literature for training hardware using FPGAs. Although our implementation trains a much smaller network model, it reduces the training latency by a factor of 40. The energy efficiency of [30] is 7.5x higher than that of the corresponding CPU implementation, while in contrast, this work achieves 134x higher energy efficiency.

### C. DISCUSSIONS

To realize the training hardware, the cognition accuracy is also as important as the energy efficiency and the training latency. Our training hardware achieves an 87.97% cognition accuracy that is approximately 3% lower than that of the conventional training with backpropagation. The proposed hardware trains NNs using a very simple algorithm that calculates the weights of NNs as the mean of the temporal weights obtained from backward computation. This training algorithm can train a single binarized perceptron properly [19]; however, the proposed hardware trains a 2-layer model that is much deeper than the perceptron. To train NN models that are more complex than the 2-layer model such as LeNet-5, the training algorithm may need to be improved, which will be investigated in future work. Also, the Hamiltonian for the complex model becomes larger because the number of nodes increases. Hence, key points to implement a training hardware for the complex NN model are the configuration of the Hamiltonian and architecture of the hardware which requires less FPGA resources, and they are our another future works.

## VIII. CONCLUSION

In this article, we have demonstrated fast and energy-efficient training hardware for BCNNs using CMOS invertible logic. Due to the bidirectional operation of CMOS invertible logic, the proposed hardware directly obtains the weights without the error function that requires numerous floating-point computations. The convolution function was converted in the area-efficient Hamiltonian through overlapping nodes that are duplicated from the same element of the input feature maps or filters. For performance evaluation, the training hardware for the 2-layer BCNN was implemented in the FPGA and was used to train the modified MNIST dataset. Specifically, our hardware reduces the power dissipation and the training latency by 97.51% and 99.25%, respectively, while maintaining almost the same accuracy of 87%. The 2-layer BCNN trained by the proposed hardware is not sufficient to actually use as an application. We hope that these findings will contribute to the development of new directions in the field of hardware-based training.

## REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[3] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, and A. Acero, "Recent advances in deep learning for speech research at microsoft," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 8604–8608.

[4] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, New York, NY, USA, 2015, pp. 161–170. [Online]. Available: http://doi.acm.org/10.1145/2684746.2689060

[5] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 257–260.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 4107–4115. [Online]. Available: http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Computer Vision*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 525–542.

[8] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[9] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," 2013, *arXiv:1312.5851*. [Online]. Available: http://arxiv.org/abs/1312.5851

[10] S. Shukla *et al.*, "A scalable multi-TeraOPS core for AI training and inference," *IEEE Solid-State Circuits Lett.*, vol. 1, no. 12, pp. 217–220, Dec. 2018.

[11] B. Fleischer *et al.*, "A scalable multi-TeraOPS deep learning processor core for AI trainina and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 35–36.

[12] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 LNPU: A 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 142–144.

[13] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs," in *Proc. IEEE Int. Conf. Big Data Cloud Comput. (BDCloud), Social Comput. Netw. (SocialCom), Sustain. Comput. Commun. (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct. 2016, pp. 477–484.

[14] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, "Stochastic *p*-bits for invertible logic," *Phys. Rev. X*, vol. 7, Jul. 2017, Art. no. 031014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.7.031014

[15] R. Faria, K. Y. Camsari, and S. Datta, "Low-barrier nanomagnets as p-bits for spin logic," *IEEE Magn. Lett.*, vol. 8, pp. 1–5, 2017.

[16] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Dept. Comput. Sci. Pittsburgh, Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-84-119, 1984.

[17] W. A. Borders, A. Z. Pervaiz, S. Fukami, K. Y. Camsari, H. Ohno, and S. Datta, "Integer factorization using stochastic magnetic tunnel junctions," *Nature*, vol. 573, no. 7774, pp. 390–393, Sep. 2019, doi: 10.1038/s41586-019-1557-9.

[18] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu, "Efficient CMOS invertible logic using stochastic computing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 6, pp. 2263–2274, Jun. 2019.

[19] N. Onizawa, D. Shin, and T. Hanyu, "Fast hardware-based learning algorithm for binarized perceptrons using CMOS invertible logic," *J. Appl. Logics*, vol. 6, no. 7, pp. 41–58, Jan. 2020.

[20] Y. Lecun and C. Cortes. *The MNIST Database of Handwritten Digits*. Accessed: Aug. 5, 2019. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[21] B. Karlik and A. Vehbi, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.

[22] J. D. Biamonte, "Nonperturbative k-body to two-body commuting conversion Hamiltonians and embedding problem instances into ising spins," *Phys. Rev. A, Gen. Phys.*, vol. 77, no. 5, May 2008, Art. no. 052331. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.77.052331

[23] J. D. Whitfield, M. Faccin, and J. D. Biamonte, "Ground-state spin logic," *Europhys. Lett.*, vol. 99, no. 5, Sep. 2012, Art. no. 57004, doi: 10.1209%2F0295-5075%2F99%2F57004.

[24] B. R. Gaines, "Stochastic computing," in *Proc. Spring Joint Comput. Conf. (AFIPS)*, New York, NY, USA, Apr. 1967, pp. 149–156. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465505

[25] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.

[26] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[27] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*. [Online]. Available: https://arxiv.org/abs/1410.0759

[28] S. Vigna, "Further scramblings of Marsaglia's xorshift generators," *J. Comput. Appl. Math.*, vol. 315, pp. 175–181, May 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042716305301

[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://science.sciencemag.org/content/220/4598/671

[30] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-CNN: An FPGA-based framework for training convolutional neural networks," in *Proc. IEEE 27th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2016, pp. 107–114.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

**DUCKGYU SHIN** received the B.E. degree in electrical and communication engineering from Tohoku University, Japan, in 2019, where he is currently pursuing the M.E. degree with the Research Institute of Electrical Communication. His main research interests and activities include energy-efficient VLSI design based on CMOS invertible logic and related applications, such as hardware-based deep learning accelerators.

**NAOYA ONIZAWA** (Member, IEEE) received the B.E., M.E., and D.E. degrees in electrical and communication engineering from Tohoku University, Japan, in 2004, 2006, and 2009, respectively.

He was a Postdoctoral Fellow with the University of Waterloo, Canada, in 2011, and McGill University, Canada, from 2011 to 2013. In 2015, he was a Visiting Associate Professor with the University of Southern Brittany, France. He is currently an Assistant Professor with the Research Institute of Electrical Communication and a JST PRESTO Researcher with Tohoku University. His main research interests and activities include energy-efficient VLSI design based on asynchronous circuits and probabilistic computation and related applications, such as brain-like computers. He received the Best Paper Award in 2010 IEEE ISVLSI, the Best Paper Finalist in 2014 IEEE ASYNC, the Kenneth C. Smith Early Career Award for Microelectronics Research in 2016 IEEE ISMVL, and the MEXT Young Scientists' Prize, in 2020.

**WARREN J. GROSS** (Senior Member, IEEE) received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1996, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 1999 and 2003, respectively.

He is currently a Professor, the Louis-Ho Faculty Scholar in Technological Innovation, and the Chair of the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada. His research interests include the design and implementation of signal processing systems and custom computer architectures. He has served as the Chair of the IEEE Signal Processing Society Technical Committee on the Design and Implementation of Signal Processing Systems. He has served as the General Co-Chair of IEEE GlobalSIP 2017 and IEEE SiPS 2017 and the Technical Program Co-Chair of SiPS 2012. He has also served as the Organizer for the Workshop on Polar Coding in Wireless Communications at WCNC 2017, the Symposium on Data Flow Algorithms and Architecture for Signal Processing Systems (GlobalSIP 2014), and the IEEE ICC 2012 Workshop on Emerging Data Storage Technologies. He has served as an Associate Editor for IEEE TRANSACTIONS ON SIGNAL PROCESSING and a Senior Area Editor. He is a licensed Professional Engineer in the Province of Ontario.

**TAKAHIRO HANYU** (Senior Member, IEEE) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1984, 1986, and 1989, respectively.

He is currently a Professor and the Education/Research Councillor of the Research Institute of Electrical Communication, Tohoku University. His general research interests include nonvolatile logic circuits and their applications to ultralow-power and/or highly dependable VLSI processors and post-binary computing and its application to brain-inspired VLSI systems. He received the Sakai Memorial Award from the Information Processing Society of Japan, in 2000, the Judge's Special Award at the 9th LSI Design of the Year from the Semiconductor Industry News of Japan, in 2002, the Special Feature Award at the University LSI Design Contest from ASP-DAC, in 2007, the APEX Paper Award of the Japan Society of Applied Physics, in 2009, the Excellent Paper Award of IEICE, Japan, in 2010, the Ichimura Academic Award, in 2010, the Best Paper Award of IEEE ISVLSI 2010, the Paper Award of SSDM 2012, the Best Paper Finalist of IEEE ASYNC 2014, and the Commendation for Science and Technology by MEXT, Japan, in 2015.