IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 28, 2022, accepted April 8, 2022, date of publication April 18, 2022, date of current version April 27, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3168026

An Implementation Method Using Cut-Off Bits for Restricted Boltzmann Machines Without Random Number Generators

SANSEI HORI[1], (Student Member, IEEE), AND HAKARU TAMUKOH[1,2], (Member, IEEE)
[1]Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan
[2]Research Center for Neuromorphic AI Hardware, Kyushu Institute of Technology, Kitakyushu, Fukuoka 808-0135, Japan

Corresponding author: Sansei Hori (q899018s@mail.kyutech.jp)

This work was supported by the UENO SEIKI Next Generation Frontier Technology Collaboration Laboratory.

ABSTRACT This study proposes an implementation method of a hardware-oriented restricted Boltzmann machine (RBM) without random number generators (RNGs) that employ cut-off bits, which are obtained from fixed-point binary arithmetic operations on digital hardware, such as field-programmable gate arrays (FPGAs), instead of random numbers. Most FPGA circuits employ fixed-point binary arithmetic operations to improve hardware resource efficiency. Therefore, the proposed method applies the unique feature of the operation, which is bit width extension and cut-off bits. Stochastic neural networks, including RBMs, employ sampling processes based on a probability distribution associated with the network, and the processes require many random numbers. However, implementing RNGs in hardware is costly because it requires considerable hardware resources. The proposed method can mitigate this requirement. To validate the proposed method, we implement an RBM with the proposed method on the software, emulate fixed-point binary arithmetic operations, and train the RBM using the MNIST and Fashion MNIST datasets. Furthermore, we apply the chi-square goodness-of-fit test to evaluate the uniformity of the cut-off bits. Additionally, we compare hardware resource requirements and power consumption for the proposed method and some major RNGs, a linear feedback shift register (LFSR), and a xorshift. Experimental results showed that it was possible to use the cut-off bits for training the RBM using the datasets and clarified the properties of the cut-off bits using statistical analyses. Moreover, hardware implementation of the proposed method involved the lowest hardware resource requirements and power consumption among the RNGs compared in this study.

INDEX TERMS Field-programmable gate arrays, neural networks, random number generation, restricted Boltzmann machines.

I. INTRODUCTION

Deep learning (DL) [1], [2] has been one of the attractive topics in the research area of artificial intelligence in recent years, and many studies have proposed architectures and techniques of deep neural networks (DNNs). Moreover, DNNs are applied in many applications [3], for example, image recognition, natural language processing (NLPs), data analyses, autonomous vehicles, and robotics. These applications are applicable everywhere, such as a cloud application on a data center with massive computational resources, mobile devices, and edge devices to implement internet-of-things (IoT) [4]. However, computing systems for DNNs are imperfect. This section discusses some system problems: computational resource requirements, power consumption, and the disadvantages of cloud computing.

DNNs require many computational resources. Commonly, DNNs have many multiply-and-accumulates (MACs) operations even in the training and prediction phase. The operations are typically done by graphic processing units (GPUs) to accelerate training of a DNN or inference by a DNN, because the GPUs have a higher parallelism than central processing units (CPUs). Moreover, compute unified device architecture (CUDA) produced by NVIDIA eases the programming of DNNs using GPUs [5]. Therefore, GPU acceleration has become common practice and has spread to enterprise and personal users.

However, high-end GPUs, which accelerate DNN applications, require a higher power consumption than CPUs [6]. Studies on the power consumption of DNNs for the NLP have

The associate editor coordinating the review of this manuscript and approving it for publication was Felix Albu.

been reported [7]. The problem will be sufficiently large to ignore in applying the DL in the future.

Moreover, cloud computing has a disadvantage in that it is a communication delay concern in DNN applications [8]. Many DNNs require massive computational resources, the networks are trained on cloud servers, and the results are provided to the user as an application such as a language translation. As for using cloud service, a user should communicate with the cloud servers and transmit some data to the servers through the internet. Cloud applications have time delays in their responses. It is a critical problem for some applications that require real-time responses, such as robot control.

Developing AI-specific hardware is a possible solution for solving these problems [9]. The hardware, which has domain-specific architectures, has high parallelism to calculate MACs. Moreover, the hardware has high flexibility in memory placement and data path planning. From the flexibility, the hardware can reduce the processing time and power consumption of DNNs. In addition, the hardware has a probability of implementing the embedded use such as smartphones and robots, which have a limited power supply. When implementing the hardware in the embedded system and realizing the on-site operation of the DNNs, there is no need to transmit data to the cloud server, and it realizes a high-speed response. In recent years, some companies and research groups have proposed various hardware such as TrueNorth (IBM) [10], Loihi (Intel) [11], TPU (Google) [12], and Xavier (NVIDIA) [13].

There are two types of neural networks: deterministic and stochastic. For example, convolutional neural networks (CNNs) [14], autoencoders (AEs) [15], and chaotic Boltzmann machines (CBMs) [16] are deterministic neural networks. In contrast, stochastic neural networks include a variety of architectures such as Boltzmann machines (BMs) [17], restricted Boltzmann machines (RBMs) [18], [19], variational autoencoders (VAEs) [20], generative adversarial networks (GANs) [21], and generative moment matching networks (GMMNs) [22]. These types of networks have a sampling phase from the probability distributions trained by the dataset. In the sampling phase, the network requires many random numbers. Therefore, random number generators (RNGs) are an essential component when implementing stochastic neural networks into the hardware. Furthermore, if RNGs can be implemented into digital hardware, such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) with high parallelism, the sampling process of the neural networks can be performed in parallel. However, because the hardware resources of FPGAs and ASICs are limited, the number of RNGs that can be implemented is limited.

Two RNG implementation strategies are possible in implementing a neural network into the hardware, which requires random numbers in every unit, such as RBMs. The first strategy realizes parallel processing by implementing RNG into all units that consume random numbers. This architecture has the highest performance for generating random numbers because all units can behave in a completely parallel; however, it requires massive hardware resources to implement RNGs because all units have their own RNGs. In contrast, the second strategy shares the RNGs with some units or all units. This strategy can reduce hardware resources for RNGs; however, the high parallelism, which is one of the hardware advantages, is lost because of the sharing of RNGs and sequential distribution of generated random numbers.

We have proposed a hardware-oriented RBM implementation method without RNGs [23] to resolve this problem, which applies cut-off bits generated from fixed-point binary number operations instead of random numbers. The proposed method can reduce the hardware resources for the RNGs and realize high parallelism for generating cut-off bits instead of random numbers. Furthermore, because the circuit employing the proposed method consumes fewer hardware resources, the power consumption required to obtain the output of the circuit can be reduced compared to conventional methods. This study applied the method to an RBM, emulated fixed-point binary number operations on the software, and evaluated the training results and quality of the cut-off bits obtained from the proposed method.

Sections II and III describe hardware-oriented RNGs and the basic theory of RBMs, respectively. Section IV proposes the implementation method of an RBM for FPGAs without RNGs as in our proposed method. Section V and VI show the methodologies of the experiments and the results to evaluate the proposed method, respectively. Section VII focuses on the hardware implementation of conventional RNGs and the proposed method, and compares them. Section VIII discusses the results obtained, and Section IX concludes the paper.

II. HARDWARE RANDOM NUMBER GENERATORS

RNGs are important components of computer systems and are employed in various applications, such as in numerical simulations, cipher systems, and digital signatures. Some DNNs are also part of applications that require RNGs.

There have been a variety of previous studies on hardware implementations and algorithms for RNGs [24]. RNGs are divided into two main classes: pseudorandom number generators (PRNGs) and true random number generators (TRNGs).

PRNGs, such as linear-feedback shift registers (LFSRs) [25], xorshift [26], and chaotic algorithms [27] generate random numbers in a deterministic manner. Despite the numbers resembling true random numbers, these generated numbers are reproducible under the same PRNG initial parameters. However, if these numbers satisfy a certain criterion, they can be applied to applications that require random numbers. PRNGs are used in most cases. However, TRNGs generate true random numbers based on the non-deterministic behavior of physical phenomena such as metastabilities. These numbers cannot be reproduced even when the same generator is used.

In terms of the FPGA implementation of RNGs, there are important evaluation indicators: the quality of the

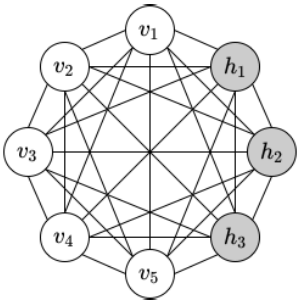


FIGURE 1. Structure of the BM. v_i and h_j are visible and hidden units, respectively. Each unit is allowed to connect with each other.

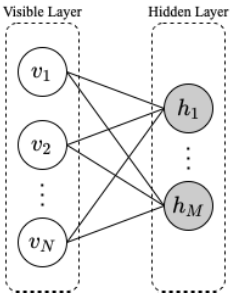


FIGURE 2. Structure of RBM. There are two layers: visible and hidden. Units are not allowed to connect with other units in the same layer.

random numbers, speed of an RNG operation, and hardware resource requirements. When implementing PRNGs in FPGAs in parallel, it is possible to reduce the latency to obtain random numbers. However, as the number of implemented PRNGs increases, hardware resource requirements increase. The requirements of several PRNG implementations are overviewed in reference [24]. However, implementing TRNGs on FPGAs requires specific modules that provide physical phenomena to generate random numbers, which is costly. Therefore, RNGs, which have low latency, fewer hardware resource requirements, and a sufficient quality of random numbers for an implemented application, are desirable components for digital hardware applications.

III. RESTRICTED BOLTZMANN MACHINES

Restricted Boltzmann machines (RBMs) are generative models categorized as stochastical neural networks. RBMs are the basic building components of some DNNs, such as DBNs. In addition, many variations of algorithms related to RBM have an interest in artificial intelligence [28]. This section describes the basic theory of RBMs and training procedures.

A. STRUCTURE AND BASIC THEORY

RBMs are one of the configurations of Boltzmann machines (BMs). The structures of a BM and RBM are shown in Figs. 1 and 2, respectively. The BM is a basic RBM model. The simplest BM is constructed using visible units that connect to each other. In this architecture, each visible unit has a binary state of zero or one, which corresponds to the observational data of the BM. Figure 1 shows that a BM has hidden units. The hidden units do not directly correspond to the observational data of the BM. However, a BM with hidden units has a high flexibility of data representation. As for RBMs, there are two layers: visible and hidden layers, which have N and M units, respectively ($v_1, v_2 \dots v_N$, and $h_1 \dots h_M$). The visible layer groups the visible units, and the hidden layer groups the hidden units. Unit belonging to the same layer do not have connections. This is a restriction on RBMs.

An RBM obtains a probability distribution, which generates trained data, and the network is often used to extract the features of a dataset in DNNs. The RBM is a component of DNNs that can be stacked in a few stages to construct a deep belief network (DBN) [29]. An RBM represents the probability distribution of each unit state, calculated in (1).

$$p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} exp\{-\Phi(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta})\}, \tag{1}$$

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{v},\boldsymbol{h}} exp\{-\Phi(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta})\}, \tag{2}$$

where $\boldsymbol{v}$ and $\boldsymbol{h}$ represent the visible and hidden unit states, respectively; $\boldsymbol{\theta}$ is a network parameter; $Z(\boldsymbol{\theta})$ is a normalization constant; and $\sum_{\boldsymbol{v},\boldsymbol{h}}$ is a partition function that calculates the sum of all combinations of $\boldsymbol{v}$ and $\boldsymbol{h}$. Φ is the energy function of the RBMs, shown in (3).

$$\Phi(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j w_{ij} v_i h_j \tag{3}$$

where v_i and h_j represent the states of the visible and hidden units, respectively; w_{ij} is the weight between the i- and $j-$th units; a_i and b_j are the biases of the visible and hidden units, respectively; and $\boldsymbol{\theta}$ is a set of network parameters.

This network operates stochastically and each unit state is determined using the firing probability calculated from the states of the units in the other layer.

B. TRAINING METHODS OF RBMS

To train the parameters $\boldsymbol{\theta}$ that define the model, RBMs or BMs with hidden units apply the maximum likelihood estimation to the model distribution indicated by $p(\boldsymbol{v}|\theta)$. Because the model distribution of the network includes $\boldsymbol{v}$ and $\boldsymbol{h}$, the probability distribution of $\boldsymbol{v}$ is obtained through marginalization, as shown in the following equation:

$$p(\boldsymbol{v}|\boldsymbol{\theta}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta}), \tag{4}$$

where $\boldsymbol{v}$ is the input data $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_D$, which are N-dimensional vectors. D is the number of data points contained in the dataset. The input data are denoted as $\boldsymbol{v}_n = \{v_1, v_2, \dots, v_N\}$. Then, the maximum likelihood estimation is applied to the likelihood function $L(\theta)$ of the input data, as follows:

$$L(\boldsymbol{\theta}) = \prod_{n=1}^{D} p(\boldsymbol{v}_n|\boldsymbol{\theta}). \tag{5}$$

This function is the target of the maximum likelihood estimation. However, the estimation requires the calculation of all combinations of the $\boldsymbol{v}$ and $\boldsymbol{h}$ states because the likelihood

function includes $Z(\boldsymbol{\theta})$, as shown in (2). As the number of units increases, the combinatorial explosion occurs, and the training method becomes an intractable problem.

To avoid the combinatorial explosion, the RBM training method employs the contrastive divergence (CD) method [30]. The training procedures of an RBM using the CD method are shown below:

1) Set training data to the visible units v_i as $v_i^{(0)}$.
2) Calculate $p_j^{(0)} = p(h_j = 1|\boldsymbol{v}, \boldsymbol{\theta})$ by (9).
3) Update the hidden unit states (if $p(h_j = 1|\boldsymbol{v}, \boldsymbol{\theta}) > r$, then $h_j = 1$, where r is a random number).
4) Calculate $p(v_i = 1|\boldsymbol{h}, \boldsymbol{\theta})$ by (10).
5) Update the visible unit states $v_i^{(1)}$ in the same way as in step 3 using $p(v_i = 1|\boldsymbol{h}, \boldsymbol{\theta})$ instead of the $p(h_j = 1|\boldsymbol{v}, \boldsymbol{\theta})$.
6) Calculate $p_j^{(1)} = p(h_j = 1|\boldsymbol{v}, \boldsymbol{\theta})$ by (9).
7) Update the parameters.

The equations for updating the parameters are shown below:

$$dw_{ij} = \varepsilon(v_i^{(0)} p_j^{(0)} - v_i^{(1)} p_j^{(1)}), \tag{6}$$

$$da_i = \varepsilon(v_i^{(0)} - v_i^{(1)}), \tag{7}$$

$$db_j = \varepsilon(p_j^{(0)} - p_j^{(1)}), \tag{8}$$

where dw_{ij}, da_i, and da_i are the gradients of the weight, visible unit bias, and hidden unit bias, respectively, and ε is the learning rate.

In RBMs, each unit has the firing probability:

$$p(h_j = 1|\boldsymbol{v}, \boldsymbol{\theta}) = \sigma\left(b_j + \sum_i w_{ij} v_i\right), \tag{9}$$

$$p(v_i = 1|\boldsymbol{h}, \boldsymbol{\theta}) = \sigma\left(a_i + \sum_j w_{ij} h_j\right), \tag{10}$$

where σ, a, and b are the sigmoid function, visible unit bias, and hidden unit bias, respectively.

C. VIEW AS AN ENCODER

From another perspective, RBMs work similar to autoencoders (AEs) [15], which are one of the neural networks that construct DNNs. AEs have input, hidden, and output layers and are unsupervised learning algorithms. The input and output layers have the same number of units, and the hidden layer has fewer units than the other layers. This structure is called an hourglass-type neural network. The AEs train the network parameters to make the output closer to the input. After training, the AEs obtain a low-dimensional data representation with essential information on the hidden layer. Therefore, AEs are networks that can encode input data to an internal representation. This feature is often used to pre-train the DNNs, and stacking the AEs results in stacked autoencoders [31].

For the RBMs, the calculations of the conditional probability distribution of the hidden layer $p(\boldsymbol{h}|\boldsymbol{v}, \boldsymbol{\theta})$ from the visible layer can be observed by encoding the input data of the RBM. In contrast, the calculations of the conditional distribution of the visible layer $p(\boldsymbol{v}|\boldsymbol{h}, \boldsymbol{\theta})$ from the hidden layer can be obtained by decoding the data. However, AEs behave deterministically, whereas RBMs behave stochastically. This is the most significant difference between the two network types.

The state of the hidden units sampled from the conditional probability distribution can be viewed as an internal representation of the input data, similar to AEs. The features apply to DNNs, e.g., deep Boltzmann machines (DBMs), which are an architecture of DNNs for classification problems.

IV. HARDWARE-ORIENTED RBM WITHOUT THE RNGS

In this section, we propose a hardware-oriented implementation method for RBMs without RNGs. The proposed method can reduce the hardware resource requirements, generate a value instead of the output of RNGs at low clock cycles, and reduce power consumption. Various studies on the hardware implementation of RBMs [32]–[36] have been reported. The proposed method has the advantage of implementation costs being related to RNGs.

A. FIXED-POINT REPRESENTATIONS

This study employs fixed-point binary numerical operations to evaluate the proposed method. Generally, software applications use floating-point representations defined by IEEE 754 [37], which have a high numerical range. Moreover, most processors in personal computers are optimized for floating-point operations.

In contrast, in the case of implementing an application into digital hardware such as FPGAs, most variables and numerical operations employ fixed-point binary number representations because floating-point arithmetic is complicated for FPGAs, and realizing them requires more hardware resources than fixed-point arithmetic.

Therefore, employing the fixed-point binary number system effectively implements an application into FPGAs, such as a neural network containing several units with high parallelism.

B. FIXED-POINT RBM EMPLOYING CUT-OFF BITS INSTEAD OF RANDOM NUMBERS

We propose a new method for implementing RBMs without RNGs in digital hardware such as an FPGA. Generally, PRNGs and TRNGs implemented in the hardware generate random numbers, as mentioned in Section II, to sample each unit state of an RMB from the firing probability. However, the hardware implementation of an RNG is costly because it requires enormous hardware resources.

Conversely, the proposed method does not require specific modules for RNGs and uses cut-off bits obtained from the fixed-point binary numerical operations during the training phase of an RBM instead of random numbers.

The proposed methods [23] use fixed-point binary numbers, which have an M-bit integer part, including a sign bit and an N-bit fractional part, as parameters of the RBM.

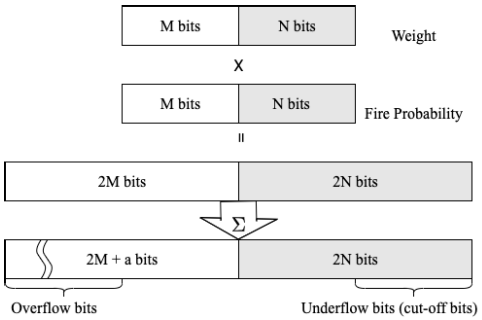


FIGURE 3. Overview of MAC operation and the proposed cut-off bit generation method.

Moreover, the proposed method uses the firing probabilities $p(v_i = 1|\boldsymbol{h}, \boldsymbol{\theta})$ instead of the state of visible units v_i in the training phase. Under this condition, many MAC operations are performed when calculating the firing probabilities of each unit by (9). Consequently, in fixed-point binary number systems, the bit width of the variables increases owing to the numerical operations. The bit width change is shown in Fig. 3 and is described in each step of obtaining the cut-off bits as follows:

1) Multiply w_{ij} and the firing probability $p(v_i)$. The result has a 2M bit integer and 2N bit fractional parts, as shown in Fig. 3.
2) Sum up all values of $w_{ij}p(v_i)$. From the summation, the number of carry bits is equal to $a = \log_2 k$. k denotes the number of terms to be summed. Therefore, the result of the summing operation has a $2M + a$-bit integer part, as shown in Fig. 3.
3) Cut off the resultant value in the integer and fractional parts to hold the initial bit width. In this operation, $(2M + a) - M$ overflow bits in the integer and N underflow bits in the fractional part are generated, as shown in Fig. 3. We employ these underflow bits instead of random numbers generated from RNGs as cut-off bits.

The proposed method employs the cut-off bits obtained during fixed-point numerical operations to eliminate RNGs from hardware, such as an RBM. Therefore, this method can release the hardware resources occupied by RNGs.

V. SOFTWARE IMPLEMENTATIONS AND TRAINING AN RBM WITH THE PROPOSED METHOD

To evaluate the proposed method with an RBM, we implemented the RBM as a C++ application and trained the RBM using the MNIST [38] and Fashion MNIST [39] datasets. This section describes the implementations and training results.

A. IMPLEMENTATIONS OF THE RBM

We implemented two types of RBMs in the software: an RBM with the proposed method and a conventional random number generator. The proposed method was implemented using the Vitis HLS environment, which is a high-level synthesis tool provided by Xilinx Inc. [40], to emulate fixed-point arithmetic. In the conventional method, RNGs are defined using a C++ random header.

TABLE 1. Comparison of the implemented software parameters.

	Proposed	Conventional
Visible units	784	784
Hidden units	150	150
Variable type	Fixed-point binary	Floating point
Precision	18 bits fraction part 14 bits integer part	C++ double (64 bit)
Learning rate	0.12	0.12
Development environment	Vitis HLS	GCC

The parameters of each software are listed in Table 1. The most different point is computational precision. The RBM with the proposed method employs fixed-point binary numbers to calculate the algorithm. It is necessary to emulate the behavior of the proposed method and implement it on an FPGA. In this study, the fixed-point variables have an 18-bit fraction part and a 14-bit integer part. In contrast, the RBM with the conventional method employs double-type variables provided by the C++ programming language.

B. EVALUATION METHODS

We implemented an RBM with the proposed method and conventional RNGs described earlier. The experimental conditions were as follows: the visible and hidden layers had 784 and 150 units, respectively, and the fixed-point numbers had an 18-bit fraction part and 14-bit integer part, as summarized in Table 1. In this definition of fixed-point variables, the fraction part was extended to 36 bits after the multiplication of two variables. After the extension, we obtained 18 cut-off bits when truncating the extended variables to save them to an original precision variable, as shown in Fig. 3. The proposed method normalizes the cut-off bits between zero and one and uses them instead of random numbers to sample the states of hidden units.

We trained the RBMs using the MNIST and Fashion MNIST datasets for 240,000 iterations. One iteration implies a training cycle that inputs an image extracted from the dataset to update the parameters. To evaluate the training result, the software dumped the parameters of weights and biases as files every 1,000 iterations during the training phase. Additionally, for the proposed method, the software dumped the cut-off bits obtained. After the training phase, we loaded the dumped files into the conventional RBM and started the test phase.

In the test phase, we input the training and testing datasets of MNIST and Fashion MNIST into the RBM and calculated the states of the visible and hidden layer units. After the calculation, we obtained the cross-entropy as follows:

$$CE = -\boldsymbol{v} ln(\sigma(\boldsymbol{h}W^T + \boldsymbol{b})) - (1 - \boldsymbol{v}) ln(1 - \sigma(\boldsymbol{h}W^T + \boldsymbol{b})), \tag{11}$$

where $\boldsymbol{v}$ and $\boldsymbol{h}$ represent the states of the visible and hidden units, respectively, W is the weight, $\boldsymbol{b}$ is the hidden unit bias, and σ is the sigmoid function. The cross-entropy errors were

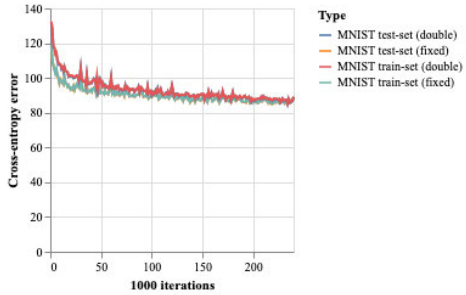


FIGURE 4. Cross-entropy errors for training and testing MNIST dataset.

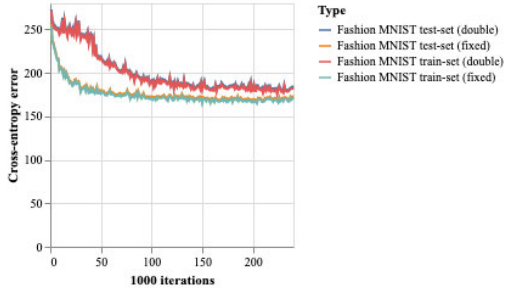


FIGURE 5. Cross-entropy errors for training and testing Fashion MNIST dataset.

TABLE 2. Minimum cross-entropy error.

Dataset	Train Method	Train Dataset	Test Dataset
MNIST	Proposed	85.178	84.871
	Conventional	85.365	85.122
Fashion MNIST	Proposed	165.071	166.694
	Conventional	175.812	177.677

calculated every 1,000 iterations. All cross-entropy errors are the averages of all input data.

C. TRAINING RESULTS

We trained the proposed and conventional RBMs using the training datasets of MNIST and Fashion MNIST datasets. After training, to validate the training result, we input the dataset into the RBMs and calculate the cross-entropy error using (11).

This experiment set the trained parameters from the proposed RBMs to the conventional RBM and calculated the cross-entropy error. Therefore, we used the proposed method for the training phase but not for the cross-entropy error calculation phase. The cross-entropy error was measured on both the training and testing MNIST and Fashion MNIST datasets. The results for these error are shown in Figs. 4 and 5, and the minimum values of the errors for each experiment are listed in Table 2. The errors with the proposed method decreased and almost reached the same levels as the conventional method; therefore, training with the proposed method successfully progressed.

Figures 6-11 show the input and output images of the RBM, which is set with the parameters trained by the proposed method. We extracted 100 images from the testing dataset from the MNIST and Fashion MNIST datasets and input them. These results show that the input images were reconstructed as output images.

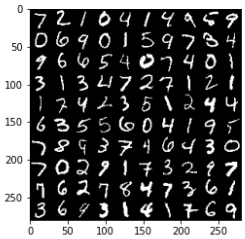


FIGURE 6. Input 100 images extracted from the testing dataset.

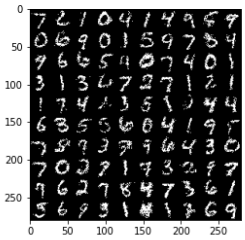


FIGURE 7. Output images from the RBM using the proposed method.

These results show that it is possible to train RBMs using the proposed method.

VI. STATISTICAL ANALYSES OF THE CUT-OFF BITS

The cut-off bits should be uniform to equally sample the state of the units from the firing probability. To evaluate the uniformity of the cut-off bits, we performed a statistical analysis using the chi-square goodness-of-fit test [41]. Some randomness test suites, such as NIST SP800-22 [42], employ a similar statistical test to evaluate uniformity.

In addition, we summarized the cut-off bits using descriptive statistical values. These values indicate the basic properties of the numbers.

This section describes the evaluation method of uniformity using the chi-square goodness-of-fit test and shows the test results and obtain descriptive statistics.

A. METHODOLOGY OF CHI-SQUARE GOODNESS-OF-FIT TEST

To evaluate the uniformity of the cut-off bits obtained using the proposed method, we performed a chi-square goodness-of-fit test. The test is an often-used statistical test for evaluating whether given data originates from a specified distribution. This study evaluated whether the cut-off bits obtained from the proposed method fit a uniform distribution.

The chi-square goodness-of-fit test procedures are as follows. First, divide the domain of the cut-off bits into l intervals, and determine frequency $f_i (i = 1, 2, \ldots, l)$ of the given cut-off bits in the $i - th$ interval (p_{i-1}, p_i). Second, calculate the theoretical frequencies of the cut-off bits F_i using (12).

$$F_i = n \times (F(p_i) - F(p_{i-1})) \quad (i = 1, 2, \ldots, l), \qquad (12)$$

where $F(z)$ is an ideal probability distribution function and n is the number of the given cut-off bits. Third, calculate the

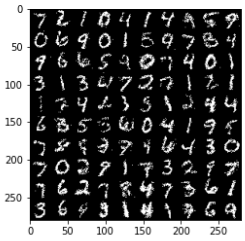


FIGURE 8. Output images from the conventional RBM.

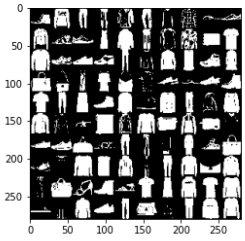


FIGURE 9. Input 100 images extracted from the Fashion MNIST testing dataset.

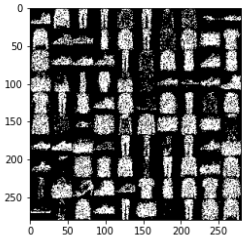


FIGURE 10. Output images from the RBM using the proposed method.

chi-square value using

$$\chi^2_{l-1} = \sum_{i=1}^{l} \frac{(f_i - F_i)^2}{F_i}. \tag{13}$$

Fourth, define the rejection region (χ^2_0, ∞) of the chi-square distribution with $l-1$ degrees of freedom under a 5% level of significance. Finally, if the chi-square value of the given cut-off bits χ^2_{l-1} is less than χ^2_0, then the numbers passed this test. In this study, l, the degree of freedom of the chi-square distribution, was set to 19.

B. THE TEST RESULTS

This validation was performed for each hidden unit because the cut-off bits were generated and consumed in every hidden unit. Figures 12 and 13 show a distribution of chi-square values for each unit during the training with the MNIST and Fashion-MNIST datasets. In these figures, the x-axis represents the hidden unit numbers, the y-axis represents the chi-square value, and the red line represents the chi-square value at a 5% significance level. In this test, if the chi-square value is below the red line, the cut-off bits pass the test and are considered to fit the uniform distribution.

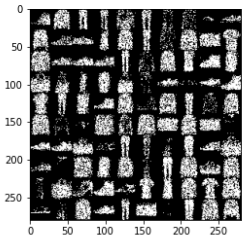


FIGURE 11. Output images from the conventional RBM.

TABLE 3. Descriptive statistics values of the cut-off bits.

	MNIST	Fashion MNIST
mean	0.4999	0.5000
SD	0.2888	0.2887
Min	0.000	0.000
25%	0.2498	0.2500
50%	0.4999	0.5000
75%	0.7500	0.7500
Max	1.000	1.000

From this result, over 90% of the hidden units obtained a uniform distribution of cut-off bits using the proposed method. However, some units did not pass the test, and the training was possible using the proposed method.

Figures 14 and 15 show the transition of the acceptance rate during the training with the MNIST and Fashion MNIST datasets. The x-axis shows the 1,000 iterations, and the y-axis shows the passing rate of the test.

These results show that the acceptance rates remain high during the training phase.

C. DESCRIPTIVE STATISTICS VALUES

We calculated the descriptive statistic values of the cut-off bits generated using the proposed method. The values provided helpful information for obtaining an overview of the basic properties of the generated values.

The descriptive statistic values of the cut-off bits when training the RBM with the MNIST and the Fashion MNIST are summarized in Table 3. In the table, SD is the standard deviation, and 25%, 50%, and 75% are the quartile points. The values describe the statistical properties of all cut-off bits generated during the training phase of the RBM.

Moreover, according to the results, the cut-off bits were uniformly distributed. This result is an essential property for using given numbers instead of random numbers.

VII. CONSIDERATIONS OF HARDWARE IMPLEMENTATIONS

We synthesized conventional PRNGs, xorshift and LFSR, and the proposed method to compare the hardware resource requirements and power consumption. Figure 16 shows the synthesized circuits in this experiment. These circuits have the single PRNG or the proposed method described in Verilog HDL.

This section describes the architecture of each implemented logic, shows the synthesized results, which are estimations of the hardware resource requirements and clock

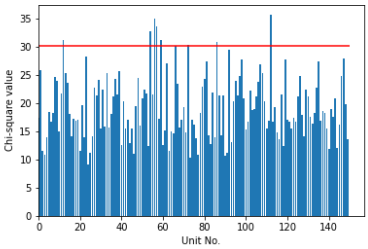


FIGURE 12. Chi-square values with MNIST dataset. Test passed: 141/150 units.

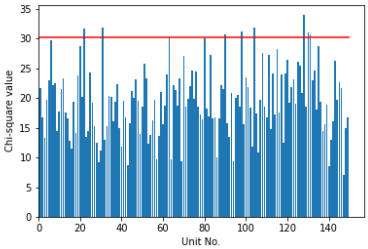


FIGURE 13. Chi-square values with Fashion MNIST dataset. Test passed: 141/150 units.

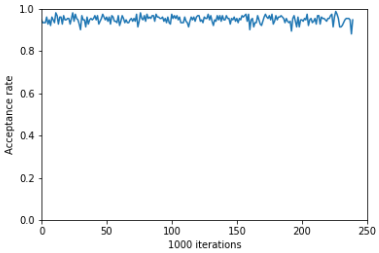


FIGURE 14. Acceptance rate of the obtained random numbers when trained with the MNIST dataset.

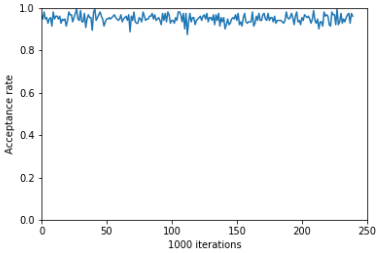


FIGURE 15. Acceptance rate of the obtained random numbers when trained with the Fashion MNIST dataset.

TABLE 4. Estimations of clock cycles, resource utilization and power consumption from synthesis report for each circuits.

Method	xorshift	LFSR	Proposed
Clock cycles [Clock]	1	32	1
LUT resources	33	6	1
FF resources	160	70	18
Dynamic Power [W]	0.036	0.003	0.006
Device Static [W]	0.158	0.158	0.158
Total On-Chip Power [W]	0.195	0.162	0.164

cycles for obtaining the output, and shows the estimation results of the power consumption of the logic.

First, Fig. 16 (a) shows an implementation of the LFSR, which has a 32-bit shift register, and provides feedback on the output of the register to generate pseudorandom numbers. The LFSR generates the bits to be fed back into the shift register by performing XOR operations on the extracted bits. The bit extraction locations on the shift register, namely taps, are defined for the LFSR to realize the longest pseudorandom output period [43]. In this study, the tap positions were the first, second, 22nd, and 32nd bits of the shift register. Furthermore, this logic had a 5-bit counter for generating a valid output signal. The signal indicates that the shift register is completed using new bits.

Second, Fig. 16 (b) shows the implementation of the xorshift PRNG. The PRNG comprises internal states, shift operations, and XOR operations. This logic has four 32-bit registers that maintain the internal state of X, Y, Z, and W, and a 32-bit register that latches an output value. Moreover, "$\ll x$" and "$\gg x$" operators in the figure indicate an x-bit left-shift and x-bit right-shift operation, respectively.

Third, Fig. 16 (c) shows the implementation of the proposed method. In this case, the hardware logic requires only the cut-off operation and latches the result into a 32-bit register.

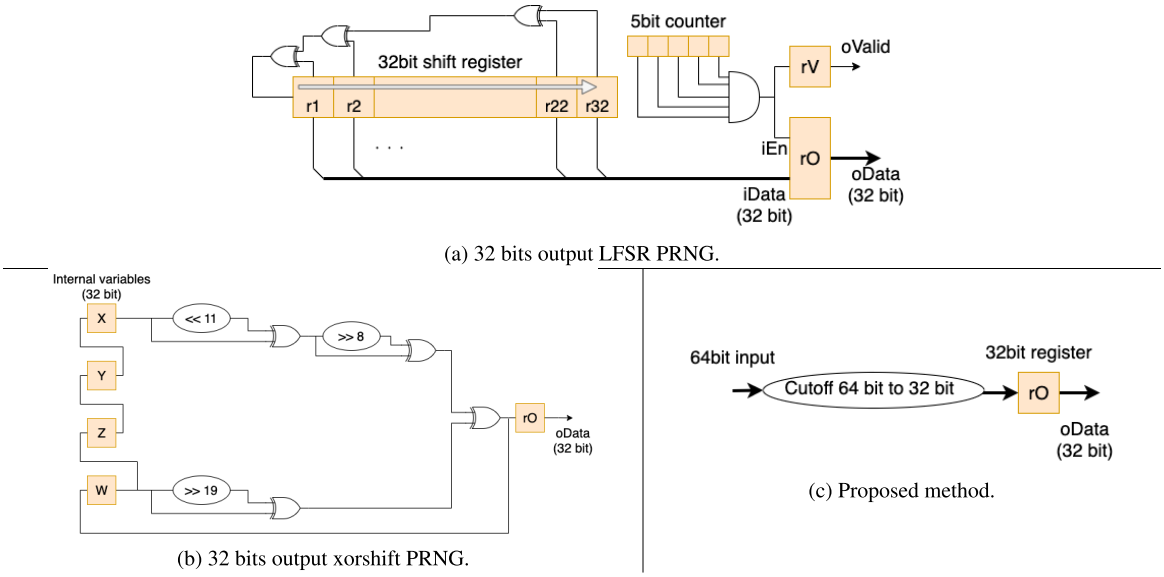
Table 4 lists the synthesized result of each logic and clock cycle estimation to obtain the output. The synthesis environment is the Xilinx Vivado tool, and the target device is the Xilinx Kintex-7 evaluation board (KC705) [44]. In this result, the look-up-table (LUT) realizes logical operations, and the flip-flop (FF) latches the data. Moreover, the dynamic power in the table is the power consumption for the calculation on the implemented logic within the FPGA, and the static power in the table is what the implemented logic consumes to maintain the essential FPGA operation. From these results, the proposed method requires minimum hardware resources on the circuits. The power consumption was estimated using Xilinx Vivado tools under the 100 [MHz] clock settings. The tools consider many conditions to estimate power consumption, such as current leakage inside the device, clock frequency, power supply level, implemented circuit design, and device family [45]. From these results, the total On-Chip Power of the LFSR was the lowest, but the PRNG required 32 clock cycles to obtain an output. It is difficult to conclude that the LFSR requires the lowest hardware resources, and the estimation of power consumption is discussed in the next section.

VIII. DISCUSSIONS

This section discusses the training results of the RBMs, statistical analyses of the cut-off bits obtained using the proposed method, advantages of the proposed method from the perspective of the hardware implementation, and applicability of the proposed method to other neural networks.

A. TRAINING RESULTS

The RBMs using the proposed method can be trained using the MNIST and Fashion MNIST datasets, as evident from



(a) 32 bits output LFSR PRNG.

(b) 32 bits output xorshift PRNG.

(c) Proposed method.

FIGURE 16. Implementations of LFSR, xorshift, and proposed method programmed by Verilog HDL. Yellow rectangles indicate FFs.

Figs. 4 and 5. The cross-entropy errors in each experimental condition with the proposed method are closer to those of the conventional method applying double-precision and software RNGs. In the training result of Fashion MNIST, the error behaved differently from that of the conventional method in the early training phase. This occurred because of the complexity of Fashion MNIST, which may affect the fixed-binary numerical operations. The training results of MNIST had smaller errors than those of Fashion MNIST, which can be attributed to its.

B. STATISTICAL ANALYSES

The distribution of the cut-off bits fit a uniform distribution of more than 90% during the training phase, based on the chi-square goodness-of-fit test. Additionally, the passing rate of the test did not decrease during training. Moreover, the descriptive statistical values also show the uniformity of the distribution of the cut-off bits. In contrast, some units did not pass the test during the training iterations. However, the RBMs could be trained because the number of hidden units was sufficient for training the datasets.

However, we cannot conclude that the cut-off bits are pseudorandom numbers in this study, even if the RBM can be trained using the proposed method. The bits should pass stricter statistical randomness tests, such as NIST SP800-22, to show that they are pseudorandom numbers. Note that a developer should consider the quality and property of the cut-off bits to be sufficient for the requirements when applying the proposed method, instead of random numbers, for any application.

C. COMPARISONS OF THE IMPLEMENTATION RESULTS

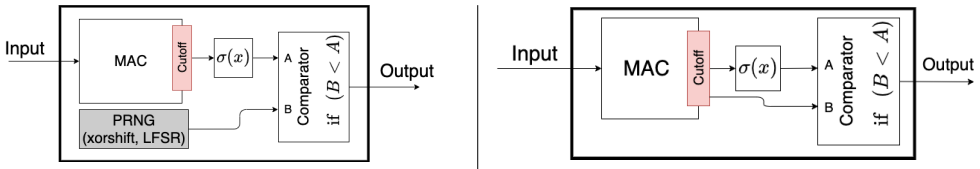
From the results of the hardware implementations for the proposed method, LFSR and xorshift required 6 and 33 times more LUTs, respectively, and 3.9 and 8.9 times more FFs, respectively. These results show that the proposed method consumes less from the hardware resources implemented into the FPGA than the other methods. Furthermore, xorshift and the proposed method require one clock cycle to obtain an output, but LFSR requires 32 clock cycles. This is because the LFSR must fill out the 32-bit shift register using the fed-back bit, which is provided each clock cycle. Therefore, the proposed method requires fewer hardware resources than conventional PRNGs without sacrificing clock cycles to obtain an output.

Figure 17 (a) and (b) show examples of architectures of the conventional and proposed methods, respectively. The MAC and $\sigma(x)$ shown in the figure are basic units used to realize the neural networks. The cut-off logic reverts the bit width of the MAC output to the original width and is an essential unit. The first figure uses a PRNG to generate random numbers and supply them to the comparator, which determines the unit state from the firing probability. Some conventional implementations [32], [34]–[36] also employ random number generators. However, the second figure does not have a PRNG and employs the cut-off bits used as the comparator input. The implemented logic decreases when employing the proposed method.

Table 5 lists the power consumption estimations to obtain an output from the PRNGs and the proposed method circuits. These values were calculated from the power estimation reports provided by the Xilinx tool [45] after the synthesis and clock cycles to obtain an output. This estimation relies on the premise that the clock speed is 100 [MHz], and the target device is the Xilinx Kintex-7 evaluation board (KC705). The power consumptions listed in table 5 is calculated as follows:

$$E = P \times cycles \times 1/f, \tag{14}$$

where E, P, $cycles$, and f denote the power consumption, dynamic power, clock cycles, and clock frequency, respectively.



(a) Conventional. (b) Proposed.

FIGURE 17. Example hardware architectures with the conventional and proposed methods. The common components of this architecture are the MAC logic, sigmoid function, and comparator.

TABLE 5. Power estimations of each implementation of each circuit.

Method	xorshift	LFSR	Proposed
Clock cycles [Clock]	1	32	1
Dynamic Power [W]	0.036	0.003	0.006
Power Consumption [nJ] (Dynamic Power)	0.36	0.96	0.06

Therefore, the proposed method can be implemented in an FPGA with fewer hardware resources and a lower power consumption. Furthermore, the proposed method can obtain an output within a clock cycle. These features are important advantages over the conventional method.

D. APPLICABILITY OF OTHER METHODS

The proposed method may be applicable to other stochastic neural networks and training methods such as DBM and dropout. First, the DBMs are structured by stacking the RBMs as layers. To train a DBM, each layer is trained as an RBM, which is pretraining. There is a possibility that the proposed method is applicable to pretraining. Second, dropout is one of the training methods of the DNNs, such as convolutional neural networks (CNNs), which are often used for image classification. During the training phase, the dropout controls the overfitting by randomly enabling network units. Therefore, this method requires RNGs to generate randomness, and thus proposed method has applicability to be applied. Moreover, CBMs [16] are one of the implementations of a BM without RNGs. However, our proposed method has an advantage in the applicability of other than BM.

IX. CONCLUSION

This study proposes an FPGA implementation method without PRNGs for applications that require random numbers, such as stochastic neural networks, which apply cut-off bits generated from fixed-point binary operations, instead of random numbers. To validate the proposed method, we applied it to the RBM and trained it with the MNIST and Fashion MNIST datasets emulating the fixed-point binary operations on the software. Additionally, we performed a chi-square goodness-of-fit test to evaluate the uniformity of the distribution of the cut-off bits obtained from the proposed method.

Furthermore, we synthesized the single circuits of conventional PRNGs, LFSR, and xorshift, and the proposed method was implemented using Verilog HDL to compare the hardware resource requirements. The results show that the requirements of the proposed method were the least compared to those of the other methods. Moreover, we estimated the power consumption of each circuit to obtain an output, and the proposed method consumed the least power compared to the others. Therefore, this study proved that the proposed method has the capability to implement stochastic applications in an FPGA without PRNGs.

However, this study did not mention that the cut-off bits are random numbers because they should pass a rigorous statistical test to be considered as random numbers. Therefore, when employing the proposed method, the user should consider the quality requirements of the random number for its desired application.

Future work should consider conducting further randomness tests and statistical analyses for the cut-off bits, applying the proposed method elsewhere, and implementing the proposed method in FPGAs for practical uses.

ACKNOWLEDGMENT

The authors would like to thank the Prof. Takashi Morie for having discussions and advice regarding this work.

REFERENCES

[1] 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.
[2] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 8595–8598.
[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, Feb. 2015.
[4] X. Ma, T. Yao, M. Hu, Y. Dong, W. Liu, F. Wang, and J. Liu, "A survey on deep learning empowered IoT applications," *IEEE Access*, vol. 7, pp. 181721–181732, 2019.
[5] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for?" *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008, doi: 10.1145/1365490.1365500.
[6] 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.
[7] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," 2019, *arXiv:1906.02243*.
[8] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.
[9] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225134–225180, 2020.
[10] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[11] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
[12] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, New York, NY, USA, 2017, pp. 1–12, doi: 10.1145/3079856.3080246.
[13] NVIDIA. *JETSON AGX Xavier Series*, Accessed: Feb. 16, 2022. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetso%n-agx-xavier/
[14] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1237–1242.
[15] G. W. Cottrell and P. Munro, "Principal components analysis of images via back propagation," *Proc. SPIE*, vol. 1001, pp. 1070–1077, Oct. 1988.
[16] I. Kawashima, T. Morie, and H. Tamukoh, "FPGA implementation of hardware-oriented chaotic Boltzmann machines," *IEEE Access*, vol. 8, pp. 204360–204377, 2020.
[17] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognit. Sci.*, vol. 9, no. 1, pp. 147–169, 1985.
[18] A. Fischer and C. Igel, "An introduction to restricted Boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Berlin, Germany: Springer, 2012, pp. 14–36.
[19] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," Dept. Comput. Sci., Univ. Tronto, Tronto, ON, Canada, Tech. Rep. UTML TR 2010-003, 2010.
[20] D. P Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.
[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, vol. 27. Red Hook, NY, USA: Curran Associates, 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97%b1afccf3-Paper.pdf
[22] Y. Li, K. Swersky, and R. Zemel, "Generative moment matching networks," in *Proc. 32nd Int. Conf. Mach. Learn.*, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, Jul. 2015, pp. 1718–1727. [Online]. Available: https://proceedings.mlr.press/v37/li15.html
[23] S. Hori, T. Morie, and H. Tamukoh, "Restricted Boltzmann machines without random number generators for efficient digital hardware implementation," in *Artificial Neural Networks and Machine Learning—ICANN 2016*. Cham, Switzerland: Springer, 2016, pp. 391–398.
[24] M. Bakiri, C. Guyeux, J.-F. Couchot, and A. K. Oudjida, "Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses," *Comput. Sci. Rev.*, vol. 27, pp. 135–153, Feb. 2018.
[25] R. C. Tausworthe, "Random numbers generated by linear recurrence modulo two," *Math. Comput.*, vol. 19, no. 90, pp. 201–209, 1965.
[26] G. Marsaglia, "Xorshift RNGs," *J. Stat. Softw.*, vol. 8, no. 14, pp. 1–6, 2003.
[27] P. Dabal and R. Pelka, "A chaos-based pseudo-random bit generator implemented in FPGA device," in *Proc. 14th IEEE Int. Symp. Design Diag. Electron. Circuits Syst.*, Apr. 2011, pp. 151–154.
[28] N. Zhang, S. Ding, J. Zhang, and Y. Xue, "An overview on restricted Boltzmann machines," *Neurocomputing*, vol. 275, pp. 1186–1199, Jan. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217315849
[29] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2014.
[30] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.
[31] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[32] S. K. Kim, L. C. McAfee, P. L. Mcmahon, and K. Olukotun, "A highly scalable restricted Boltzmann machine FPGA implementation," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 367–372.
[33] S. K. Kim, P. L. Mcmahon, and K. Olukotun, "A large-scale architecture for restricted Boltzmann machines," in *Proc. 18th IEEE Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, 2010, pp. 201–208.
[34] D. L. Ly and P. Chow, "High-performance reconfigurable hardware architecture for restricted Boltzmann machines," *IEEE Trans. Neural Netw.*, vol. 21, no. 11, pp. 1780–1792, Nov. 2010.
[35] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. J. Yoo, "A 1.93 TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *IEEE ISSCC Dig. Tech. Papers*, Dec. 2015, pp. 80–82.
[36] K. Ueyoshi, T. Asai, and M. Motomura, "Scalable and highly parallel architecture for restricted Boltzmann machines," in *Proc. RISP Int. Workshop Nonlinear Circuits, Commun. Signal Process.*, 2015, pp. 369–372.
[37] *IEEE Standard for Floating-Point Arithmetic*, Standard IEEE Std 754-2008, 2008, pp. 1–70.
[38] 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.
[39] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[40] Xilinx. *Xilinx Software Tools*. Accessed: Jan. 26, 2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis.html
[41] K. Pearson, "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *Philos. Mag.*, vol. 50, no. 320, pp. 157–175, 1900, doi: 10.1080/14786440009463897.
[42] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, S. Leigh, M. Levenson, M. Vangel, N. Heckert, and D. Banks. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762
[43] Xilinx. *Efficient Shift Registers, LFSR Counters, Long Pseudo-Random Sequence Generators (XAPP052)*. Accessed: Jan. 17, 2022. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp052.% pdf
[44] Xilinx. *Xilinx Kintex-7 FPGA KC705 Evaluation Kit*. Accessed: Jan. 26, 2022. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html
[45] Xilinx. *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*. Accessed: Feb. 25, 2022. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug%907-vivado-power-analysis-optimization.pdf

SANSEI HORI (Student Member, IEEE) received the bachelor's degree from the Tokyo University of Agriculture and Technology, Japan, in 2014, and the master's and Ph.D. degrees from the Kyushu Institute of Technology, Japan, in 2016 and 2022, respectively. He is currently a Postdoctoral Research Fellow with the UENO SEIKI Next Generation Frontier Technology Collaboration Laboratory, Kyushu Institute of Technology. He is a member of IEICE.

HAKARU TAMUKOH (Member, IEEE) received the B.Eng. degree from Miyazaki University, Japan, in 2001, and the M.Eng. and Ph.D. degrees from the Kyushu Institute of Technology, Japan, in 2003 and 2006, respectively. He was a Postdoctoral Research Fellow with the Kyushu Institute of Technology, from 2006 to 2007. He was an Assistant Professor with the Tokyo University of Agriculture and Technology, from 2007 to 2013. He is currently a Professor with the Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology. His research interests include digital hardware design, soft-computing, and home-service robots. He was the author of works that won the Best Paper Award at IEEE/INNS IJCNN 2019, the Best Live Demonstration Award at IEEE ISCAS 2019, and the Best Paper Award at ICONIP 2013. He is a member of IEICE and JNNS.