PAPER Special Section on Parallel and Distributed Computing and Networking

Dither NN: Hardware/Algorithm Co-Design for Accurate Quantized Neural Networks

Kota ANDO[†a)], **Kodai UEYOSHI**[††], ***Student Members***, **Yuka OBA**[††], **Kazutoshi HIROSE**[††], **Ryota UEMATSU**[††], **Takumi KUDO**[††], ***Nonmembers***, **Masayuki IKEBE**[††], **Tetsuya ASAI**[††], **Shinya TAKAMAEDA-YAMAZAKI**[†††,††††], ***and*** **Masato MOTOMURA**[†], ***Members***

SUMMARY Deep neural network (NN) has been widely accepted for enabling various AI applications, however, the limitation of computational and memory resources is a major problem on mobile devices. Quantized NN with a reduced bit precision is an effective solution, which relaxes the resource requirements, but the accuracy degradation due to its numerical approximation is another problem. We propose a novel quantized NN model employing the "dithering" technique to improve the accuracy with the minimal additional hardware requirement at the view point of the hardware-algorithm co-designing. Dithering distributes the quantization error occurring at each pixel (neuron) spatially so that the total information loss of the plane would be minimized. The experiment we conducted using the software-based accuracy evaluation and FPGA-based hardware resource estimation proved the effectiveness and efficiency of the concept of an NN model with dithering.

key words: *neural network, dithering, error diffusion, FPGA, hardware-oriented neural network algorithm*

1. Introduction

Deep neural network has enabled various artificial intelligence (AI) applications in many fields, such as virtual assistants, self-piloting robots and cars, smart home devices, and game players. They are supported by the enormous evolution of neural network structures (models) with improvements of the recognition accuracy. However, the explosion of the computational and memory resource requirements of these highly-developed neural network models has become the problem, especially looking at the era of the Internet of Things (IoT) approaching rapidly. To reduce this resource problem, many *approximate* neural network algorithms and accelerator architectures based on them, which utilize a reduced-precision arithmetic and data storage, have been researched and presented [1]–[5].

In this paper, we attempt to improve the recognition accuracy of the low-precision quantized neural network models under the limited hardware resource. We import the *dithering* technique from the field of image processing into neural network; dithering is a commonly-used technique in image/signal processing to quantize an image/signal to low precision while keeping the gradation of the source signal by diffusing the quantization error occurring at each pixel (neuron) to its neighboring pixels, by which the total (or average) quantization error of entire the plane is minimized.

We extend the quantized neural network by introducing the *error diffusion* method, the basic and lightest algorithm for dithering. This dithering extension is quite hardware-friendly because it requires only two additional operations of addition and comparison, which are used as the primitive operations in a neural network, therefore no additional arithmetic units are needed. This allows the dithering algorithm to be easily integrated into an existing approximate neural network accelerator architecture with a little modification. To verify the merits of the dithering algorithm, we conduct a hardware resource evaluation using experimental architectures we implemented, as well as the software-based accuracy evaluation.

This work is based on our previous conference paper [6]. The main contribution of this paper is the following: we explained the proposed algorithm and architectures in more detail, and we conducted the evaluation using well-known network structures that are easier to reproduce. We also expanded the concept of the dithering into the neural networks with fixed-point and logarithmic quantization, not only for binary, by presenting typical procedures and architectures with those quantization methods; they are described as Appendix since the reproducible evaluation for them has not been completed.

The rest of the paper is composed as follows. In Sect. 2, we take a brief look at the prior work on hardware-oriented low-precision neural networks, and on their accelerator architectures. In Sect. 3, the base algorithm of the dithering in the field of digital signal processing, which motivates our work attempting to improve the accuracy of the low-precision neural networks, is explained. Then in Sect. 4, the algorithm is modified and extended so that it can be used with neural network hardware. In Sect. 5, we discuss applying the back propagation training method to the proposed dithering neural network, and another dithering technique is discussed as well. In Sect. 6, we conduct the accuracy evaluation using software simulation, and we evaluate the hardware efficiency of the proposed method by FPGA prototype architectures. We conclude the paper in Sect. 7. Conceptual proposal of using the dithering in generic low-precision

Manuscript received January 8, 2019.
Manuscript revised June 2, 2019.
Manuscript publicized July 22, 2019.

[†]The authors are with the Tokyo Institute of Technology, Yokohama-shi, 226–8502 Japan.
[††]The authors are with the Hokkaido University, Sapporo-shi, 060–0814 Japan.
[†††]The author is with The University of Tokyo, Tokyo, 113–8656 Japan.
[††††]The author is with JST PRESTO, Kawaguchi-shi, 332–0012 Japan.
a) E-mail: ando.kota@artic.iir.titech.ac.jp

DOI: 10.1587/transinf.2019PAP0009

hardware is presented in Appendix.

2. Related Work

Multiplication is the costliest computation of the neural network, and the memory occupation is caused mostly for the weights. Multiplication requires a lot of hardware resource; the gate count is in proportion to the square of the bit width, in contrast to addition whose gate count is in linear proportion. Data amount, on the other hand, gets larger as the network model becomes more complex, owing to the demands on the practical applications. To reduce these computation and memory resource requirements, mainly for mobile applications that cannot hire large memory and energy-hungry high-end processors, many hardware-aware approximate neural network algorithms have been proposed.

The redundancy of a neural network model has been researched since the dawn of the deep learning applications. It is known that a neural network does not require much higher numerical precision, especially in the inference phase. Studies have been conducted that use fixed-point expression rather than floating point, aiming at simplifying the computation (both multiplication and addition) [7], [8]. An approach called *dynamic fixed-point* that allows the bit width in a single network to vary has been proposed [9]–[11]. In the most aggressive case, the bit precision alternates during the inference on an accelerator according to the occurrence of the latest arithmetic overflow/underflow.

BinaryConnect [12] appeared in the second half of 2015 as the first successful neural network model that uses binary weights. It *binarizes* the weights into ±1 while keeping the activations linear, which allows the maltiply-accumulation (MAC) operation between an activation and a weight to be calculated as an addition. During the training phase, the full-precision weights must be kept since the update of the weights takes place in full-precision as a part of back propagation. When the training phase is complete, the weights are statically binarized, and in the inference phase only the binarized weights are retained.

The first feasible network model that binarizes both the weights and the activations was *BinaryNet* [1] in 2016, which is also called binarized neural network or binary neural network (BNN). Its binarized weights and activations can be *multiplied* by a bit-wise XNOR (exclusive NOR) operation; therefore, the heavy multiplication step can be eliminated. The success of this model is attributed to "batch normalization" [13]. Batch normalization regularizes the distribution of the activations statistically before binarizing them, which tolerates the statistical changes in the input activations among batches in the training phase; thus preventing the network from overfitting/diverging and accelerating the training.

These binary network models often result in lower recognition accuracy due to their extreme approximation. There are several approaches that use *quantized* numerical expression other than binarization. Ternary weight network [14], or ternary neural network, quantizes the weights into ternary expression with the values of −1, 0, and +1, which could achieve a more accurate weight representation than the binary one in a certain situation. *LogNet* [2] quantizes the weights and activations in the logarithmic representation, which replaces the costly multiplication with a simple addition while retaining the resolution of the most frequently occurring numbers valued at around 0.

XNOR-Net [15] appeared just after BinaryNet. It conducts the binary weight-activation XNOR multiplication with a few real-valued scaling factors. Residual binary neural network (*ReBNet*) [16] is an extension of the binary neural network that binarizes both the activation and weights, intended to obtain the appropriate approximation by a linear combination of the binary activations. It binarizes the activations gradually into a sequence of binary numbers using multiple thresholds and "residual errors". The resulting multiple binary activations are multiplied (XNORed) with a single binary weight, which are then linearly combined using the scaling factors.

3. Dithering

In this section, we explain the base algorithm of "dithering" to accommodate to neural networks.

3.1 Dither in Signal Processing

Dithering is a commonly-used technique in digital signal processing, used to reduce the effect of quantization errors. When a source signal is "quantized" into another signal digitally, quantization errors (the differences between the source and quantized signal) always occur. For example, when a photograph in 24-bit color is converted to 8-bit color, the details and gradients will be lost and parts that had similar colors may become indistinguishable.

Dithering is often used with binary quantization where the quantized signal only takes two states (in 1-bit digital signal, they are '0' and '1'). In binary quantization, a threshold is set, and an input value is quantized into '1' when it is greater than the threshold; otherwise it becomes '0'. This means that if many of the input samples have values slightly lower than threshold, they always become '0', and as a result, the total quantization error becomes quite large. A solution is to stochastically quantize nearly half of those middle-valued samples into '0' and the other half into '1'. This would equalize the quantization errors among the input samples so that the total quantization error becomes minimal because half of the output values will have negative errors (deficit of the output), while the others will have positive errors (surplus of the output). This idea of stochastically quantizing the inputs is called "dithering."

3.2 Error Diffusion

One of the most widely used approaches of dithering is the "error diffusion" method. In the following explanation, consider that the N-length source signal S takes a real value in

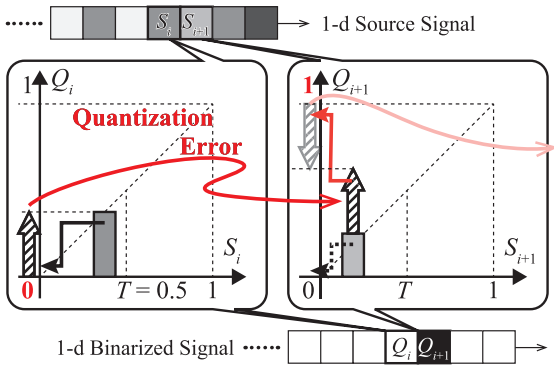

Fig. 1 Error diffusion on 1-d signal quantization.

[0, 1] at the i-th sample S_i for each i ($i = 1, \cdots, N$), the resulting quantized signal Q takes a binary value (0 or 1) at the i-th sample Q_i, and the threshold is T ($0 < T < 1$). Note that the error diffusion algorithm does not limit the output signal to be binary; it can be utilized with any kind of quantization; however, we use binary quantization as an example here. As shown in Fig. 1 and Algorithm 1, the basic idea of this method is to "integrate" (or "accumulate") the quantization error occurring at each sample. The current quantization error E (surplus/deficit of the quantized value when it is 1/0) is added to the next source value S_{i+1} before thresholding. This works to cancel the total (or average) quantization error among overall samples as described above. This algorithm is quite similar to the "delta-sigma modulation" that is used in ADCs (analog-to-digital converters).

Let us give an example. If the first input is $S_1 = 0.3$, the second is $S_2 = 0.4$, and the threshold is $T = 0.5$, the quantized values without dithering would be $Q_1 = Q_2 = 0$ because both inputs do not exceed the threshold. In this case, the total quantization error is $\sum(S_i - Q_i) = 0.7$. When we apply the error diffusion method to these inputs, the first quantized output is $Q_1 = 0$ since the initial value of the integrated quantized error E is 0; therefore, the error becomes $E = 0.3 - 0 = 0.3$. Then, this error $E = 0.3$ is added to the next input ($S_2 + E = 0.4 + 0.3 = 0.7$), and the resulting output is $Q_2 = 1$ because it is greater than the threshold T. With the integrated quantization error, even a smaller input could produce an output 1, and vice versa. The total quantization error is now $(0.3 - 0) + (0.4 - 1) = -0.3$, whose absolute value is smaller than 0.7 of the case without dithering.

3.3 Dithering on 2-d Signals

We have explained the error diffusion on a 1-d sequence, and this can be easily extended to 2-d signals (such as images) by propagating the quantization errors in the plane. The simplest way to achieve this is through a "1-directional" method that retains an integrated quantization error *for each row* as shown in Fig. 2 (a), *i.e.* there is no such relationship between the rows. There are several algorithms used in image processing that *diffuse* the quantization errors of neighboring pixels (Fig. 2 (b): an example of "2-directional" error diffusion). These multidirectional methods result in better image quality when used for image conversion, but the computational complexity would increase. We discuss these 2-d dithering algorithms later from the viewpoint of hardware implementation.

Algorithm 1 binary-quantization of 1-d signal using error diffusion

Input: $S = \{S_1, \cdots, S_N\}$:
N-length sequence of a source signal (analog or digital)
Input: T: Threshold
Output: $Q = \{Q_1, \cdots, Q_N\}$:
N-length sequence of the binary-quantized signal
1: $E \Leftarrow 0$
2: **for** i in $\{1, \cdots, N\}$ **do**
3: **if** $(S_i + E) \geq T$ **then**
4: $Q_i \Leftarrow 1$
5: **else**
6: $Q_i \Leftarrow 0$
7: **end if**
8: $E \Leftarrow (S_i + E) - Q_i$
9: **end for**
10: **return** Q

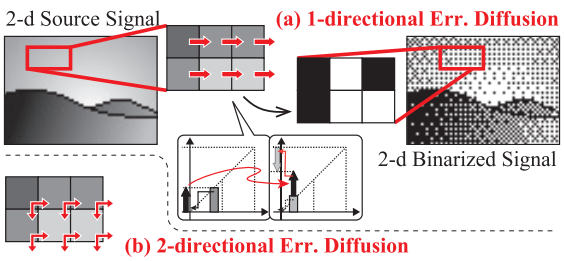

Fig. 2 Error diffusion in 2-d image. (a) "1-directional" method and (b) "2-directional" method.

4. Neural Network with Dither

We have described the base algorithm of dithering, which minimizes the total quantization error for low-precision quantization. In this section, we attempt to apply it to neural network models to improve the accuracies of approximate neural network algorithms for mobile/embedded applications while minimizing additional requirement of hardware resources.

4.1 Prerequisite

Many convolutional neural network (CNN) based processors integrated on ASICs (application-specific ICs) and FPGAs feature output-parallel processing, where the processing engines (PEs) form an array and each PE computes an output neuron. On this parallel processor, each PE usually has a multiplier and an accumulator to calculate the MAC operations between the inputs and weights, and the PE array sequentially scans all the input neurons while accumulating the products generated by the multipliers.

We assume this output-parallel processor as the main target of our work. As described later, this type of parallelism in which the output partial results stay in the accu-

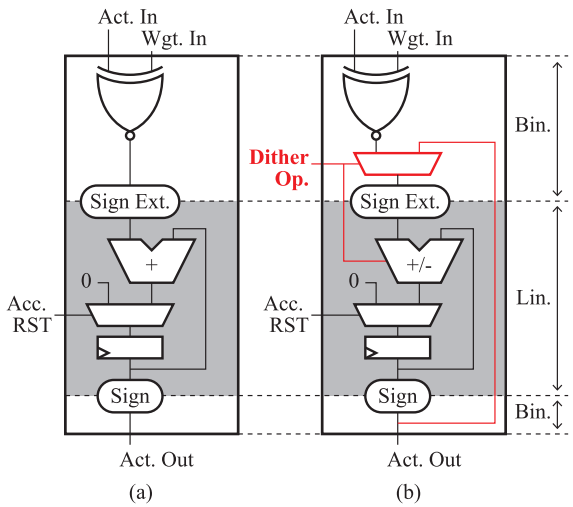

Fig. 3 (a) A general and simplified form of an XNOR-accumulator-type PE. (b) An XNOR-accumulator PE with error diffusion operation mode.

mulators and the inputs travel is suitable for the dithering algorithm.

4.2 Error Diffusion in Output-Parallel Binary Architecture

4.2.1 Baseline Binary Neural Network

We first discuss the general architecture with dithering for a binary neural network. Binary neural networks [1] restrict the activations and weights to either -1 or $+1$, unlike standard binary quantization used in signal processing, which uses values of 0 and 1. This is because the binary-quantized network is a special case of fixed-point (or linear integer) quantization where the bit width is 1, and therefore, only the sign bit remains. As proposed in [1], the multiplication between two binary-quantized values produces only four possible situations and can be executed as an XNOR (exclusive NOR) operation with the values $-1/+1$ being denoted as logic 0/1 respectively. A neuron of a binary neural network layer acquires the binary (± 1) pairs of weights and activations, conducts XNOR operation (*multiplication*) on each pair, sums all the resulting products, and picks the sign of the sum as the output activation (*non-linear function*).

To implement this logic on an output-parallel array, usually an XNOR-accumulator-based PE is used (Fig. 3 (a)). The accumulator on a PE sums the *binary* products calculated by the XNOR gate from all the input activations in a *linear* manner.

4.2.2 Applying Error Diffusion

The summation of each neuron produces an integer value in the range of $-N$ to N for N input activations, and the resulting output activation is ± 1 depending on the sign. The quantization error can be defined in a similar manner to the basic explanation provided in Sect. 3.2, *i.e.* by simply subtracting the resulting activation ± 1 from the sum.

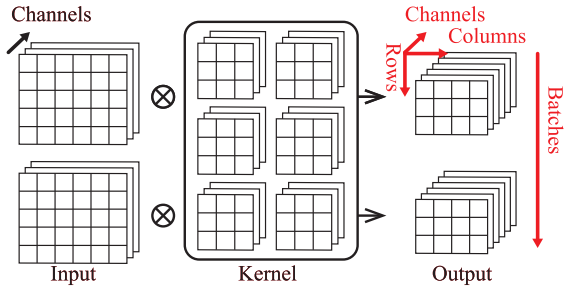

Fig. 4 Four axes of the activation of a convolutional layer.

In the binary neuron processing on an output-parallel PE, the accumulator retains the final sum at the end of input scanning, and the value is reset to 0 immediately after the sign is assigned as the output. On the other hand, the quantization error to be added to the next neuron's weighted-sum in the dithering algorithm is calculated using the sum and sign of the current neuron. Thus, we can compute the accumulation of the quantization error by the following steps: 1) scan the inputs and perform XNOR-accumulation similar to the operation in standard binary neuron processing; 2) calculate the sign bit (the output activation) upon completion of the input scan; 3) subtract this ± 1 value from the sum in the accumulator instead of the reset operation; and 4) scan the next input. Therefore, we do not need additional circuit components in the XNOR-accumulator-based PE; all that is required is a simple selector circuit to select the mode of the accumulator for the initial reset, accumulation, or quantization error calculation, as shown in Fig. 3 (b).

The dependencies among the error diffusion sets in the output activations of the current neuron affect the next neuron; however, this would not be a problem for the output-parallel processing. A typical convolutional layer of a neural network has four axes in the output activation: batches, rows (height), columns (width), and channels (Fig. 4). We introduce the error diffusion technique only within a plane, *i.e.* the axes of rows and columns of each channel in each batch. Therefore, the axes of the batches and channels are unaffected. Most output-parallel architectures map the axes of the channels and rows onto their PE arrays mainly because of the efficiency of data delivery achieved. As long as we use the 1-directional error diffusion, the quantization error is only accumulated along a row; as each row/channel can be computed individually, the output-row-and-channel-parallel processing works well with dithering. In such a configuration, each PE in the array is assigned to an output neuron in a row of a channel and operates sequentially along the columns, as indicated in Fig. 5. In other words, error diffusion algorithms can be computed in a channel–row–column loop with the outer two axes unrolled spatially onto the output-parallel PE array. The output–input–channel parallelism is also suitable; in this case, the PEs in a column correspond to an output neuron, a row corresponds to an input, all the PEs individually calculate the MACs of the inputs and weights, and the partial sums finally obtained in the PEs are gathered and summed in a column-wise manner.

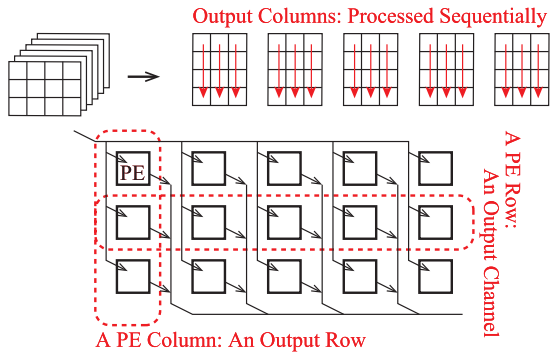

Fig. 5 Row- and channel-parallel PE. The output columns are processed sequentially, which harmonizes with dithering.

4.3 Combination with ReLU Activation Function

4.3.1 Activation Function in Conventional Models

A convolutional or fully-connected layer in a neural network employs a non-linear function called *activation function*. The activation function, which is applied to the weighted-sum of each neuron, is a key technique for neural networks. It represents a primitive non-linear response by itself, and this non-linearity helps the network in obtaining an acceptable approximation of the complex behavior of an unknown implicit function.

Rectified linear unit (*ReLU*), is the most popular type of activation function in standard (linearly expressed) neural network models. This unit allows a positive value and blocks a negative value (set to 0). This lightweight implementation of a non-linearity requires checking only the sign of the input, and it is widely used because it is friendly with the training algorithms that use back propagation.

In a binary neural network, obtaining the sign of the sum is also a kind of activation function called *binarization* or simply *Sign function*. It can be interpreted as a thresholding function that checks if the weighted-sum exceeds 0 or not and can be generalized for any threshold value via a bias term.

4.3.2 Activation Function and Dithering

The output of the binarization using dithering approaches the identity transformation when the input is large enough and the output is "demodulated" by blurring it (equivalent to "integrating" or "applying low-pass filter"). This fact suggests that dithering does not have any non-linear effects. Therefore, we must consider introducing a mechanism for obtaining non-linearity in the error diffusion.

One possible option is to use dithering after a standard ReLU activation function. This is quite straight-forward and expects the ReLU activation function to provide the non-linearity and dithering to remap the result of the ReLU spatially. However, there are some disadvantages in this method: the computational complexity would increase owing to the separate operations for ReLU and dithering, and it may generate +1 in the neurons that the non-linear ReLU previously deactivated.

Another option is to customize the error diffusion algorithm, namely *positive-only error diffusion*, such that it takes the quantization error into consideration *only when the raw sum (S_i) is greater than the threshold*. Similar to the ReLU, the positive-only error diffusion suppresses a negative sum value to −1 unconditionally without using/accumulating the quantization error. When the sum has a positive value, it works like the baseline error diffusion, *i.e.* some of the positive inputs become −1 outputs by considering the quantization error or *total surplus*. This method could not only operate as a single activation function whose computational complexity would be reduced compared to the separated ReLU-dither structure but also properly evaluate the neurons having negative sums, which should be deactivated to obtain the non-linearity.

4.4 Working with Other Low-Precision Networks

The idea of dithering using error diffusion algorithm on a low-precision hardware can be naturally extended to any kind of quantization methods other than binarization, as long as the quantization error can be defined using the input and output of the quantization. We present basic concepts of using the dithering with the logarithmic and fixed-point quantization in Appendix, since they have not been synthesized and evaluated.

5. Discussion

5.1 Back Propagation Methods for Dither NN

When we intend to use a certain operation in the hidden layers of a neural network, its derivative *must* be defined for applying the back propagation. We discuss the appropriate realization of the derivative of the dithering operation.

The strictest and most accurate form of the derivative of the error diffusion is to mathematically differentiate it with no doubt. However, defining the strict derivative of the error diffusion algorithm is very difficult because it employs discontinuous non-linear operations, and because the computation of a neuron could depend on the results of many other neurons in the plane. Instead, we attempted to use another function to adopt back propagation into error diffusion.

5.1.1 Identity Function

First, we viewed the dither as an identity transformation. Since dithering aims to represent a richer expression by a poor quantized signal, the output must be similar to the input. The derivative of an identity function is simply the constant 1. This supposition worked well with some relatively small neural network models, but it was unstable. One reason for this failure is that the identity supposition is *too sensitive* (or *too much linear*) that many undesirable responses

of the input/output activations (they are expected to be suppressed in a part of the non-linearity of the standard activation functions) affected the weights.

5.1.2 Active-Only Propagation

The basic principle of the back propagation is to claim the responsibility of the input for the output result. The idea is to pass the output error (or *delta*; not to be confused with the quantization error) only to the neuron that was activated as '+1' at the positive-only dithering and activation, similarly to the back propagation for max pooling. This was totally unsuccessful, probably because it was the opposite of the case of identity assumption... it was *so dull* that the necessary feedbacks were trapped.

5.1.3 Partial Linear Approximation (Near-Threshold Propagation)

This method propagates the output error when the weighted-sum value before quantization is near the threshold and is very similar to the piecewise linear approximation of the Sign function of a standard binary neural network. It worked perfectly with the models we tested, from narrow and shallow networks to deep networks. We view the two characteristics of this setting: 1) the weights that must be updated waver near the threshold; 2) this near-threshold back propagation, where the neurons with the weighted-sum values in a certain range are updated, could harmonize with the characteristic of the dithering where the location of the neurons being activated is changed by the error diffusion.

5.1.4 Delayed-Linear Derivative

The same calculation as that for the error diffusion is used, as *delta-sigma modulation* in the field of digital audio. This is a form of ADC, which generates a pulse-density-modulated binary stream from the source analog signal. The transfer function of this modulation is denoted as $Y = z^{-1}X$, so the output sequence is linear to the input with 1-cycle delay. Based on this observation, we tested the *delayed-linear* derivative, where the output errors propagate linearly to the input with a 1-pixel shift. This implementation worked well in most of the experimental cases. We should select the near-threshold or this delayed-linear back propagation considering their computational load and non-linearity that would affect the training speed and final accuracy.

5.2 Complex Dithers

We mentioned that the error diffusion algorithm in image processing could be more complex for perceptually better image quality. This tends to enlarge the area for distribution of the quantization errors; for example, the Floyd–Steinberg dithering distributes the error of a pixel to 4 neighboring pixels. Many such algorithms distribute the error unevenly, *i.e.* they use multiple scaling factors for each direction.

However, we believe that this complex dithering can hardly be used with neural networks, because such processing procedures are too complicated to be integrated in an accelerator. The 1-directional error diffusion we experimented is the most friendly to the hardware implementation. Moreover, we tested a uniform 4-directional error diffusion algorithm (distributes the error of a pixel to the right, lower-left, lower, and lower-right pixels with magnifying 1/4) by software simulation, but the recognition accuracy did not improve as much as that in the case of the 1-directional version.

6. Evaluation

We evaluate the proposed algorithm from the viewpoints of accuracy and hardware cost. First, we conduct the simulation-based accuracy comparison by training a 10-layer binary CNN model with/without dithering. Second, we construct test architectures for the CNN processing, and evaluate the hardware impact of adopting the dithering. Through the above, we show that the proposed algorithm can achieve higher accuracy than conventional binary quantization while minimizing the hardware cost.

6.1 Experimental Setup

6.1.1 Training Environment

To organize the neural network models, we used TensorFlow [17], Keras [18], and PyTorch [19] frameworks running on GPGPU servers. We customized the frameworks to apply several non-standard operations. We implemented the forward and backward operations of the error diffusion using the TensorFlow and PyTorch C++/CUDA API according to the discussion in Sect. 5.1. The *Sign* activation function and weight binarization method proposed in [1] were also included in our implementation. In the process of the hyperparameter exploration, we partially used Optuna [20], though the configurations of the final experiments were manually fixed to equalize the environments along them.

The network models we tested are shown in Tables 1 and 2, namely Model A and Model B. Both in the "w/o Dither" and "w/ Dither" cases, the first convolutional layers accept the full-range (24-bit color) images and binary weights, and the later layers have the binarized activations and weights, except for the final readout layer that remains floating-point due to the Softmax operation. This would not be a limitation when the model is offloaded on an accelerator; most implementations of the neural network accelerators have the input layer preprocessed by their host CPUs, and the final readout Softmax layer is only applied in the training phase and is replaced with a linear or binary activation layer (this is possible because the Softmax operation is monotonic and only the location of the maximal value is important). The models were trained using the CIFAR-10 image recognition dataset [21] in Models A and

Table 1 Test Network Model Architectures A

#	Layer Type	Activation F.		Output Size
		w/o Dither	w/ Dither	
0	INPUT	-		$32 \times 32 \times 3$
1	CONV	Sign	**Dither**	$30 \times 30 \times 128$
2	CONV	Sign	**Dither**	$30 \times 30 \times 128$
-	MaxPool	Sign	Sign	$15 \times 15 \times 128$
3	CONV	Sign		$15 \times 15 \times 256$
4	CONV	Sign		$15 \times 15 \times 256$
-	MaxPool	Sign		$7 \times 7 \times 256$
5	CONV	Sign		$7 \times 7 \times 512$
6	CONV	Sign		$7 \times 7 \times 512$
-	MaxPool	Sign		$3 \times 3 \times 512$
7	FC	Sign		1,024
8	FC	Sign		1,024
9	FC	SoftMax		10

In *Layer Type*, we represent the input layer as INPUT, convolutional layers with 3×3 kernel as CONV, fully-connected layers as FC, max pooling layers with 2×2 window as MaxPool. In *Activation F.*, sign activation function is denoted as Sign and 1-directional dithering as Dither. A batch normalization layer is included in each CONV layer but is not shown in the table.
The first CONVs #1 in both case are in the "valid" mode, and the others are in the "same" padding mode.
Output Size refers to (Output height) × (Output width) × (Output channels) for the input and convolutional layers and (Output neurons) for the fully-connected layers.

B, and SVHN [22] dataset in Model B. To use CIFAR-10 and SVHN whose input size is 32×32 in Model B with the input size 224×224, we enlarged the input images by bilinear interpolation. We used Adam optimizer with the learning rate decay strategy starting from 0.01 and halved every 25 epochs (Model A), and from 0.005 being halved every 20 epochs (Model B). We applied the data augmentation technique to improve the generalization capability of the networks.

The reason why we did not apply the dithering in DW-CONV layers in Model B is that they are followed by CONV layers with 1×1 kernels (*i.e.* pointwise convolution layers). The "demodulation" of the dithered signal in the context of signal processing is done by applying a low-pass filter; in a convolutional neural network, the convolutional kernels are expected to act as the filters. This suggests that the size of convolution kernels, as well as the number and location of layers with dithering, would be a key when we explore the hyperparameter space.

6.1.2 Target FPGA

We designed the prototype architectures in Verilog HDL, and synthesized them using Xilinx Vivado 2017.4. We chose the Xilinx Zynq-7000 XC7Z020 FPGA mounted on the ZedBoard evaluation kit as the target because it is a middle-range system-on-chip (SoC) that features the ARM processor coupled with user logic on a single FPGA, which is an acceptable prototype candidate for mobile applications.

6.2 Accuracy

The accuracy evaluation was conducted on a GPGPU workstation using a 10-layer tiny CNN models "Model A" (Table 1) with CIFAR-10 dataset and a MobileNet [23]-based CNN models "Model B" using CIFAR-10 and SVHN format 2 dataset. We first pre-trained a binary neural network model with the same structure as the "w/o Dither" model for 200/100 epochs for Models A/B, and we then trained the two models "w/o Dither" and "w/ Dither" for 200/50 more epochs starting with the pre-trained weights.

The Model A "w/ Dither" achieved 87.14% accuracy with two layers employing the dithering, whereas the baseline "w/o Dither" model was 85.83% accurate. This result outperformed the previous multithreshold model [16], although a straightforward comparison is impossible due to the differences in the model structure. It should be noted that the proposed method could be used with other state-of-the-art algorithms including [16].

Model B trained with CIFAR-10 dataset showed the accuracy 75.15%/73.45% with/without dithering, while the model trained using SVHN format 2 achieved 91.64%/90.91% with/without dithering, respectively. The improvement with SVHN dataset is not greater than that with CIFAR-10 dataset, although the baseline accuracy with SVHN is higher than CIFAR-10. One possible explanation of this is the following: dithering remaps the multi-level bit precision on the spatial resolution using multiple low-precision pixels, which assumes the large enough area of smooth gradation; the pictures of digits in the SVHN dataset did not match this assumption.

Again, the error-diffusion-based dithering algorithm can be utilized with any settings of the dataset, network model structure, and optimizer, as long as the model has 2-d (height, width) or higher-dimensional image-like activations with quantization, however it works well especially with spatially smooth data like objects in natural photos.

6.3 FPGA Implementation

To evaluate the hardware impact on the dithering operation, we implemented two prototype PE-array-based parallel architectures on an FPGA with/without dithering support. This design only supports the processing of a convolutional layer of a neural network for simplicity.

Figure 6 (a) (b) (c) (d) indicates the prototype architectures. The PE array (Fig. 6 (a)) forms a primitive binary neural network accelerator based on the typical output-input-channel parallelism, where each PE row corresponds to an input channel and each PE column corresponds to an output channel. This is a binary-only subset of a single core of the architecture proposed in [4]; the weights and inputs/outputs are all in 1-bit. In this configuration, an input activation is shared among multiple output channels (*i.e.* among PEs in a row), and an output is calculated in a column, with a word of the weight RAM being distributed bit-wise to the PEs and the Activation PE (Act-PE; Fig. 6 (c)) summing up the partial sums (weighted-sums) accumulated in each PE (shifted column-wise sequentially) and generating an output activation.

Table 2 Test Network Model Architectures B: MobileNet v1

#	Layer Type	Activation F.		Output Size
		w/o Dither	w/ Dither	
0	INPUT	-		224 × 224 × 3
1	CONV 3-s2	Sign	**Dither**	112 × 112 × 32
2-1	DW-CONV 3-s1	Sign	Sign	112 × 112 × 32
2-2	CONV 1-s1	Sign	**Dither**	112 × 112 × 64
3-1	DW-CONV 3-s2	Sign	Sign	56 × 56 × 64
3-2	CONV 1-s1	Sign	**Dither**	56 × 56 × 128
4-1	DW-CONV 3-s1	Sign		56 × 56 × 128
4-2	CONV 1-s1	Sign		56 × 56 × 128
5-1	DW-CONV 3-s2	Sign		28 × 28 × 128
5-2	CONV 1-s1	Sign		28 × 28 × 256
6-1	DW-CONV 3-s1	Sign		28 × 28 × 256
6-2	CONV 1-s1	Sign		28 × 28 × 256
7-1	DW-CONV 3-s2	Sign		14 × 14 × 256
7-2	CONV 1-s1	Sign		14 × 14 × 256(*)
8~12-1	DW-CONV 3-s1	Sign		14 × 14 × 512
8~12-2	CONV 1-s1	Sign		14 × 14 × 512
13-1	DW-CONV 3-s2	Sign		7 × 7 × 512
13-2	CONV 1-s1	Sign		7 × 7× 1,024
14-1	DW-CONV 3-s1	Sign		7 × 7× 1,024
14-2	CONV 1-s1	Sign		7 × 7× 1,024
-	AvePool	-		1 × 1× 1,024
15	CONV 1-s1	SoftMax		1 × 1 × 10

In *Layer Type*, we represent the input layer as INPUT, standard and pointwise convolutional layers with $k \times k$ kernel and stride s as CONV k-ss, depthwise convolution as DW-CONV-k-ss, and an average pooling with 7×7 kernel and window (also known as global average pooling) layer as AvePool. In *Activation F.*, sign activation function is denoted as Sign, and 1-directional dithering as Dither. A batch normalization layer is included in each CONV or DW-CONV layer but is not shown in the table. All the CONV and DW-CONV layers are in "same" padding mode.
Output Size refers to (Output height) × (Output width) × (Output channels).
(*) Although the output size of the layer #7-2 is $14 \times 14 \times 512$ in [23], we utilized $14 \times 14 \times 256$ due to the resource limitation of the training environment.

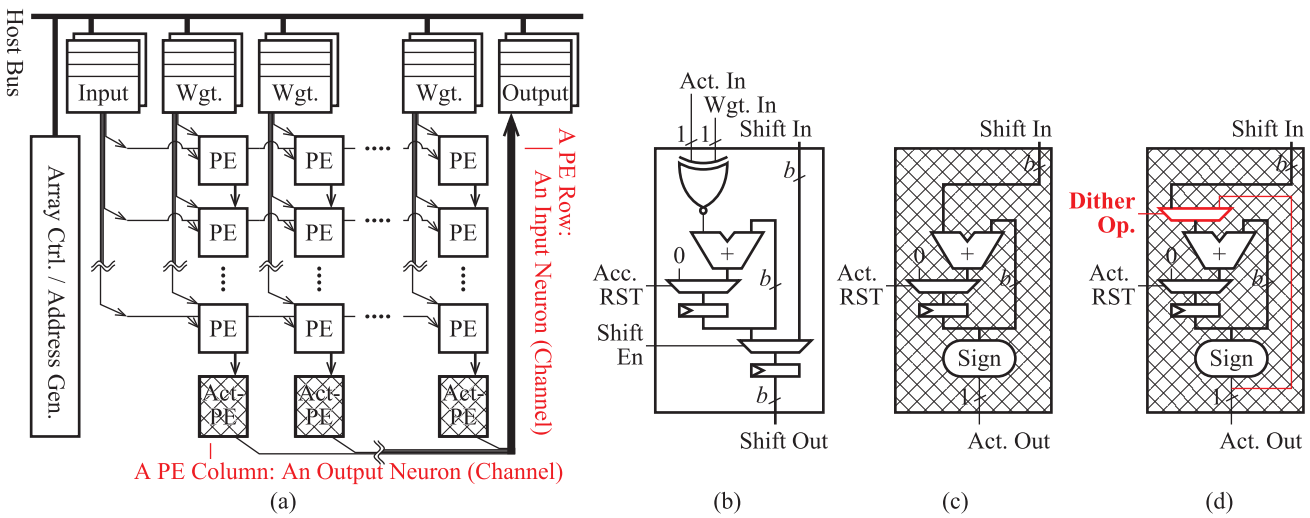

Fig. 6 Prototype array architecture. The arithmetic bit width is denoted as b. (a) Whole PE array with the input/weight/output RAMs and the controller, (b) a PE to calculate the weighted sum of an input channel, (c)(d) Act-PEs with/without dithering support to sum the results of PEs in a column and apply the Sign function to the sum.

The only difference between the two architectures with/without dithering is the type of Act-PE (without dithering Fig. 6 (c) or dithering (d)) used. The dithering Act-PE in Fig. 6 (d) has additional multiplexers to select either RESET or DITHER operation at the end of the input accumulation. Repeatedly, the dithering (error diffusion) operation can be performed as a subtraction (accumulation), for which the adder in an accumulator of each Act-PE can be used. Therefore, the overhead of the hardware resource for adopting the dithering operation would not be significant. In addition, since the dithering operation is conducted *instead of* the accumulator reset operation, no additional clock cycles are needed, as shown in Figs. 7 and 8.

Here, the implementation results of the architectures are shown in Table 3. The table includes only the PE array and its corresponding controllers and RAMs; any other parts such as data transfer circuits are not included. In this evaluation, we used the Xilinx Vivado software and the Zynq-7000 XC7Z020 FPGA, as mentioned above, with both architectures synthesized/PARed by the "Area Exploration" strategy. As seen in the table, the LUT and register usage would increase by less than 1% upon adding the dithering operation, and the RAM usage does not change, because the dithering operation does not need any additional arithmetic

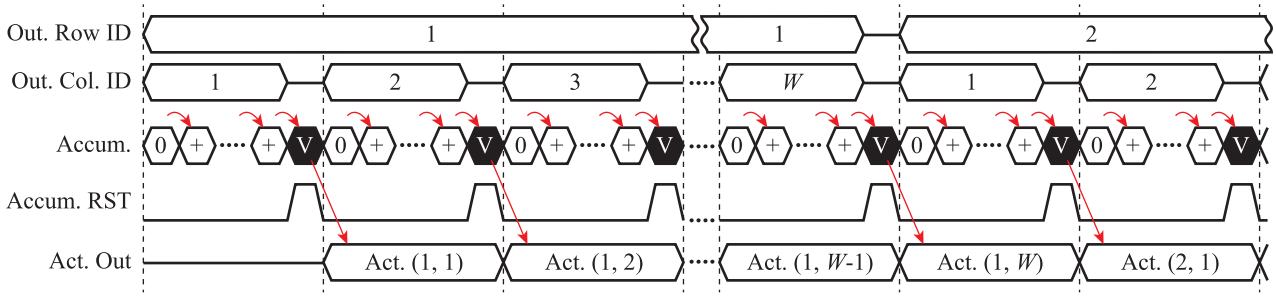

Fig. 7 Timing chart of the PE array without dithering. The channel axes are not shown. *W* denotes the output width, 'Accum.' is the value of the accumulator, and '+' and 'V' means 'accumulation' and 'valid' respectively.

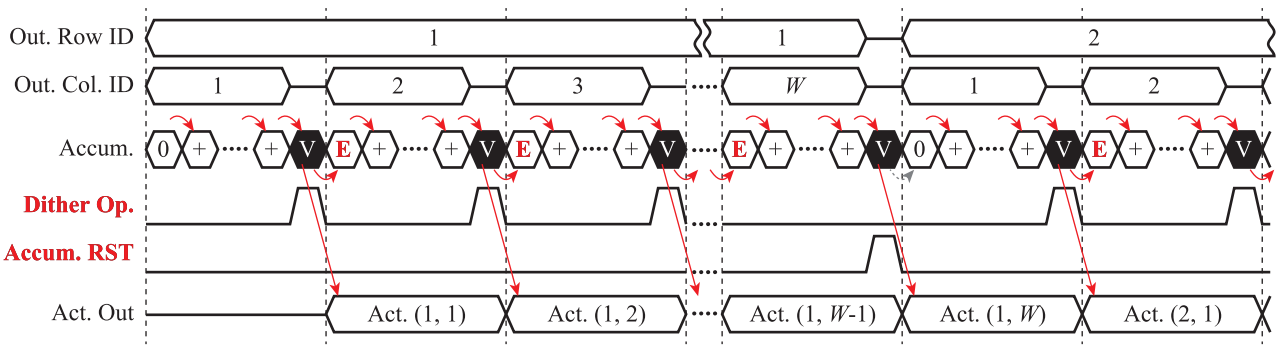

Fig. 8 Timing chart of the PE array with dithering. 'E' in 'Accum.' denotes the quantization error calculated using the previous 'V' and 'Act. Out' values.

Table 3 Implementation Result of the Prototype Architectures

Resource		Avail.	w/o Dither		w/ Dither	
			Util.	←%	Util.	←%
LUTs		53,200	18,515	34.8	18,560	34.9
Registers		106,400	19,586	18.4	19,622	18.4
BRAMs	[Tiles]	140	72	51.4	72	51.4
	[kb]	5,040	2,592		2,592	
Block IOs		200	0	0.0	0	0.0
(Accuracy A [%])		CIFAR-10		85.83		87.14
(Accuracy B [%])		CIFAR-10		73.45		75.15
		SVHN		90.91		91.64

* For both architectures, the bit width of the accumulators was set to 12; the numbers of PE rows and columns were set to 16; and all the RAMs were 16-bit 4k-word dual-bank.

units. Therefore, the proposed dithering algorithm is proved to be a hardware-friendly technique that can be utilized with a very few additional hardware resources.

7. Conclusion

We have proposed the *dithering neural network* to improve the accuracy of the quantized neural network models. The experimental accelerator architectures using binary neural network have proved that the proposed concept can be efficiently realized without deviating from the nature of the quantized neural network on the limited hardware resources.

The proposed method is the first contribution, to the best of our knowledge, that enhances the quantized neural network with a very few additional hardware resource requirement by importing the quantization error minimizing technique from the field of image processing, as the fruit of the hardware-algorithm co-designing strategy.

As a future work, there is room for exploring the optimal implementation of the inference hardware (such as parallelism, memory architecture, *etc.*) and the optimal model constructing and training methods (for example, automatic model optimization framework taking account of the relationship between the layers with and without dithering).

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers JP18J20307, JP18H05288, and JST PRESTO JPMJPR18M9.

References

[1] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1," CoRR, vol.abs/1602.02830, pp.1–11, 2016.
[2] D. Miyashita, E.H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," CoRR, vol.abs/1603.01025, pp.1–10, 2016.
[3] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "BRein Memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," IEEE J. Solid-State Circuits, vol.53, no.4, pp.983–994, 2018.
[4] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J.

Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "QUEST: A 7.49TOPS multi-purpose log-quantized DNN inference engine stacked on 96MB 3D SRAM using inductive-coupling technology in 40nm CMOS," 2018 IEEE International Solid - State Circuits Conference - (ISSCC), pp.216–218, Feb. 2018.

[5] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," 2018 IEEE International Solid - State Circuits Conference - (ISSCC), pp.218–220, Feb. 2018.

[6] K. Ando, K. Ueyoshi, Y. Oba, K. Hirose, R. Uematsu, T. Kudo, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, and M. Motomura, "Dither NN: An accurate neural network with dithering for low bit-precision hardware," 2018 International Conference on Field-Programmable Technology (FPT), pp.9–16, Dec. 2018.

[7] M. Courbariaux, Y. Bengio, and J. David, "Low precision arithmetic for deep learning," CoRR, vol.abs/1412.7024, 2014.

[8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," Proc. 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, pp.1737–1746, JMLR.org, 2015.

[9] B. Moons, B.D. Brabandere, L.V. Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), pp.1–8, March 2016.

[10] B. Moons and M. Verhelst, "A 0.3 –2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," 2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits), pp.1–2, June 2016.

[11] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable cnn-rnn processor for general-purpose deep neural networks," 2017 IEEE International Solid-State Circuits Conference (ISSCC), pp.240–241, Feb. 2017.

[12] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," CoRR, vol.abs/1511.00363, 2015.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," CoRR, vol.abs/1502.03167, 2015.

[14] F. Li and B. Liu, "Ternary weight networks," CoRR, vol.abs/1605.04711, pp.1–5, 2016.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," Computer Vision – ECCV 2016, Lecture Notes in Computer Science, vol.9908, pp.525–542, Springer International Publishing, Cham, 2016.

[16] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "ReBNet: Residual binarized neural network," 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.57–64, 2018.

[17] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[18] F. Chollet et al., "Keras." https://keras.io, 2015.

[19] "PyTorch." https://pytorch.org.

[20] Preferred Networks, Inc., "Optuna: Define-by-run hyperparameter optimization framework." https://optuna.org/, 2018.

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Computer Science Department, University of Toronto, Jan. 2009.

[22] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A.Y. Ng, "Reading digits in natural images with unsupervised feature learning," NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 01 2011.

[23] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," CoRR, vol.abs/1704.04861, 2017.

Appendix: Hardware Implementation of Other Quantization Techniques with Dithering

We have discussed and evaluated the dithering mainly in the binary neural network and its hardware as an example. The algorithm would be extended to any kind of quantization methods, such as ternary [14], logarithmic [2], and linear fixed-point, since any quantization technique produces the quantization error.

Here, we discuss applying the error diffusion to a typical hardware architecture for neural networks with fixed-point and logarithmic quantization. Quite similar methodology to the binary accelerator with dithering we used for evaluation could be utilized here, therefore the hardware overhead would not be significant even for that quantized hardware, although they have not been synthesized and evaluated.

A.1 Fixed-Point Linear Quantization

Fixed-point linear quantization has been utilized in various neural network accelerator arcitectures. In the fixed-point quantization with the B_i-bit integer part and B_f-bit fractional part (*i.e.* $QB_i.B_f$ format), the multiplication of two numbers produces the product with the $2B_i$ integer and $2B_f$ fractional bits, and they are then truncated to the original B_i-and-B_f-bit expression, as shown in Fig. A· 1. Here, the remainder bits of the integer part corresponds to the arithmetic overflow, but that of the fractional part is simply ignored, thus this causes the *quantization error*.

In a real fixed-point MAC operation, the bit truncation usually takes place after the completion of the accumulation, not just after the multiplication, therefore the accumulator has the bit width for at least $2B_i$-and-$2B_f$-bit number. An activation function, such as ReLU, is applied to the produced sum. Thus the same thing as the binary neural network — the quantization error occurring at the end of MAC computation appears also in the fixed-point quantization. The computation of the dithering on the fixed-point number is quite similar to that of binary, but the biggest difference from the binary is that the resultant number is still in a multi-level number. As indicated in Fig. A· 2, the difference between the accumulated sum and the truncated result is added at the accumulation of the next output neuron as the *quantization error*. Similarly to the case of binary, the method where the quantization error is accumulated in the accumulator instead of the reset operation can be used here. If the bit truncation is implemented as *rounding toward zero*, the quantization error can be computed by picking the lower fractional bits (masking the larger bits and extending its sign bit) with no explicit subtraction as shown in Fig. A· 3.

A.2 Logarithmic Quantization

Neural network using logarithmic quantization was proposed in [2], where both the activation and weights are rep-

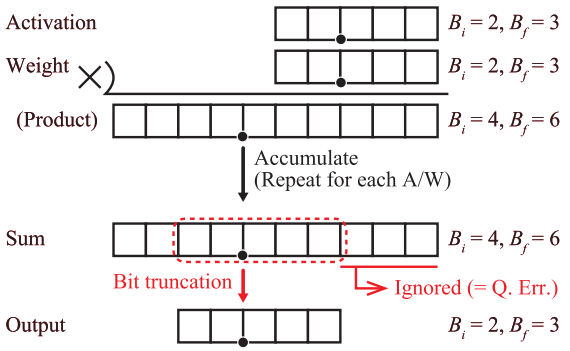

Fig. A· 1 Typical procedure of the MAC operation of the fixed-point activations and weights.

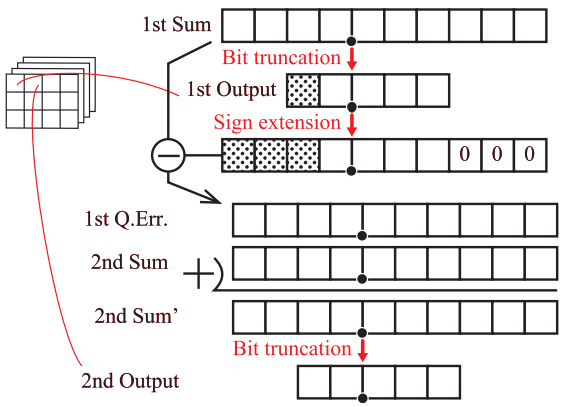

Fig. A· 2 Dithering operation in the fixed-point computation.

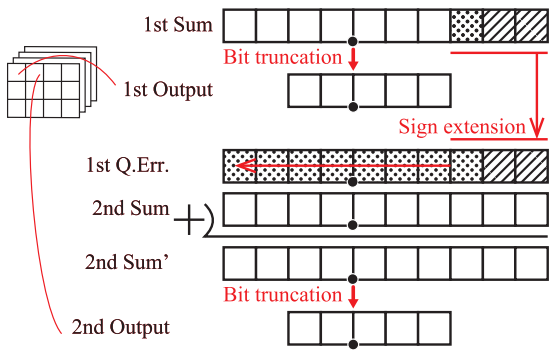

Fig. A· 3 Dithering operation in the fixed-point computation when the truncation is *rounding toward zero*.

resented in base-2 logarithm. The advantage of the logarithmic quantization is that the multipliers can be eliminated, because the multiplication between the weight and activation is replaced by the addition between the logarithms of them.

Though [2] presented a technique to compute all the operation including arithmetic addition in the logarithmic domain, one possible and reasonable realization of the logarithmic quantization is to represent/multiply the activation and weight in the logarithmic domain and to accumulate the product in the linear domain. As shown in Fig. A· 4,

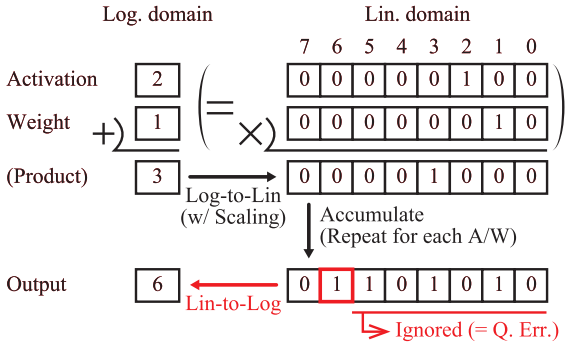

Fig. A· 4 Typical procedure of the MAC operation of the logarithmic activations and weights. In this example, the scaling factor is omitted for the simplicity, and the sign is coded in the *signed absolute* expression.

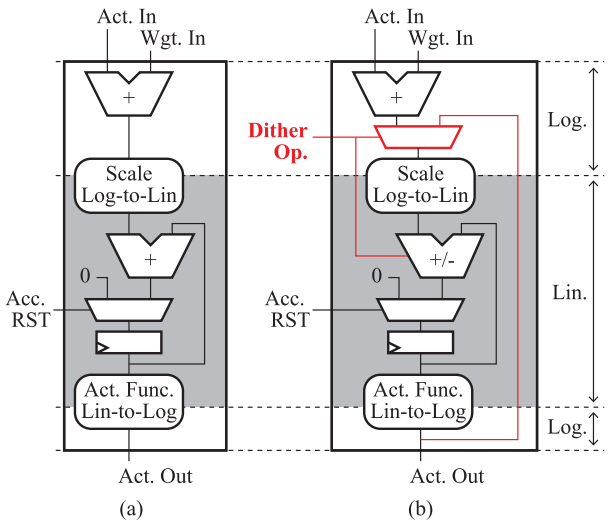

Fig. A· 5 (a) A typical logarithmic PE (for output-pixel-parallel computation). (b) Applying the dithering in the logarithmic PE. These PEs are a straightforward extension from the binary ones in Fig. 3.

after the multiplication (addition between the logalithmic-represented activation and weight), the product is converted into the linear expression with proper scaling, and then it is accumulated in the linear domain, and finally the sum is activated and quantized in logarithmic expression as the output activation at the end of accumulation. Therefore, the *quantization error* can be defined as the difference between the linearly accumulated sum and the real value of the logarithmic-represented activation.

Figure A· 5 indicates the ideal and primitive architecture to process dithering in logarithmic neural networks. Since the accumulation of the products of the activations and weights is conducted in linear domain, the same processing method as the case of fixed-point can be used; the quantization error to be accumulated is computed as the subtraction between converted logarithmic-expressed activation and the accumulator value using the adder of the accumulator.

Kota Ando received his B.E. and M.E. degrees in electronics at Hokkaido University, Japan, in 2016 and 2018, respectively. He is now pursuing his Ph.D. study at Tokyo Institute of Technology. His research interests cover reconfigurable architectures, memory-centric processing, and hardware-aware algorithms for efficient deep learning processing. He received the Best Student Presentation Award from the Technical Committee on Reconfigurable Systems of IEICE, Japan, in 2016 and 2017, the Best Student Poster Award from the Technical Committee on Integrated Circuits and Devices of IEICE in 2018, and the Best Paper Award at the 2018 International Conference on Field-Programmable Technology. He has been a JSPS Research Fellow since 2018. He is a student member of IEICE and IEEE.

Kodai Ueyoshi received his B.E. and M.E. degrees from Hokkaido University, Japan, in 2015 and 2017, respectively. He is currently pursuing his Ph.D. study on efficient hardware architecture for machine learning systems at the same university. Current topics include deep learning hardware accelerators and memory-error tolerance analysis for deep learning system. He is a JSPS Research Fellow from 2017. He received the ISSCC Silkroad Award in 2018 and the JSPS Ikushi Prize in 2019.

Yuka Oba received her B.E. degree in electronics from Hokkaido University, Japan, in 2018. She is now pursuing her M.E. study at the same university. Her current research interests are in algorithms for constructing/converting hardware-ready neural network models.

Kazutoshi Hirose received his B.E. and M.E. degrees in electronics from Hokkaido University, Japan, in 2017 and 2019, respectively. He is now pursuing his Ph.D. study at the same university. He is interested in deep neural networks and its hardware-aware algorithms. He received the Young Researcher Award from the IPSJ Special Interest Group on System Architecture and the Best Student Presentation Award from the Technical Committee on Computer Systems of IEICE, Japan, in 2017. He has been a JSPS Research Fellow since 2019.

Ryota Uematsu received his B.E. and M.E. degrees in electronics from Hokkaido University, Japan, in 2017 and 2019, respectively. He is interested in high-level hardware optimizer, and power reduction techniques in backend design flow. He received the Young Researcher Award from the Workshop on Synthesis And System Integration of Mixed Information technologies of IEEE in 2018.

Takumi Kudo received his B.E. degree in electronics from Hokkaido University, Japan, in 2018. He is now pursuing his M.E. study at the same university. His current interest includes mixed-paradigm neural network models aware of hardware processing.

Masayuki Ikebe received his B.S., M.S., and Ph.D. degrees in electrical engineering from Hokkaido University, Sapporo, Japan, in 1995, 1997, and 2000, respectively. During 2000-2004, he worked for the Electronic Device Laboratory, Dai Nippon Printing Corporation, Tokyo, Japan, where he was engaged in the research and development of wireless communication systems and image processing systems. Presently, he is a Professor and leads the Integrated Quantum Systems group at the Research Center For Integrated Quantum Electronics (RCIQE), Hokkaido University. His current research interests are analog and mixed-signal IC design with an emphasis on CMOS image sensor, THz imaging devices and intelligent image processing systems. He is a member of IEICE and IEEE.

Tetsuya Asai received his B.S. and M.S. degrees in electronic engineering from Tokai University, Japan, in 1993 and 1996, respectively, and his Ph.D. degree from Toyohashi University of Technology, Japan, in 1999. He is now a Professor in the Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan. His research interests are focused on developing intelligent integrated circuits and their computational applications. Current topics that he is involved with include emerging research architectures, deep learning accelerators, and device-aware neuromorphic VLSIs.

Shinya Takamaeda-Yamazaki received the B.E, M.E, and D.E degrees from Tokyo Institute of Technology, Japan in 2009, 2011, and 2014 respectively. From 2011 to 2014, he was a JSPS research fellow (DC1). From 2014 to 2016, he was an assistant professor of Nara Institute of Science and Technology, Japan. From 2016 to 2019, he was an associate professor of Hokkaido University, Japan. Since 2018, he has been a researcher of JST PRESTO. Since 2019, he has been an associate professor of The University of Tokyo, Japan. His research interests include computer architecture, high-level synthesis, and machine learning acceleration. He is a member of IEEE, IEICE, and IPSJ.

Masato Motomura received the B.S., M.S., and Ph.D. degrees in electrical engineering from Kyoto University, Kyoto, Japan, in 1985, 1987, and 1996, respectively. In 1987, he joined the NEC central research laboratories, where he worked on various hardware architectures including string search engines, multi-threaded on-chip parallel processors, embedded DRAM field-programmable gate array (FPGA) hybrid systems, memory-based processors, and reconfigurable systems. From 2001 to 2008, he was with NEC Electronics where he led research and business development of dynamically reconfigurable processor (DRP) that he invented. He was also a visiting researcher at MIT laboratory for computer science from 1991 to 1992. From 2011 to 2019, he was a professor at Hokkaido University. He has been a Professor at Tokyo Institute of Technology, Japan, since 2019. His current research interests include reconfigurable and parallel architectures for deep neural networks, machine learning, annealing machines, and intelligent computing in general. Dr. Motomura is a member of IEICE, IPSJ, and EAJ. He was a recipient of the IEEE JSSC Annual Best Paper Award in 1992, the IPSJ Annual Best Paper Award in 1999, the IEICE Achievement Award in 2011, and the ISSCC Silkroad Award as the corresponding author in 2018, respectively.