Learning accelerator of deep neural networks with logarithmic quantization

Takeo Ueki†,Keisuke Iwai,Takashi Matsubara,and Takakazu Kurokawa
Department of Computer Science
National Defence Academy of Japan
Yokosuka, Japan
†Email: em56031@nda.ac.jp

***Abstract*—In recent years, Deep Neural Network (DNN)s have become widely spread. Several high-throughput hardware implementations for DNNs have been proposed. One of the key points for hardware implementations of DNNs is to reduce their circuit scale because DNNs require a lot of product-sum operations. Previous papers presented some accelerators using logarithmic quantization to reduce circuit scale by replacing multipliers with shifters. However, most of them are implemented only for inference, and we can find no previous paper using logarithmic quantization for learning. In this paper, a learning accelerator using a logarithmic quantization is proposed. The accelerator uses a stochastic approach for logarithmic quantization. It enables DNNs to learn using logarithmic quantization values. Preliminary evaluation confirmed that DNNs using the proposed method can achieve recognition of MNIST with an accuracy of 97.8%. This evaluation result proves that the proposed method can be used for learning DNNs. The accelerator using the proposed method is implemented on a field-programmable gate array (FPGA) (Xilinx Zynq7020) and synthesized with Xilinx Vivado 2017.1. As a result, it is confirmed to have 2.9 times smaller circuit scale compared with the same FPGA accelerator using 32-bit fixed point number.**

***Index Terms*—neural networks, logarithmic computation, approximate computing, field-programmable gate arrays**

I. INTRODUCTION

In recent years, Deep Neural Network (DNN)s have become widespread. DNN is one of the machine learning methods. DNN requires a lot of data samples and product-sum operations. For example, the first layer of vgg16: A network for image recognition proposed by Simonyan et al. in 2015 [1], requires a 86,704,128 multiplications in inference. For this reason, several implementations with a low power consumption have been proposed. There are several implementation methods which can learn / infer with low power consumption.

Logarithmic quantization is a method to reduce computing resource for DNNs. In this method, values are dealed with 4 to 7-bit width, and shifters or adders are used in place of multipliers for matrix multiplications in learning phase. Therefore, less data memory is required and the operation circuit can be small with logarithmic quantization. Because of these characteristics, a hardware implementations with logarithmic quantization have been proposed [2]. However, this implementation is designed for inference devices, not for learning device. In this paper, an accelerator with logarithmic quantization adopting a stochastic approach is proposed. The rest of this paper is organized as follows. Section 2 gives related works of logarithmic quantization and clarifies the problems of conventional methods. Section 3 presents our quantization method to overcome the problems of conventional methods. Section 4 compares the recognition accuracy of the network by conventional method and the proposed method, and evaluates the proposed quantization method as a preliminary evaluation. Section 5 implements an accelerator with proposed method on an FPGA. Section 6 evaluates its characteristic by comparing with conventional implementation. Finally, Section 7 describes a summary of results and future issues.

II. LOGARITHMIC QUANTIZATION

Logarithmic quantization is a technique using power-of-two approximation. The logarithmic quantization number is represented with a sign bit and an exponent part. For example, logarithmically quantizing a 32-bit fixed-point number can be represented by 1 bit for sign bit and 5 bits for exponent part.

A method proposed by Miyashita et al. (2016) [3] performs quantization based on the following scheme.

$$\mathrm{LogQuant}(\mathrm{x}, \mathrm{bitwidth}, \mathrm{FSR}) := \begin{cases} 0 & \text{if } x = 0, \\ 2^{\hat{x}} & \text{otherwise} \end{cases} \tag{1}$$

where

$$\hat{x} = \mathrm{Clip}\left(\lfloor x \rfloor, \mathrm{FSR} - 2^{\mathrm{bitwidth}}, \mathrm{FSR}\right), \tag{2}$$

$$\mathrm{Clip}(x, \min, \max) = \begin{cases} 0 & \text{if } x \leq \min, \\ \max - 1 & \text{if } x \geq \max, \\ x & \text{otherwize} \end{cases} \tag{3}$$

[3] investigated the influence of logarithmic quantization on the point of recognition accuracy by inserting logarithmic quantization layers to a network. According to their investigation, a decline of the accuracy of the quantized network is very small compared with the non-quantized network.

Quantization Neural Netowrk (QNN)s are proposed as quantized networks in various ways [4]. In the study of QNNs, the networks constructed on the basis of the scheme proposed in [3] are compared with networks based on binarization such as Binary-Connect [5]. According to this work, logarithmic quantization requires 6 bits for numerical expression when

978-1-5386-7447-5/18/$31.00 ©2018 IEEE
DOI 10.1109/IIAI-AAI.2018.00133

computing logarithmic quantization of weights, loss gradients and activations.

There is a study focusing on the reduction of the amount of memory by logarithmic quantization [6]. This study presents that the quantization makes the memory consumption in forward-propagation and back-propagation a quarter and halves the other in parameter-updating. LogNet is an implementation of the network inference device using logarithmic quantization [2]. This is an implementation of a convolution neural network. It performs the convolution products by only Shift-Add convolutions, and enables weights be distributed across the computing fabrics.

These related works have suggested methods of performing logarithmic quantization of forward-propagation of neuron outputs and back-propagation of gradients, and methods of using logarithmically quantized values for parameter-updating, etc. However, there is no proposal for inference / learning using logarithmic quantization method. Similarly, an accelerator of inference apparatus using logarithmic quantization has been proposed. However, we can find no previous work applying logarithmic quantization to learning apparatus.

III. STOCHASTIC APPROACH TO LOGARITHMIC QUANTIZATION

In related works, there are some problems in learning phase. Their approach is deterministic, so that the difference between two values cannot be represented in quantizing halfway value generated in learning phase.

For example, information will loss in the case that a parameter that has very large value is updated with small learning rate. Therefore, updating will be invalidated.

The stochastic approach solves this problem. However, it needs amount of computing resource because the logarithmic function is a non-linear function. To cope with this problem, logarithmic function is approximated with equation (4) in our method. In this way, equation (2) is replaced to equation (5) for training neural network with logarithmic quantization.

$$\log_2 x \simeq \lfloor \log_2 |x| \rfloor + \left\lfloor \frac{x}{2^{\lfloor \log_2 |x| \rfloor}} \right\rfloor - 1 \tag{4}$$

$$\hat{x} = \mathrm{Clip}\Big(\lfloor \log_2 |x| \rfloor + \left\lfloor \frac{x}{2^{\lfloor \log_2 |x| \rfloor}} - N \right\rfloor, \mathrm{FSR} - 2^{\mathrm{bitwidth}}, \mathrm{FSR} \Big) \tag{5}$$

Here, N is the uniformed random number of $[1, 2)$. Quantization with noise enables fractions to be represented by stochastic approach.

IV. PRELIMINARY EVALUATION

A. Evaluation Settings

Preliminary evaluation is conducted to evaluate recognition accuracy of stochastic logarithmic quantization, Five networks for MNIST [7] are prepared by Keras [8]; an open source python framework for neural networks.

TABLE I: Detailed Configuraton of the Network Used for Preliminary Evaluation

Layer Type	Properties
Conv2D1	filter:$5 \times 5 \times 32$,stride:1 padding:2
BatchNorm1	
Act1(Relu)	
MaxPooling2D1	filter:2×2,stride:2,padding:2
Conv2D2	filter:$5 \times 5 \times 64$,stride:1 padding:2
BatchNorm2	
Act2(Relu)	
MaxPooling2D2	filter:2×2,stride:2 padding:2
FC1	filter:3136×512
BatchNorm3	
Act3(Relu)	
FC2	filter:512×10
Act4(Softmax)	

TABLE II: Environment of Preliminary Evaluation

Item	Content
Framework	Keras on TensorFlow r1.4
Benchmark	MNIST (50,000 for training, 10,000 for test)
Optimizer	Adam ($\mathrm{lr} = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$)
Batch Size M	256
Epochs	100

Detailed configurations of the networks is shown in Table I. The networks have 2 each of a convolution layer and pooling layer, and 2 full-connection layers. The evaluation environment is shown in Table II.

Four networks use the proposed method (LogQuant5 to LogQuant8). Here, "LogQuant$n$" means that values of the network is quantized with n bits (including sign bit). These networks' layers quantize the values logarithmically before they are passed to the next layer. In addition, all layer parameters are updated with values quantized logarithmically.

Another network uses 32-bit floating point number (Float32). It is maked for the baseline.

B. Evaluation Results

TABLE III: Accuracy of the Networks

Network	LogQuant5	LogQuant6	LogQuant7	LogQuant8	Float32
Accuracy	9.8%	27.5%	97.8%	97.7%	99.4%

As shown in Table III, LogQuant7 performs 97.8% accuracy when the proposed method is applied to all layers. Compared to Float 32, recognition accuracy is reduced by 1.6 points. However, sufficient accuracy can not be obtained when quantization is performed with 6 bits or less.

Accordingly, it is necessary to perform quantization with a bit width of 7 bits or more when using the proposed method.

V. ACCELERATOR ARCHITECTURE

A. Overview

The proposed architecture of the accelerator of stochastic logarithmic quantization neural networks has the following characteristics.

- This hardware is SIMD based accelerator processor.
- It has a processing unit tile (PU Tile), a data memory (Data Mem), an instruction memory (Inst Mem), a scalar queue, and a control unit (Ctrl Unit).
- Each PU performs calculations on the neurons.
- Its instruction set architecture is 32-bit fixed length and it takes an intermediate architecture between the accumulator architecture and the memory-to-memory architecture.

Fig. 1 illustrates the design of accelerator. The accelerator is connected to an on-chip ARM CPU (on the left side of the figure) and controlled by driver software on the CPU. PU Tile is a core part of this accelerator and executes operations. Inst Mem and Data Mem consist dual port block RAMs; on-chip hard macros in FPGAs. One of the ports is connected to the PU Tile and the other to the CPU. Therefore, CPU and PU Tile can access the memories simultaneously. Ctrl Unit controlls PU Tile based on command from the driver software. Scalar queue is controlled by the driver and supplies scalar input to the PU Tile. Currently, the accelerator has only one PU Tile, but there are plans to be arranged in the future.

The accelerator is driven by the following procedures.

1) Push linear scaled scalar inputs on the scalar queue.
2) Write first instruction address to a program counter (PC) in the Ctrl Unit.
3) Start counting up the PC.

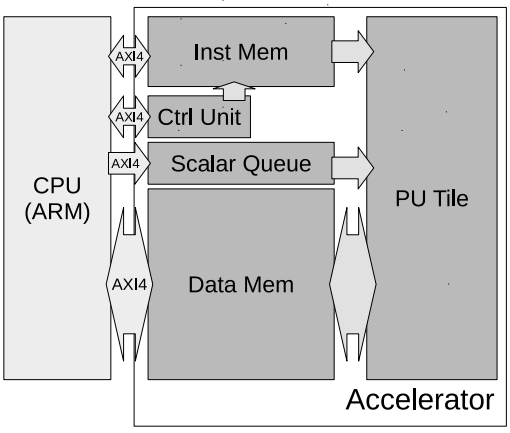

Fig. 1: Accelerator Design.

B. Data Path

The data path is shown in Fig. 2. Most parts of data path are contained in PU Tile.

PU Tile is a block that contains 16 processing unit (PU)s, one adder tree and 16 logarithmic quantizers (LogQuant). It is necessary to decide the execution order of instructions at compile time considerly since the PU Tile can only write or read at a time and there is no mechanism for stalling the pipeline. The register file has read-only port and write-only port independently. The computed result coming out of the PU enters the adder tree and is subjected to horizontal addition processing as necessary. The output data is stored to the register, or written back to the Data Mem through a LogQuant.

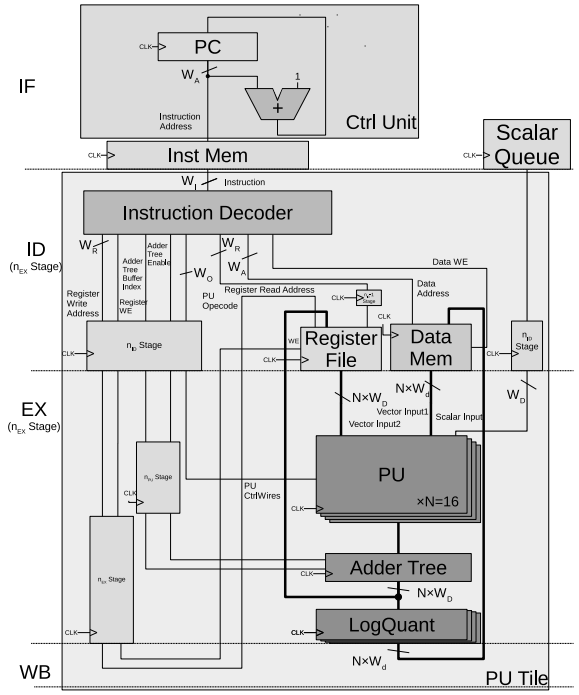

Fig. 2: Data Path of Accelerator.

C. PU: Processing Unit

PU consists of a preprocessor, arithmetic shifter, adder, accumulator and pipeline registers as shown in Fig. 3. Saturation processing is performed when overflow or underflow occurs in the course of each operation.

The preprocessor implements a function such as square root, ReLU (Rectified Linear Unit) and derivative of ReLU. The square root can be composed of only comparators and shifters since it is performed on logarithmic quantized values. The preprocessor makes it possible to calculate the activation function and the standard deviation used in batch normalization. The arithmetic shifter shifts the 32-bit fixed point number received as input with the logarithmic quantization number, and takes the place of the multiplier. In order to make it function as a multiplier for fixed point number, a mechanism for adjusting the shift amount by the number of digits of the decimal part is provided. The adder adds the output of the shifter to the value of the accumulator, but it may be turned off depending on the situation. Accumulators and pipeline registers are general flip-flops with clock enable.

D. LogQuant: Logarithmic Quantizer

Fig. 4 shows the structure of a logarithmic quantizer. The logarithmic quantizer consists of a random number generator (Xorshift[9]) and several logic circuits.

The principle is briefly described as follows. First, a random number is added to an input value (32-bit fixed point number) and only the carry received by each digit is extracted (1). At the same time, the most significant digit of the input value is taken out as a 5-bit integer by the look-up table (LUT) (2). The output value is obtained by adding the value of (1) to (2).

Thus, the same effect as equation (5) can be obtained.

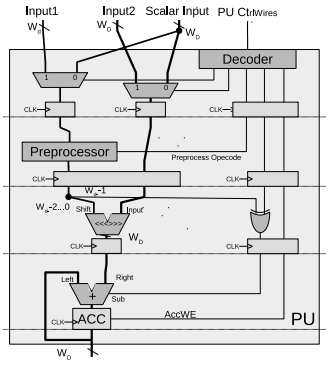

Fig. 3: Structure of PU.

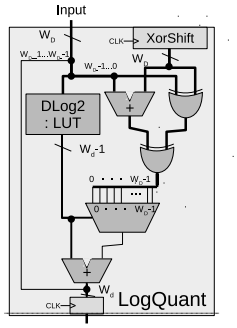

Fig. 4: Structure of LogQuant.

VI. EVALUATIONS

A. Evaluation Settings

The proposed accelerator is implemented in order to evaluate the circuit size and power consumption.

For overall evaluation, the accelerator is synthesized for Zynq 7020 from Xilinx. An ARM CPU is installed as an on-chip hard macro, and the proposed accelerator also uses this CPU for control. The accelerator is synthesized to operate at 100 MHz. The depth of the instruction memory is set with 8,192 words, and the depth of the data memory is set with 2,048 words. Two versions of 6 bits and 7 bits (LogQuant6, LogQuant7) are prepared. Another version using 32 bit fixed point number is also prepared as a comparison target and synthesized under similar conditions (FixedPoint32). Circuit synthesis and estimation of power consumption is done by Xilinx Vivado 2017 r1.

B. Circuit Size

Table IV shows size of the synthesized circuit. In all methods, 8 BRAMs are used in instruction memory.

Compared with FixedPoint32, the number of BRAMs used for the instruction memory and the data memory is less than 1/2 for both LogQuant6 and LogQuant7. This is because the bit width per PU unit is limited to one byte unit for convenience of interaction with the CPU. Number of LUTs of LogQuant6 and one of LogQuant7 are about 1/3 times and about 2/3, compared with FixedPoint32 respectively. Although Number of Flip-Flop (FF)s of LogQuant6 and one of Fixed-Point32 are approximately equal, LogQuant7 requires twice FFs as FixedPoint32. This is due to bitwidth of acccumulators and registers. Multiplexser (MUX)s are not consumed as much as LUTs or FFs.

C. Power Consumption

Table V illustrates power consumption of the accelerator. LogQuant7 has double power consumption of LogQuant6. It can be said that the circuit scale is also reflected in the power consumption. However, FixedPoint32 has smaller power consumption than LogQuant6 and LogQuant7, although having a larger circuit scale. It is considered that FixedPoint32 is adapted to the device characteristics.

TABLE IV: Circuit Size

Term	LUT	FF	MUX	BRAM
LogQuant6	8,443	7,083	80	16
LogQuant7	15,606	13,227	272	16
FixedPoint32	24,750	7,150	32	40

TABLE V: Power Consumption (W)

Term	PU Tile	Inst Mem	Data Mem	Total
LogQuant6	0.15	0.030	0.020	0.21
LogQuant7	0.32	0.030	0.020	0.38
FixedFloat32	0.063	0.027	0.099	0.20

VII. CONCLUSION AND FUTURE WORKS

A method of logarithmic quantization by stochastic approach and an accelerator architecture on FPGA by the method are proposed in this paper. The method is confirmed to be sufficiently available by preliminary evaluation. In addition, the accelerator is confirmed to be smaller circuit scale than the one of the conventional method. However the accelerator needs more power consumption than the one of the conventional methods and required detailed control from the CPU because of the insufficient function of the control unit.

In the future work, the accelerator must be modified to require low power consumption and to be controlled directly from the control unit by enhancement of the control unit.

REFFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014. arXiv: 1409.1556.

[2] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*, Mar. 2017, pp. 5900–5904.

[3] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, 2016. arXiv: 1603.01025.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *CoRR*, 2016. arXiv: 1609.07061.

[5] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 3123–3131.

[6] K. Hirose *et al.*, "Logarithmic compression for memory footprint reduction in neural network training," in *5th International Workshop on Computer Systems and Architectures (CSA 2017)*, Aomori, Japan, 2017, pp. 291–297.

[7] Y. LeCun, C. Cortes, and C. J.C.Burges. (2010). MNIST handwritten digit database, [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[8] F. Chollet *et al.* (2015). Keras, [Online]. Available: https://github.com/keras-team/keras.

[9] G. MARSAGLIA, "Xorshift rngs," *J. Stat. Software*, vol. 8, no. 14, pp. 1–6, 2003.