Implementation of DNN on a RISC-V Open Source Microprocessor for IoT devices

1st Jingyong Cai
Graduate School of Engineering
Tokyo University of Agriculture and Technology
Tokyo, Japan
s174740v@st.go.tuat.ac.jp

2nd Masashi Takemoto
BeatCraft Inc.
Tokyo, Japan
lesser@beatcraft.com

3rd Hironori Nakajo
Institute of Engineering
Tokyo University of Agriculture and Technology
Tokyo, Japan
nakajo@cc.tuat.ac.jp

***Abstract*—Logarithmic Quantization[1] and feature extraction enable us to reduce model parameters to a great extent. Based on these methods, we have implemented a small sized DNN on a RISC-V microprocessor with RAM of only 16KB. We also propose a feature extraction algorithm which outperforms the original fully connected neural network and reduces inputs by 12.25× at the same time. MNIST[2] dataset is used as our training samples and Chainer[8] is used to train the network. As the result, we reduced weights size by nearly 86× from 49.625KB to 0.578KB which make it possible to store these weights in arrays and load them directly into the RAM.**

***Index Terms*—Feature Extraction, RISC-V, Logarithmic Quantization, DNN**

I. Introduction

Advantages of small neural networks such as low latency, privacy and "always on" reliability are detailed in [3]. Low cost and easy accessibility are main motivations behind this research. We hold the belief that if IoT can connect human and things, smart little chips will make this connection even closer and more interactive.

Our research aims to explore the possibility and practicability of implementing a small sized DNN on a microprocessor with limited RAM (16KB in our case). A traditional method would store the weight and bias parameters in the external flash first, then load them into the RAM piece by piece for calculation. However, this method is exceedingly inefficient and unlikely to be implemented in practical applications. Therefore, to meet the needs of real-world applications on low cost ends, it is necessary to store the whole network parameters directly in the RAM. In order to achieve this, we propose a feature extraction algorithm to lessen the input neurons (12.25× reduction in our case) before they are fed into small sized neural networks having one hidden layer of 16 or 50 neurons. After we trained the network, weights and biases are extracted and weights are logarithmic quantized to further reduce memory requirement. Finally, we implemented a handwritten digit recognition neural network on the Hifive1 board with good efficiency and accuracy. Section two details Implementation processes. Section three introduces our testing platform and accuracy of the quantized neural network.

II. Extraction and Quantization of Model Parameters

A. Image Acquisition

MNIST dataset is used as our training samples owing to its simplicity. Handwritten digital images usually have clear contours and large portions of blank background which allows for considerable dimension reduction. For MNIST dataset, there is no shortage of demonstration examples from modern deep learning frameworks. Among these frameworks Chainer is used as our testing platform since its model parameters are easy to be extracted. Besides, network definition in Chainer is clear and easy. Each sample will be converted to a numpy matrix of 784 elements in a row. Before we perform the feature extraction, matrices are reshaped to 28 rows × 28 columns.

B. Feature Extraction

Among several classifiers surveyed by Patel et al.[6], feature extraction combined with MLP gives the highest recognition rate for handwritten digits. Four View Projection Profiles algorithm[4] with MLP, the best method in the survey, reaches recognition rate of 98.73%. However, their samples are normalized to 100 by 100 sizes which are vastly more dimensions than MNIST samples. To fill the gap, we have made some modifications to the Four View Projection Profiles and our algorithm outputs four peak sums in each direction. The process is explained as follows:

- Divide each (28,28) matrix to 16 zones (each of 7 rows × 7 columns).
- Find the horizontally, vertically, left diagonal and right diagonal sums in each line of 16 zones.
- Store 4 peak values of each direction.
- Generate feature vectors of 64 elements.

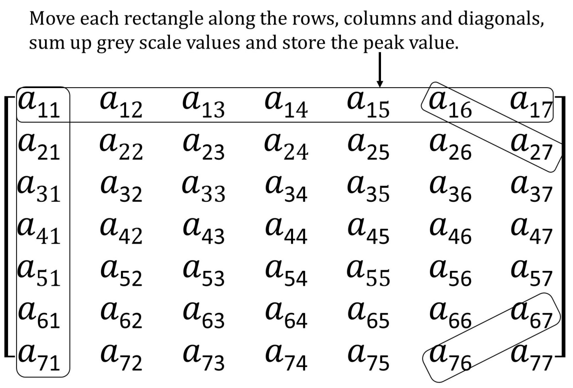
Move each rectangle along the rows, columns and diagonals, sum up grey scale values and store the peak value.
a_{11} a_{12} a_{13} a_{14} a_{15} a_{16} a_{17}
a_{21} a_{22} a_{23} a_{24} a_{25} a_{26} a_{27}
a_{31} a_{32} a_{33} a_{34} a_{35} a_{36} a_{37}
a_{41} a_{42} a_{43} a_{44} a_{45} a_{46} a_{47}
a_{51} a_{52} a_{53} a_{54} a_{55} a_{56} a_{57}
a_{61} a_{62} a_{63} a_{64} a_{65} a_{66} a_{67}
a_{71} a_{72} a_{73} a_{74} a_{75} a_{76} a_{77}

Fig. 1. Process of the algorithm.

Fig. 1. illustrates the specific process of the proposed algorithm. Compared with the Four View Projection Profiles algorithm, we firstly change one of the horizontal projections to left diagonal projection. Adding left diagonal projection enables our algorithm to detect curves in four directions. Secondly, instead of generating average of four sums in each zone, our algorithm outputs four peak sums as four different features. This step ensures more detailed information in each zone is retained.

Fig. 2. and Fig. 3. show the comparison between the original data sample and extracted feature vectors. Fig. 2. is a sample from MNIST, its greyscale has been reversed to be better compared with Fig 3. In Fig. 3., each zone contains 4 feature vectors. To better visualize it, different vector values are expressed through variations in length. In addition, we retained feature vectors' original orientation when drawing these line segments.

As can be seen from Fig. 3., although we have compressed the original image to a great extent, the information of the original image is still well preserved. Moreover, background noise is largely taken away from the original sample. Although it might be hard to recognize with human eyes, neural networks should be able to abstract characteristics and differences between samples.

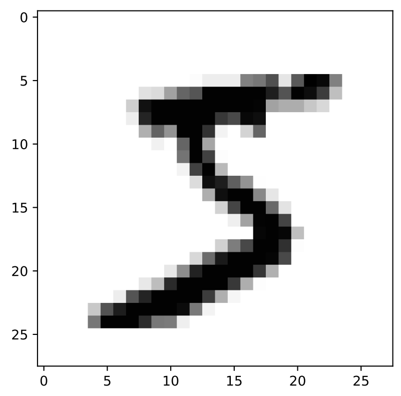

Fig. 2. Original MNIST data sample (28 × 28 pixels).

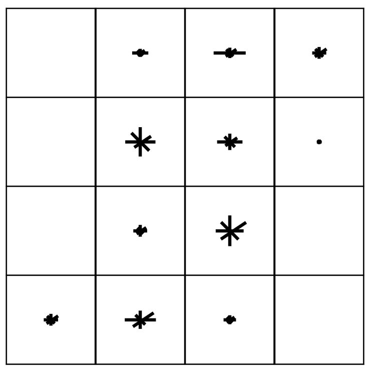

Fig. 3. Visualization of the feature vectors (16 zones).

TABLE I
AVERAGE ACCURACY OF 1000 EPOCHSE

Input size	Accuracy
784 (without feature extraction)	94.125
64 (with feature extraction)	96.035

C. Training and Evaluation of Proposed Algorithm

On account of the fact that adding depth of the network will exponentially increase both the weights size and the amount of computation, we choose single hidden layer neural network as the classifier. The algorithm is tested on neural networks contain 784 or 64 input neurons, one hidden layer of 16 neurons and an output layer of 10 neurons. Table 1 gives evaluation result of our algorithm.

As the result, our feature extraction algorithm reduced the input neurons by 12.25×. Moreover, combination of feature extraction and neural network outperforms the original fully connected neural network.

D. Logarithmic Quantization

Quantization of parameters could reduce the memory consumption by a large scale without affecting the accuracy. Song et al.'s work[5] achieved storage reduction from $35\times$ to $49\times$ via combination of three techniques: Pruning, Quantization and Huffman Coding. Based on the fact that weights in a trained network naturally have non-uniform distributions, Daisuke et al. proposed logarithmic data representation[1] that encode the weights of trained network to 3 bits without significant decrease in accuracy. The logarithmic quantization algorithm (LogQuant) we used operations as below:

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

where

FSR: Full scale range,

$$\widetilde{x} = \text{Clip}(\text{Round}(\log_2(|x|)), \text{FSR-}2^{\text{bitwidth}}, \text{FSR}), \quad (2)$$

$$\text{Clip(x,min,max)} = \begin{cases} 0 & \text{x} \leq \text{min}, \\ \text{max-1} & \text{x} \geq \text{max}, \\ \text{x} & \text{otherwise} \end{cases} \quad (3)$$

LogQuant can not only reduce the weights size, but also benefit IoT devices since it can eliminates the floating point and multiplication operations. This can be explained as follows:

if:

$$x = \log_2^a, \; w = \log_2^b$$

then:

$$wx = 2^{a+b} \; = \text{Bitshift}(1, a + b)$$

The demerit of LogQuant is the loss of sign bit. Since Chainer uses minus weights, we have to extract the sign bits first. Thankfully Numpy's mask rule and element-wise operations[7] offers an efficient and convenient way to perform this:

```
def signBitsMaskArray(weights_array):
        mask  = weights_array < 0
        mask = mask * (-1)
        mask[mask > -1] = 1
        return mask
...
QanWeWithSign = mask * QanWeWithoutSign
...
```

After LogQuant, each weight parameter will occupy four bits including one sign bit. In consideration of the fact that memory access is expensive, we combine two weights into one byte and utilize CPU to extract them afterwards. Since there are only 26 biases in our small neural network, quantization of biases is unnecessary.

E. Visualization of Logarithmic Quantization

Unlike linear quantization, enlarge the bit width of logarithmic quantization can only cause the quantized weights to concentrate around zero. Fig. 4. and Fig. 5. show the output of logarithmic quantization, both original weights and quantized weights gather around zero. Fig. 6. shows that regardless of how much we increase the bit width of LogQuant, weights that are away from zero are not affected. For instance, LogQuant can only generate two values that are greater than 0.2.

One important point is that when increasing the number of the hidden layer neurons, the Law of Large Numbers begins to function. When the number of hidden layer neurons is relatively small, increasing the number of neurons will cause the weights distribution to become smooth. This also means that weights with larger value are eliminated when increasing the number of hidden layer neurons.

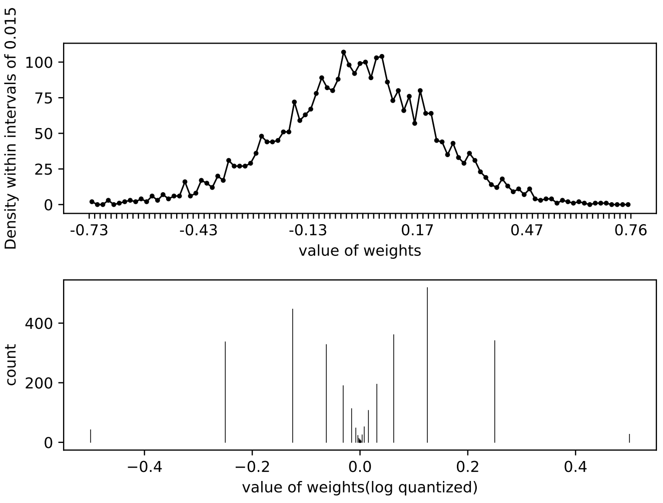

Fig. 4. 7 bits logarithmic quantization (50 neurons in hidden layer).

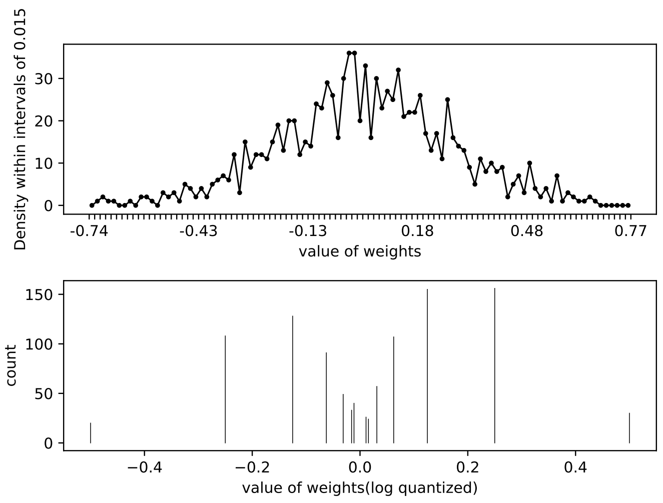

Fig. 5. 3 bits logarithmic quantization (16 neurons in hidden layer)

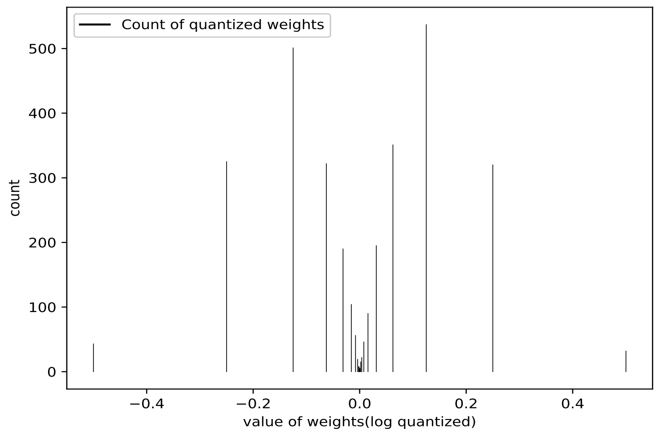

Fig. 6. 32 bits logarithmic quantization (50 neurons in hidden layer)

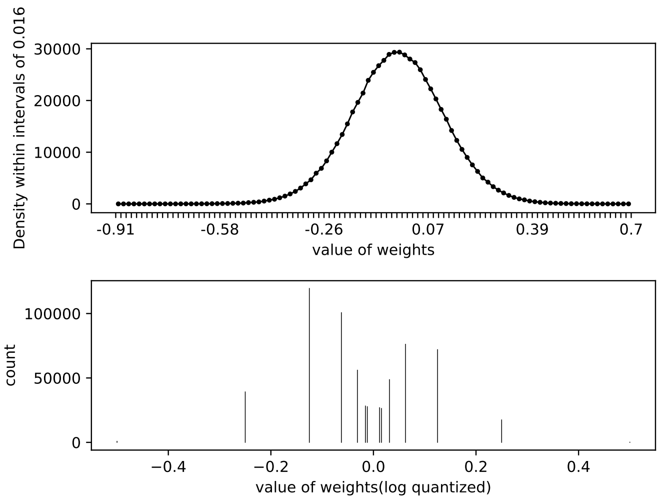

Fig. 7. 3 bits logarithmic quantization (10000 neurons in hidden layer)

Fig. 7. shows an extreme example when the hidden layer neuron size reaches 10000. The density distribution shows that weights density near two sides are almost squeezed to zero. It is also noticeable that proportion of quantized weights on two sides are small compared to Fig. 4. and Fig. 5.. This phenomenon, from another point of view, well explains why Daisuke et al.'s work on large neural networks can achieve no deterioration in accuracy. The larger layer size we use, the more concentration around value zero we will get which leads to concordance between original and quantized weights.

III. RISC-V Microprocessor and Accuracy Evaluation

After we trained our neural network, model parameters are quantized and extracted which could be easily stored in arrays. Compared with the original fully connected neural network, our method reduces the weights size from 49.625KB to 0.578KB, which achieves nearly 86× reduction. Total model parameters size is reduced from 49.723KB to 0.68KB (3 bits quantization, 16 hidden layer neurons).

Freedom E310 development board is chosen as our testing platform (on Sifive Hifive-1 board). The following explains why we choose a RISC-V processor to perform the implementation:

- Open source which allows further modifications.
- Clock speed: 320+ MHz, Performance: 1.61 DMIPs/MHz, 2.73 Coremark/MHz.
- GNU tool chain support.
- Open and extensible ISA which allows customized commands.

Although the training processes are time-consuming and require a large amount of computations, a trained model is purely a group of matrices. We exported these matrices to arrays in C and performed several recognition tests on the Hifive-1 board. We also added some control experiments with small feasibility on Hifive-1 for reference, the results are listed in table 2.

TABLE II
ACCURACY RESULTS

Bit width and hidden layer size	Accuracy(%)
3 bits with 16 neurons	82.48
3 bits with 32 neurons	82.55
3 bits with 64 neurons	90.28
3 bits with 128 neurons	93.66
3 bits with 256 neurons	93.78
3 bits with 10240 neurons	95.88
3 bits with 102400 neurons	96.60
7 bits with 16 neurons	74.74
7 bits with 32 neurons	90.30
7 bits with 64 neurons	90.08
7 bits with 128 neurons	94.58
7 bits with 256 neurons	94.36
7 bits with 10240 neurons	96.58
7 bits with 102400 neurons	96.20
32 bits with 16 neurons	85.48
32 bits with 32 neurons	86.42
32 bits with 64 neurons	93.22
32 bits with 128 neurons	94.92
32 bits with 256 neurons	94.82
32 bits with 10240 neurons	96.36
32 bits with 102400 neurons	96.90

Combination of feature extraction algorithm and LogQuant achieves reasonable accurate recognition rate. Although Daisuke et al.'s work shows negligible or no loss in classification performance when LogQuant been applied to fully connected layers[1]. Our work reveals that small sized DNNs cannot maintain the accuracy merit of logarithmic quantization. After quantized, the network's recognition performance decreased substantially compared to the results in Table 1. There is a trade-off between accuracy and efficiency, especially on resource limited microchips.

IV. Conclusion and Future Works

Our work explores the practicability of implementing a small sized DNN on a microprocessor of extremely limited RAM and computational capability. Here we summarize the work flow of the entire processes targeting on embedded devices of limited resources. This should work on other datasets like fashion mnist, cifar10 or real world application datasets.

- Design a feature extraction algorithm which performs dimension reduction and preserves the relevant characteristics.
- Decide a network that meets the precision of target application. Shallow depth with hidden layer of considerable size is preferred by cause of the Law of Large Numbers.
- Logarithmic Quantization of model parameters.
- Export the model parameters to the target device(s).

During our experiments, we found that logarithmic quantization suffers from small sized layers. We set it as a future plan to mathematical model the performance deterioration of logarithmic quantization under certain circumstances and how to avoid it. Besides, we have yet to test how the quantization of inputs will affect the performance of the recognition accuracy. Since quantization on both ends can increase the efficiency, we will focusing on speed acceleration in the next stage.

References

[1] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. ”Convolutional neural networks using logarithmic data representation.” arXiv preprint arXiv:1603.01025 (2016).

[2] LeCun, Yann, Corinna Cortes, and C. J. Burges. ”MNIST handwritten digit database.” AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist 2 (2010).

[3] Iandola, Forrest, and Kurt Keutzer. ”Small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures.” Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion. ACM, 2017.

[4] Garg, Mamta, and Deepika Ahuja. ”A Novel Approach to Recognize the off-line Handwritten Numerals using MLP and SVM Classifiers.” International Journal Of Computer Science & Engineering Technology (IJCSET) ISSN (2013): 2229-3345.

[5] Han, Song, Huizi Mao, and William J. Dally. ”Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.” arXiv preprint arXiv:1510.00149 (2015).

[6] Patel, Ishani, Virag Jagtap, and Ompriya Kale. ”A Survey on Feature Extraction Methods for Handwritten Digits Recognition.” International Journal of Computer Applications 107.12 (2014).

[7] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[8] Tokui, S., Oono, K., Hido, S. and Clayton, J., Chainer: a Next-Generation Open Source Framework for Deep Learning, Proceedings of Workshop on Machine Learning Systems(LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), (2015).