Quantization error-based regularization for hardware-aware neural network training

Kazutoshi Hirose\(^1\), Ryota Uematsu\(^1\), Kota Ando\(^1\), Kodai Ueyoshi\(^1\), Masayuki Ikebe\(^1\), Tetsuya Asai\(^1\), Masato Motomura\(^1\), and Shinya Takamaeda-Yamazaki\(^1\)\(^{b)}\)

\(^1\) Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan

\(^{a)}\) hirose.kazutoshi.8m@ist.hokudai.ac.jp
\(^{b)}\) takamaeda@ist.hokudai.ac.jp

Received February 21, 2018; Revised April 27, 2018; Published October 1, 2018

Abstract: We propose “QER”, a novel regularization strategy for hardware-aware neural network training. Although quantized neural networks reduce computation power and resource consumption, it also degrades the accuracy due to quantization errors of the numerical representation, which are defined as differences between original numbers and quantized numbers. The QER solves such the problem by appending an additional regularization term based on quantization errors of weights to the loss function. The regularization term forces the quantization errors of weights to be reduced as well as the original loss. We evaluate our method by using MNIST on a simple neural network model. The evaluation results show that the proposed approach achieves higher accuracy than the standard training approach with quantized forward propagation.

Key Words: deep neural network, quantization, regularization

1. Introduction

Deep neural network (DNN) technology is widely used for various machine-learning applications, such as object detection [1, 2], three-dimensional (3D) semantic segmentation [3, 4], speech recognition [5], and translation [6]. While DNNs are very powerful, their computing resource demands are steadily growing. The state-of-the-art DNNs are growing in computational complexity and memory footprint. For instance, ResNet [7], which achieved a high classification accuracy value in the 2015 ILSVRC competition [8], contains 152 layers, whereas AlexNet [9] proposed in 2012 contains only eight layers. Subsequently, DenseNet [10], proposed in 2017, achieved higher accuracy but also has many parameters.

In server environments, high-performance but power-consuming GPUs are essential for both training
and inference. In contrast, recent IoT systems such as embedded devices do not allow the employment of such power-consuming accelerators, e.g., GPUs, owing to the cost and energy requirements. Thus, a hardware-focused approach that uses less energy and has a smaller memory footprint must be considered.

One of the hardware-aware DNN approaches is quantization, which replaces floating point representations with low complexity representations, such as a fixed point [11], binary [12], and logarithmic [13]. These reduce both computing complexity and memory footprint. For example, BinaryNet [12], which has binary weight and binary activation, no longer requires energy- and area-consuming multiplications. Instead, the multiplications are replaced with XNOR operations, which can be implemented by a very simple circuit such that energy consumption is dramatically reduced [14]. In the log domain, multiplications are replaced with additions according to the log rules. Therefore, it is suitable for low-power hardware [15].

The primary problem in quantization is accuracy degradation owing to its lower numerical representation [16]. With quantization, pre-learned weights in the full-precision representation are converted into quantized values. Therefore, quantization errors occur between the original and quantized values. Although a lower precision quantization can effectively reduce the computation complexity and memory footprint, it degrades the accuracy of the neural network. Thus, a trade-off generally exists between low numerical precision and accuracy.

We herein propose a quantization-error-aware training method for higher accuracy in quantized neural networks. Our method focuses on the quantization error at the training phase and introduces a novel regularization term based on the quantization error. Because quantization errors typically degrade the accuracy, the errors must be reduced at the training phase. The proposed method appends an additional regularization term calculated by quantization errors in weights to the loss function.

This paper is based on our previous work [17]. The following are the primary supplementary contributions of this paper: we considered using another regularization method based on the L1 norm, evaluated not only the weight-quantized neural networks but also the weight-and-activation-quantized neural network, and measured the convergence speed.

The rest of this paper is presented as follows: in section 2, we confirm the current quantized neural network. In section 3, we propose a regularization method. In section 4, we evaluate the proposed method using MNIST. Subsequently, in section 5, we present some related studies. Finally, we conclude this paper in section 6.

2. Quantized neural network

Quantization in neural network represents the original floating point values by using reduced information, such as fixed point and binary. A recent advanced technique in quantization uses a logarithmic representation to increase both the dynamic range and resolution of numerical values. In this work, we focus on logarithmic quantization, linear quantization, and binarization. We first present the general neural networks using floating point representation, and then the quantized neural networks.

The feed forwarding of general neural networks is calculated by repeating the following:

\[
u_l = x_{l-1} \cdot w_l \quad (1)
\]

\[
u_l = \text{BatchNormalization}(u_l) \quad (2)
\]

\[x_l = \phi(u_l) \quad (3)
\]

where \(w_l\) are the weights in the \(l\)-th layer, \(x_0\) are the input values, and \(x_l\) are the activation values. Batch normalization (BN) [18] is a method for high convergence speed and high accuracy. The recent neural network model often has BN. \(\phi(\cdot)\) is a nonlinearity activation function applied pointwise to the output of the batch normalization. ReLU is the most often used activation function.

\[
\text{ReLU}(x) = \begin{cases} 
x & \text{if } w \geq 0 \\
0 & \text{otherwise}
\end{cases}
\]

In a quantized neural network, both weights and activations are quantized. When it quantizes
activations, the quantization function is used as an activation function. First, logarithmic quantization is defined as follows:

$$w^q = \text{LogQuant}(w, \text{bitwidth}, V_{\text{max}}) = \text{Clip}(\text{AP}^2(w), V_{\text{min}}, V_{\text{max}})$$ (5)

$$V_{\text{min}} = V_{\text{max}} - (2^{\text{bitwidth}-1} - 1)$$ (6)

$$\text{AP}^2(w) = \text{sign}(w) \times 2^{\text{round}(|\log_2|w|)}$$ (7)

where $w$ is the original weight, $\text{bitwidth}$ is the bit width of the quantized weights that contain a sign bit, and $V_{\text{max}}$ and $V_{\text{min}}$ are the maximum and minimum scale ranges, respectively. $\text{AP}^2(\cdot)$ is the approximate power of two. Figure 1(a) shows the example of a logarithmic quantization. Next, we present the linear quantization. We use the following quantization method:

$$w^q = \text{LinearQuant}(w, \text{bitwidth}) = \left\lceil w \times \frac{V_{\text{max}}}{w_{\text{max}}} \right\rceil \times \frac{|w|_{\text{max}}}{V_{\text{max}}} \quad \text{max}$$ (8)

$$V_{\text{max}} = (2^{\text{bitwidth}-1} - 1)$$ (9)

Figure 1(b) shows the example of linear quantization. The binary quantization is defined as follows:

$$w^q = \text{Binarize}(w) = \text{sign}(w) = \begin{cases} +1 & \text{if } w \geq 0 \\ -1 & \text{otherwise} \end{cases}$$ (11)

The goal of training is to obtain the parameters that have less prediction errors of the network, via back propagation.

$$L(w) = E(w^q)$$ (12)

$$w^* = \arg \min_w L(w)$$ (13)

where $E(\cdot)$ is an error function such as the mean log-likelihood, and $L(\cdot)$ is the objective function. We herein apply these quantization methods to the parameters except the biases in all the layers.
However, quantization errors occur between the original and quantized values, as shown in Fig. 1(c), (d). Using parameters that contain quantization errors, we cannot obtain the correct update amounts. The update amount is obtained from the training that assumes quantized parameters. Therefore, the parameters can update correctly when they are their quantized values. Hence, it is important that the parameters have less quantization errors.

3. QER: Quantization-error-based regularization
Quantization errors (QE) have never been considered in the past approaches of quantized neural networks. We propose a quantization-error-aware training method that uses the QE as a regularization term. A QE is defined by the following expression:

\[ QE(w) = w - w^q \]  

where \( w^q \) is the quantized weight values whose expressions are represented by logarithmic quantization, linear quantization, or binarization. Subsequently, we define the quantization-error-based regularization (QER) term as follows:

\[
QER_2(w) = \|w - w^q\|_2
\]

\[
QER_1(w) = \|w - w^q\|_1
\]

These are based on the L2-norm and L1-norm. The QER term is appended to the objective function as follows:

\[
L(w) = E(w) + \lambda QER(w)
\]

where \( E(w) \) is the loss function and \( \lambda \) is the adaptive rate of QER. Thus, the weights are updated via the following optimization problem, as well as the general neural network training.

\[
w^* = \arg \min_w L(w)
\]

Weight decay using the L2-norm and L1-norm regularization is a common technique in neural network training for the generalization of weights to prevent overfitting [19]. These regularizations function to limit the weight divergence. In contrast, the QER forces the weights to be closer in values to their quantized values and the weights can obtain less quantization errors. By updating the weights to minimize \( L(w) \), both quantization errors and loss are gradually reduced.

Because our method is a training technology, we can use a trained network for the prediction without modifying the existing hardware architecture such as [14, 15]. Furthermore, the QER processing is scarce compared to the whole training step in the optimization code.

4. Experiments
We performed experiments on the proposed method to apply the QER in the neural network training. We compared the proposed method to the original quantized network without the QER in terms of accuracy and convergence. While we do not necessarily aim for the state-of-the-art performance, we endeavored to achieve the floating point network accuracy using the quantized network with QER. Algorithm 1 describes the training algorithm used in our experiments.

4.1 Weight-quantized network
First, we evaluated \( QER_2 \) on a weight-quantized network whose activation values are floating-point representations. By being in an ideal representation of activation, it is possible to evaluate only the effect of weight quantization and the effect of applying QER. The evaluations in this section are preliminary and were almost reported in our previous work [17]. The numerical representations of weights are logarithmic quantization and binarization. In the logarithmic quantization, all the weights were obtained by \( \text{LogQuant}(w, 4, 1) \). We did not use batch normalization to observe only the
Algorithm 1 Quantized network with QER

Require: a minibatch of inputs and targets \((x_0, x^*)\), previous weights \(w\), previous learning rate \(\eta^t\).
Ensure: updated weights \(w^{t+1}\), updated learning rate \(\eta^{t+1}\)

1. Forward propagation
   for \(l = 1 \text{ to } L\) do
      \(w^l_q \leftarrow \text{Quantize}(w_l)\)
      \(u_l \leftarrow x_{l-1} \cdot w_l\)
      \(u_l \leftarrow \text{BatchNormalize}(u_l)\)
      if \(l < L\) then
         \(x_l \leftarrow \text{Activate}(u_l)\)
      end if
   end for
   \(L \leftarrow E(w) + \lambda \text{QER}(w)\)

2. Backward propagation
   Compute \(\frac{\partial L}{\partial u_l}\) knowing \(u_L\) and \(x^*\)
   for \(l = L \text{ to } 1\) do
      \(\frac{\partial L}{\partial w_l} \leftarrow \frac{\partial L}{\partial u_l} \cdot w^q_l\)
      \(\frac{\partial L}{\partial w_l} \leftarrow \frac{\partial L}{\partial u_l}^T \cdot u^q_{l-1}\)
   end for

3. Accumulating the parameter gradients
   for \(l = 1 \text{ to } L\) do
      \(w^{l+1} \leftarrow \text{Update}(w_l, \eta^l, \frac{\partial L}{\partial w_l})\)
      \(\eta^{l+1} \leftarrow \alpha \eta^l\)
   end for

effects of QER. When the activations are floating-point representations, the accuracy can converge without batch normalization. We evaluate the relevance to batch normalization in the next section, which explains the case of quantized activation. The optimization solver is Adam [20], which is widely adopted in deep learning. The learning rate is 0.001. By trial and error, we set \(\lambda\) to start at 0.001, and be amplified by a factor 1.2 every 10 epochs. The used benchmarks are MNIST [21] and CIFAR-10 [22].

1. MNIST: The MNIST dataset [21] consists of 70000 28 \(\times\) 28 grayscale images representing digits ranging from 0 to 9, with 7000 images per digit. This dataset contains 60000 training images and 10000 test images. We did not use data augmentation.

2. CIFAR-10: The CIFAR-10 dataset [22] consists of 60000 32 \(\times\) 32 color images in 10 classes, with 6000 images per class. This dataset contains 50000 training images and 10000 test images. We did not use data augmentation.

For MNIST, we used a multilayer perceptron (MLP) that consists of two hidden layers (784-256-256-10 neurons). For CIFAR-10, we used a simple convolutional neural network (CNN) that consists of two convolution layers, two max-pooling layers, and three full-connection layers as follows:

C3-64, MP2, C3-64, MP2, FC4096-384, FC384-192, FC192-10
where C3-\(n\) is a 3 \(\times\) 3 convolution layer for \(n\)-channels, MP2 is a 2 \(\times\) 2 max-pooling layer, and FC is a fully connected layer.

Table I shows the results of the test classification accuracy. Figures 2, 3 shows their convergence graph. In both benchmarks and both cases of the logarithmic quantization and binarization, the proposed approach using QER achieves a higher accuracy rate. In particular, the logarithmic quantization with QER exceeds the baseline network trained and tested in floating point. Therefore, we can confirm that QER is effective in increasing the accuracy of the weight-quantized network.
\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
 & MNIST & CIFAR-10 \\
\hline
Float & 0.9777 & 0.6941 \\
LogQuantize (4bit) w/o QER & 0.9773 & 0.6844 \\
 & w/ QER & 0.9783 & 0.7031 \\
Binarize (1bit) w/o QER & 0.9664 & 0.6724 \\
 & w/ QER & 0.9709 & 0.6839 \\
\hline
\end{tabular}
\caption{Accuracy for weight-quantized network.}
\end{table}

\begin{figure}[h]
\centering
\subfloat[Logarithmic]{
\includegraphics[width=0.4\textwidth]{fig2a.png}
\label{fig:2a}}
\subfloat[Binary]{
\includegraphics[width=0.4\textwidth]{fig2b.png}
\label{fig:2b}}
\caption{Convergence of accuracy on MNIST.}
\end{figure}

\begin{figure}[h]
\centering
\subfloat[Logarithmic]{
\includegraphics[width=0.4\textwidth]{fig3a.png}
\label{fig:3a}}
\subfloat[Binary]{
\includegraphics[width=0.4\textwidth]{fig3b.png}
\label{fig:3b}}
\caption{Convergence of accuracy on CIFAR-10. The used numerical representations of weights are logarithmically obtained by $LogQuant(x, 4, 1)$.}
\end{figure}

\subsection{4.2 Weight-and-activation-quantized neural network}
Next, we expand QER to the weight-and-activation-quantized neural network. In an actual hardware, it is common to implement a network that quantizes not only the weight but also the activation. In this section, we demonstrate the QER effects even in practical situations. In neural network practices, various hyperparameters such as learning rate, network structure, and weight decay are used. The hyperparameters vary with the training algorithm and application. Hence, we need to obtain an optimal hyperparameter $\lambda$ by comparing the training results. Initially, we evaluated the optimal values $\lambda$ for QER$_2$ and QER$_1$, respectively, and confirmed that the weights actually moved to their quantized values. Subsequently, we tested various quantization methods using the obtained value $\lambda$ and verified the cases where the QER effect was observed. Finally, we described the relation with batch normalization and the relation with training speed.

\textbf{Adaptive rate} The adaptive rate $\lambda$ indicates the contribution of QER to the objective function. We trained the MNIST on the MLP while changing $\lambda$ from 1 to 0 to obtain an optimal value.
Fig. 4. MNIST accuracy on MLP using $QER_2$. The used numerical representations of weights and activations are 4-bit logarithmic.

Fig. 5. MNIST accuracy on MLP using $QER_1$. The used numerical representations of weights and activations are 4-bit logarithmic.

Fig. 6. Weight distributions. The upper row shows the normal training without QER; the bottom shows the training with QER. The vertical blue lines indicate the power-of-two values.

As an example, the used numerical representation is logarithmic, and the number of bits used for representing the parameters is four. The optimization solver is Adam [20] and the learning rate is 0.001. The result shows that the experiments using $QER_2$, $\lambda = 0.01$ attain the best prediction accuracy, as shown in Fig. 4. Similarly, Fig. 5 shows that $\lambda = 0.001$ achieves the best accuracy on the network using $QER_1$. In both cases of $QER_2$ and $QER_1$, higher accuracy is achieved compared to the standard approach. Our results show that QER also improves the prediction accuracy in the weight-and-activation-quantized neural network. For the following discussions, we use the $\lambda$ values obtained in the experiments above. It is noteworthy that lambda could not be optimal in all cases because the optimal hyperparameter depends on the training methods, as described above.

**Quantization error** QER forces the weights to be closer their quantized values. We confirmed the distribution of the weights that are trained using 4-bit logarithmic. As shown in Fig. 6, the
Table II. MNIST accuracy for MLP with batch normalization. Adaptive rate of $QER_2$ is $\lambda = 0.01$, and adaptive rate of $QER_1$ is $\lambda = 0.001$. The accuracy written in bold characters exceed the baseline accuracy whose representation is floating point.

| bit width | w/o $QER$ | w/ $QER_2$ | $\Delta$ | w/ $QER_1$ | $\Delta$ |
|-----------|-----------|------------|---------|-----------|---------|
| Float     | 32        | 0.9748     |         |           |         |
| Logarithmic | 5        | 0.9720     | **0.9843** | +0.0123 | **0.9996** | +0.0276 |
|           | 4        | 0.9680     | **0.9859** | +0.0179 | **0.9912** | +0.0192 |
|           | 3        | 0.9609     | 0.9303   | −0.0306 | 0.9250   | −0.053  |
| Linear    | 6        | 0.9674     | **0.9753** | +0.0079 | 0.7308   | −0.2366 |
|           | 5        | 0.9448     | 0.9481   | +0.0033 | 0.8871   | −0.0617 |
|           | 4        | 0.9703     | 0.9695   | −0.0008 | 0.8626   | −0.1077 |
| Binary    | 1        | 0.9248     | 0.7602   | −0.1645 | 0.2225   | −0.7023 |

Weights gradually gather in the power-of-two values as training progressed. It is known that there exist numerous suboptimal solutions to achieve a high accuracy in a neural network. However, the size of suboptimal solutions in a quantized neural network will be limited to a subset of the original neural network without quantization, because some of the suboptimal solutions in the original neural network are not allowed due to the quantization of numerical representation. Since weight values are updated based on the loss function in the neural network training without QER, some suboptimal solutions not suitable to the quantization tend to be explored. In contrast, using QER helps to efficiently survey a suitable suboptimal solution to achieve high accuracy in the quantized network. Therefore, it facilitated in improving the accuracy in quantized neural network.

$QER_2$ vs. $QER_1$ We defined $QER_2$ and $QER_1$, based on the L2 norm and L1 norm respectively. Since the update amount with $QER_2$ is calculated based on the $\lambda$ and the quantization error, tuning $\lambda$ is relatively easy. In contrast, the update amount of $QER_1$ is calculated directly by the $\lambda$. It is important to carefully decide the value of lambda because lambda for $QER_1$ is more sensitive than lambda for $QER_2$. An optimum value can be found in this experiment; however, it is not always found in the actual DNN tuning. Hence, our discussion is focused on $QER_2$, whose adaptive rate is tuned easily.

Quantization methods We evaluated QER using other quantization methods by comparing to the standard approach without QER. The evaluated numerical representation is logarithmic (5-bit, 4-bit, 3-bit), linear (6-bit, 5-bit, 4-bit), and binary (1-bit). Table II shows the results of the accuracy, and Figs. 7 and 8 show their convergences. In the results using $QER_2$, we observed that $QER_2$ increases the accuracy when the bit width is sufficient (logarithmic 5-bit, 4-bit, linear 6-bit, 5-bit). In particular, the 4-bit and 5-bit logarithmic, and the 6-bit linear with $QER_2$ exceeds the baseline network trained and tested in floating point. However, $QER_2$ reduces the accuracy of the network using the low bit width including binary. In these cases, training without QER cannot converge satisfactorily compared to floating-point training in Fig. 7(a). The baseline networks with low bit width has a large vibrational amplitude, as shown in Fig. 7(e), (h). As QER is a kind of disturbance for a loss function, it is considered to negatively affect the model if the model does not converge. Therefore, it is important that the training model can converge sufficiently using the sufficient bit width to express the values. Meanwhile, the quantization points are far apart if the bit width is low. The weights have converged to their quantized values completely such as approximately one on the bottom row in Fig. 6. Hence, it is revealed that training using QER in the low bit width is difficult. To achieve a high precision with a small bit width using QER, the adaptive rate must be decided dynamically such as in Adam, which computes individual adaptive learning rates for different
Fig. 7. Accuracy convergences on MLP network using $QER_2$. Here, BN means batch normalization. (a) is a baseline convergence trained in floating point. (c), (d), (f), (g) are the convergences that have good $QER_2$ effect with batch normalization. (b), (e), (h) are the convergences when a bad $QER_2$ effect occurs. In all cases, a convergence with higher accuracy gains a better $QER_2$ effect.

parameters from the estimates of the first and second moments of the gradients.

In the evaluation using $QER_1$, the representation of the 4-bit and 5-bit logarithmic achieved higher accuracies than the floating point network and the case of using $QER_2$. However, the prediction accuracy with linear quantization is reduced and it did not converge (Fig. 8(f), (g), (h)) despite having the sufficient bit width. The $\lambda$ for $QER_1$, which is obtained by training on the logarithmic quantization, is assumed an unsuitable value for the linear quantization because it is difficult to tune, as mentioned above. Therefore, $\lambda$ must be reset to a proper value suitable for the quantization method by comparing the training results.

**Batch normalization** The experiments above of section 4.2 used QER with batch normalization. Typically, training without batch normalization does not contribute the sufficient accuracy
Fig. 8. Accuracy convergences on MLP using QER for each numerical representation. (a) is a baseline convergence-trained floating point. (c), (d) are the convergences when a good QER effect occurs with batch normalization. (b), (e), (f), (g), (h) do not converge by QER. In these cases, the predetermined and common hyperparameter $\lambda$ seems to be unsuitable.

convergence when the activation is quantized. Therefore, to adjust the range of activation that is quantizing, it is common to use batch normalization in quantized neural networks. However, we conducted experiments Table III without batch normalization to evaluate the relevance between QER and the accuracy convergence. In particular, we discuss only the results of logarithmic quantization whose $\lambda$ is optimal because it has been decided by training on the logarithmic quantization. Table II and Table III show that better accuracy improvements can be provided using QER$_2$ and QER$_1$ in combination with batch normalization in the 4-bit and 5-bit logarithmic, respectively. Therefore, we can conclude that the hypothesis above that does not work with QER if the model does not converge sufficiently, is correct. QER is a technology to accentuate the model performance and is not a technology to improve models that cannot be trained properly. The quantized neural network is not perfect by only using the QER. Thus, a cooperative design between model definition and the hyperparameter is important.

Training speed QER is useful not only for accuracy improvement but also for training speed acceleration. As shown above, we confirmed that QER achieves a higher accuracy value compared
Table III. MNIST accuracy for MLP without batch normalization. Training conditions other than batch normalization are the same as in Table II.

| w/o/ batch normalization | bit width | w/o QER | w/ QER2 | Δ   | w/ QER1 | Δ   |
|--------------------------|-----------|---------|---------|-----|---------|-----|
| Float                    | 32        | 0.9538  |         |     |         |     |
| Logarithmic              | 5         | 0.9407  | 0.9424  | +0.0017 | 0.9203 | −0.0204 |
|                         | 4         | 0.9408  | 0.9408  | 0.0000 | 0.9216 | −0.0192 |
|                         | 3         | 0.9567  | 0.9283  | −0.0284 | 0.9300 | −0.0267 |
| Linear                   | 6         | 0.9196  | 0.9249  | +0.0053 | 0.4695 | −0.4501 |
|                         | 5         | 0.9155  | 0.9201  | +0.0046 | 0.2150 | −0.7005 |
|                         | 4         | 0.9372  | 0.9341  | −0.0031 | 0.1154 | −0.8218 |
| Binary                   | 1         | 0.1135  | 0.7356  | +0.6221 | 0.1135 | 0.0000  |

to the baseline model. In other words, QER reduced the number of training steps required to reach the maximum accuracy of the baseline model. For instance, the network using a 4-bit logarithmic representation without $QER_2$ reached the highest accuracy of 0.9680 within 48 epochs (Fig. 7(d)). In the case of using $QER_2$, it reached a 0.9680 accuracy with only four epochs. In our experiments, we achieved a baseline-top accuracy with 56.8% epoch on average when $QER_2$ is in effect.

5. Related works

Some quantization- or hardware-aware weight compression techniques have been proposed. Shin et al. proposed a weight-compression technique using a lookup table (LUT) [23]. Gysel et al. proposed a fine-tuning technique for hardware-oriented weight quantization [24]. These approaches aimed to optimize weights with high prediction accuracy for limited numerical representations. Our work differs from these approaches in terms of quantization error, but can be used simultaneously with these techniques.

Loss-aware binarization [25] uses a proximal Newton algorithm with diagonal Hessian approximation that directly minimizes the loss with respect to the binarized weights. Our work is similar to this work as it also focuses on the adverse effect of quantization. Because our approach aims to improve the prediction accuracy via the regularization term, it can be applied to quantization types other than binarization.

BitNet$^1$ [26] also utilizes quantization error for regularizing the bit width. This method controls the expressive power of the network by dynamically quantizing the range and set of values that the parameters can take. This method optimizes the bit width in each layer while reducing the quantization error. Because hardware such as embedded devices prefer a simple and monotonous architecture, we used the fixed bit width numbers.

6. Conclusion

We proposed a quantization-error-aware training method for hardware-oriented neural network implementations. Our approach appends on an additional regularization term, based on the quantization errors of weights, to the loss function. When the quantization bit width is sufficient to express distinct features, it achieved a higher accuracy value compared to the baseline quantized neural network. However, it requires the stable accuracy convergence of the baseline network to affect our approach. Further, a proper hyperparameter of the adaptive rate is required.

Future work includes considering an additional universal algorithm for various quantization methods or various bit widths without tuning the hyperparameters. Additionally, the proposed method can be applied to more complicated neural network structures and can be evaluated on larger datasets.

---

$^1$It is noteworthy that the paper of BitNet [26] has been uploaded to ArXiv after we submitted the previous paper to SGAI 2017.
Acknowledgments

This work was supported by JST ACCEL Grant Number JPMJAC1502, Japan.

References

[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A.C. Berg, “Ssd: Single shot multibox detector,” To appear. 2016.

[2] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517–6525, 2017.

[3] C.R. Qi, L. Yi, H. Su, and L.J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” In I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pp. 5105–5114. Curran Associates, Inc., 2017.

[4] S. Song, F. Yu, A. Zeng, A.X. Chang, M. Savva, and T. Funkhouser, “Semantic scene completion from a single depth image,” Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[5] Y. LeCun, Y. Bengio, and G. Hinton, Nature, 2016.

[6] M. Johnson, M. Schuster, Q.V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean, “Google’s multilingual neural machine translation system: enabling zero-Shot translation,” ArXiv e-prints, November 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei, “ImageNet large scale visual recognition challenge,” International Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015.

[9] A. Krizhevsky, I. Sutskever, and G.E. Hinton, “Imagenet classification with deep convolutional neural networks,” In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc., 2012.

[10] G. Huang, Z. Liu, L. van der Maaten, and K.Q. Weinberger, “Densely connected convolutional networks,” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[11] D.D. Lin, S.S. Talathi, and V.S. Annapureddy, “Fixed point quantization of deep convolutional networks,” ArXiv e-prints, November 2015.

[12] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1,” ArXiv e-prints, February 2016.

[13] D. Miyashita, E.H. Lee, and B. Murmann, “Convolutional neural networks using logarithmic data representation,” CoRR, abs/1603.01025, 2016.

[14] K. Ando, K. Oriro, K. Ueyoshi, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, T. Kuroda, and M. Motomura, “BRein Memory: A 13-layer 4.2 K neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos,” In 2017 IEEE Symposium on VLSI Circuits (VLSI-Circuits), pp. C24–C25, Kyoto, Japan, 2017.

[15] K. Ando, K. Oriro, K. Ueyoshi, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, T. Kuroda, and M. Motomura, “BRein Memory: A 13-layer 4.2 K neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos,” In 2017 IEEE Symposium on VLSI Circuits (VLSI-Circuits), pp. C24–C25, Kyoto, Japan, 2017.

[16] S. Hashemi, N. Anthony, H. Tann, R.I. Bahar, and S. Reda, “Understanding the impact of precision quantization on the accuracy and energy of neural networks,” In Design, Automation Test in Europe Conference Exhibition (DATE), 2017, pp. 1474–1479, March 2017.
[17] K. Hirose, K. Ando, K. Ueyoshi, M. Ikebe, T. Asai, M. Motomura, and S. Takamaeda-Yamazaki, “Quantization error-based regularization in neural networks,” In Max Bramer and Miltos Petridis, editors, Artificial Intelligence XXXIV, pp. 137–142, Cham, 2017. Springer International Publishing.

[18] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” In Francis R. Bach and David M. Blei, editors, ICML, volume 37 of JMLR Workshop and Conference Proceedings, pp. 448–456. JMLR.org, 2015.

[19] K. Janocha and W.M. Czarnecki, “On loss functions for deep neural networks in classification,” ArXiv e-prints, February 2017.

[20] D.P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” ArXiv e-prints, December 2014.

[21] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.

[22] A. Krizhevsky, V. Nair, and G. Hinton, Cifar-10 (canadian institute for advanced research).

[23] 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,” In 2017 IEEE International Solid-State Circuits Conference (ISSCC), pp. 240–241, February 2017.

[24] P. Gysel, M. Motamedi, and S. Ghiasi, “Hardware-oriented approximation of convolutional neural networks,” ArXiv e-prints, April 2016.

[25] L. Hou, Q. Yao, and J.T. Kwok, “Loss-aware binarization of deep networks,” ArXiv e-prints, November 2016.

[26] A. Raghavan, M.R. Amer, and S. Chai, “Bitnet: Bit-regularized deep neural networks,” CoRR, abs/1708.04788, 2017.