Post-training Quantization with Multiple Points: Mixed Precision without Mixed Precision

Xinghcao Liu[1], **Mao Ye**[1], **Dengyong Zhou**[2], **Qiang Liu** [1]

[1] The University of Texas at Austin [2] Google Brain

Abstract

We consider the post-training quantization problem, which discretizes the weights of pre-trained deep neural networks without re-training the model. We propose *multipoint quantization*, a quantization method that approximates a full-precision weight vector using a linear combination of multiple vectors of low-bit numbers; this is in contrast to typical quantization methods that approximate each weight using a single low precision number. Computationally, we construct the multipoint quantization with an efficient greedy selection procedure, and adaptively decides the number of low precision points on each quantized weight vector based on the error of its output. This allows us to achieve higher precision levels for important weights that greatly influence the outputs, yielding an "effect of mixed precision" but without physical mixed precision implementations (which requires specialized hardware accelerators (Wang et al. 2019)). Empirically, our method can be implemented by common operands, bringing almost no memory and computation overhead. We show that our method outperforms a range of state-of-the-art methods on ImageNet classification and it can be generalized to more challenging tasks like PASCAL VOC object detection.

Introduction

The past decade has witnessed the great success of deep neural networks (DNNs) in many fields. Nonetheless, DNNs require expensive computational resources and enormous storage space, making it difficult for deployment on resource-constrained devices, such as devices for Internet of Things (IoT), processors on smart phones, and embeded controllers in mobile robots (Howard et al. 2017; Xu et al. 2018).

Quantization is a promising method for creating more energy-efficient deep learning systems (Han, Mao, and Dally 2015; Hubara et al. 2017; Zmora et al. 2018; Cheng et al. 2018). By approximating real-valued weights and activations using low-bit numbers, quantized neural networks (QNNs) trained with state-of-the-art algorithms (e.g., Courbariaux, Bengio, and David 2015; Rastegari et al. 2016; Louizos et al. 2018; Li, Dong, and Wang 2019) can be shown to perform similarly as their full-precision counterparts (e.g., Jung et al. 2019; Li, Dong, and Wang 2019).

This work focuses on the problem of *post-training quantization*, which aims to generate a QNN from a pretrained full-precision network, without accessing the original training data (e.g., Sung, Shin, and Hwang 2015; Krishnamoorthi 2018; Zhao et al. 2019; Meller et al. 2019; Banner, Nahshan, and Soudry 2019; Nagel et al. 2019; Choukroun, Kravchik, and Kisilev 2019). This scenario appears widely in practice. For example, when a client wants to deploy a full-precision model provided by a machine learning service provider in low-precision, the client may have no access to the original training data due to privacy policy. In addition, compared with training QNNs from scratch, prost-training quantization is much more efficient computationally.

Mixed precision is a recent advanced technology to boost the performance of QNNs (Wang et al. 2019; Banner, Nahshan, and Soudry 2019; Gong et al. 2019; Dong et al. 2019). The idea is to assign more bits to important layers (or channels) and less bits to unimportant layers/channels to better control the overall quantization error and balance the accuracy and cost more efficiently. The difficulty, however, is that current mixed precision methods require specialized hardware (e.g., Wang et al. 2019). Most commodity hardware do not support efficient mixed precision computation (e.g. due to chip area constraints (Horowitz 2014)). This makes it difficult to implement mixed precision in practice, despite that it is highly desirable.

In this paper, we propose *multipoint quantization* for post-training quantization, which can achieve the flexibility similar to mixed precision, but uses only a single precision level. The idea is to approximate a full-precision weight vector by a linear combination of multiple low-bit vectors. This allows us to use a larger number of low-bit vectors to approximate the weights of more important channels, while use less points to approximate the insensitive channels. It enables a flexible trade-off between accuracy and cost at a per-channel basis, while using only a single precision level. Because it does not require physical mixed precision implementation, our method can be easily deployed on commodity hardware by common operands.

We propose a greedy algorithm to iteratively find the optimal low-bit vectors to minimize the approximation error. The algorithm sequentially adds the low-bit vector that yields largest improvement on the error, until a stopping criterion is met. We develop a theoretical analysis, showing that the error decays exponentially with the number of low-bit vectors used. The fast decay of the greedy algorithm ensures small overhead after adding these additional points.

Our multipoint quantization is computationally efficient. The key advantage is that it only involves multiply-accumulate (MAC) operations during inference, which has been highly optimized in normal deep learning devices. We adaptively decide the number of low precision points for each channel by measuring its output error. Empirically, we find that there are only a small number of channels that require a large number of points. By applying multipoint quantization on these channels, the performance of the QNN is improved significantly without any training or fine-tuning. Empirically, it only brings a negligible increase of memory cost.

We conduct experiments on ImageNet classification with different neural architectures. Our method performs favorably against the state-of-the-art methods. It even outperforms the method proposed by Banner, Nahshan, and Soudry (2019) in accuracy, which exploits physical mixed precision. We also verify the generalizability of our approach by applying it to PASCAL VOC object detection tasks.

Related Works

Quantized neural networks has made significant progress with training (Courbariaux, Bengio, and David 2015; Han, Mao, and Dally 2015; Zhu et al. 2016; Rastegari et al. 2016; Mishra et al. 2017; Zmora et al. 2018; Cheng et al. 2018; Krishnamoorthi 2018; Li, Dong, and Wang 2019). The research of post-training quantization is conducted for scenarios when training is not available (Krishnamoorthi 2018; Meller et al. 2019; Banner, Nahshan, and Soudry 2019; Zhao et al. 2019). Hardware-aware Automated Quantization (Wang et al. 2019) is a pioneering work to apply mixed precision to improve the accuracy of QNN, which needs fine-tuning the network. It inspired a line of research of training a mixed precision QNN (Gong et al. 2019; Dong et al. 2019). Banner, Nahshan, and Soudry (2019) first exploits mixed precision to enhance the performance of post-training quantization.

Using multiple binary filters to approximate a full-precision filter has been investigated in supervised learning of binary neural networks (Guo et al. 2017; Lin, Zhao, and Pan 2017; Zhuang et al. 2019). Our work differs from these works in two dimensions: (1) Instead of supervised learning, our work focuses on post-training quantization, where only a small fraction of data can be used. (2) Our work goes beyond binary case. Our method and theory are applicable for quantization with arbitrary bits. This is non-trivial since, with higher bits, the weights are not simply ± 1, leading to a much more difficult optimization problem.

Method

We start by introducing the background on post-training quantization. Then we discuss the main framework of multipoint quantization, its application to deep neural networks, and its implementation overhead.

Preliminaries: Post-trainig Quantization

Given a pretrained full-precision neural network f, the goal of post-training quantization is to generate a quantized neural network (QNN) $\tilde{f}$ with high performance. We assume the full training dataset of f is unavailable, but there is a small *calibration dataset* $D = \{\boldsymbol{x}_i\}_{i=1}^N$, where N is a very small size, e.g. $N = 256$. The calibration set is used only for choosing a small number of hyperparameters of our algorithm, and we can not directly train $\tilde{f}$ on it because it is too small and would cause overfitting.

The b-bit linear quantization amounts to approximate real numbers using the following quantization set $\mathcal{Q}$,

$$\mathcal{Q} = K \times [-1 : \epsilon_b : 1] \, + \, B, \quad \epsilon_b := \frac{1}{2^{b-1} - 1}, \tag{1}$$

where $[-1 : \epsilon_b : 1]$ denotes the uniform grid on $[-1, 1]$ with increment ϵ_b between elements, and $K > 0$ is a scaling factor that controls the length of $\mathcal{Q}$ and B specifies center of $\mathcal{Q}$.

Then we map a floating number t to $\mathcal{Q}$ by,

$$\tilde{t} = [t]_{\mathcal{Q}} := \underset{z \in \mathcal{Q}}{\arg\min} \, |t - z|, \tag{2}$$

where $[\cdot]_{\mathcal{Q}}$ denotes the nearest rounding operator w.r.t. $\mathcal{Q}$. For a real vector $\boldsymbol{t} = (t_1, t_2, \ldots, t_d) \in \mathbb{R}^d$, we map it to $\mathcal{Q}^d$ by,

$$\tilde{\boldsymbol{t}} = [\boldsymbol{t}]_{\mathcal{Q}} = ([t_1]_{\mathcal{Q}}, [t_2]_{\mathcal{Q}}, \ldots, [t_d]_{\mathcal{Q}}). \tag{3}$$

Further, $[\cdot]_{\mathcal{Q}}$ can be generalized to higher dimensional tensors by first stretching them to one-dimensional vectors then applying Eq. 3.

Since all the values are larger than K (or smaller than $-K$) will be clipped, K is also called the *clipping factor*. Supposing $\mathcal{Q}$ is used to quantize vector $\boldsymbol{t}$, a naive choice of K is the element with the maximal absolute value in $\boldsymbol{t}$. In this case, no element will be clipped. However, because the weights in a layer/channel of a neural network empirically follows a bell-shaped distribution, properly shrinking K can boost the performance. Different clipping methods have been proposed to optimize K (Zhao et al. 2019).

There are two common configurations for post-training quantization, *per-layer quantization* and *per-channel quantization*. Per-layer quantization assigns the same K and B for all the weights in the same layer. Per-channel quantization is more fine-grained, and it uses different K and B for different channels. The latter can achieve higher precision, but it also requires more complicated hardware design.

Multipoint Quantization and Optimization

We propose *multipoint quantization*, which can be implemented with common operands on commodity hardware.

Consider a linear layer in a neural network, which is either a fully-connected (FC) layer or a convolutional layer. The weight of a channel is a vector for FC layer, or a convolution kernel for convolutional layer. For simplicity, we only introduce the case of FC layer in this section. It can be easily generalized to convolutional layers. Supposing the input to this layer is d-dimensional , then the real-valued weight of a channel can be denoted as $\boldsymbol{w} = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$. Multipoint quantization approximates $\boldsymbol{w}$ with a weighted sum of a set of low precision weight vectors,

$$\tilde{\boldsymbol{w}} = \sum_{i=1}^{n} a_i \tilde{\boldsymbol{w}}_i, \tag{4}$$

Algorithm 1 Optimization of Problem 5

```
1: Input: weight $\boldsymbol{w}$, integer $n$, maximal step size for grid
   search $\eta$, a fixed quantization set $\mathcal{Q}$.
2: Initialize the residual $\boldsymbol{r}_1 = \boldsymbol{w}$.
3: for $i = 1 : n$ do
4:     Compute $\Delta_{\boldsymbol{r}_i}$, the minimal gap of $\boldsymbol{r}_i$, as definition 1.
5:     Set step size $\gamma$ and search range $I$ as Eq. 11.
6:     Solve Eq. 9 for $a_i^*$ by grid search in $I$ with step size $\gamma$.
7:     Set $\boldsymbol{w}_i^* = \left[\frac{\boldsymbol{r}_i}{a_i^*}\right]_{\mathcal{Q}}$, $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_i^* \boldsymbol{w}_i^*$.
8: end for
9: Return $\{a_i^*, \tilde{\boldsymbol{w}}_i^*\}_{i=1}^n$.
```

where $a_i \in \mathbb{R}$ and $\tilde{\boldsymbol{w}}_i \in \mathcal{Q}^d$ for $\forall i = 1, \ldots, n$. Multipoint quantization allows more freedom in representing the full-precision weight. Naive quantization approximates a weight by the nearest grid points, while multipoint quantization approximates it with the nearest point on the linear segments. If we release the constraint $a_1 + a_2 = 1$, we can actually represent every point on the 2-dimensional planar with multipoint quantization.

Given a fixed n, we want to find optimal $\{a_i^*, \tilde{\boldsymbol{w}}_i{}^*\}_{i=1}^n$ that minimizes the ℓ_2-norm between the real-valued weight and the weighted sum,

$$\{a_i^*, \tilde{\boldsymbol{w}}_i{}^*\}_{i=1}^n = \underset{\{a_i, \tilde{\boldsymbol{w}}_i\}_{i=1}^n}{\arg\min} \left|\left| \boldsymbol{w} - \sum_{i=1}^n a_i \tilde{\boldsymbol{w}}_i \right|\right|. \quad (5)$$

Problem 5 yields a difficult combinatorial optimization. We are able to get exact approximation when $n = d + 1$ by taking $a_i = w_i$ and $\tilde{\boldsymbol{w}}_i = one\_hot(i)$, where $one\_hot(i)$ is a one hot vector with the i-th element as 1 and other elements as 0. However, d is always large in deep neural networks, and our goal is to approximate $\boldsymbol{w}$ with a small enough n. Hence, we propose an efficient greedy method for solving it, which sequentially adds the best pairs $(a_i, \tilde{\boldsymbol{w}}_i)$ one by one. Specifically, we obtain the i-th pair $(a_i, \tilde{\boldsymbol{w}}_i)$ by approximate the residual from the previous pairs,

$$(a_i^*, \tilde{\boldsymbol{w}}_i^*) = \underset{a,\, \tilde{\boldsymbol{w}}}{\arg\min} ||\boldsymbol{r}_i - a\tilde{\boldsymbol{w}}|| \quad (6)$$

where $\boldsymbol{r}_i$ is the residual from the first $i - 1$ pairs,

$$\boldsymbol{r}_1 = \boldsymbol{w}; \;\; \boldsymbol{r}_i = \boldsymbol{w} - \sum_{j=1}^{i-1} a_j^* \tilde{\boldsymbol{w}}_j^*, \quad \forall\, i = 2, \ldots, n \quad (7)$$

For a fixed a, we have,

$$\begin{aligned} \tilde{\boldsymbol{w}}_i^*(a) &= \underset{\tilde{\boldsymbol{w}}}{\arg\min} ||\boldsymbol{r}_i - a\tilde{\boldsymbol{w}}|| \\ &= \underset{\tilde{\boldsymbol{w}}}{\arg\min} \left|\left| \frac{\boldsymbol{r}_i}{a} - \tilde{\boldsymbol{w}} \right|\right| = \left[\frac{\boldsymbol{r}_i}{a}\right]_{\mathcal{Q}}. \end{aligned} \quad (8)$$

Now we only need to solve optimal a,

$$a_i^* = \underset{a}{\arg\min} \left|\left| \boldsymbol{r}_i - a \left[\frac{\boldsymbol{r}_i}{a}\right]_{\mathcal{Q}} \right|\right|. \quad (9)$$

Because $[\cdot]_{\mathcal{Q}}$ is not differentiable, it is hard to optimize a by gradient descent. Instead, we adopt grid search to find a_i^* efficiently. Once the optimal a_i^* is found, the corresponding $\tilde{\boldsymbol{w}}_i^*$ is,

$$\tilde{\boldsymbol{w}}_i^* = \left[\frac{\boldsymbol{r}_i}{a_i^*}\right]_{\mathcal{Q}}. \quad (10)$$

Choice of Parameters for Grid Searching a_i^*: Grid search enumerates all the values from set $[I_{min} : \gamma : I_{max}]$, and selects the value that achieves the lowest error. The parameters of grid search, search range and step size, are defined as the interval $I = [I_{min}, I_{max}]$ and the increment γ respectively. The choice of search range I and step size γ are critical. We first define minimal gap of vector , and then give the choice of search range and step size.

The minimal gap is the minimal distance between two elements in a vector $\boldsymbol{t}$. It restricts the maximal value of step size.

Definition 1 *Given vector $\boldsymbol{t} = (t_1, \ldots, t_d) \in \mathbb{R}^d$, the* minimal gap *of $\boldsymbol{t}$ is,*

$$\Delta_{\boldsymbol{t}} = \min_{i,j} \frac{|\,|t_i| - |t_j|\,|}{2}, \quad \textit{s.t. } i, j \in \{1, 2, \ldots, d\} \textit{ and } t_i \neq t_j.$$

Then we propose the following choice of I and γ,

$$I = [0\;,\; 2(2^{b-1} - 1)||\boldsymbol{r}_i||]; \;\; \gamma = \min(\frac{\Delta_{\boldsymbol{r}_i}}{2^{b-1} - 1}, \eta), \quad (11)$$

where η is a predefined maximal step size to accelerate convergence. In Sec. , we show that by choosing I and γ like this, our algorithm is guaranteed to converge to zero. As n increases, the dimension of the approximation set increases. Intuitively, the nearest distance from an arbitrary point to the approximation set decreases exponentially with n. We rigorously prove that the greedy algorithm decays in an exponential rate in Sec. . Algorithm 1 recaptures the optimization procedure.

Multipoint Quantization on Deep Networks

We describe how to apply multipoint quantization to deep neural networks. Using multipoint quantization can decrease the quantization error of a channel significantly, but every additional quantized filter requires additional memory and computation consumption. Therefore, to apply it to deep networks, we must select the important channels to compensate for their quantization error with multipoint quantization.

For a layer L with d-dimensional input, we adopt a simple criterion, output error, to determine the target channels. Output error is the difference of the output of a channel before and after quantization. Suppose the weight of a channel is $\boldsymbol{w}$, its output error is defined as,

$$e(\boldsymbol{w}, \tilde{\boldsymbol{w}}, D_L) = \mathbb{E}_{\boldsymbol{x} \sim D_L} ||\boldsymbol{w}^\top \boldsymbol{x} - \tilde{\boldsymbol{w}}^\top \boldsymbol{x}||_2^2, \quad (12)$$

where D_L is the input batch to L, collected by running forward pass of f with calibration set D. Our goal is to keep the output of each channel invariant. If $e(\boldsymbol{w}, \tilde{w}, D_L)$ is larger than a predefined threshold ϵ, we apply multipoint quantization to this channel and increase n until $e(\boldsymbol{w}, \tilde{\boldsymbol{w}}, D_L) < \epsilon$.

Algorithm 2 Generate QNN with Multipoint Quantization

```
1: Input: A full-precision network $f$, a predefined threshold $\epsilon$, a calibration set of data points $D = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$.
2: Run forward pass of $f$ with calibration set $D = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ to get the input batch $D_L = \{\boldsymbol{x}_L^{(i)}\}_{i=1}^N$ for each layer $L$ in $f$,
3: for each layer $L$ in $f$ do
4:     for each channel $k$ in layer $L$ do
5:         $\tilde{\boldsymbol{w}}_k \leftarrow [\boldsymbol{w}_k]_Q$.
6:         if $e(\boldsymbol{w}_k, \tilde{\boldsymbol{w}}_k, D_L) > \epsilon$ then
7:             Apply multipoint quantization with Algorithm 1 and keep increasing $n$ until $e(\boldsymbol{w}_k, \tilde{\boldsymbol{w}}_k, D_L) < \epsilon$. Get $\{a_i^*, \tilde{\boldsymbol{w}}_i{}^*\}_{i=1}^n$.
8:             $\tilde{\boldsymbol{w}}_k \leftarrow \sum_{i=1}^n a_i^* \tilde{\boldsymbol{w}}_i{}^*$
9:         end if
10:    end for
11: end for
12: Return QNN $\tilde{f}$
```

A similar idea is leveraged to determine the optimal clipping factor K^*,

$$K^* = \arg\min_K \sum_{\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{\boldsymbol{x}\sim D_L} ||\boldsymbol{w}^\top \boldsymbol{x} - \tilde{\boldsymbol{w}}^\top \boldsymbol{x}||_2^2. \tag{13}$$

Here, $\mathcal{W}$ is the set of weights sharing the same K. For per-layer quantization, $\mathcal{W}$ is contains the weights of all the channels in a layer. For per-channel quantization, $\mathcal{W}$ contains only one element, which is the weight of a channel.

Analysis of Overhead

We introduce how the computation of dot product can be implemented with common operands when adopting multipoint quantization. Then we analyze the overhead of memory and computation.

For $d-$dimensional input and weight with N bits, computing the dot product requires d multiplications between two $N-$bit integers. The result of the dot product is stored in a 32-bit accumulator, since the sum of the individual products could be more than N bits. The above operation is called *Multiply-Accumulate (MAC)*, which has been highly optimized in modern deep learning hardware (Chen et al. 2016). The 32-bit integer is then quantized according to the quantization scheme of the output part.

Now we delve into the computation pipeline when $\tilde{\boldsymbol{w}} = \sum_{i=1}^n a_i \tilde{\boldsymbol{w}}_i$. Because $a_i \in \mathbb{R}$, we transform them to a hardware-friendly integer representation beforehand,

$$a_i \approx \frac{A_i}{2^p}, A_i = [2^p \times a_i] \tag{14}$$

Here, p determines the precision of the quantized a_i. We use the same p for all the weights with multipoint quantization in the network. A_i are 32-bit integers. The quantization of a_i can be performed off-line before deploying the QNN. We point out that,

$$\begin{aligned}\tilde{\boldsymbol{w}}^\top \boldsymbol{x} &= a_1 \tilde{\boldsymbol{w}}_1^\top \boldsymbol{x} + \cdots + a_n \tilde{\boldsymbol{w}}_n^\top \boldsymbol{x} \\ &\approx \frac{A_1 \tilde{\boldsymbol{w}}_1^\top \boldsymbol{x} + \cdots + A_n \tilde{\boldsymbol{w}}_n^\top \boldsymbol{x}}{2^p}\end{aligned} \tag{15}$$

Method	Memory	MULs	ADDs
Naive	dN	dN^2	$(d-1)N$
Multipoint	$ndN + 32n$	$n(dN^2 + 32^2)$	$n(d-1)N+$ $32(n-1)$

Table 1: Comparison of memory and computation consumption between a naively quantized layer and a layer using multipoint quantization.

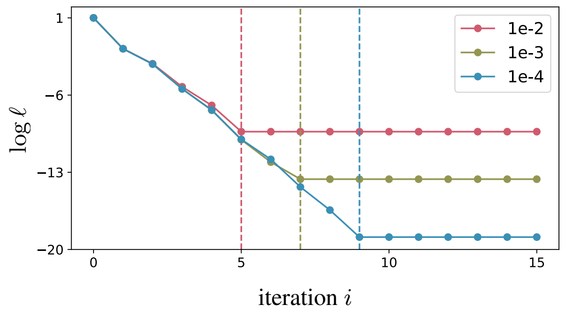

Figure 1: A toy experiment where $\boldsymbol{w}$ is randomly generated. The y-axis refers to $\log \ell$. We test three different step sizes γ for grid search. As ℓ gets smaller, $||\boldsymbol{r}||$ approaches zero and $\Delta_{\boldsymbol{r}}$ also becomes smaller. The dashed line indicates the step that $\gamma > \Delta_{\boldsymbol{r}}$ for the first time. Before the dashed line, ℓ decays exponentially. After the dashed line, the grid search does not have enough precision and thus the residual can no more be further reduced.

We divide the computation into three steps.

Step 1: Matrix Multiplication In the first step, we compute $(\tilde{\boldsymbol{w}}_1{}^\top \boldsymbol{x}, \ldots, \tilde{\boldsymbol{w}}_n{}^\top \boldsymbol{x})$. The results are stored in the 32-bit accumulators.

Step 2: Coefficient Multiplication & Summation The second step first multiplies A_i with $\tilde{\boldsymbol{w}}_i^\top \boldsymbol{x}$, containing n times of multiplication between two 32-bit integers. Then we sum $A_i \tilde{\boldsymbol{w}}_i^\top \boldsymbol{x}$ together with $n-1$ times of addition.

Step 3: Bit Shift Finally, the division with 2^p can be efficiently implemented by shifting p bits of $\sum_{i=1}^n A_i \tilde{\boldsymbol{w}}_i^\top \boldsymbol{x}$ to the left. We ignore the computation overhead in this step.

Overall Storage/ Computation Overhead: We count the number of binary operations following the same bit-op computation strategy as Li, Dong, and Wang (2019); Zhou et al. (2016). The multiplication between two N-bit integer costs N^2 binary operations. Suppose we have a weight vector $\boldsymbol{w} \in \mathbb{R}^d$. We compare the memory cost and the computational cost (dot product with $N-$bit input $\boldsymbol{x}$) between naive quantization $\tilde{\boldsymbol{w}} = [\boldsymbol{w}]_Q$ and multipoint quantization $\sum_{i=1}^n a_i \tilde{\boldsymbol{w}}_i$. The results are summarized in Table 1. Because d is always large in neural networks, so the memory and the computation overhead is approximately proportional to the number n.

Theoretical Analysis

In this section, we give a convergence analysis of the proposed optimization procedure. We prove the quantizataion error of the proposed greedy optimization decays exponentially w.r.t. the number of points.

Suppose that we want to quantize a real-valued d-dimensional weight $\boldsymbol{w} \in \mathbb{R}^d$. For simplicity, we assume a binary precision $b = 2$ in this section, which leads to $\mathcal{Q} = [-1, 0, 1]$. Our proof can be generalized to $b > 2$ easily. We follow the notations in Section . At the i-th iteration, the residual $\boldsymbol{r}_i$, a_i^* and $\tilde{\boldsymbol{w}}_i^*$ are defined by Eq. (7), Eq. (9) and Eq. (10), respectively. The minimal gap of a vector $\boldsymbol{t}$, $\Delta_{\boldsymbol{t}}$, is defined in Definition (1).

Let the loss function be $\ell(\boldsymbol{r}, a, \boldsymbol{w}) = ||\boldsymbol{r} - a\boldsymbol{w}||$. Now we can prove the following rate under mild assumptions.

Theorem 1 (Exponential Decay) *Suppose that at the i-th iteration of the algorithm, a_i^* is obtained by grid searching a from the range $(0 : \gamma_i : 2(2^{2-1} - 1)||\boldsymbol{r}_i||]$, where γ_i is the step size of the grid search. Assume that $\gamma_i \leq \min(\Delta_{\boldsymbol{r}_i}, \eta)$ for any step i before termination, where η is a predefined maximal step size. We have*

$$\ell(\boldsymbol{r}_i, a_i^*, \boldsymbol{w}_i^*) = \mathcal{O}\left(\exp\left(-ci\right) + \eta\right),$$

for some constant $c > 0$.

The proof is in Appendix A. Note that η is usually much smaller than the exponential term and thus can be ignored. Theorem 1 suggests that if we use sufficiently small step size ($\gamma_i \leq \min(\Delta_{\boldsymbol{r}_i}, \eta)$) for the optimization, the loss will decrease exponentially. Because of the exponentially fast decay of the algorithm, we find that $n \leq 2$ for most of the channels using multipoint quantization in practice. Fig. 1 justifies our theoretical analysis by a toy experiment.

Experiments

We evaluate our method on two tasks, ImageNet classification (Krizhevsky, Sutskever, and Hinton 2012) and PASCAL VOC object detection (Everingham et al. 2007). Our evaluation contains various neural networks.

Experiment Results on ImageNet Benchmark

We evaluate our method on the ImageNet classification benchmark. For fair comparison, we use the pretrained models provided by PyTorch [1] as others (Zhao et al. 2019; Banner, Nahshan, and Soudry 2019). We take 256 images from the training set as the calibration set. Calibration set is used to quantize activations and choose the channels to perform multipoint quantization. To improve the performance of low-bit activation quantization, we pick the optimal clipping factor for activations by minimizing the mean square error (Sung, Shin, and Hwang 2015). Like previous works, the weights of the first and the last layer are always quantized to 8-bit (Nahshan et al. 2019; Li, Dong, and Wang 2019; Banner, Nahshan, and Soudry 2019). For all experiments, we set the maximal step size for grid search in Eq. 9 to $\eta = \frac{1}{2^{10}}$.

We report both model size and number of operations under different bit-width settings for all the methods. The first and the last layer are not counted. We follow the same bit-op computation strategy as Li, Dong, and Wang (2019); Zhou et al. (2016) to count the number of binary operations. One OP is defined as one multiplication between an 8-bit weight and an 8-bit activation, which takes 64 binary operations. The multiplication between a m-bit and a n-bit integer is counted as $\frac{mn}{64}$ OPs.

[1]https://pytorch.org/

We provide two categories of results here: per-layer quantization and per-channel quantization. In per-layer quantization, all the channels in a layer exploit the same K and B. In per-channel quantization, each channel has its own parameter K and B. For both settings, we test six different networks in our experiments, including VGG-19 with BN (Simonyan and Zisserman 2014), ResNet-18, ResNet-101, WideResNet-50 (He et al. 2016), Inception-v3 (Szegedy et al. 2015) and MobileNet-v2 (Sandler et al. 2018).

Per-layer Quantization For per-layer quantization, we compare our method with a state-of-the-art (SOTA) baseline, Outlier Channel Splitting (OCS) (Zhao et al. 2019). OCS duplicates the channel with the maximal absolute value and halves it to mitigate the quantization error. For fair comparison, we choose the best clipping method among four methods for OCS according to their paper (Sung, Shin, and Hwang 2015; Migacz 2017; Banner, Nahshan, and Soudry 2019). We select the threshold ϵ such that the OPs of the QNN with multipoint quantization is about 1.15 times of the naive QNN. For fair comparison, we expand the network with OCS until it has similar OPs with the QNN using multipoint quantization. The results without multipoint quantization (denoted 'w/o Multipoint' in Table. 2) serve as another baseline. We quantize the activations and the weights to the same precision as the baselines. Experiment results are presented in Table. 2. It shows that our method obtains consistently significant gain on all the models compared with 'w/o Multipoint', with little increase on memory overhead. Our method also consistently outperforms the performance of OCS under any computational constraint. Especially, on ResNet-18, ResNet-101 and Inception-v3, our method surpasses OCS by more than 2% Top-1 accuracy. OCS cannot quantize MobileNet-v2 due to the group convolution layers, while our method nearly recovers the full-precision accuracy. Our method achieves similar performance with Data Free Quantization (Nagel et al. 2019) (71.19% Top-1 accuracy with 8-bit MobileNet-v2), which focuses on 8-bit quantization on MobileNets only. Note that this method is orthogonal to ours and we expect to obtain more improvement by combining with it.

Per-channel Quantization For per-channel quantization, we compare our method with another SOTA baseline, Banner, Nahshan, and Soudry (2019). Banner, Nahshan, and Soudry (2019) requires physical per-channel mixed precision computation since it assigns different bits to different channels. We denote it as 'Mixed Precision (MP)'. All networks are quantized with asymmetric per-channel quantization ($B \neq 0$). Since per-channel quantization has higher precision, weight clipping is not performed for naive quantization, which means that $K = \max(|\boldsymbol{w}|)$. We quantize both weights and activations to 4 bits. Experiment results are presented in Table. 3.

Our method outperforms MP on VGG19-BN and Inception-v3 even without weight clipping. After performing weight clipping with Eq. 13, our method beats MP on 5 out of 6 networks, except for ResNet-101. On VGG19-BN, Inception-v3 and MobileNet-v2, compared with MP, the Top-1 accuracy of our method after clipping is more than

Model	Bits (W/A)	Method	Acc (Top-1/Top-5) (%)	Size	OPs
VGG19-BN	32/32	Full-Precision	74.24/91.85	76.42MB	-
	4/8	w/o Multipoint	60.81/83.68	9.55MB	9.754G
		OCS (Zhao et al. 2019)	62.11/84.59	10.70MB	10.924G
		Ours	**64.06/86.14**	**9.59MB**	**10.923G**
ResNet-18	32/32	Full-Precision	69.76/89.08	42.56MB	-
	4/8	w/o Multipoint	54.04/78.10	5.32MB	847.78M
		OCS (Zhao et al. 2019)	58.05/81.57	6.20MB	988.51M
		Ours	**61.68/84.03**	**5.37MB**	**983.22M**
ResNet-101	32/32	Full-Precision	77.37/93.56	161.68MB	-
	4/8	w/o Multipoint	61.04/83.02	20.21MB	3.841G
		OCS (Zhao et al. 2019)	70.27/89.73	23.40MB	4.448G
		Ours	**73.09/91.34**	**20.86MB**	**4.446G**
WideResNet-50	32/32	Full-Precision	78.51/94.09	262.64MB	-
	4/8	w/o Multipoint	61.78/83.60	31.83MB	5.639G
		OCS (Zhao et al. 2019)	68.54/88.68	35.97MB	6.372G
		Ours	**70.47/89.43**	**32.08MB**	**6.365G**
Inception-v3	32/32	Full-Precision	77.45/93.56	82.96MB	-
	4/8	w/o Multipoint	5.17/12.85	10.37MB	2.846G
		OCS (Zhao et al. 2019)	8.49/17.75	12.16MB	3.338G
		Ours	**33.89/56.07**	**10.42MB**	**3.337G**
Mobilenet-v2	32/32	Full-Precision	71.78/90.19	8.36MB	-
	8/8	w/o Multipoint	0.06/0.15	2.090MB	299.49M
		OCS (Zhao et al. 2019)	N/A	N/A	N/A
		Ours	**70.70/89.70**	**2.091MB**	**357.29M**

Table 2: Per-layer quantization on ImageNet Benchmark (W=Weight, A=Activation, M=10^6, G=10^9, Acc=Accuracy). Bold refers to the method with highest Top-1 accuracy. Note that OCS cannot be applied to MobileNet-V2 because it cannot deal with group convolution.

2% higher. In the experiments, all the memory overhead is smaller than 5% and the computation overhead is no more than 17% compared with the naive QNN.

Experiment Results on PASCAL VOC Object Detection Benchmark

We test Single Shot MultiBox Object Detector (SSD), which is a well-known object detection framework. We use an open-source implementation [2]. The backbone network is VGG16. We apply per-layer quantization and per-channel quantization on all the layers, excluding localization layers and classification layers. Due to the GPU memory constraint, the calibration set only contains 6 images. We measure the mean average precision (mAP), size and OPs of the quantized model. We perform activation clipping and weight clipping for both settings.

In per-layer quantization, our method increases the performance of the baseline by over 1% mAP (72.86% $\longrightarrow$ 74.10%). When weight is quantized to 3-bit, our method boost the baseline by 4.38% mAP (42.56% $\longrightarrow$ 46.94%) with little memory overhead of 0.01MB. Our method also performs well in per-channel quantization. It improves the baseline by 0.41% mAP for 4-bit quantization and 1.09% mAP for 3-bit quantization. Generally, our method performs better when the bit width goes smaller.

[2]https://github.com/amdegroot/ssd.pytorch

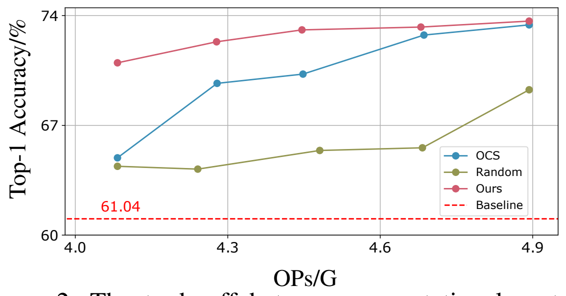

Figure 2: The trade-off between computational cost and performance of a quantized ResNet-101 (W4A8). 'Baseline' is the naive QNN without multipoint quantization (OPs=3.841G). 'Random' uses multipoint quantization but channels are randomly added, while our method adds channels according to their output error.

Analysis of the Algorithm

We provide a case study of ResNet-101 under per-layer quantization to analyze the algorithm. More results can be found in the appendix.

Computation Overhead and Performance: Fig. 2 demonstrates how the performance of different methods changes as the computational cost changes. Our method obtains huge gain with only a little overhead. OCS cannot perform comparably with our method at the beginning, but it catches up when the computational cost is large enough. The performance of 'Random' is consistently the worst among

Model	Bits (W/A)	Method	Acc (Top-1/Top-5) (%)	Size	OPs
VGG19-BN	32/32	Full-Precision	74.24/91.85	76.42MB	-
	4/4	w/o Multipoint	52.08/76.19	9.55MB	4.877G
		MP (Banner, Nahshan, and Soudry 2019)	70.59/90.08	9.55MB	4.877G
		Ours	71.96/90.75	9.63MB	5.525G
		Ours + Clip	**72.78/91.23**	**9.58MB**	**5.354G**
ResNet-18	32/32	Full-Precision	69.76/89.08	42.56MB	-
	4/4	w/o Multipoint	57.00/80.40	5.32MB	423.89M
		MP (Banner, Nahshan, and Soudry 2019)	64.78/85.90	5.32MB	423.89M
		Ours	64.29/85.59	5.39MB	494.16M
		Ours + Clip	**65.89/86.68**	**5.41MB**	**470.89M**
ResNet-50	32/32	Full-Precision	76.15/92.87	89.44MB	-
	4/4	w/o Multipoint	65.88/86.93	11.18MB	992.28M
		MP (Banner, Nahshan, and Soudry 2019)	72.52/90.80	11.18MB	992.28M
		Ours	71.88/90.43	11.33MB	1.148G
		Ours + Clip	**72.67/91.11**	**11.32MB**	**1.128G**
ResNet-101	32/32	Full-Precision	77.37/93.56	161.68MB	-
	4/4	w/o Multipoint	69.67/89.21	20.21MB	1.920G
		MP (Banner, Nahshan, and Soudry 2019)	**74.22/91.95**	**20.21MB**	**1.920G**
		Ours	71.56/90.36	20.82MB	2.177G
		Ours+Clip	72.85/91.16	21.04MB	2.189G
Inception-v3	32/32	Full-Precision	77.45/93.56	82.96MB	-
	4/4	w/o Multipoint	12.12/25.24	10.37MB	1.423G
		MP (Banner, Nahshan, and Soudry 2019)	60.64/82.15	10.37MB	1.423G
		Ours	61.22/83.27	10.44MB	1.692G
		Ours+Clip	**65.49/86.72**	**10.38MB**	**1.519G**
Mobilenet-v2	32/32	Full-Precision	71.78/90.19	8.36MB	-
	4/4	w/o Multipoint	6.86/16.76	1.04MB	74.87M
		MP (Banner, Nahshan, and Soudry 2019)	42.61/67.78	1.04MB	74.87M
		Ours	27.52/50.80	1.05MB	91.16M
		Ours+Clip	**55.54/79.10**	**1.045MB**	**85.88M**

Table 3: Per-channel quantization on ImageNet Benchmark (W=Weight, A=Activation, M=10^6, G=10^9, MP=Mixed Precision, Acc=Accuracy). Note that MP requires specialized hardware. Bold refers to the method with highest Top-1 accuracy. 'Clip' means using the optimal clipping factor K^* by solving Eq. 13.

(a) Per-layer Quantization

W/A	Method	mAP(%)	Size(MB)	OPs(G)
32/32	FP	77.43	100.24	-
4/8	w/o Multipoint	72.86	12.53	15.69
	Ours	**74.10**	**12.63**	**17.58**
3/8	w/o Multipoint	42.56	9.40	11.76
	Ours	**46.94**	**9.41**	**12.18**

(b) Per-channel Quantization

W/A	Method	mAP(%)	Size(MB)	OPs(G)
32/32	FP	77.43	100.24	-
4/4	w/o Multipoint	73.17	12.53	7.843
	Ours	**73.58**	**12.62**	**8.636**
3/3	w/o Multipoint	59.37	9.40	4.412
	Ours	**60.46**	**9.43**	**4.733**

Table 4: Post-training quantization result on SSD-VGG16 (mAP=mean average precision, FP=Full-Precision). Bold refers to the method with highest mAP.

all three methods, implying the importance of choosing appropriate channels for multipoint quantization.

Where Multipoint Quantization is Applied: Fig 3 shows the relative increment of size in each layer. We observe that the layers close to the input have more relative increment of size compared with later layers.

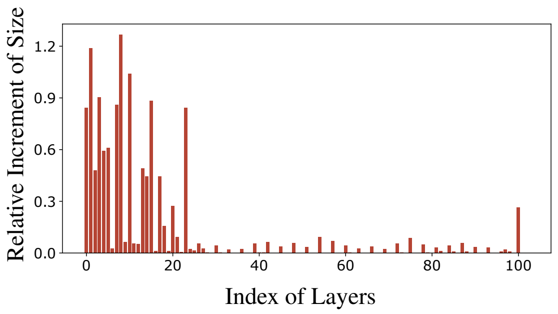

Figure 3: Relative increment of size in each layer of a quantized ResNet-101 with multipoint quantization. The layers close to the input have large relative increment, while the layers after the 30-th layer have only negligible increment.

Conclusions

We propose multipoint quantization for post-training quantization, which is hardware-friendly and effective. It performs favorably compared with state-of-the-art methods.

References

Banner, R.; Nahshan, Y.; and Soudry, D. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, 7948–7956.

Chen, Y.-H.; Krishna, T.; Emer, J. S.; and Sze, V. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52(1): 127–138.

Cheng, J.; Wang, P.; Li, G.; Hu, Q.; and Lu, H. 2018. Recent Advances in Efficient Computation of Deep Convolutional Neural Networks. *arXiv preprint arXiv:1802.00939* .

Choukroun, Y.; Kravchik, E.; and Kisilev, P. 2019. Low-bit quantization of neural networks for efficient inference. *arXiv preprint arXiv:1902.06822* .

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, 3123–3131.

Dong, Z.; Yao, Z.; Gholami, A.; Mahoney, M. W.; and Keutzer, K. 2019. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. In *The IEEE International Conference on Computer Vision (ICCV).*

Everingham, M.; Van Gool, L.; Williams, C. K. I.; Winn, J.; and Zisserman, A. 2007. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Gong, C.; Jiang, Z.; Wang, D.; Lin, Y.; Liu, Q.; and Pan, D. Z. 2019. Mixed Precision Neural Architecture Search for Energy Efficient Deep Learning. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–7. IEEE.

Guo, Y.; Yao, A.; Zhao, H.; and Chen, Y. 2017. Network sketching: Exploiting binary structure in deep cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5955–5963.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Horowitz, M. 2014. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 10–14. IEEE.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* .

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18(1): 6869–6898.

Jung, S.; Son, C.; Lee, S.; Son, J.; Han, J.-J.; Kwak, Y.; Hwang, S. J.; and Choi, C. 2019. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4350–4359.

Krishnamoorthi, R. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* .

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Li, Y.; Dong, X.; and Wang, W. 2019. Additive Powers-of-Two Quantization: A Non-uniform Discretization for Neural Networks. *arXiv preprint arXiv:1909.13144* .

Lin, X.; Zhao, C.; and Pan, W. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, 345–353.

Louizos, C.; Reisser, M.; Blankevoort, T.; Gavves, E.; and Welling, M. 2018. Relaxed quantization for discretized neural networks. *arXiv preprint arXiv:1810.01875* .

Meller, E.; Finkelstein, A.; Almog, U.; and Grobman, M. 2019. Same, same but different-recovering neural network quantization error through weight factorization. *arXiv preprint arXiv:1902.01917* .

Migacz, S. 2017. 8-bit inference with tensorrt. In *GPU technology conference*, volume 2, 7.

Mishra, A.; Nurvitadhi, E.; Cook, J. J.; and Marr, D. 2017. WRPN: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134* .

Nagel, M.; van Baalen, M.; Blankevoort, T.; and Welling, M. 2019. Data-Free Quantization Through Weight Equalization and Bias Correction. *arXiv preprint arXiv:1906.04721* .

Nahshan, Y.; Chmiel, B.; Baskin, C.; Zheltonozhskii, E.; Banner, R.; Bronstein, A. M.; and Mendelson, A. 2019. Loss Aware Post-training Quantization. *arXiv preprint arXiv:1911.07190* .

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, 525–542. Springer.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* .

Sung, W.; Shin, S.; and Hwang, K. 2015. Resiliency of deep neural networks under quantization. *arXiv preprint arXiv:1511.06488* .

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8612–8620.

Xu, X.; Ding, Y.; Hu, S. X.; Niemier, M.; Cong, J.; Hu, Y.; and Shi, Y. 2018. Scaling for edge inference of deep neural networks. *Nature Electronics* 1(4): 216.

Zhao, R.; Hu, Y.; Dotzel, J.; De Sa, C.; and Zhang, Z. 2019. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. In *International Conference on Machine Learning*, 7543–7552.

Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; and Zou, Y. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* .

Zhu, C.; Han, S.; Mao, H.; and Dally, W. J. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* .

Zhuang, B.; Shen, C.; Tan, M.; Liu, L.; and Reid, I. 2019. Structured binary neural networks for accurate image classification and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 413–422.

Zmora, N.; Jacob, G.; Zlotnik, L.; Elharar, B.; and Novik, G. 2018. Neural Network Distiller. doi:10.5281/zenodo.1297430. URL https://doi.org/10.5281/zenodo.1297430.