In-Hindsight Quantization Range Estimation for Quantized Training

Marios Fournarakis, Markus Nagel
Qualcomm AI Research*
mfournar@qti.qualcomm.com, markusn@qti.qualcomm.com

Abstract

Quantization techniques applied to the inference of deep neural networks have enabled fast and efficient execution on resource-constraint devices. The success of quantization during inference has motivated the academic community to explore fully quantized training, i.e. quantizing back-propagation as well. However, effective gradient quantization is still an open problem. Gradients are unbounded and their distribution changes significantly during training, which leads to the need for dynamic quantization. As we show, dynamic quantization can lead to significant memory overhead and additional data traffic slowing down training. We propose a simple alternative to dynamic quantization, in-hindsight range estimation, that uses the quantization ranges estimated on previous iterations to quantize the present. Our approach enables fast static quantization of gradients and activations while requiring only minimal hardware support from the neural network accelerator to keep track of output statistics in an online fashion. It is intended as a drop-in replacement for estimating quantization ranges and can be used in conjunction with other advances in quantized training. We compare our method to existing methods for range estimation from the quantized training literature and demonstrate its effectiveness with a range of architectures, including MobileNetV2, on image classification benchmarks (Tiny ImageNet & ImageNet).

1. Introduction

Deep Neural Networks (DNNs) have become the state-of-the technique for a wide range of Computer Vision (CV) applications, such as image recognition, object detection or machine translation. However, as the accuracy and effectiveness of these networks grow, so does their size. The high computational cost and memory footprint can impede the deployment of such networks to resource-constrained devices, such as smartphones, wearables or drones. Fortunately, in recent years low-bit network quantization for neural network inference has been extensively studied and in combination with dedicated hardware utilizing efficient fixed-point operations, they have succeeded in accelerating DNN inference [7, 13, 11, 12, 3].

However, training DNNs still predominately relies on floating-point format. As we move towards a world of privacy-preserving and personalized AI, we can expect increased requirements for training on edge devices that do not have the computational resources of servers. This raises the need for more power-efficient training techniques for neural networks. Quantizing back-propagation can provide considerable acceleration and power efficiency but the noise induced by gradient quantization can be detrimental to the network's accuracy [25, 23]. Recent work has shown that *quantized training*, can achieve accuracy within $1\% - 2\%$ of floating-point (FP32) training in a range of CV tasks and models [25, 19, 2]. In most cases, this is possible by quantizing the weights, activations and activation gradients to 8-bits and maintaining certain operations in floating-point, such as batch-normalization [6] or weight updates.

A key challenge present throughout the quantized training literature is how to set the quantization range for gradients [2, 25, 15, 1, 21, 22]. Because gradients are unbounded, choosing the quantization range appropriately is important to keep the quantization error in check. Some existing methods use the min-max range of the gradient tensor [21, 22, 24] whereas others use a moving average of the tensor's statistics [23]. [2] goes even further and propose a per-sample quantization of gradient tensor. However, in all cases there is a common theme: to determine the quantization parameters of the tensor, these methods require access to the unquantized gradient tensor. In other words, they perform *dynamic quantization*.

Dynamic quantization can reduce the quantization error as the quantization grid is better utilized but comes with significant memory overhead [4, 8, 20]: the quantization range depends on the full tensor output, therefore, the entire full precision tensor needs to be written to memory before it can be quantized. For typical layers in common DNNs, this can lead up to $8\times$ more memory transfer. In section 3.2, we discuss in more detail the implications of dynamic quanti-

*Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

zation for fixed-point accelerator hardware and quantify the associated overhead.

In this work, we provide a hardware-friendly alternative to dynamic quantization for quantized training, called *in-hindsight range estimation*. Our approach uses the quantization ranges from previous iterations to quantize the current tensor. This method allows us to use pre-computed quantization ranges to accelerate training and reduce the memory overhead. We use a moving average of the quantization range and update the ranges with statistics extracted from the accumulator in an online fashion.

We evaluate our proposed framework on Tiny ImageNet and ImageNet datasets. We show that our method achieves comparable accuracy to dynamic quantization when applied to activations and gradients. Our approach is intended as a drop-in replacement for estimating quantization ranges and can be used in conjunction with other advances in quantized training.

2. Related Work

In this section, we outline some of the relevant work in the area of quantized training. We split the contribution into two sections. First, we discuss general advances in quantized training and, second, we focus on the details of quantization range estimation by relevant work. We concentrate on activation and gradient quantization because they rely on dynamic quantization. Weights can be quantized offline as they do not depend on the input data.

2.1. Quantized Training

The first attempt in fully quantized training dates back to 2015 [5]. The authors train a 3-layer convolutional network on CIFAR-10 and MNIST and introduce stochastic rounding as an unbiased quantization operator. DoReFa-Net [24] train low-bit AlexNet with BatchNorm on ImageNet but struggle to close the gap to full precision models when training from scratch. Range Normalization is introduced by [1] as a fixed-point friendly alternative to BatchNorm and they observe no accuracy drop when training ResNet-50 on ImageNet. [15] formulate a theoretical framework for finding the optimal bit-width for all quantized tensor in fully quantized training and perform the weight update in fixed-point but keep BatchNorm operations in floating-point. WAGE [21] adopt a layer-wise scaling factor instead of using BatchNorm and quantize gradients to 8-bits while keeping the weight update to 16-bits. WAGEUBN [22] expand on WAGE and implement a quantized implementation of BatchNorm for the forward and backward pass. Deviation Counteractive Learning Rate Scaling [25] uses an exponential decaying learning rate based on the cosine distance between the full precision and quantized gradients to stabilize training. The authors measure a 22% training time reduction for ResNet-50 on ImageNet and apply their training framework to MobileNetV2 and Inception V3. Recently, it was shown that ResNet-50 can be trained within 1% of FP32, using a hybrid of INT4 forward quantization and a novel Radix-4 FP4 format for the gradients [19].

2.2. Quantization Range Estimation

In earlier attempts [5], a fixed-point format with a fixed decimal point was used for activations and gradients throughout training. DoReFa-Net clip the activations to the $[0, 1]$ range and use the dynamic min-max range to quantize the gradients. WAGE and WAGEUBN use a pre-defined scale factor α for each layer's activation which depends on the network's structure and the min-max range for the gradients. [15] fix the activation range to $[0, 2]$ and use an exponential moving average of the gradient's standard deviation to calculate the gradient bit-width. Models with RagenNorm [1] split the activation and gradient tensor in chunks and use the average minimum and maximum of the chunks to estimate the quantization range. [23] use an exponential moving average of the maximum absolute value for activation and gradients. Direction sensitive gradient clipping [25] periodically updates the gradient clipping range by searching for the clipping values that minimizes the angle between the FP32 and quantized gradients. To the best of our knowledge, this is the only existing method that to an extend avoids dynamic quantization

In all previous cases, the statistics are computed over the complete tensor. [2] observe that per-tensor quantization of gradients does not utilize the quantization grid efficiently and instead propose a per-sample quantization and a Block Householder decomposition of the gradient tensor to better spread the signal across the tensor.

3. Problem Formulation

In this section, we outline how quantized training works using typical fixed-point accelerator architecture and explain how dynamic quantization can lead to memory and compute overhead. In this work, we concentrate on hardware dedicated to fixed-point operations to extract the biggest power gains from quantized tensor operations.

3.1. Quantized Training Framework

In figure 1 we present a compute graph for the forward (left) and backward pass (right) for our quantized training framework. The forward pass is very similar to that quantization-aware training (QAT) [9, 7]. The weights $\mathbf{W}$ are typically stored in higher precision (16-bits or FP32) to allow the accumulation of small gradients during training. The weight tensor is quantized to a low-bit representation $\widetilde{\mathbf{W}}$ through the quantization function $Q_{\mathbf{W}}(\cdot)$ before it is loaded into the Multiply & Accumulate (MAC) array. The quantized input $\widetilde{\mathbf{X}}$ is also loaded into the MAC array

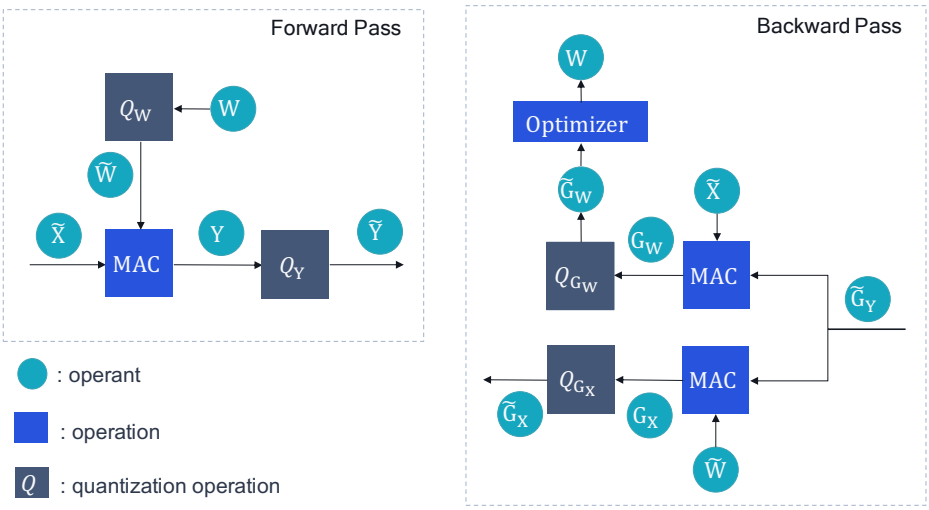

Figure 1: Forward and backward pass for quantized training pipeline.

to compute the linear operations of the layer. As $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{W}}$ are typically larger than the MAC array, the output is calculated over multiple compute cycles. The output of the MAC array $\mathbf{Y}$ is typically in 32-bits and is thus followed by another quantization step, $Q_{\mathbf{Y}}(\cdot)$, to convert it to the required bit-width.

During the backward pass, the quantized activation gradient $\widetilde{\mathbf{G}}_{\mathbf{Y}}$ is used to calculated the weight gradient $\mathbf{G}_{\mathbf{W}}$ and input gradient $\mathbf{G}_{\mathbf{X}}$. The input gradient is typically quite large and is always quantized to a low-bit representation $\widetilde{\mathbf{G}}_{\mathbf{X}}$ before it is propagated to the preceding layer. The weight gradient can be re-quantized to a lower-bit representation, but it is also common in literature to keep it in full precision. In this work, we also keep the weight gradient in FP32 and we denote $Q_{\mathbf{G_X}}(\cdot) = Q_{\mathbf{G}}(\cdot)$ for clarity.

For fully quantized training, we need to specify the quantization ranges of at least three quantizers $Q_{\mathbf{W}}(\cdot)$, $Q_{\mathbf{Y}}(\cdot)$ and $Q_{\mathbf{G}}(\cdot)$. Because the weights are independent of the data, the quantization range for the weights can be precomputed and be stored in memory. However, this is not the case for the activations and gradient, as they depend on the current batch. To address this issue most existing techniques assume *dynamic quantization* for these quantizers. In the following section, we discuss what exactly is dynamic quantization and its implications for quantized training. It is important to be able to adjust the quantization ranges of gradients during training because the gradient distribution changes significantly during training [19, 25, 23].

3.2. Dynamic Quantization

Dynamic quantization uses the statistics of the full precision tensor to quantize it. Assuming a quantization range of $(q_{\text{min}}, q_{\text{max}})$, then in its simplest form dynamic quantization uses the *min-max range* of the full tensor $\mathbf{G}$:

$$q_{\text{min}} = \min \mathbf{G}, \quad q_{\text{max}} = \max \mathbf{G} \tag{1}$$

Figure 2 illustrates how matrix multiplication is computed in a typical fixed-point accelerator. The logic consists of a fixed-size MAC array and accumulators. Typically, in deep learning the size of matrices that are multiplied exceed the size of the MAC array. For this reason, the computation takes places in slices until the whole matrix multiplication is completed. The output of the accumulator is typically in higher bit-width (32-bits) to avoid overflow and is normally followed by a quantization step.

In the case of *static* quantization, the quantization ranges of the output $(q_{\text{min}}, q_{\text{max}})$ are known in advance. Therefore, each output from the accumulator can be quantized directly and be stored in memory in a low-bit representation. On the other hand, in *dynamic* quantization, the ranges depend on the output itself. To extract the necessary statistics, all the outputs of the accumulator have to first be written in memory. After the quantization ranges have been calculated, the tensor is brought back to the compute unit to be quantized and then stored back in memory.

It is evident, that dynamic quantization can lead to significant memory overhead and extra data movement. Our analysis in section 6 shows that dynamic quantization can lead up to $8\times$ additional memory transfer depending on the size of the layer. A study between static and dynamic quantization for an MLP in PyTorch 1.4 on a CPU also showed that dynamic quantization leads on average to a 20% latency increase for inference. Unfortunately, dynamic quantization for convolutions is not implemented in PyTorch, which might be due to the even larger overhead associated with large feature maps.

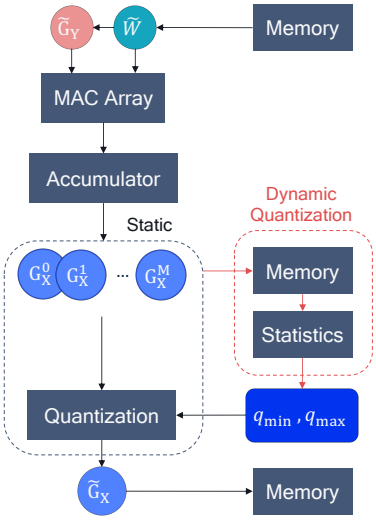

Figure 2: Quantized neural network accelerator diagram. The MAC array size is fixed which means that output tensor can only be computed in slices. In the case of static quantization the quantization parameters are know in advance and the accumulator output is directly quantized. For dynamic quantization all outputs have to be written to memory before they can be quantized.

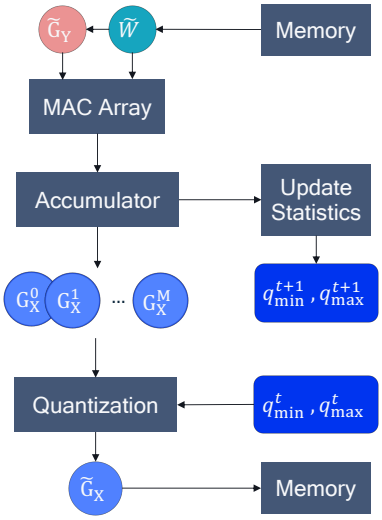

Figure 3: General framework for in-hindsight range estimation. Using pre-computed quantization ranges enables efficient static quantization of the output. Additional logic is needed at the accumulator level update the ranges for the next step.

4. In-Hindsight Range Estimation

Our proposed method aims at preventing the need for dynamic quantization during quantized training and unlock the speed-ups provided by dedicated fixed-point hardware. The method involves two key steps:

1. Use *pre-computed* quantization parameters to quantize the current tensor.
2. Extract *statistics* from the current tensor in an online fashion to update the quantization parameters for the next iteration.

Figure 3 shows a general framework of how in-hindsight range estimation can be implemented in hardware. The benefit of this approach is that the pre-computed quantization enables fast and efficient static quantization. The required statistics should be easy to calculate at the accumulator or quantization level, to reduce the computational overhead of the method. Such statistics can be the min and max statistics or the saturation ratio[1]. In some cases, extracting these statistics may require appropriate hardware logic around the accelerator.

4.1. In-Hindsight Min-Max

We propose an instance of our framework that uses the min-max statistics, which we call *in-hindsight min-max*. In this method, we define the quantization range as the exponential moving average of the tensor's min-max statistics. To quantize the tensor at step t, we use the estimate of quantization ranges from the previous iteration. While the output is computed, appropriate logic keeps track of the min-max statistics from the accumulator. These statistics are then used to update the quantization ranges for the next iteration as soon as the complete output tensor has been calculated. The quantization ranges are calculated as:

$$q_{\text{min}}^{t} = (1-\eta)\min \mathbf{G}^{t-1} + \eta q_{\text{min}}^{t-1} \tag{2}$$

$$q_{\text{max}}^{t} = (1-\eta)\max \mathbf{G}^{t-1} + \eta q_{\text{max}}^{t-1} \tag{3}$$

where η is the momentum term. To initialize the scheme ($t = 0$), we can use the min-max range of the first batch, namely $q_{\text{min}}^{0} = \min \mathbf{G}^{0}$ and $q_{\text{max}}^{0} = \max \mathbf{G}^{0}$. If we replace the $\mathbf{G}^{t-1}$ with $\mathbf{G}^{t}$ in the above equations we end up with a dynamic quantization method called *running min-max*, which is common in post-training quantization [9] and adopted for gradient quantization by [23].

5. Experiments

To verify the effectiveness of our method we conduct experiments on the Tiny ImageNet [10] and full ImageNet [14] benchmarks. We train ResNet18 on ImageNet and a modified version on Tiny ImageNet [18]. We also train our own version VGG16 [17] and MobileNetV2 [16] on Tiny ImageNet.

[1]The proportion of values that lie outside the quantization grid.

5.1. Range Estimation Methods Comparison

Typically, in literature, we only observe final results for the fully quantized setting, making it difficult to assess the impact of the individual quantization choices for each tensor. In this section, we aim at better understanding the impact of the individual range estimation methods for either gradients or activations quantization at the final accuracy. We compare our hardware friendly method, in-hindsight min-max range estimation, to the commonly used dynamic quantization methods: current min-max [24, 21, 22, 25] and running min-max [9, 23] estimators.

We further compare to Direction Sensitive Gradient Clipping (DSGC) [25]. DSGC searches for the optimal clipping values that maximizes the cosine similarity between the quantized and full precision tensor. In our implementation of the method, we use golden section search to find the optimal quantization ranges, as the authors do not provide implementation details. Because of the computational cost of such optimization, the quantization range is only updated periodically. We use an update interval of 100 iteration as per the authors. Note, this method is a hybrid between static quantization and dynamic quantization. It uses HW friendly static quantization for most iterations but the update step can be very expensive, as it requires estimating the objective function (cosine similarity) at multiple clipping thresholds.

We also experimented with using an exponential moving average of the gradient variance [15] to define the quantization ranges. However, we found that it made training unstable despite an extensive hyper-parameter search for momentum and the number of standard deviations.

Experimental Setup We conduct our study on Tiny ImageNet with the modified ResNet18 and train for 90 epochs using SGD with initial learning rate of 0.1 and momentum of 0.9. The learning rate is reduced by a factor of 10 at epochs 30 and 60 and we use a weight decay of 1e-4. For all different range estimation methods on the gradient and activations tensor, we tune the hyper-parameter (e.g. momentum term) individually and present the results with the best hyper-parameters.

Gradient Quantization To see the effect of range estimation for gradient quantization, we keep the forward pass in full precision and only quantize the activation gradient to 8-bit using asymmetric uniform quantization with stochastic rounding [5].

We see that in hindsight min-max performs on par and even better than the other dynamic quantization methods. Its performance is also comparable to DSGC, which is a hybrid between dynamic and static quantization method. However, in-hindsight range estimation relies on simple statistics, whereas DSGC requires a computationally intensive parameter search. We also found that under certain seeds training with DSGC can become quite unstable, which is also reflected by the larger standard deviation of the final result. Current min-max underperforms all other methods. This analysis demonstrates that switching to a better range estimator could be very beneficial before attempting more complex solutions.

Method	Static	Val. Acc. (%)
FP32	n.a.	58.97 ± 0.13
Current min-max	✗	59.14 ± 0.23
Running min-max	✗	59.25 ± 0.55
DSGC [25]	✗	59.35 ± 0.95
In-hindsight min-max	✓	59.46 ± 0.71

Table 1: Gradient quantization range estimators comparison. Validation accuracy (average of 5 seeds) and standard deviation for ResNet-18 on Tiny ImageNet.

Method	Static	Val. Acc. (%)
FP32	n.a.	58.97 ± 0.13
Current min-max	✗	59.00 ± 0.31
Running min-max	✗	59.28 ± 0.25
DSGC [25]	✗	59.09 ± 0.01
In-hindsight min-max	✓	59.30 ± 0.19

Table 2: Activation quantization range estimator comparison. Validation accuracy (average of 5 seeds) and standard deviation for ResNet-18 on Tiny ImageNet.

It is also interesting to observe that gradient quantization leads to accuracy improvements compared to FP32 training across the board likely due to its regularization effect.

Activation Quantization In this study, we explore the same range estimation methods for activation quantization. We keep the weights and backward pass in full precision and only quantize the activations using asymmetric uniform quantization.

Similar to gradient quantization, our method is on par with dynamic quantization methods and outperforms FP32. For activation quantization, in-hindsight min-max and running min-max perform significantly better than the other methods. DSGC performs worse in activation quantization, which is not surprising, as its objective function was designed with gradient distributions in mind. Once again current min-max trails the other methods.

5.2. Fully Quantized Training Results

We now demonstrate the effectiveness of our method for fully quantized training on ImageNet and Tiny ImageNet

Gradient Method	Activation Method	Static	ResNet18	VGG16	MobileNetV2
FP32	FP32	n.a.	58.97 ± 0.13	53.79 ± 0.30	59.61 ± 0.37
Current min-max	Current min-max	✗	58.77 ± 0.73	53.28 ± 0.43	58.88 ± 0.73
Running min-max	Running min-max	✗	59.20 ± 0.25	53.36 ± 0.27	59.69 ± 0.09
DSGC [25]	Current min-max	✗	59.07 ± 0.33	52.84 ± 0.28	59.10 ± 0.44
In-hindsight min-max	In-hindsight min-max	✓	58.99 ± 0.44	53.25 ± 0.41	59.28 ± 0.20

Table 3: Results on Tiny ImageNet. Average validation accuracy (%) with standard deviation: 5 seeds for ResNet18 & VGG16 and 3 seeds for MobilenetV2.

Gradient Method	Activation Method	Static	ResNet18
FP32	FP32	n.a.	69.75
Current min-max	Current min-max	✗	69.21 ± 0.06
Running min-max	Running min-max	✗	69.35 ± 0.16
In-hindsight min-max	In-hindsight min-max	✓	69.37 ± 0.11

Table 4: Validation accuracy (%) results on ImageNet. Average validation accuracy (%) of 3 seeds with standard deviation.

benchmarks and compare against other dynamic quantization methods. We apply in-hindsight min-max to both activations and gradients. We also compare against using running and current min-max for both tensors. The earlier is a variation of the method adopted by quantification parameter adjustment [23] but without the bit-with adjustment, whereas the latter is similar to the quantized training framework of [2] when using per-tensor quantization. Finally, for Tiny ImageNet, we further compare to quantized training framework of DSGC without the adaptive learning rate.

Experimental Setup Weights, activations and gradients are quantized to 8-bits (W8/A8/G8). We use uniform stochastic quantization for gradients and standard uniform quantization for the weights and activations for all layers, including the first and last layer of the networks. The weights are always quantized with the current min-max method. For the DSGC method, we follow the author's approach and dynamically quantize activations using current min-max.

We use a momentum of 0.9 for running min-max and in-hindsight min-max although we observe little sensitivity to that parameter. We also found that both methods benefit from an initial calibration step when used for activation quantization. By calibration, we mean feeding a few batches of data through the network to calibrate the quantization ranges before training starts.

All models are trained using SGD with momentum of 0.9. ResNet18 is trained as described in the previous section. VGG16 is trained for 90 epochs with initial learning rate of 0.01. The learning rate is reduced by a factor of 10 at epochs 60 and 80. We train MobileNetV2 for 120 epochs with initial learning of 0.01 for the depthwise-separable layers and 0.1 for all other layers in the network. We found that this heterogeneous learning rate stabilizes quantized training for all methods. It also leads to no accuracy drop for FP32 training compared to a homogeneous learning rate of 0.1. We use a cosine annealing schedule with a final learning rate of 1e-5 and a weight decay of 2e-5.

Results Discussion The results for Tiny ImageNet are shown in table 2. Across all 3 models, our hardware-friendly in-hindsight min-max performs on par with the significantly less efficient dynamic quantization methods. Only running min-max outperforms this slightly for MobileNetV2. We observe a similar trend for full ImageNet training (cf. table 3) where in-hindsight min-max performs similar to running min-max and marginally outperforms the commonly used current min-max. In summary, for all cases, 8-bit in-hindsight quantization is close to the FP32 baseline (within .5%) while fully utilizes the advantages of common fixed point accelerators.

6. Memory Transfer Comparison

In this section, we compare the memory transfer associated with static and dynamic quantization. We show it here for the forward pass, the backwards pass follows analogously (see figure 1).

In static quantization, the quantized weight tensor $\widetilde{\mathbf{W}}$ (C_{in} input channels, C_{out} output channels, $k \times k$ kernel size) and the quantized input $\widetilde{\mathbf{X}}$ of size $W \times H$ are sequentially loaded to the MAC array. The output of the accumulator is then quantized and stored in memory. The total memory

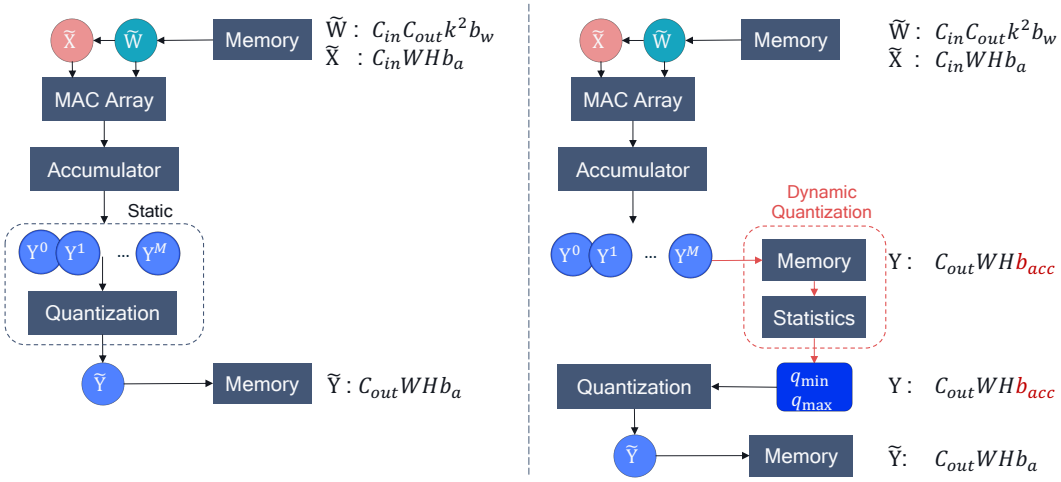

Figure 4: Comparison of memory movements associated with static (left) and dynamic (right) quantization in a fixed-point neural network accelerator.

Network	Conv	C_{in}	C_{out}	$W \times H$	Static	Dynamic	Delta
ResNet18	3×3	64	64	56×56	428 KB	1996 KB	+366%
ResNet18	3×3	256	256	14×14	674 KB	1066 KB	+58%
MobileNetV2	1×1	16	96	112×112	1374 KB	10782 KB	+685%
MobileNetV2	3×3 (DW)	96	96	112×112	882 KB	4410 KB	+400%
MobileNetV2	3×3 (DW)	960	960	7×7	100 KB	468 KB	+366%

Table 5: Memory movement costs comparison between static and dynamic quantization for various layers in ResNet18 and MobileNetV2 on ImageNet ($b_w = b_a = 8$-bits, $b_{\text{acc}} = 32$-bits, DW = depthwise separable).

cost for static quantization in bits is given by:

$$\underbrace{C_{\text{in}}C_{\text{out}}k^2b_w}_{\text{weight kernel}} + \underbrace{C_{\text{in}}WHb_a}_{\text{input feature}} + \underbrace{C_{\text{out}}WHb_a}_{\text{output feature}} \tag{4}$$

where b_a and b_w are the activation and weight bit-width, respectively. As we discussed in section 3.2, dynamic quantization requires writing the accumulator output first in memory. After the statistics have been calculated the quantization parameters are updated and the output is then brought back to the compute unit to be quantized. The quantized output is then written back to memory. To total memory cost of dynamic quantization is given by:

$$\begin{aligned}\underbrace{C_{\text{in}}C_{\text{out}}k^2b_w}_{\text{weight kernel}} + \underbrace{C_{\text{in}}WHb_a}_{\text{input feature}} + \underbrace{C_{\text{out}}WHb_{\text{acc}}}_{\text{save acc output}} \\ + \underbrace{C_{\text{out}}WHb_{\text{acc}}}_{\text{load acc output}} + \underbrace{C_{\text{out}}WHb_a}_{\text{save quantized output}}\end{aligned} \tag{5}$$

where b_{acc} is the bit-width of the accumulator. Figure 4 illustrates the memory movement associated with every step.

In table 5, we compare the memory movement cost of static and dynamic quantization for typical layers in ResNet18 and MobileNetV2 using $b_w = b_a = 8$-bits and $b_{\text{acc}} = 32$-bits. The exact overhead of dynamic quantization depends on many parameters, such as the input size, number of channels and type of kernel. In most cases, the extra memory movement is about $4\times$. Only in later layers in ResNet18, where the weight tensor is significantly larger than the input feature map, is the overhead lower. In the extreme case of certain point-wise convolutions in MobileNetV2, the memory movement of dynamic quantization can be $8\times$ higher than for static quantization.

7. Conclusions

In this paper, we provide a general framework for estimating quantization ranges in the context of quantized training that overcomes the need for dynamic quantization. Our proposed approach, *in-hindsight range estimation*, uses past estimates of the quantization parameters to enable static quantization of the tensor in question. It relies on extracting simple statistics from the output tensor in an online fashion to update the quantization parameters for the next iteration. While this is a general framework, we propose a specific variant, called *in-hindsight min-max* that uses the min-max statistics.

We demonstrate the effectiveness of our methods on pop-

ular image classification benchmarks by comparing them to other dynamic quantization techniques found in literature. We show that *in-hindsight min-max* performs on par with the best scoring dynamic range methods while reducing significantly the memory overhead associated with dynamic quantization. It can be used as a drop-in replacement method for estimating quantization ranges that can better utilize the common fixed-point accelerators for quantized training.

References

[1] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks, 2018.

[2] Jianfei Chen, Yu Gai, Zhewei Yao, Michael W. Mahoney, and Joseph E. Gonzalez. A statistical framework for low-bitwidth training of deep neural networks, 2020.

[3] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. *International Conference on Learning Representations (ICLR)*, 2020.

[4] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[5] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision, 2015.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

[8] Raghuraman Krishnamoorthi. Introduction to quantization on pytorch, https://pytorch.org/blog/introduction-to-quantization-on-pytorch/#post-training-static-quantization. Accessed: 2021-04-15.

[9] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[10] Y. Le and X. Yang. Tiny imagenet visual recognition challenge. 2015.

[11] Markus Nagel, Rana Ali Amjad, Marinus van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *Proceedings of Machine Learning and Systems 2020*, pages 7696–7705. 2020.

[12] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. *International Conference on Computer Vision (ICCV)*, 2019.

[13] Bita Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, Alessandro Forin, Haishan Zhu, Taesik Na, Prerak Patel, Shuai Che, Lok Chand Koppaka, Xia Song, Subhojit Som, Kaustav Das, Saurabh Tiwary, Steve Reinhardt, Sitaram Lanka, Eric Chung, and Doug Burger. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. In *NeurIPS 2020*. ACM, November 2020.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[15] Charbel Sakr and Naresh Shanbhag. Per-tensor fixed-point quantization of the back-propagation algorithm, 2018.

[16] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[18] Lei Sun. Resnet on tiny imagenet. 2017.

[19] Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi (Viji) Srinivasan, and Kailash Gopalakrishnan. Ultra-low precision 4-bit training of deep neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1796–1807. Curran Associates, Inc., 2020.

[20] TensorFlow. Post-training quantization, https://www.tensorflow.org/lite/performance/post_training_quantization. Accessed: 2021-04-15.

[21] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks, 2018.

[22] Yukuan Yang, Shuang Wu, Lei Deng, Tianyi Yan, Yuan Xie, and Guoqi Li. Training high-performance and large-scale deep neural networks with full 8-bit integers, 2019.

[23] Xishan Zhang, Shaoli Liu, Rui Zhang, Chang Liu, Di Huang, Shiyi Zhou, Jiaming Guo, Qi Guo, Zidong Du, Tian Zhi, and Yunji Chen. Fixed-point back-propagation training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[24] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2018.

[25] Feng Zhu, Ruihao Gong, Fengwei Yu, Xianglong Liu, Yanfei Wang, Zhelong Li, Xiuqi Yang, and Junjie Yan. Towards unified int8 training for convolutional neural network, 2019.