Supplemental Material: Learnable Lookup Table for Neural Network Quantization

Longguang Wang[1], Xiaoyu Dong[2,3], Yingqian Wang[1], Li Liu[1], Wei An[1], Yulan Guo[1*]
[1]National University of Defense Technology [2]The University of Tokyo [3]RIKEN AIP
{wanglongguang15,yulan.guo}@nudt.edu.cn

Section I presents experimental details. Section II illustrates the derivation of gradients in our method. Section III includes additional analyses. Finally, Section IV discusses potential negative societal impact of our method.

I. Experimental Details

I.I. Experiments on CIFAR-10

During training, the original 32×32 images were padded with 4 pixels on each side. Then, 32×32 patches were randomly cropped and horizontally flipped. The stochastic gradient descent (SGD) method with momentum of 0.9 was used for optimization. All quantized models were trained for 200 epochs with a mini-batch size of 128. For ResNet-20, the learning rate was initially set to 0.01 and decayed by a factor of 10 at epoch 80 and 120, with the weight decay being set to 10^{-4}. The gradients were clipped with a maximum L_2 norm of 5. For VGG-Small, the learning rate was initially set to 0.02 and decayed by a factor of 10 at epoch 80 and 160, with the weight decay being set to 5×10^{-4}. The gradients were clipped with a maximum L_2 norm of 3. τ_0, τ_1, and M in Eq. 3 were empirically set to 1, 10^{-3}, and 50 for both ResNet-20 and VGG-Small.

I.II. Experiments on ImageNet

During training, the original images were resized, cropped to 224×224 and randomly flipped horizontally for data augmentation. The SGD method with momentum of 0.9 was used for optimization. All quantized models were trained for 120 epochs with a mini-batch size of 1024. The learning rate was initially set to 0.01 and decayed by a factor of 10 at epoch 30, 60, and 90. Weight decay was set to 10^{-4} for 4/3-bit quantization and set to 2×10^{-5} for 2-bit quantization. τ_0, τ_1, and M in Eq. 3 were empirically set to 1, 10^{-3}, and 30, respectively.

I.III. Experiments on SR

During training, 12 low-resolution patches with their corresponding high-resolution counterparts were randomly cropped. Then, data augmentation was performed through random rotation and random flipping. The Adam [2] method with $\beta_1 = 0.9$ and $\beta_2 = 0.99$ was used for optimization. An L_1 loss between SR results and HR images was used as the loss function. The initial learning rate was set to 10^{-4} and halved every 10 epochs. The gradients were clipped with a maximum L_2 norm of 5. All quantized models were trained for 40 epochs. τ_0, τ_1, and M in Eq. 3 were empirically set to 1, 10^{-5}, and 10, respectively.

I.IV. Experiments on Point Cloud Classification

Following [3, 4], we sampled 1024 points from each object as the input of the network. During training, batch size was set to 24. The Adam [2] method with $\beta_1 = 0.9$ and $\beta_2 = 0.99$ was used for optimization. A cross-entropy loss was used as the loss function. The learning rate was initialized as 0.001 and multiplied with 0.7 after every 20 epochs. All models were trained for 200 epochs. τ_0, τ_1, and M in Eq. 3 were empirically set to 1, 10^{-3}, and 50, respectively.

II. Gradient Derivation

The gradients of lookup tables (*e.g.*, $\frac{\partial \bar{a}}{\partial p_1}$), float values (*e.g.*, $\frac{\partial \bar{a}}{\partial a}$), and scale parameters (*e.g.*, $\frac{\partial \bar{a}}{\partial s_a}$) are derived as follows. Using these gradients, the lookup tables can be optimized with the network in an end-to-end manner.

$$\begin{aligned}\frac{\partial \bar{a}}{\partial p_1} &= \frac{\partial(s_a \cdot \mathbb{Q}(\hat{a}, Q_a))}{\partial p_1} \\ &= s_a \frac{\boxed{\partial \mathbb{Q}(\hat{a}, Q_a)} = p_1 (\text{Sec. 3.3})}{\partial p_1} \\ &= s_a\end{aligned} \tag{I}$$

$$\begin{aligned}\frac{\partial \overline{a}}{\partial a} &= \frac{\partial(s_a \cdot \mathbb{Q}(\hat{a}, Q_a))}{\partial a} \\ &= s_a \frac{\partial \mathbb{Q}(\hat{a}, Q_a)}{\partial a} \\ &= s_a \underbrace{\boxed{\frac{\partial \mathbb{Q}(\hat{a}, Q_a)}{\partial \hat{a}}}}_{=1(\text{STE})} \frac{\partial \hat{a}}{\partial a} \\ &= s_a \frac{\partial \hat{a}}{\partial a} \\ &= s_a \frac{\partial \text{clip}(a/s_a)}{\partial a} \\ &= \begin{cases} 1, & \text{if } a < s_a \\ 0, & \text{otherwise} \end{cases}\end{aligned} \tag{II}$$

$$\begin{aligned}\frac{\partial \overline{a}}{\partial s_a} &= \frac{\partial(s_a \cdot \mathbb{Q}(\hat{a}, Q_a))}{\partial s_a} \\ &= \frac{\partial s_a}{\partial s_a}\mathbb{Q}(\hat{a}, Q_a)) + s_a \frac{\partial \mathbb{Q}(\hat{a}, Q_a)}{\partial s_a} \\ &= \underbrace{\boxed{\mathbb{Q}(\hat{a}, Q_a))}}_{=p_1(\text{Sec. 3.3})} + s_a \underbrace{\boxed{\frac{\partial \mathbb{Q}(\hat{a}, Q_a)}{\partial \hat{a}}}}_{=1(\text{STE})} \frac{\partial \hat{a}}{\partial s_a} \\ &= p_1 + s_a \frac{\partial \text{clip}(a/s_a)}{\partial s_a} \\ &= \begin{cases} p_1 - \frac{a}{s_a}, & \text{if } a < s_a \\ p_1, & \text{otherwise} \end{cases}\end{aligned} \tag{III}$$

III. Additional Analyses

In this section, we first conduct experiments to investigate the hyper-parameters in our temperature scheduler (Eq. 6). Then, we provide additional analyses regarding the evolution of lookup tables during training. Next, we conduct efficiency evaluation of our lookup tables. Finally, we discuss the difference of our method with several previous relevant approaches.

III.I. Temperature Scheduler

We use different combinations of hyper-parameters to train quantized ResNet-20 on CIFAR-10 and then compare their performance. Comparative results are listed in Table I. We first trained quantized ResNet-20 using different M. It can be observed that the model with $M = 50$ achieves comparable or better performance for different bit-widths. Consequently, M is set to 50 as the default setting. Then, we fixed M and trained quantized ResNet-20 using different τ_1. Since the model with $\tau_1 = 10^{-3}$ produces higher overall accuracy, it is used as the default setting.

Table I. Top-1 accuracy (%) achieved by our quantized ResNet-20 with different settings of the temperature scheduler.

Model	τ_0	τ_1	M	Bit-width (W/A)		
				4/4	3/3	2/2
ResNet-20 (FP: 92.96))			25	92.62	92.10	90.12
	1	10^{-3}	50	**92.71**	92.17	**90.63**
			75	92.59	**92.20**	90.22
		10^{-4}		92.66	92.05	90.42
	1	10^{-3}	50	**92.71**	92.17	**90.63**
		10^{-2}		92.68	**92.39**	90.39

III.II. Evolution of Lookup Tables

We study the evolution of lookup tables for (1) weights and activations, (2) different intervals of float values, and (3) different layers. Specifically, we compare the number of quantization levels in our lookup tables at different epochs during training. Results are shown in Fig. I.

Weights vs. Activations. From Fig. I(a), we can see that the evolution of lookup tables for weights and activations are quite different. Particularly, the evolution of lookup tables for weights starts earlier than those for activations. However, the lookup tables for activations converge faster than those for weights. We suppose that since the distributions of activations are usually less steeper than weights, the lookup tables for activations are easier to be optimized.

Different Intervals of Float Values. From Fig. I(b) and (c), we can see that the evolution of different intervals in our lookup table varies a lot. The interval near 1 starts earlier and converges faster than the interval near 0. We suppose that since more float values fall into the intervals near 0, these intervals are more difficult to be optimized.

Different Layers. As shown in Fig. I(d) and (e), the evolution of lookup tables in various layers are also different. The difference is more significant for lookup tables of weights. Specifically, the evolution of lookup tables starts from shallow layers and then gradually goes to deeper layers.

Overall, the evolution of our lookup tables is adaptive to the distributions of float values in different intervals and layers such that better performance can be achieved.

III.III. Efficiency Evaluation

We use 4-bit ResNet-18 to evaluate the efficiency of our lookup tables on different types of hardwares. Detailed experimental settings are as follows:

- GPU (RTX 2080Ti): Win10, CUDA 10.1, PyTorch 1.1.0
- CPU (Intel i9-9900K): Win10, PyTorch 1.1.0
- Mobile Processor (Kirin810): HarmonyOS 2.0.0
  - ∗ Mobile CPU: PyTorch Mobile ∗ Mobile NPU: TFLite + NNAPI

Using simple linear quantizers, PACT has shorter inference time but its accuracy is limited (Table II). Using

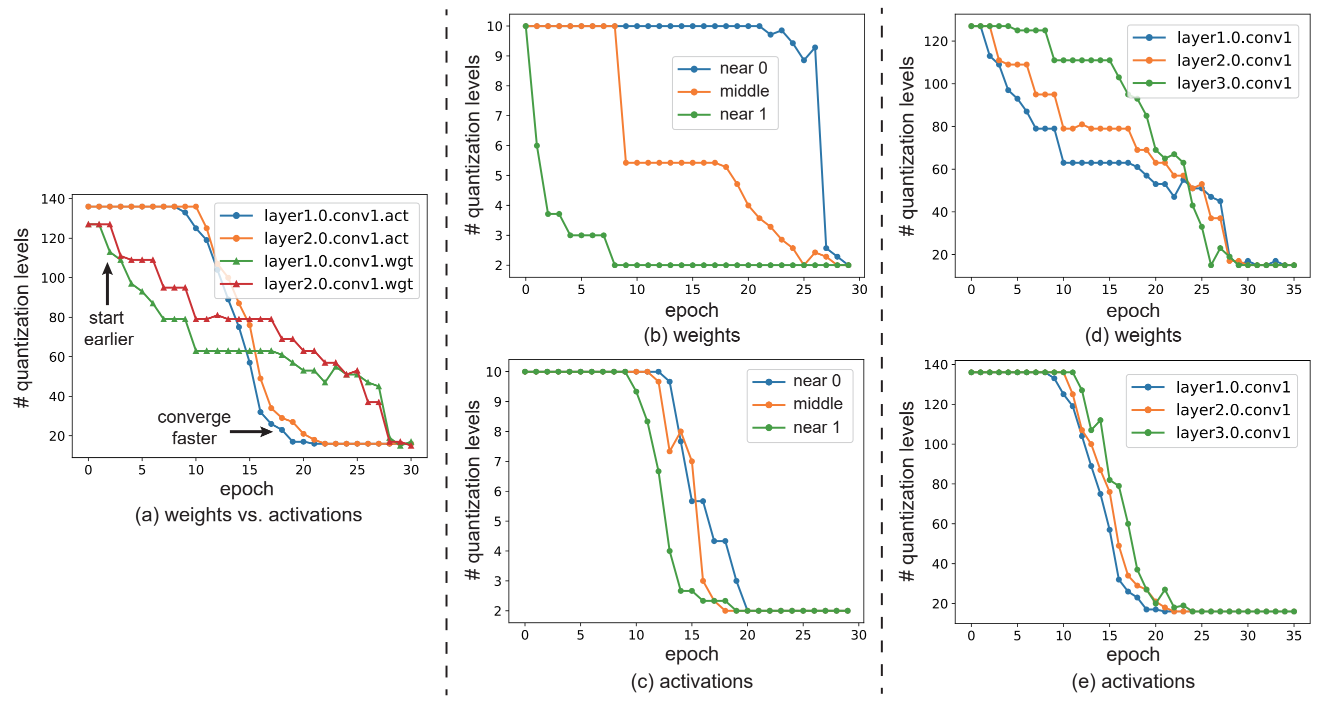

Figure I. Evolution of the number of quantization levels in the lookup tables for 4/4-bit quantization. We compare the number of quantization levels in the lookup tables for (a) weights and activations, (b-c) different intervals of float values, and (d-e) different layers.

Table II. Results achieved by 4-bit ResNet-18 on ImageNet.

Method	Model Size	Time (Online Activation Quantization)				Acc
		GPU	CPU	M-CPU	M-NPU	
PACT	7553.7KB	1ms	11ms	20ms	10ms	69.2
QIL	7553.7KB	3ms	90ms	120ms	120ms	70.1
QNet	7553.7KB	2ms	85ms	95ms	100ms	69.7
LLT (Ours)	7554.9KB	2ms	20ms	40ms	24ms	**70.4**

learnable quantizers, QIL and QNet produce higher performance. However, their complicated quantizers introduce considerable computational overhead. In contrast, our LLT has very small additional computational cost and achieves higher accuracy while being efficient enough. This further demonstrates the practicability of our method.

From Table II we can further see that our method is hardware friendly and achieves promsing efficiency on different types of general hardwares. On GPUs, our LLT is as fast as other methods. On other devices with limited resources where quantized networks are usually deployed, our LLT achieves better accuracy with higher efficiency as compared to QIL and QNet. We believe the advantages of our method can be further increased on specially-designed hardwares with better support of lookup operations.

III.IV. Difference with Previous Works

Similar to our method, [5] also addresses the problem of learning thresholds during network quantization. However, our method has three major differences. **1) Motivation:** [5] aims to partition float parameters into balanced bins to achieve histogram equalization of quantized values. In contrast, our method aims to use a learnable lookup table as the quantization function to adapt to the bell-shaped distributions of float values. **2) Methodology:** [5] uses distorted weights and percentiles to calculate quantized values while our method learns a lookup table for quantization. **3) Applicability:** [5] can only be used to quantize weights while our method can be used to quantize both weights and activations.

Although [1] makes similar attempts to use lookup tables for network quantization, our method has two major differences. **1) Methodology:** [1] uses k-means method to calculate quantized weights while our method is fully differentiable and can be trained in an end-to-end manner. **2) Applicability:** [1] can only be used to quantize weights while our method can be used to quantize both weights and activations.

Furthermore, we compare our method to [5] and [1] on ImageNet. It can be observed from Table III that our method produces much better performance for different bit-widths.

Table III. Top-1/Top-5 accuracy (%) on ImageNet.

Method	32-bit	4-bit	2-bit
[5]	69.8/89.1	- / -	59.4/82.0
[1]	69.8/89.1	68.4/ -	64.2/ -
LLT (Ours)	69.8/89.1	**70.4/89.6**	**66.0/86.2**

IV. Potential Negative Societal Impact

One potential negative impact would be that the characteristics (*e.g.*, the statistics of weights as shown in Fig. 6) of a quantized network could be altered from its baseline model due to the quantization error inside the network. Consequently, related techniques (*e.g.*, existing network initialization methods and BN layers) may not be directly used for quantized models. However, a thorough analysis on the changes of characteristics is out of the scope of this paper.

Acknowledgments

The authors would like to thank anonymous reviewers for their insightful suggestions. Xiaoyu Dong was supported by RIKEN Junior Research Associate Program.

References

[1] Fabien Cardinaux, Stefan Uhlich, Kazuki Yoshiyama, Javier Alonso García, Lukas Mauch, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Iteratively training look-up tables for network quantization. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):860–870, 2020. 3

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1

[3] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 1

[4] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *CVPR*, pages 3173–3182, 2021. 1

[5] Shu-Chang Zhou, Yu-Zhi Wang, He Wen, Qin-Yao He, and Yu-Heng Zou. Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, 32(4):667–682, 2017. 3