Centered Symmetric Quantization for Hardware-Efficient Low-Bit Neural Networks: Supplementary Materials

Faaiz Asim*[1]
faaizasm@unist.ac.kr
Jaewoo Park*[2]
hecate64@unist.ac.kr
Azat Azamat[3]
azatkariuly@unist.ac.kr
Jongeun Lee†[1]
jlee@unist.ac.kr

Dept. of Electrical Engineering[1]
Dept. of Physics[2]
Dept. of CSE[3]
Ulsan National Institute of Science and Technology (UNIST)
Ulsan, Korea

A Variations of CSQ in Previous Works

As defined in Appendix B CSQ stipulates uniform step size and perfect symmetry between positive and negative quantization levels. Therefore, CSQ should not be restricted to the quantizer formulated in (4). CSQ should be considered as a family of quantizers with uniform step size and perfectly symmetric quantization levels. [2, 3, 4, 7, 10] all belong to the CSQ family.

[4] statistically computes the optimal scale for quantization bins using (14). c_1 and c_2 are coefficients depending on the quantization precision.

$$\alpha_w^* = c_1 * \sqrt{E\left(w^2\right)} - c_2 * E(|w|) \tag{14}$$

Their quantizer results in perfectly symmetric quantization levels because they use same scale for positive and negative range of weights distribution. Therefore their quantizer belongs to the the CSQ family.

[7] uses a soft quantizer that converges to CSQ during training. Their quantization is given by (15).

$$Q_S(x) = \begin{cases} l, & x < l \\ u, & x > u \\ l + \Delta\left(i + \frac{\varphi(x)+1}{2}\right), & x \in \mathcal{P}_i \end{cases} \tag{15}$$

*equal contribution
†Jongeun Lee is the corresponding author (E-mail: jlee@unist.ac.kr).

Where $\varphi(x)$ is a differential asymptotic function used to approximate uniform quantizer. They use tanh function to approximate the quantizer which results in perfectly symmetric quantization levels.

[2] proposes a weight quantizer similar to PACT [5]. They parameterize and learn the clipping parameter during training. Their quantizer is shown in (19).

$$\hat{w} = 0.5\left(|w+\alpha_w| - |w-\alpha_w|\right) \tag{16}$$

$$\hat{w}' = \frac{\hat{w}}{2\alpha_w} + 0.5 \tag{17}$$

$$Q(w)' = \left\lceil \hat{w}' \cdot (2^n - 1) \right\rfloor / (2^n - 1) \tag{18}$$

$$Q(w) = 2\alpha_w \left(Q(w)' - 0.5\right) \tag{19}$$

Their quantizer has the same clipping parameter for positive and negative range of distribution. Therefore, their quantizer is perfectly symmetric and behaves same as CSQ defined in (4) during inference.

The quantizer proposed in [10] also has perfectly symmetric quantization levels. Their quantizer is formulated as (22).

$$x_n = \text{clip}\left(\frac{x-l}{u-l}, 0, 1\right) \tag{20}$$

$$x_q = \frac{\text{round}\left(\left(2^b-1\right)x_n\right)}{2^b-1} \tag{21}$$

$$Q_W(x) = 2\left(x_q - 0.5\right) \tag{22}$$

Because, [10] normalizes weights before quantization in (20), their quantizer is invariant to any affine transformation applied to the input weights. This behaviour is unique to [10] and is not shared by any quantizer in CSQ family. However, since the weights distribution has near zero mean, the difference between [10] and other variants CSQ should be negligible in practice.

[3] parametrizes and learns the quantization interval. Their quantizer is formulated as (23).

$$\begin{aligned} \eta_w &= \left\lfloor \left(\text{clip}\left(\tfrac{w}{v_w}, -1, 1\right) + 1\right) / 2 \cdot \left(2^b - 1\right) \right\rceil \\ \bar{w} &= \left(\eta_w \cdot \tfrac{1}{2^b-1} \cdot 2 - 1\right) \cdot v_w \end{aligned} \tag{23}$$

They do not discuss the quantization levels resulting from their quantizer but their quantizer behaves same as the quantizer defined in (4).

B Alternatives to CSQ

From the perspective of quantization levels, one can consider the following quantization schemes **Extended Symmetric Quantization (ESQ)**, on the other hand, uses one more quantization level to achieve exact zero representation and symmetry; i.e., $L = -2^{b-1}$ and

$U = 2^{b-1}$. However, since ESQ requires more than b-bit precision, it is not feasible for practical deployment, but included here for comparison. **Non-uniform Symmetric Quantization (NSQ)** is same as ESQ without zero representation. NSQ achieves perfect symmetry with exactly 2^b quantization levels, but the step size is not uniform, leading to a completely different quantizer function and QAT methods. Since NSQ is non-uniform quantization, we do not consider it for the scope of this paper. **Centered Symmetric Quantization (CSQ)** stipulates *uniform step size* and *perfect symmetry* between the positive and negative sides, while compromising on the exact representation of zero (*e.g.* $\{-1.5,-0.5,0.5,1.5\}$ for 2-bit). It can also be represented using integers only by scaling all quantization levels by 2 ($\{-3,-1,1,3\}$ for 2-bit). For the remainder of the paper we focus on CSQ and CLQ, as they are the most practical.

C Our Training Method

We have employed the method presented in [6] to train the quantization parameters. To optimize the step size s using gradient descent, we use the following derivative formula.

$$\frac{\partial \hat{v}}{\partial s} = \begin{cases} -v/s + \dot{v} & \text{if } -Q < \dot{v} < Q \\ -Q & \text{if } \dot{v} \leq -Q \\ Q & \text{if } \dot{v} \geq Q \end{cases} \tag{24}$$

Then computing the loss gradient w.r.t. step size is straightforward. Similar to gradient scaling in [6], the gradient of step size is scaled by factor $g = 1/\sqrt{N_W 2^p}$, where N_W is the number of weight parameters. The weights are initialized as $2\langle|v|\rangle/\sqrt{Q}$, where $\langle.\rangle$ represents the notation for mean of a distribution. Figure 2b shares the gradient of step size parameter at 2-bit precision. Our training method is based on LSQ [6] but it should be noted that CSQ can be used with any training method.

D Limitaions of Affine Quantizer

We have defined affine quantizer by [9] in (7). Another variant of affine quantizer has also been proposed by [1]:

$$\bar{x} = \left\lfloor \text{clip}\left(\frac{x-z}{s}, n, p\right)\right\rceil, \quad \hat{x} = s \cdot \bar{x} + z \tag{25}$$

where n and p are clipping intervals. [1] does not restrict their zero-point to integer values. This approach can lead to significant hardware overhead. To tackle this problem they propose the affine quantizer for activations only, in which case zero-point can be implemented simply as a bias term, since

$$\hat{w} \cdot \hat{x} = (\bar{w} \cdot s_w)(\bar{x} \cdot s_x + z) = \bar{w}\bar{x}s_w s_x + \underbrace{z s_w \bar{w}}_{\text{bias}} \tag{26}$$

and $\bar{w}$ is known in advance. This resolves the hardware overhead problem of affine quantization for activation. However, the same method cannot be applied to weight because it would involve pre-computing $\bar{x}$ in advance, which is impossible.

Affine quantization for weights still results in some hardware overhead as the the zero-point of weights quantizer can not be implemented as the bias. This can be shown as:

$$\hat{w}\hat{x} = (\bar{w} \times s_w + z_w)(\bar{x} \times s_x + z_x) = \bar{w}\bar{x}s_w s_x + \underbrace{z_x s_w \bar{w} + z_w z_x}_{\text{bias}} + \underbrace{z_w s_x \bar{x}}_{\text{overhead}} \tag{27}$$

This is because $\bar{x}$ can not be pre-computed as it will change with each input. Similarly weight quantization overhead is shown by [9] as:

$$y(k,l,n) = s_w s_x \,\text{conv}\,(\bar{w}(k,l,m;n) - z_w, \bar{x}(k,l,m) - z_x) \tag{28}$$

$$y(k,l,n) = \text{conv}\,(\bar{w}(k,l,m;n), \bar{x}(k,l,m)) - z_w \sum_{k=0}^{K-1}\sum_{l=0}^{K-1}\sum_{m=0}^{N-1} \bar{x}(k,l,m) \tag{29}$$

$$- z_x \sum_{k=0}^{K-1}\sum_{l=0}^{K-1}\sum_{m=0}^{N-1} \bar{w}(k,l,m;n) + z_x z_w \tag{30}$$

Considering these limitations of affine quantizer we have proposed CSQ as an approximation of affine quantizer which can be efficiently realized on hardware.

E Derivation of CSQ Hwardware Realization via Affine Quantizer

Let us first consider the multiplication between two values $\hat{w}, \hat{x}$ quantized by affine quantizer. Here we use relaxed affine quantizer because CSQ cannot be represented by integer zero-point. The multiplication operation can be computed as follows:

$$\hat{w}\hat{x} = ((\bar{w} - z_w) \times s_w)((\bar{x} - z_x) \times s_x) = \bar{w}\bar{x}s_w s_x - \underbrace{z_x s_w s_x \bar{w} + z_w z_x s_w s_x}_{\text{bias}} - \underbrace{z_w s_x s_x \bar{x}}_{\text{overhead}} \tag{31}$$

where $\bar{w}\bar{x}$ can be computed using a standard multiplier without any overhead and $z_x s_w s_x \bar{w} + z_w z_x s_w s_x$ can be computed in advance and folded into the bias term, causing no runtime overhead. However, the term $z_w s_x s_x \bar{x}$ cannot be computed in advance and must result in runtime overhead.

In Section 3.3 we show that CSQ can be considered as a special case of relaxed affine quantizer with a specific zero-point value. Based on this, we propose an efficient method to realize CSQ on standard multipliers in Section 4.1. The method in (9) can be shown for full convolution layer as:

$$y(k,l,n) = \bar{s}_w \bar{s}_x \,\text{conv}\,(\bar{w}(k,l,m;n), \bar{x}(k,l,m)) - \underbrace{0.5 \times \bar{s}_w \bar{s}_x \sum_{k=0}^{K-1}\sum_{l=0}^{K-1}\sum_{m=0}^{N-1} \bar{x}(k,l,m)}_{\text{overhead}} \tag{32}$$

This method results in marginal overhead but it is significantly lower than that of (relaxed) affine quantizer because multiplying by 0.5 does not need any logic gate (just wires are enough) and there is no real-valued parameter unlike relaxed affine quantizer. This method can be used to realize CSQ on standard multipliers with very small overhead, as demonstrated by our experiments in Section 5.3.

Table 5: Comparison of CSQ with affine quantizer using BRECQ on ImageNet.

		Accuracy		
Network	**Method**	2/32	3/32	4/32
ResNet-18		*Full Precision: 71.08*		
	BRECQ	66.60	**69.82**	**70.64**
	BRECQ-CSQ (ours)	**66.93**	69.81	*70.62*
ResNet-50		*Full Precision: 77.00*		
	BRECQ	72.16	**75.68**	**76.41**
	BRECQ-CSQ (ours)	**72.24**	75.52	76.35

F Additional Experiments

F.1 Post Training Quantization Results (Comparison with BRECQ)

BRECQ [] uses affine quantizer with integer zero-point, which extends the linear quantizer in a different direction than our CSQ. Thus it is very interesting to see how our method compares with BRECQ. Note that integer zero-point requires some additional hardware overhead whereas CSQ does not. At the same time, the two schemes are orthogonal in the sense that one can combine both schemes, viz. integer zero-point and CSQ, so that zero-point take any integer or half-integer value, at the cost of some hardware overhead. Here we compare only CSQ vs. BRECQ, but not the combination. Since we have proposed CSQ for weight quantization, we only quantize the weights in BRECQ experiments.

To implement CSQ we simply fix the zero point value to CSQ which is shown as Z_{CSQ} in (8). The shared results may be different from reported results. The results have been reproduced using the officially shared code. We share our results on ResNet-18 and ResNet-34. We used channel-wise quantization and all the hyper-parameters are same as shown in []. Table 5 shows the experimental results using BRECQ on ImageNet data. It can be seen that at 2-bit precision CSQ outperforms affine quantizer. This is especially interesting because unlike affine quantization, CSQ does not incur any hardware overhead. At 3 and 4-bit precision, affine quantizer gives better performance. However, CSQ also provides competitive results. This shows that CSQ is a very strong approximation of (relaxed) affine quantizer.

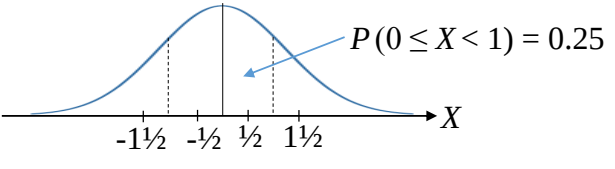


Figure 5: With 2-bit CSQ, quantization levels (shown on the x-axis) can resemble Gaussian distribution while, at the same time, real-valued weight data are also uniformly mapped to them (scale parameter s is omitted for brevity).

Table 6: Comparison of CSQ vs. CLQ using knowledge distillation training on ImageNet.

	Top-1 Accuracy @ Precision		
Network	ResNet-18		
	Full Precision: 70.52		
Precision (W/A)	2/2	3/3	4/4
CLQ (LSQ) + Knowledge Distillation	66.99	69.77	**70.63**
CSQ + Knowledge Distillation (ours)	**67.24**	**69.90**	70.56

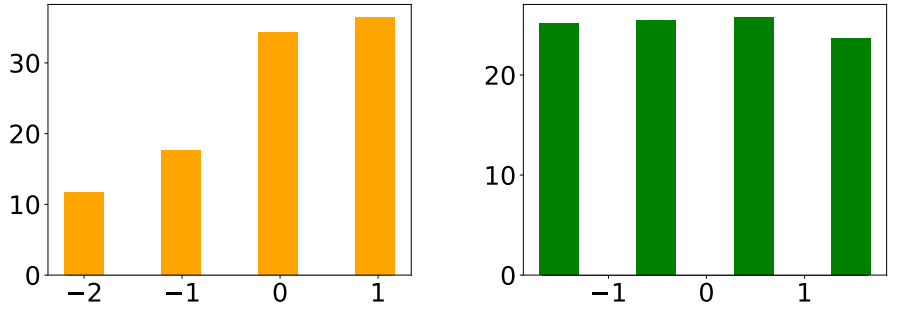
F.2 Additional Experiments Using Knowledge Distillation

To further demonstrate the generalization-ability of CSQ, we conduct some experiments with knowledge distillation [8] loss. We use LSQ [6] for training the quantization parameters and show that CSQ performs superior to CLQ at low precision even when we use advanced training methods such as knowledge distillation over state-of-the-art quantization-aware training method.

The experimental methodology and training setups are the same as described in Section 5.1, except that we use a weight decay of 0.25e-4 and knowledge distillation loss for all experiments. For knowledge distillation loss we set temperature as 1 and give equal weight to the standard loss and the distillation loss following [6]. Our experimental results in Table 6 show that at extremely low precision, i.e., at 2- and 3-bit, CSQ outperforms CLQ. This is consistent with experimental results in Table 3.

G Why CSQ is always better than CLQ at 2-bit

In the weight quantization error experiment of Section 3.1, optimizing for minimum quantization error faces the challenge of maximizing the utility of limited quantization levels. The utility is maximized (i) if each quantization level has equal number of real values mapped to it, in the same way as the Lloyd-Max quantization [17] is optimal. Also it can be helped a lot (ii) if the quantization levels have the same distribution as the underlying data. In other words, for the first objective the underlying data (when mapped to quantization levels) should be distributed as uniformly as possible, while for the second objective the distribution of quantization levels should resemble that of the underlying data (*e.g.*, Gaussian). This usually creates conflicting requirements, but not in the case of 2-bit CSQ. At 2-bit, CSQ has only three quantization thresholds, $\{-1,0,1\}$, and therefore can satisfy both requirements simultaneously: (i) real-valued weight data are uniformly distributed across quantization levels, and at the same time, (ii) quantization levels follow Gaussian (or any symmetric) distribution, as illustrated in Figure 5. Note that this is not possible for 2-bit CLQ, as in other precision for either CLQ or CSQ, which explains why CSQ always shows better performance than CLQ at 2-bit. From the figure, the scale parameter can be determined as $s = \Phi^{-1}(0.75)$, where Φ is the CDF of standard normal distribution, since $P(0 \leq X < s) = 0.25$ when $X \sim N(0,1)$). Also our experimental results confirm that indeed real-valued weight data are mapped uniformly to 2-bit CSQ quantization levels as shown in Figure 6.



(a) CLQ 2-bit (b) CSQ 2-bit

Figure 6: How real-valued weight data are mapped to CLQ vs. CSQ after training ResNet-18 on ImageNet.

H Exhaustive Search Method for Quantization Error Experiments

We present quantization error experiments in Section 3.1. To find the step size that minimizes the Quantization Error we use an exhaustive search method. Our exhaustive search method is very similar to the method used by [8] for their quantization error experiments. The exhaustive search goes as follows. First, we initialize step sizes as:

$$s_0 = \frac{\langle |w| \rangle}{2^p - 1} \tag{33}$$

where w represents full precision weights,$\langle . \rangle$ represents the mean operation, and p is the bit-width. Then for the search space $S = \{0.01s_0, 0.02s_0, 0.03s_0, \cdots, 5s_0\}$, we exhaustively find the value of $s \in S$ that minimizes the target quantization error metric. This helps us find the minimum quantization error using CLQ and CSQ, for any given bit-width. Experimental results and analysis has been presented in Section 3.1.

I More about Zero-point

The difference between z_{CLQ} and z_{CSQ} is constant at 0.5 (see (8)) while the possible range of values that can be taken by the quantizer grows exponentially with b. Thus we can give the percentage difference between zero-points of CLQ vs. CSQ relative to the entire quantization range by:

$$D = \frac{z_{\text{CLQ}} - z_{\text{CSQ}}}{2^b} \times 100 = \frac{1}{2^{b+1}} \times 100 \tag{34}$$

This equation implies that D decreases as we increase the precision. For example, at 2-bit precision, D is 12.5% but at 4-bit precision, D reduces to mere 3.125%. This shows that at higher precision the difference between CSQ and CLQ becomes negligible compared to the entire distribution range and they should provide similar performance.

J Exact Zero Representation vs. Perfect Symmetry

Any linear quantization to b-bit precision results in 2^b quantization levels. Therefore, inclusion of zero among the quantization points, naturally results in an asymmetry in the number of positive and negative quantization levels. While exact representation of zero may be important due to common operations like zero padding [9], it is also true that the asymmetry among positive and negative quantization levels grows large as precision becomes low. Thus one of our aims in this paper is to explore the trade-off between exact zero representation and perfect symmetry in the context of weight quantization of neural networks.

J.1 Zero Representation

CSQ does not provide an exact representation for zero. Instead, zero or values slightly less than zero are rounded to -0.5 while values slightly greater than zero are rounded to 0.5 (before scale factor). Despite this it has been shown in Section 3.1 that CSQ can reduce quantization error compared with CLQ. Furthermore, our experimental results have also shown superior performance with CSQ. This serves as an evidence that exact zero representation is not critical for weight quantization, especially at low precision (< 4 bits).

J.2 Zero Padding

Zero padding is needed for activation only, not for weight. In case, if zero padding were used for weight as well, we can simulate the exact effect of rounding zeros during QAT, and adjust weight accordingly. So it would not contribute to any performance degradation.

Now, vector and tensor processors (*e.g.* TPU) must process an array of values together, and may "fill" some elements with zeros as needed. This *zero filling* is needed for both weight and activation. For CSQ weight, inexact zero representation may introduce an error or discrepancy between algorithm and realization. This error can be eliminated or minimized, depending on hardware dataflow, by resetting corresponding activation values to zeroes or filling with both $+0.5$ and -0.5.

J.3 Why Propose CSQ for Weight Only

Activation frequently involves zero padding. Therefore, activation quantization strictly demands exact zero representation to support zero padding. Furthermore ReLU is frequently used activation function which results in unsigned activation distribution, whereas CSQ can only be used for signed distributions. Therefore, we recommend that using CSQ for weight quantization and CLQ for activation quantization, which provides the best performance with no hardware overhead.

References

[1] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–697, 2020.

[2] Yoonho Boo, Sungho Shin, Jungwook Choi, and Wonyong Sung. Stochastic precision ensemble: self-knowledge distillation for quantized deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6794–6802, 2021.

[3] Peng Chen, Jing Liu, Bohan Zhuang, Mingkui Tan, and Chunhua Shen. Aqd: Towards accurate quantized object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 104–113, 2021.

[4] Jungwook Choi, Pierce I-Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Bridging the accuracy gap for 2-bit quantized neural networks (qnn). *arXiv preprint arXiv:1807.06964*, 2018.

[5] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[6] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2019.

[7] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4852–4861, 2019.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

[10] Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6448–6457, 2021.

[11] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

[12] Joel Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12, 1960.