Post-Training Piecewise Linear Quantization for Deep Neural Networks *(Supplementary Material)*

Jun Fang[1], Ali Shafiee[1], Hamzah Abdel-Aziz[1], David Thorsley[1], Georgios Georgiadis[2⋆], and Joseph H. Hassoun[1]

[1] Samsung Semiconductor, Inc.
{jun.fang, ali.shafiee, hamzah.a, d.thorsley, j.hassoun}@samsung.com
[2] Microsoft georgios.georgiadis@microsoft.com

In this supplement, we expand our discussion on piecewise linear quantization (PWLQ) with additional analysis and experiments. In particular, we discuss the following:

- Section 1 provides details about the algorithm of finding optimal breakpoints.
- Section 2 reports additional experimental results of activation quantization.
- Section 3 discusses the details of hardware implementation for PWLQ. The impact on energy and latency is provided.

1 Optimal Breakpoint Inference

To best apply PWLQ, we need to find the optimal breakpoint. One approach is to assume that weights and activations satisfy Gaussian or Laplacian distributions, and then use the formulated PDF/CDF to solve the convex problem of minimizing the quantization error by gradient descent.

However, the iterative process of gradient descent increases the computation complexity and the overall latency during inference. Therefore, it is better to use a simple and fast one-shot approach to approximate the optimal breakpoint, provided the approximation error is small.

1.1 Derivation of Approximation Formula

We first collect the maximum over standard deviation of per-channel weights in a pre-trained Inception-v3 model, which is computed as $\max(W_i[j,:,:,:])/\text{std}(W_i[j,:,:,:])$ to estimate the range value m for the normalized quantization range $[-m, m]$, where W_i is the weight tensor at i-th layer and $W_i[j,:,:,:]$ is the weight sub-tensor at j-th output channel of W_i. In Figure 1, we found that over 95% of m values are located between 3 and 10.

We then use gradient descent under the Gaussian assumption to find optimal breakpoints for the quantization range values $m \in [3, 10]$. Using the gradient descent results, we compute the following approximation of the optimal breakpoint for a normalized Gaussian distribution: $p^*/m = \ln(0.8614m + 0.6079)$.

⋆ Work performed while at Samsung Semiconductor, Inc.

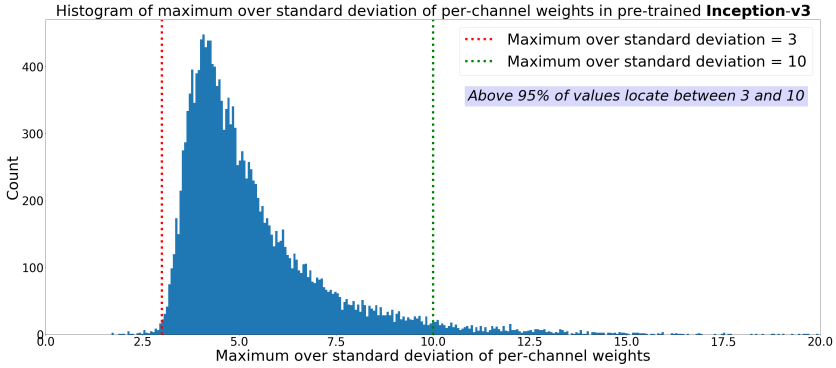

Fig. 1. Histogram of maximum over standard deviation of per-channel weights in a pre-trained Inception-v3 model. More than 95% of the values are located between 3 and 10

Figure 2 indicates how accurate our approximated versions of the optimal breakpoints are for $m \in [3, 10]$ under the Gaussian distribution assumption. Furthermore, experimental results in Table 1 show that the approximations obtain almost the same quantization error and same top-1 accuracy compared to gradient descent on the piecewise linearly quantized model.

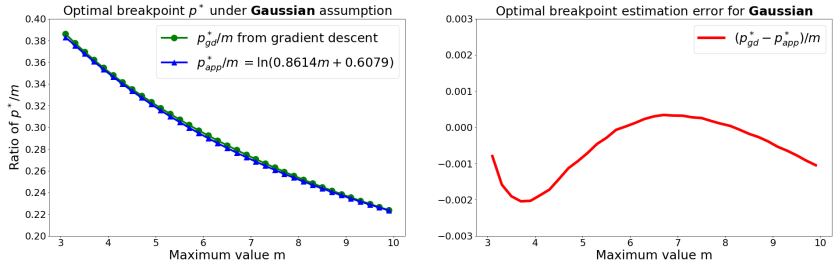

Fig. 2. Approximation formula of the optimal breakpoint p for Gaussian distributions. Left: approximation formula of p in term of the ratio p/m. Right: approximation error. Notation p_{gd}: optimal breakpoint found by gradient descent. Notation p_{app}: optimal breakpoint approximated by the one-shot formula.

1.2 PWLQ with Other Assumptions

Just as discussed above for Gaussian distributions, we also extend PWLQ to Laplacian distributions. The approximated version of the optimal breakpoint for a normalized Laplacian is: $p^*/m = 0.8030\sqrt{m} - 0.3167$. We show the approximation error of the Laplacian case in Figure 3. It has the same magnitude of approximation error as the Gaussian case in Figure 2.

Table 1. Different methods of finding the optimal breakpoint for PWLQ: gradient descent and approximation under Gaussian distribution assumption on Inception-v3, top-1 accuracy baseline is 77.49%. MSE: mean squared error of all quantized weights.

Breakpoint Methods	8-bit		7-bit		6-bit		5-bit		4-bit	
	Top1%	MSE	Top1%	MSE	Top1%	MSE	Top1%	MSE	Top1%	MSE
Gradient Descent	77.51	7.56e-8	77.50	3.07e-7	77.40	1.27e-6	77.16	5.41e-6	75.74	2.30e-5
Approximation	77.52	7.57e-8	77.49	3.08e-7	77.42	1.27e-6	77.15	5.42e-6	75.72	2.31e-5

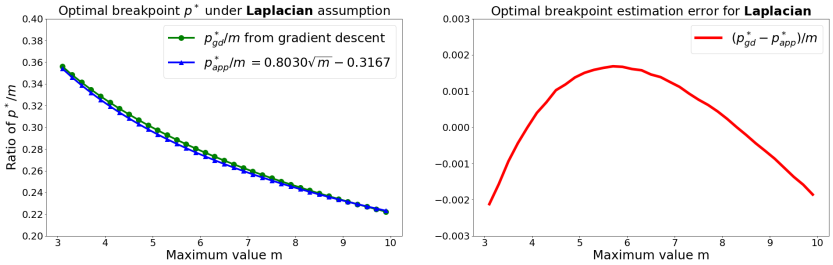

Fig. 3. Approximation formula of the optimal breakpoint p for Laplacian distributions. Left: approximation formula of p in term of the ratio $p = m$. Right: approximation error. Notation p_{gd}: optimal breakpoint found by gradient descent. Notation p_{app}: optimal breakpoint approximated by the one-shot formula.

Table 2. Comparison results of top-1 accuracy (%) for PWLQ with different assumptions. Gaussian/Laplacian/Search are three optimal breakpoint selection methods with Gaussian/Laplacian or no specific distribution assumptions, respectively.

Network	Method	8-bit	8+BC	6-bit	6+BC	4-bit	4+BC
Inception-v3 (77.49)	Gaussian	77.52	77.53	77.42	77.48	75.72	**76.45**
	Laplacian	77.48	77.53	77.44	**77.52**	75.67	75.97
	Search	77.52	**77.55**	77.35	77.39	75.34	76.40
ResNet-50 (76.13)	Gaussian	76.10	**76.10**	76.03	76.08	74.28	75.62
	Laplacian	76.08	76.07	76.03	**76.13**	73.03	75.60
	Search	76.08	76.10	76.08	76.09	72.55	**75.71**
MobileNet-v2 (71.88)	Gaussian	71.59	**71.73**	70.82	71.58	54.34	69.22
	Laplacian	71.65	71.67	71.20	**71.60**	53.70	68.99
	Search	71.64	71.63	70.75	71.53	56.13	**69.28**

Another approach is a simple three-stage coarse-to-fine grid search algorithm 1 to minimize the quantization error without any need of specific distribution assumptions. Compared to the breakpoint inference methods based on Gaussian or Laplacian assumptions, the coarse-to-fine search is generally guaranteed to achieve smaller quantization error at a cost of longer search time. We provide a comparison result for all three methods in Table 2. There is no clear evidence in this table to show that one of them generally performs better than the other two methods. We therefore only report the results with Gaussian assumptions to show the robustness of the PWLQ approach.

Algorithm 1: Coarse-to-fine search for the optimal breakpoint.

```
Input: A tensor T need to be quantized
Output: Optimal breakpoint p
p = 0.5;   m = max(abs(T))
for stage in [1,2,3] do
    grid = 0.1^stage
    range = 5 if stage == 1 else 10
    start = p =m - grid * range
    end = p =m + grid * range
    for p =m in [start: grid: end] do
        Apply piecewise linear quantization scheme
        Compute mean squared quantization error
        Select p =m having smallest quantization error
```

1.3 PWLQ with Multiple Breakpoints

In this section, we extend PWLQ to more general cases with n multiple breakpoints $\mathbf{p} = (p_1, p_2, ...p_n)$ under the feasible domain $\mathbf{D} = \{(p_1, p_2, ...p_n) \,|\, 0 < p_1 < p_2 < ... < p_n < m\}$. We denote by $p_0 = 0$, $p_{n+1} = m$ and calculate the expected squared quantization error similarly to Equation (4) in the main paper:

$$\mathsf{E}(\varepsilon_{pw}^2; b, m, \mathbf{p}) = C(b-1) \sum_{l=0}^{n} \Big\{ (p_{l+1} - p_l)^2 \big[F(p_{l+1}) - F(p_l) \big] \Big\}. \tag{1}$$

Accordingly, the optimal breakpoints $\mathbf{p}^* = (p_1^*, p_2^*, ...p_n^*)$ can be estimated by minimizing the expected squared quantization error:

$$\mathbf{p}^* = \underset{\mathbf{p} \in \mathbf{D}}{\arg\min}\, \mathsf{E}(\varepsilon_{pw}^2; b, m, \mathbf{p}). \tag{2}$$

In practice, we assume the tensor values satisfy Gaussian or Laplacian distribution with formulated PDF/CDF. We then solve the optimization problem (2) by using gradient descent. In our experiments with up to three breakpoints, the

gradient descent performs well and converges quickly. We also note the convexity of the quantization error with two-to-three breakpoints in our experiments and Figure 4 shows a typical example for two breakpoints. However, as we discussed in Section 4 of the main paper and Section 3 of the supplementary material, more breakpoints introduce more hardware overhead, we suggest using one breakpoint on weights to maintain the simplicity of the inference algorithm and the hardware implementation.

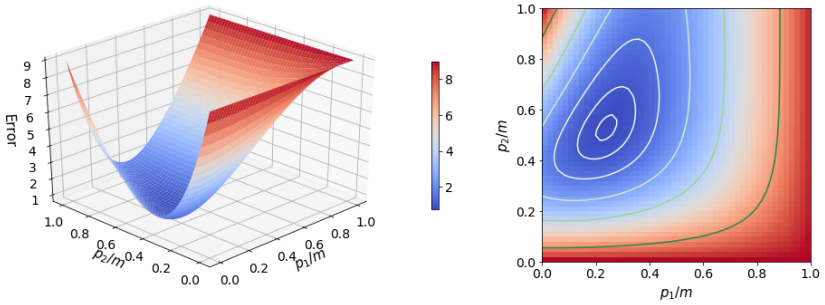

Fig. 4. Quantization error $\frac{1}{C(b\ \ 1)}\mathsf{E}(\ ''^2_{pw}; b; m; \mathsf{p})$ of PWLQ ($m = 3$) with two breakpoints under Gaussian distribution assumption. Left: 3D surface plot. Right: 2D contour plot.

2 Activation Quantization

In this section, we present more experimental results of activation quantization.

2.1 Activation Ranges

Before applying quantization to activations, we first need to find the activation range boundaries $[r_l, r_u]$. These ranges ($\Delta = r_u - r_l$) introduce two sources of error to the activation quantization: one is from the clipping of outliers, and another one is from the affine projection of floating-point to low-precision representations. Shorter quantization ranges increase the clipping error and decrease the projection error. Conversely, longer quantization ranges decrease the clipping error but increase the projection error. Therefore, a good quantization method on activations should balance the quantization error of clipping and projection.

We compare here the *top-k* median and percentile-based approaches to clip the activation ranges. As explained in the main paper, the *top-k* median method computes the median of the *top-k* smallest and *top-k* largest of the sorted activation values X_{sort}, i.e., $r_l = \text{median}(X_{sort}[:k])$ and $r_u = \text{median}(X_{sort}[-k:])$. As opposed to computing the median, the percentile-based approach utilizes

a parameter γ to determine the range, i.e., $r_l = \gamma$-th percentile of X_{sort} and $r_u = (1-\gamma)$-th percentile of X_{sort}.

We show the experimental results for $k = 5, 10, 20$ and $\gamma = 0.1, 0.01, 0.001$ in Table 3. In most of the cases (5 out of 9), the top-10 median method achieves the best accuracy among all six configurations. Therefore, we only report the results obtained from the top-10 median method in this work.

Table 3. Top-1 accuracy (%) of different clipping methods for activation ranges. Weights are quantized per-channel into 4, 6, or 8 bits using PWLQ, activations are uniformly quantized per-layer into 8 bits

Method	Parameter	Inception-v3 (77.49)			ResNet-50 (76.13)			MobileNet-v2 (71.88)		
		8-bit	6-bit	4-bit	8-bit	6-bit	4-bit	8-bit	6-bit	4-bit
Top-k Median	$k = 5$	77.50	77.41	75.81	76.07	75.95	74.18	71.66	70.83	54.25
	$k = 10$	**77.52**	77.42	**75.75**	76.09	**76.05**	**74.28**	71.59	70.82	**54.34**
	$k = 20$	77.45	77.32	75.71	**76.12**	75.99	74.23	71.66	**70.90**	54.23
Percentile Based	$= 0.1$	75.85	75.78	73.78	75.27	75.26	73.29	70.91	70.12	51.91
	$= 0.01$	77.31	77.30	75.51	76.04	75.95	74.13	71.53	70.76	54.02
	$= 0.001$	77.53	**77.47**	75.76	76.10	75.96	74.22	**71.67**	70.72	54.27

2.2 PWLQ on Activations

Once we have the activation ranges, we can apply PWLQ on activations. We show the comparison results of applying uniform quantization and PWLQ *only* on activations in Table 4. In addition, we also provide the experimental results of applying PWLQ on *both* weights and activations in Table 5. Again, our PWLQ achieves notable superior performance against uniform quantization at a cost of hardware overhead.

Table 4. Top-1 accuracy (%) of different quantization methods for activations. Weights are uniformly quantized per-channel into 8 bits with bias correction, activations are quantized per-layer into 4, 6, or 8 bits

W/A Method	Inception-v3 (77.49)			ResNet-50 (76.13)			MobileNet-v2 (71.88)		
	8/8	8/6	8/4	8/8	8/6	8/4	8/8	8/6	8/4
Uniform / Uniform	77.52	76.86	63.36	76.14	75.61	62.35	71.58	67.64	1.41
Uniform / PWLQ	**77.53**	**77.45**	**75.95**	**76.15**	**76.09**	**75.27**	**71.76**	**71.40**	**67.14**

Similar to the case of applying PWLQ on weights as we discussed in the main paper, applying PWLQ *only* on activations with one breakpoint requires two separate computational paths on each non-overlapping activation region. While applying PWLQ on *both* weights and activations, we require computational paths

Table 5. Top-1 accuracy (%) of different quantization methods for both weights and activations. Weights are quantized per-channel with bias correciton and activations are quantized per-layer

W/A Method	Inception-v3 (77.49)			ResNet-50 (76.13)			MobileNet-v2 (71.88)		
	8/8	6/6	4/4	8/8	6/6	4/4	8/8	6/6	4/4
Uniform / Uniform	77.52	76.67	31.72	**76.14**	75.47	59.84	71.58	66.86	0.34
PWLQ / PWLQ	**77.53**	**77.49**	**74.91**	76.12	**76.08**	**74.85**	**71.84**	**71.45**	**63.64**

for each combination of weight regions and activation regions. We discuss the details more precisely in the following.

For the case of applying PWLQ with one breakpoint on both weights W and activations X, the algorithm breaks the ranges into non-overlapping regions R_{w_1}, R_{w_2} for weights and R_{x_1}, R_{x_2} for activations. We set offsets $z_{w_1} = 0, z_{w_2} = p_w, z_{x_1} = 0, z_{x_2} = p_x$ and denote scaling factors by $s_{w_1}, s_{w_2}, s_{x_1}, s_{x_2}$ in $R_{w_1}, R_{w_2}, R_{x_1}, R_{x_2}$, respectively. We also define by $\langle \cdot, \cdot \rangle_{R_{w_i;x_j}}$ the associated partial vector inner product, and W_{q_i}, X_{q_j} the associated quantized integer vector of W, X in region R_{w_i}, R_{x_j} for $i = 1, 2$ and $j = 1, 2$. Then each combination of weight region R_{w_i} and activation region R_{x_j} has a computational path $P_{i,j}$, which is calculated as:

$$\begin{aligned} P_{i,j} &= \langle s_{w_i} W_{q_i} + z_{w_i} I,\ s_{x_j} X_{q_j} + z_{x_j} I \rangle_{R_{w_i;x_j}} \\ &= C_{1,i,j} \langle W_{q_i},\ X_{q_j} \rangle_{R_{w_i;x_j}} + C_{2,i,j} \langle X_{q_j},\ I \rangle_{R_{w_i;x_j}} + C_{3,i,j}, \end{aligned} \tag{3}$$

where $C_{1,i,j} = s_{w_i} s_{x_j}$, $C_{2,i,j} = z_{w_i} s_{x_j}$ and $C_{3,i,j} = s_{w_i} z_{x_j} \langle W_{q_i},\ I \rangle_{R_{w_i;x_j}} + z_{w_i} z_{x_j} \langle I,\ I \rangle_{R_{w_i;x_j}}$ are constant terms that can be pre-computed offline.

As indicated by (3), when $C_{1,i,j}$ and $C_{2,i,j}$ are non-zeros, each path $P_{i,j}$ requires two accumulators for the terms of $\langle W_{q_i},\ X_{q_j} \rangle_{R_{w_i;x_j}}$ and $\langle X_{q_j},\ I \rangle_{R_{w_i;x_j}}$. However, note that the offsets $z_{w_1} = 0$ and then $C_{2,1,j} = z_{w_1} s_{x_j} = 0$, thus $P_{1,j}$ only needs one accumulator for $\langle W_{q_1},\ X_{q_j} \rangle_{R_{w_1;x_j}}$. Therefore, in total, applying PWLQ on both weights and activations requires six accumulators: one of each for $P_{1,1}$ and $P_{1,2}$, plus two of each for $P_{2,1}$ and $P_{2,2}$. Compared to applying PWLQ only on weights, it needs three additional accumulators for supporting PWLQ also on activations, which translates to more hardware overhead. Applying PWLQ with multiple breakpoints on both weights and activations can be extended similarly.

3 Hardware Implementation

In this section, we describe how PWLQ can be applied on a MAC-based DNN accelerator. First, we explain the operation of a uniformly quantized accelerator and then we extend it to support PWLQ.

Figure 5 (a) shows the structure of an accelerator consisting of the 2D array of MAC units and a re-scaling unit. The MAC array has n_{row} rows and n_{col}

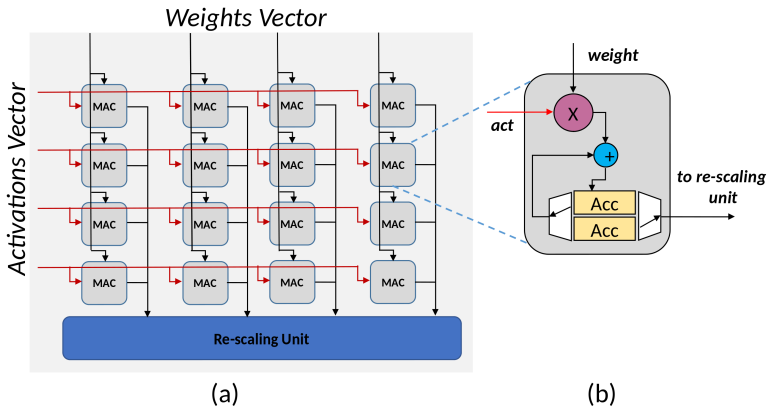

Fig. 5. MAC-based DNN accelerator for uniform quantization

columns that implement an outer product. The accelerator computes n_{row} pixels of n_{col} output channels in each round by decomposing it into a sequence of outer products and then sums the products together. The re-scaling unit consists of multiple floating-point (FP) operators to recover the FP results.

The weights and input activations are broadcast from the top and the left side of the accelerators and their products are accumulated into the accumulator inside each MAC unit (see Figure 5 (b)). Once all the pairs of input activations and their corresponding weights are broadcast, the result will be sent to the re-scaling unit, where an affine function is applied, using FP operators, to the accumulated value for recovering its FP values.

As shown in Figure 5 (b), an accumulator consists of two registers, one for accumulating the current result and one that holds the result of the prior round and serves as the input to the re-scaling unit. This approach increases efficiency, as the MAC unit does not need to wait for re-scaling unit to process its result. Consequently, it enables optimizing the number of FP operators in the re-scaling unit. For each MAC, the re-scaling unit is only required after multiple cycles of accumulations. Thus, MAC units can share a limited number of FP operators. The number of accumulation cycles might be different for different layers. If the number of accumulation cycles is larger than n_{cycle} in most cases, then re-scaling unit requires $\frac{n_{row} \times n_{col}}{n_{cycle}}$ FP operators to execute, without causing any stall.

PWLQ keeps the organization of the accelerator intact, however, it imposes overhead in each unit, as follows (see Figure 6):

(1) Besides broadcasting weight and activation values, PWLQ requires broadcasting their corresponding regions.
(2) For each of the extra regions we need to allocate two accumulation registers, as each region has its own offset value. The first one is to accumulate the product value for this region (product accumulator). The second one is to add all the activations that their corresponding weights belong to this region