Supplementary Material: Accurate Post Training Quantization With Small Calibration Sets

Itay Hubara [*1 2] **Yury Nahshan** [*1] **Yair Hanani** [1] **Ron Banner** [1] **Daniel Soudry** [2]

1. Size of calibration set

Fully Connected layers Let's assume that we have weights of size $W \in R^{M \times N}$ and input and output are of sizes N and M respectively. Recalling Eq. (2) and setting $Y = WX$ and $W' = W + V$ results in:

$$\left(\hat{\Delta}_{w'}, \hat{\Delta}_x, \hat{V}\right) = \underset{\Delta_w, \Delta_x, V}{\arg\min} ||Y - Q_{\Delta_{w'}}(W') \cdot Q_{\Delta_x}(X)||^2,$$

For simplicity we assume that Δ_x is fixed and define $X_q = Q_{\Delta_x}(X)$, $W_q = Q_{\Delta_{w'}}(W')$. Therefore, if we have B unique samples, then the problem we aim to solve have the following structure:

$$\begin{pmatrix} w_{11}x_{11} & \dots & w_{1N}x_{N1} \\ w_{11}x_{11} & \dots & w_{1N}x_{N2} \\ \dots & \ddots & \dots \\ w_{M1}x_{1B} & \dots & w_{MN}x_{NB} \end{pmatrix} = \begin{pmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{MB} \end{pmatrix}$$

Notice that in the above equations, for each output we have a different set of parameters, therefore we can examine each output separately. For a single output we are in scalar linear regression with N parameters and B equations. If $B \geq N$ we are under-parameterized, and if $B < N$ we are over-parameterized.

Convolution layers layers Similarly, for convolution layers with C_o output channels, C_i input channels, and kernel size k each element of the output is a dot product of $C_i \cdot k \cdot k$ parameters. We have in total $Co \times H \times W$ outputs where H, W is the output height and width. Thus we need $B \geq \frac{C_i \cdot k^2}{HW}$ samples to avoid over-fitting, where B is the number of unique samples.

[*]Equal contribution [1]Habana Labs – An Intel company, Caesarea, Israel [2]Department of Electrical Engineering - Technion, Haifa, Israel. Correspondence to: Itay Hubara <itayhubara@gmail.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

2. Reconstruction and re-fusing of Batch Normalization

In this section, we provide more details on Batch Normalization reconstruction and re-fusing procedure.

Reconstructing BN layers: Consider a Batch Normalization layer with parameters γ_o, β_o that fused into previous convolutional layer weight and bias. Fusing batch normalization layer transforms weights and bias as following:

$$W_i' = W_i \frac{\gamma_o}{\sigma}; \qquad b_i' = \frac{\gamma_o}{\sigma}(b_i - \mu) + \beta_o; \tag{1}$$

To reconstruct the batch normalization, we would like to initialize μ, σ^2, as well as the BN parameters γ_r and β_r (r for "reconstructed") so that the reconstructed BN is approximately identity fig. 1.

$$BN_r(x) = \gamma_r \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta_r \approx x \tag{2}$$

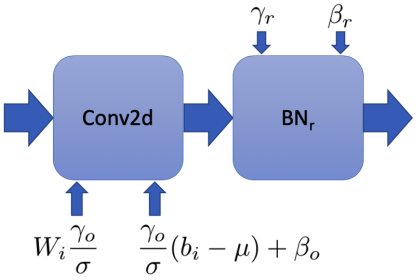

Figure 1.

To do so, first we initialize the reconstructed BN layers by setting the following parameters (denoted by r):

$$\mu = \beta_r = \beta_o; \qquad \sigma^2 = \gamma_o^2 \qquad \gamma_r = \sqrt{\gamma_o^2 + \epsilon} \tag{3}$$

so that $BN_r(x) = x$.

Now, we can update μ and σ^2 by collecting running mean and running variance on the calibration data. We stress that the BN parameters, γ_r, β_r, do not change while applying BN tuning, as we only invoke forward propagation.

Re-fusing BN layers: After BNT phase we need to fuse Batch Normalization layer again into convolution weights and bias. Regular batch normalization fusing will cause degradation due to quantization of the weights. To resolve this issue we can leverage per-channel quantization setting we use.

Denote s_{w_i}, z_{w_i} scale and zero point of the weigh, the quant/dequant operation defined as:

$$W_q = s_{w_i}\left(\left\lfloor \frac{W}{s_{w_i}} - \left\lfloor \frac{z_{w_i}}{s_{w_i}} \right\rceil \right\rceil + \left\lfloor \frac{z_{w_i}}{s_{w_i}} \right\rceil\right) \tag{4}$$

We can fuse parameters of the batch normalization layer as following:

$$\begin{aligned} W_i' &= W_i \frac{\gamma_r}{\sigma_x}; \qquad b_i' = \frac{\gamma_r}{\sigma_r}(b_i - \mu_x) + \beta_r \\ s_{w_i}' &= \frac{\gamma_r}{\sigma_x} s_{w_i}; \qquad z_{w_i}' = \frac{\gamma_r}{\sigma_x} z_{w_i} \end{aligned} \tag{5}$$

Finally we can show that transformations Eq. (5) equivalent to $\frac{\gamma_r}{\sigma_r} W_q$

$$\begin{aligned} W_q' &= s_{w_i}'\left(\left\lfloor \frac{W'}{s_{w_i}'} - \left\lfloor \frac{z_{w_i}'}{s_{w_i}'} \right\rceil \right\rceil + \left\lfloor \frac{z_{w_i}'}{s_{w_i}'} \right\rceil\right) \\ &= \frac{\gamma_r}{\sigma_r} s_{w_i}\left(\left\lfloor \frac{W}{s_{w_i}} - \left\lfloor \frac{z_{w_i}}{s_{w_i}} \right\rceil \right\rceil + \left\lfloor \frac{z_{w_i}}{s_{w_i}} \right\rceil\right) \\ &= \frac{\gamma_r}{\sigma_r} W_q \end{aligned} \tag{6}$$

3. Additive loss assumption for integer-programming

Suppose the loss function of the network $\mathcal{L}$ depends on a certain set of variables (weights, activations, etc.), which we denote by a vector $\mathbf{v}$. We would like to measure the effect of adding quantization noise to this set of vectors.

Since the quantization is emulated with additive noise, the loss is smooth and thus can be expanded to the Taylor series:

$$\Delta\mathcal{L} = \mathcal{L}(\mathbf{v} + \varepsilon) - \mathcal{L}(\mathbf{v}) = \tag{7}$$

$$= \frac{\partial \mathcal{L}^{\mathcal{T}}}{\partial \mathbf{v}}\varepsilon + \varepsilon^T \frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{v}}\varepsilon + O\left(\|\varepsilon\|^3\right). \tag{8}$$

One can see from Eq 7 that when the quantization error ε is sufficiently small, the overall degradation $\Delta\mathcal{L}$ can be approximated as a sum of N independent degradation processes by neglecting the quadratic terms ε^2:

$$\Delta\mathcal{L} \approx \frac{\partial \mathcal{L}^{\mathcal{T}}}{\partial \mathbf{v}}\varepsilon = \sum_i^n \frac{\partial \mathcal{L}}{\partial v_i} \cdot \varepsilon_i \tag{9}$$

We note that (Lin et al., 2016; Choukroun et al., 2019) used a similar assumption with respect to the additivity of quantization noise.

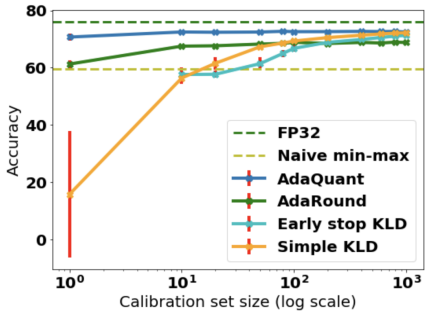

Figure 2. Calibration size ablation study with additional early-stop plot.

4. Experimental Details

In all our experiments, we used a small subset of the training set to run our methods. Specifically, for vision models, we used 1000 unlabeled images from the ImageNet training set (single image for each class) as a calibration set. For the Bert model, we used one paragraph from the training set. All presented methods AdaQuant, BNT, BT, and IP, performed well on such small calibration set producing SOTA results. Next we detail our setting for each of the technique in our pipelines

4.1. AdaQuant

AdaQuant optimization problem defined as following except zero-point of the quantizer which we omitted from eq.(3):

$$\left(\hat{\Delta}_w, \hat{\Delta}_x, \hat{V}_W, \hat{V}_b\right) = \underset{\substack{\Delta_w, \Delta_x \\ V_w, V_b}}{\arg\min} ||WX + b - Q_{\Delta_w}(W + V_W) \cdot Q_{\Delta_x}(X) - Q(b + V_b)||$$

Technically to find a solution for eq.(3), we use Adam optimizer with different learning rates per type of parameters. We set different learning rates for weight, bias, and quantization parameters of input and weights. After experimenting with different models, we found that the same set of LR parameters worked for each model. The learning rates are $1e-5, 1e-3, 1e-1, 1e-3$ for weight, bias, quantization parameters of the inputs, and weights, respectively.

For vision models, we used 1000 unlabeled images from the ImageNet training set (single image for each class), running Adam optimizer for 100 iterations and a batch-size of 50 unless otherwise stated. For BERT-base model, we used one paragraph from the training set, running Adam optimizer for 50 - 100 iterations depending on the type of layer. Learning rates and batch size are the same as of vision models. In fig-(1) we aimed to answer the following question: *Assuming you have a small calibration set and no resources constraints (time,power) which method is the most accurate and robust* Out method were evaluated by

running each experiments five times and reporting mean and standrad deviation. Here, in fig. 2, we add an additional naive early-stop plot on top of QAT-KLD experiment. We split the calibration data into two equal sets and train on half the examples while evaluation our performance on the other half Both *KLD* experiments used an SGD optimizer over 10 epochs; starting with learning rate of 0.1 and decreasing it by 1e-2 factor after 2 and 8 epochs. We also conducted *KLD* experiments with Adam optimizer and learning rate of 1e-3 where performed but their results were inferior. As can be seen in the plot AdaQuant is superior to other methods and remarkably excels on small calibration sets. As can be seen in fig. 2 the early exit results were inferior to the QAT-KLD as they use much smaller training set. However, other types of training-validation splits (e.g. 80-20) may boost the results.

4.2. Integer Programming

Our IP method requires two steps, the first is measuring the properties of each layer, and the second is applying the program based on these measurements with user defined constraint. As reference, we measure the loss (can also be accuracy) of the base precision model on the calibration set. Next, we measure the sensitivity of each layer by evaluating a model where all layers are qunatize to the base-precision but one layer that is quantized to lower precision (e.g., all 8-bit but one layer with 4-bit). The $\Delta\mathcal{L}_l$ in Eq. 3 is defined as the difference between the reference model loss and the measured loss. If a layer is robust to quantization, $\Delta\mathcal{L}_l$ will be small, and if a layer is sensitive to quantization, $\Delta\mathcal{L}_l$ will be large. The performance gain in the case of compression, is simply the model parameters size difference when lowering the precision of the examined layer. Hence, if a layer has N parameters, the performance gain when changing from 8-bit to 4-bit result in compression gain of $\Delta\mathbb{P}_l = N * 8 - N * 4 = 4N$. In the second stage, we run the integer program based on the sensitivity and compression measured on each layer along with the user defined constraint.

4.3. Batch Normalization and Bias Tuning

The Batch Norm tuning phase is the most lightweight phase of the pipeline. We found empirically less than ten iterations of statistics update are sufficient. We also found that as compression growth, more iterations of batch norm tuning are required. At the bias tuning phase, we perform 200 iterations of fine-tuning with the learning-rate of 0.1.

4.4. Methods complexity

We experimented with several methods each with its own complexity. For integer programming we used pulp library optimized for CPU, thus running IP for layer-wise bit allocation takes seconds at most. As stated in section 3.3, para-norm requires only a few model iterations (we used 100). AdaQuant complexity is slightly higher as it requires tuning the weight parameters. Yet, only 100 training iterations per layer are needed. Thus for ResNet50, even the most time-consuming version, seq-AdaQuant, takes less than 5 minutes on one device (GeForce 1080). Thus, our methods are two orders of magnitude faster than QAT.

	Num of iterations	Requires BP	time (min)
Integer programming	-	No	0.1-0.2
Para-Normalization	100	No	1-2
Seq-AdaQuant	100	Yes	5

Table 1. Methods complexity in terms of number of iterations required, whether or no back-propagation (BP) is required, and the time as measured over one GeForce 1080 GPU.

5. Code

For all our vision dataset we used the default *torchvision* pre-trained model. For BERT-base experiment we finedtuned on SQUAD1.1 dataset and provide the script for that as a part of our repository. Our code can be found at: *https://github.com/papers-submission/CalibTIP*.

References

Choukroun, Y., Kravchik, E., Yang, F., and Kisilev, P. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3009–3018. IEEE, 2019.

Lin, D., Talathi, S., and Annapureddy, S. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pp. 2849–2858, 2016.