Post-Training Quantization for Vision Transformer

https://arxiv.org/pdf/2106.14156

3 Methodology

In this section, we elaborate on the proposed mixed-precision post-training quantization scheme for the vision transformer. The similarity-aware quantization for linear layers and ranking-aware quantization for self-attention layers are presented. In addition, the bias correction method for optimization and the mixed-precision quantization based on nuclear norm of the attention map and output feature are introduced.

3.1 Preliminaries

A standard transformer receives an input as a 1-D sequence of token embeddings, so the vision transformers usually reshape the image $\mathbf{I} \in \mathbb{R}^{H\times W\times C}$ into a sequence of flatted 2D patches $I^p \in \mathbb{R}^{n\times(P^2\cdot C)}$. Here, H and W are the height and width of the original image and (P, P) is the resolution of each image patch, $n = \frac{HW}{P^2}$ is then the effective sequence length for the transformer. Usually, the vision transformers use constant widths through all of its layers, so a trainable linear projection maps each vectorized patch to the model dimension d. Thus, the input to the first transformer layer is:

$$\mathbf{X}_1 = [x_{class}; I_1^p\mathbf{W}_1^E; \cdots; I_n^p\mathbf{W}_n^E] + \mathbf{E}^{pos}, \tag{1}$$

$$\text{where } \mathbf{W}^E \in \mathbb{R}^{(P^2\cdot C)\times d}, \mathbf{E}^{pos} \in \mathbb{R}^{(n+1)\times d} \tag{2}$$

A standard transformer layer includes two main modules: Multi-Head Self Attention (MSA) and Multi-Layer Perceptron (MLP) module. For the l-th transformer layer, suppose the input to it is $\mathbf{X}_l \in \mathbb{R}^{n\times d}$, the attention scores computed by the dot product of queries and keys can be formulated as:

$$\mathbf{A}_l = \mathbf{Q}_l\mathbf{K}_l^{\mathrm{T}} = \mathbf{X}_l\mathbf{W}_l^Q\mathbf{W}_l^{K^{\mathrm{T}}}\mathbf{X}_l^{\mathrm{T}}, \tag{3}$$

Then the softmax function is applied on the normalized scores to get the output and the output of the multi-head self attention module is:

$$\mathrm{MSA}(\mathbf{X}_l) = \mathrm{Softmax}(\frac{1}{\sqrt{d}}\mathbf{A}_l)\mathbf{X}_l\mathbf{W}_l^V \cdot \mathbf{W}_l^O. \tag{4}$$

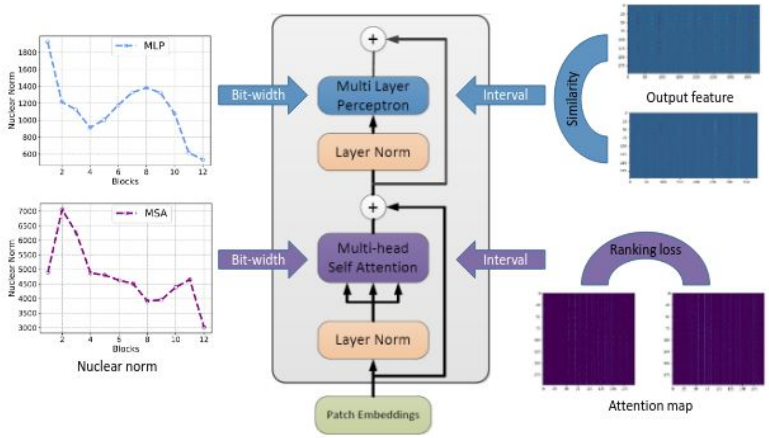


Figure 1: Diagram of the proposed mixed-precision post-training quantization method for vision transformer. The similarity-aware and ranking-aware quantization are designed for finding the optimal quantization interval of the linear operations and self-attention layers. The bit-widths of transformer layers are determined based on the nuclear norm of the attention map and the output feature.

The MLP module contains two linear layers parameterized by $\mathbf{W}^1 \in \mathbb{R}^{d \times d_f}, b^1 \in \mathbb{R}^{d_f}$ and $\mathbf{W}^2 \in \mathbb{R}^{d_f \times d}, b^2 \in \mathbb{R}^d$ respectively, where d_f is the number of neurons in the intermediate layer of MLP. Denote the input to MLP as $\mathbf{Z}_l \in \mathbb{R}^{n \times d}$, the output is then computed as:

$$\mathrm{MLP}(\mathbf{Z}_l) = \mathrm{GeLU}(\mathbf{Z}_l\mathbf{W}^1 + b^1)\mathbf{W}^2 + b^2. \tag{5}$$

Combining Eq. (4) and (5), the forward propagation for the l-th transformer layer can be formulated as:

$$\mathbf{Z}_l = \mathrm{LN}(\mathbf{X}_l + \mathrm{MSA}(\mathbf{X}_l)), \tag{6}$$
$$\mathbf{X}_{l+1} = \mathrm{LN}(\mathbf{Z}_l + \mathrm{MLP}(\mathbf{Z}_l)). \tag{7}$$

where LN represents the layer normalization.

The most computational costs of vision transformer lie on the large matrix multiplication in MSA and MLP module. Following the mainstream quantization methods for CNNs [7, 21], we quantize all the weights and inputs involved in matrix multiplication. For weight quantization, we quantize the weights $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O, \mathbf{W}^1, \mathbf{W}^2$ in Eq. (4) and (5) for all transformer layers, as well as the linear embedding $\mathbf{W}^E$ in Eq. (1). Besides these weights, we also quantize the inputs of all linear layers and matrix multiplication operations. Following the methods in [22, 34], we do not quantize the softmax operation and layer normalization, because the parameters contained in these operations are negligible and quantizing them may bring significant accuracy degradation.

3.2 Optimization for Post-Training Quantization

For post-training quantization, we need to restrict the floating-numbers to a finite set of values. The choice of quantization intervals is critical for quantization and one popular option is to use a uniform quantization function, where the data range is equally split:

$$\Psi_\Delta(\mathbf{Y}) = \mathrm{Clip}(\mathrm{Round}(\frac{\mathbf{Y}}{\Delta}), -2^{b-1}, 2^{b-1} - 1). \tag{8}$$

where Δ is the quantization interval, b is the quantization bit-width and $\mathbf{Y}$ is a tensor representing weights or inputs. Clip denotes that elements in the tensor that exceed the ranges of the quantized domain are clipped.

Similarity-Aware Quantization for Linear Operation For the linear operations in the MSA module and MLP module of the l-th transformer layer, the original output can be computed as $\mathbf{O}_l = \mathbf{X}_l\mathbf{W}_l$. The uniform quantization for the weights and inputs and the corresponding dequant operation can be described as:

$$\widehat{\mathbf{O}}_l = \Psi_{\Delta_l^X}(\mathbf{X}_l)\Psi_{\Delta_l^W}(\mathbf{W}_l) \cdot \Delta_l^W \cdot \Delta_l^X. \tag{9}$$

where $\widehat{\mathbf{O}}_l$ denotes the outputs of the quantized layer. From Eq. (8) and Eq. (9), it can be seen that the quantization intervals actually control the clipping thresholds in quantization process, which affects the similarity between original output feature maps and quantization feature maps to a great extent. Therefore, we are motivated to focus on optimizing the quantization intervals for both weights Δ_l^W and inputs Δ_l^X to improve the similarity between $\mathbf{O}_l$ and $\widehat{\mathbf{O}}_l$, where inputs X_l are generated from a given calibration dataset $\mathbf{D}$ with N samples. Specifically, the calibration dataset is much less than the common training dataset. In the l-th transformer layer, the similarity-aware quantization can be formulated as:

$$\max_{\Delta_l^W, \Delta_l^X} \frac{1}{N}\sum_{i=1}^{N} \Gamma(\mathbf{O}_l^i, \widehat{\mathbf{O}}_l^i) \quad s.t.\ \Delta_l^W, \Delta_l^X \in \mathbb{R}^+. \tag{10}$$

where $\Gamma(\mathbf{O}_l^i, \widehat{\mathbf{O}}_l^i)$ is the similarity between the original and quantized output feature maps. In this paper, we adopt Pearson correlation coefficient as the measurement for the similarity:

$$\Gamma(\widehat{\mathbf{O}}, \mathbf{O}) = \frac{\sum_{j=1}^{m}(\mathbf{O}_j - \overline{\mathbf{O}})(\widehat{\mathbf{O}}_j - \overline{\widehat{\mathbf{O}}})}{\sqrt{\sum_{j=1}^{m}(\mathbf{O}_j - \overline{\mathbf{O}})^2}\sqrt{\sum_{j=1}^{m}(\widehat{\mathbf{O}}_j - \overline{\widehat{\mathbf{O}}})^2}}. \tag{11}$$

Ranking-Aware Quantization for Self-Attention. The self-attention layer is the critical component of the transformer since it can calculate the global relevance of the features, which makes the transformer unique from the convolutional neural networks. For the calculation of self-attention (Eq. 3), we empirically find that the relative order of the attention map has been changed after quantization as shown in Fig 1, which could cause a significant performance degradation. Thus, a ranking loss is introduced to solve this problem during the quantization process:

$$\max_{\Delta_l^W, \Delta_l^X} \frac{1}{N} \sum_{i=1}^{N} \Gamma(\mathbf{O}_l^i, \widehat{\mathbf{O}}_l^i) - \gamma \cdot \mathcal{L}_{ranking} \quad s.t.\ \Delta_l^W, \Delta_l^X \in \mathbb{R}^+. \tag{12}$$

where $\mathcal{L}_{rank}$ denote the pairwise ranking based loss function, and γ is the trade-off hyper-parameter. The ranking loss can be formulated as:

$$\mathcal{L}_{ranking} = \sum_{k=1}^{h} \sum_{i=1}^{w-1} \sum_{j=i+1}^{w} \Phi((\widehat{\mathbf{A}}_{ki} - \widehat{\mathbf{A}}_{kj}) \cdot sign(\mathbf{A}_{ki} - \mathbf{A}_{kj})). \tag{13}$$

in which $\Phi(p) = (\theta - p)_+$ is hinge function with parameter θ, (h, w) are the size of matrix $\mathbf{A}$. Given a pair of examples, the loss is 0 only when the examples are in the correct order and differed by a margin.

To solve the above optimization problem, we present a simple but efficient alternative searching method for the uniform quantization of transformer layers. Firstly, the quantization interval of inputs Δ_l^X is fixed, and the quantization interval of weights Δ_l^W is optimized for adjustment. Secondly, Δ_l^W is fixed, and Δ_l^X is optimized to fine-tune the quantization interval of the inputs. Δ_l^W and Δ_l^X are alternately optimized until the target function converges or the maximum iteration is exceeded. Moreover, for fast convergence, Δ_l^W and Δ_l^X are initialized in terms of the maximum of weights or inputs respectively. For the search space of Δ_l^W and Δ_l^X, we linearly divide interval of $[\alpha\Delta_l, \beta\Delta_l]$ into C candidate options and conduct a simple search strategy on them.

Bias Correction To further reduce the biased error for the outputs raised by quantization, a bias correction method is then introduced after each search iteration. Suppose the quantization error of weights and inputs are defined as:

$$\epsilon^X = \Psi_{\Delta^X}(\mathbf{X}) \cdot \Delta^X - \mathbf{X}, \tag{14}$$

$$\epsilon^W = \Psi_{\Delta^W}(\mathbf{W}) \cdot \Delta^W - \mathbf{W}. \tag{15}$$

If the expectation of the error for output is not zero, then the mean of the output will change. This shift in distribution may lead to detrimental behavior in the following layers. We can correct this change by seeing that:

$$\mathbb{E}[\widehat{\mathbf{O}}] = \mathbb{E}[\mathbf{O}] + \mathbb{E}[\epsilon^W \mathbf{X}] + \mathbb{E}[\epsilon^X \mathbf{W}] + \mathbb{E}[\epsilon^X \epsilon^W]. \tag{16}$$

Thus, subtracting the expected error on the output from the biased output ensures that the mean for each output unit is preserved. For implementation, the expected error can be computed using the calibration data and subtracted from the layer's bias parameter, since the expected error vector has the same shape as the layer's output.

3.3 Mixed-Precision Quantization for Vision Transformer

Different transformer layers are attending to different structures, and it is expected that they exhibit different sensitivity. Thus, assigning the same number of bit-widths to all the layers is sub-optimal. As a result, we explore mixed-precision quantization, where more bits are assigned to more sensitive layers in order to retain performance. Considering the unique structure of transformer layer, we assign all the operations in the MSA or MLP modules with the same bit-width. This will also be friendly to the hardware implementation since the weights and inputs are assigned with the same bit-width.

Singular value decomposition (SVD) is an important matrix decomposition approach in linear algebra. It takes a rectangular matrix of gene expression data, whose formulation can be written as :

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}. \tag{17}$$

where the diagonal entries $\sigma_i = \mathbf{\Sigma}_{ii}$ of $\mathbf{\Sigma}$ are known as the singular values of $\mathbf{M}$. And the nuclear norm is the sum of singular values, which represents the data relevance of the matrix. In this paper, we propose to estimate the sensitivity of the transformer layer with the nuclear norm of the attention map in the MSA module and the output feature in the MLP module. The nuclear norm can be used to reduce the search space of the mixed-precision settings, while using higher bit-widths for layers that are more sensitive and vice versa. Inspired by the method in [10], we utilize a Pareto frontier approach to determine the bit-width. The main idea is to sort each candidate bit-width configuration based on the total second-order perturbation that they cause, according to the following metric:

$$\Omega = \sum_{i=1}^{L} \Omega_i = \sum_{i=1}^{L} \sum_{j=1}^{m} \sigma_j(\mathbf{Y}_i) \cdot \|\widehat{\mathbf{Y}_i} - \mathbf{Y}_i\|_2^2. \tag{18}$$

Given a target model size, we sort the candidate bit-width configuration based on their Ω value and choose the bit-width configuration with minimal Ω. The nuclear norm of the attention map and output feature in each transformer layer are shown in Figure 1. As we can see, they are various for different transformer layers.

Post-Training Sparsity -Aware Quantization

Post-Training Sparsity-Aware Quantization[PyTorch]:star:5

https://arxiv.org/pdf/2105.11010

3 The Basic Principle of SPARQ

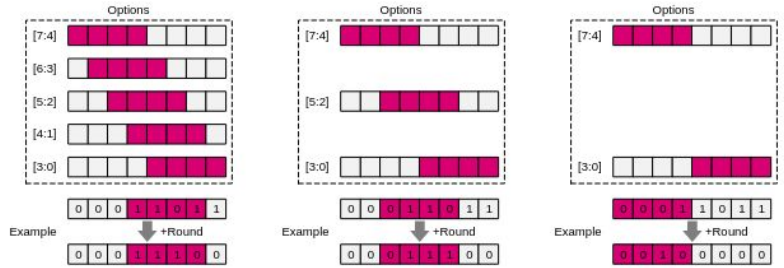
SPARQ comprises two orthogonal techniques: bSPARQ and vSPARQ. The former leverages zero-value bits to trim an 8-bit value to an n-bit value; and the latter leverages zero-value activations. Below, we describe both in detail. Throughout this work, we focus on quantizing the activations and leveraging only their sparsity, i.e., no correlation is made with the weight values, unless otherwise stated.

3.1 bSPARQ: Leveraging Bit Sparsity

Consider an already quantized 8-bit activation, x, and quantization to 4 bits (i.e., $n = 4$). bSPARQ trims the activation from 8 bits to 4 bits by inspecting the activation bits and choosing the most significant consecutive 4 bits within it, which, in practice, is achieved by searching for the first most significant toggled bit. The motivation behind bSPARQ is twofold: first, activations usually follow a bell-shaped distribution, meaning that the MSBs are usually equal to zero and, therefore, can be skipped; and second, if the MSBs are toggled, the LSBs' contribution to the entire value is insignificant. For example, given the value 00011011_2 (27_{10}), the 4-bit window will be positioned at bits [4:1] ($0001\underline{1011}_2$), thus achieving the approximated value 26_{10}. Notice that since there are five window position options, the 4-bit window is accompanied by a 3-bit identifier that corresponds to the window position—that is, how much shift-left is required on top of the four trimmed bits. In addition, to further reduce the dynamic quantization noise, we round the value within the chosen window according to the residual LSBs. bSPARQ is visually demonstrated in Figure 1.

Supporting five window options requires additional circuitry compared with, for example, three window options, since additional placement options require additional hardware support by the shift-left unit. The trade-off is, however, improved accuracy, since additional placement options introduce less quantization noise. We experiment with five, three, and two placement options, denoted as 5opt, 3opt, and 2opt, respectively. With the 3opt configuration, [7:4], [5:2], or [3:0] are chosen, and with the 2opt configuration, either [7:4] or [3:0] are chosen (we leave the analysis of asymmetrical configurations for future work). For example, given the previous value, 00011011_2, 3opt will choose bits [5:2] ($0001\underline{1011}_2$), whereas 2opt will choose bits [7:4] ($\underline{0001}1011_2$).

Relation to piecewise linear quantization. To mitigate quantization errors, previous works suggest dividing the tensor distributions into different quantization regions, each with a scaling factor of its own [6, 12, 24]. In a sense, bSPARQ is somewhat similar to those. First, each activation is assigned to a quantization range according to its value; however, we break the distributions into hardware-oriented regions of power of two. For example, for the 5opt case, the regions are $[0, 2^1 - 1]$, $[2^1, 2^2 - 1]$, and so on. As a result, values are mapped to their appropriate range by simply counting the leading zero bits. In addition, we avoid any need for preprocessing that searches for the distribution breakpoints to minimize the quantization noise. Second, each region has an individual scaling factor; however, each



(a) 5opt (b) 3opt (c) 2opt

Figure 1: Demonstration of SPARQ 8b-to-4b quantization. More window placement options (e.g., 5opt) decrease the quantization noise; however, additional hardware is needed to support many placement options.

region scaling factor is a product of a base scaling factor with the corresponding power of two. For example, in the 5opt configuration, the scaling factor of the decimal number $33_{10} = 00\underline{100001}_2$ is the original scaling factor times 2^2. This enables a relatively simple implementation with up to five regions when considering 4-bit activations, and even six and seven regions when considering 3- and 2-bit activations, respectively—as opposed to the two quantization regions used by previous works.

3.2 vSPARQ: Leveraging Sparsity with Pairs of Activations

Consider an 8-bit unsigned activation vector, $X = (x_1, \cdots, x_L)$, and an 8-bit signed weight vector, $W = (w_1, \cdots, w_L)$, both of length L. Also, consider a single MAC unit that computes a single activation-weight multiplication per cycle. vSPARQ, similar to [32, 34, 41], groups activations in pairs, to leverage the dynamic and unstructured activation sparsity. That is, the DP calculations can be formulated as:

$$X \cdot W = \sum_{i \text{ even}}^{L} x_i w_i + x_{i+1} w_{i+1} = y\,, \tag{1}$$

where y is the DP scalar result, and in our context, an output activation. For some i, if $x_i = 0$, then x_{i+1} can be used with 8-bit representation, and vice versa. If, however, both $x_i \neq 0$ and $x_{i+1} \neq 0$, and given that, for example, bSPARQ is employed, then the precision of both x_i and x_{i+1} is reduced to 4 bits. For a certain i, the vSPARQ operation can also be formulated as:

$$x_i w_i + x_{i+1} w_{i+1} = \begin{cases} x_i w_i, & \text{if } x_{i+1} = 0 \\ x_{i+1} w_{i+1}, & \text{if } x_i = 0 \\ \text{bSPARQ}(x_i) w_i + \text{bSPARQ}(x_{i+1}) w_{i+1}, & \text{otherwise} \end{cases}. \tag{2}$$

Notice that the two first case statements correspond to an 8b-8b computation, whereas the last case statement corresponds to two 4b-8b computations. The latter case is possible, since two 4b-8b multiplications are logically equivalent to a single 8b-8b multiplication, as we describe next.

8b-8b = 2x4b-8b. Given an 8-bit unsigned activation, x, and an 8-bit signed weight, w, the activation-weight multiplication can be formulated as

$$x_{[7:0]} \cdot w_{[7:0]} = \sum_{i=0}^{7} 2^i x_i \cdot w_{[7:0]} = \left(\sum_{i=0}^{3} 2^{i+4} x_{i+4} + \sum_{i=0}^{3} 2^i x_i \right) \cdot w_{[7:0]} \\ = 2^4 x_{[7:4]} \cdot w_{[7:0]} + x_{[3:0]} \cdot w_{[7:0]}, \tag{3}$$

where the $[b:a]$ notation represents the b-to-a range in bits, the two activation-weight multiplications are 4b-8b wide, and the 2^4 is equivalent to a 4-bit shift-left operation.

By considering an additional weight input as well as dynamic shift-left operations, we can reuse the multipliers and achieve a multiplier capable of either one 8b-8b multiplication or two *independent*

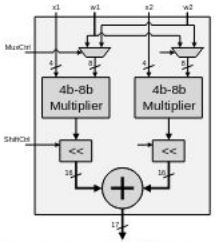


Figure 2: Equation (4) hardware implementation.

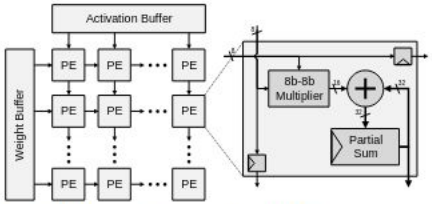


Figure 3: Illustration of a conventional 8b-8b output stationary systolic array.

4b-8b multiplications with a dynamic range:

$$2^{\text{opt}_1} x_{\text{in1,4b}} \cdot w_{\text{in1,8b}} + 2^{\text{opt}_2} x_{\text{in2,4b}} \cdot w_{\text{in2,8b}}, \tag{4}$$

where the activation and weight inputs are 4 bits and 8 bits long, respectively. Equation (4) resembles a FP representation; however, the "opt" configurations are not necessarily continuous, as in 3opt and 2opt. Figure 2 illustrates how Equation (4) is mapped to hardware. The two 4b-8b multipliers correspond to $x_{\text{in1}} \cdot w_{\text{in1}}$ and $x_{\text{in2}} \cdot w_{\text{in2}}$, and the two shift-left units ($\ll$) correspond to 2^{opt_1} and 2^{opt_2}. The adder corresponds to the addition of the two groups, and the multiplexers, which are not explicitly formulated in Equation (4), are used to choose dynamically between w_{in1}, w_{in2}, or select both, during execution. We use this multiplier instead of the conventional one used in well-known hardware structures.

GPTQ: ACCURATE POST-TRAINING QUANTIZATION FOR GENERATIVE PRE-TRAINED TRANSFORMERS

https://arxiv.org/pdf/2210.17323v2

3 BACKGROUND

Layer-Wise Quantization. At a high level, our method follows the structure of state-of-the-art post-training quantization methods (Nagel et al., 2020; Wang et al., 2020; Hubara et al., 2021; Frantar et al., 2022), by performing quantization layer-by-layer, solving a corresponding reconstruction problem for each layer. Concretely, let $\mathbf{W}_\ell$ be the weights corresponding to a linear layer ℓ and let $\mathbf{X}_\ell$ denote the layer input corresponding to a small set of m data points running through the network. Then, the objective is to find a matrix of quantized weights $\widehat{\mathbf{W}}$ which minimizes the squared error, relative to the full precision layer output. Formally, this can be restated as

$$\text{argmin}_{\widehat{\mathbf{W}}} ||\mathbf{W}\mathbf{X} - \widehat{\mathbf{W}}\mathbf{X}||_2^2. \tag{1}$$

Further, similar to (Nagel et al., 2020; Li et al., 2021; Frantar et al., 2022), we assume that the quantization grid for $\widehat{\mathbf{W}}$ is fixed before the process, and that individual weights can move freely as in (Hubara et al., 2021; Frantar et al., 2022).

Optimal Brain Quantization. Our approach builds on the recently-proposed Optimal Brain Quanization (OBQ) method (Frantar et al., 2022) for solving the layer-wise quantization problem defined above, to which we perform a series of major modifications, which allow it to scale to large language models, providing more than *three orders of magnitude* computational speedup. To aid understanding, we first briefly summarize the original OBQ method.

The OBQ method starts from the observation that Equation (1) can be written as the sum of the squared errors, over each row of $\mathbf{W}$. Then, OBQ handles each row $\mathbf{w}$ independently, quantizing one weight at a time while always updating all not-yet-quantized weights, in order to compensate for the error incurred by quantizing a single weight. Since the corresponding objective is a quadratic,

whose Hessian is $\mathbf{H}_F = 2\mathbf{X}_F\mathbf{X}_F^\top$, where F denotes the set of remaining full-precision weights, the greedy-optimal weight to quantize next, which we denote by w_q, and the corresponding optimal update of all weights in F, denoted by $\boldsymbol{\delta}_F$, are given by the following formulas, where $\text{quant}(w)$ rounds w to the nearest value on the quantization grid:

$$w_q = \text{argmin}_{w_q} \frac{(\text{quant}(w_q) - w_q)^2}{[\mathbf{H}_F^{-1}]_{qq}}, \quad \boldsymbol{\delta}_F = -\frac{w_q - \text{quant}(w_q)}{[\mathbf{H}_F^{-1}]_{qq}} \cdot (\mathbf{H}_F^{-1})_{:,q}. \tag{2}$$

OBQ quantizes weights iteratively using these two equations, until all the weights of $\mathbf{w}$ are quantized. This is done efficiently, avoiding expensive full recomputations of $\mathbf{H}^{-1}$, by removing the qth row and column of $\mathbf{H}$, which is necessary after quantizing w_q, directly in the inverse via one step of Gaussian elimination. Namely, the updated inverse is given by the formula

$$\mathbf{H}_{-q}^{-1} = \left(\mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}^{-1}_{:,q} \mathbf{H}^{-1}_{q,:}\right)_{-p}. \tag{3}$$

This method comes with a vectorized implementation, handling multiple rows of $\mathbf{W}$ in parallel. Eventually, the algorithm can achieve reasonable runtimes on medium-sized models: for instance, it can fully quantize the ResNet-50 model (25M parameters) in $\approx$ 1 hour on a single GPU, which is roughly in line with other post-training methods achieving state-of-the-art accuracy (Frantar et al., 2022). However, the fact that OBQ's runtime for a $d_{\text{row}} \times d_{\text{col}}$ matrix $\mathbf{W}$ has *cubic* input dependency $O(d_{\text{row}} \cdot d_{\text{col}}^3)$ means that applying it to models with billions of parameters is extremely expensive.

4 The GPTQ Algorithm

Step 1: Arbitrary Order Insight. As explained in the previous section, OBQ quantizes weights in greedy order, i.e. it always picks the weight which currently incurs the least additional quantization error. Interestingly, we find that, while this quite natural strategy does indeed seem to perform very well, its improvement over quantizing the weights in arbitrary order is generally small, in particular on large, heavily-parametrized layers. Most likely, this is because the slightly lower number of quantized weights with large individual error is balanced out by those weights being quantized towards the end of the process, when only few other unquantized weights that can be adjusted for compensation remain. As we will now discuss, this insight that *any fixed order may perform well*, especially on large models, has interesting ramifications.

The original OBQ method quantizes rows of $\mathbf{W}$ independently, in a specific order defined by the corresponding errors. By contrast, we will aim to quantize the weights of *all rows in the same order*, and will show that this typically yields results with a final squared error that is similar to the original solutions. As a consequence, the set of unquantized weights F and similarly $\mathbf{H}_F^{-1}$ is always the same for all rows (see Figure 2 for an illustration). In more detail, the latter is due to the fact that $\mathbf{H}_F$ depends only on the layer inputs $\mathbf{X}_F$, which are the same for all rows, and not on any weights. Therefore, we have to perform the update of $\mathbf{H}_F^{-1}$ given by Equation (3) only d_{col} times, once per column, rather than $d_{\text{row}} \cdot d_{\text{col}}$ times, once per weight. This reduces the overall runtime from $O(d_{\text{row}} \cdot d_{\text{col}}^3)$ to $O(\max\{d_{\text{row}} \cdot d_{\text{col}}^2, d_{\text{col}}^3\})$, i.e., by a factor of $\min\{d_{\text{row}}, d_{\text{col}}\}$. For larger models, this difference consists of several orders of magnitude. However, before this algorithm can actually be applied to very large models in practice, two additional major problems need to be addressed.

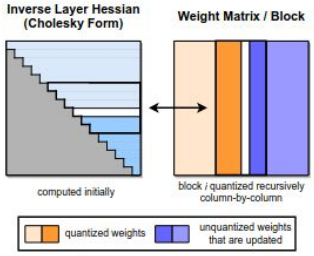


Figure 2: GPTQ quantization procedure. Blocks of consecutive *columns* (bolded) are quantized at a given step, using the inverse Hessian information stored in the Cholesky decomposition, and the remaining weights (blue) are updated at the end of the step. The quantization procedure is applied recursively inside each block: the white middle column is currently being quantized.

Step 2: Lazy Batch-Updates. First, a direct implementation of the scheme described previously will not be fast in practice, because the algorithm has a relatively low compute-to-memory-access ratio. For example, Equation (3) needs to update all elements of a potentially huge matrix using just a

few FLOPs for each entry. Such operations cannot properly utilize the massive compute capabilities of modern GPUs, and will be bottlenecked by the significantly lower memory bandwidth.

Fortunately, this problem can be resolved by the following observation: The final rounding decisions for column i are only affected by updates performed on this very column, and so updates to later columns are irrelevant at this point in the process. This makes it possible to "lazily batch" updates together, thus achieving much better GPU utilization. Concretely, we apply the algorithm to $B = 128$ columns at a time, keeping updates contained to those columns and the corresponding $B \times B$ block of $\mathbf{H}^{-1}$ (see also Figure 2). Only once a block has been fully processed, we perform global updates of the entire $\mathbf{H}^{-1}$ and $\mathbf{W}$ matrices using the multi-weight versions of Equations (2) and (3) given below, with Q denoting a set of indices, and $\mathbf{H}^{-1}_{-Q}$ denoting the inverse matrix with the corresponding rows and columns removed:

$$\boldsymbol{\delta}_F = -(\mathbf{w}_Q - \text{quant}(\mathbf{w}_Q))([\mathbf{H}^{-1}_F]_{QQ})^{-1}(\mathbf{H}^{-1}_F)_{:,Q}, \tag{4}$$

$$\mathbf{H}^{-1}_{-Q} = \left(\mathbf{H}^{-1} - \mathbf{H}^{-1}_{:,Q}([\mathbf{H}^{-1}]_{QQ})^{-1}\mathbf{H}^{-1}_{Q,:}\right)_{-Q}. \tag{5}$$

Although this strategy does not reduce the theoretical amount of compute, it effectively addresses the memory-throughput bottleneck. This provides an order of magnitude speedup for very large models in practice, making it a critical component of our algorithm.

Step 3: Cholesky Reformulation. The final technical issue we have to address is given by numerical inaccuracies, which can become a major problem at the scale of existing models, especially when combined with the block updates discussed in the previous step. Specifically, it can occur that the matrix $\mathbf{H}^{-1}_F$ becomes indefinite, which we notice can cause the algorithm to aggressively update the remaining weights in incorrect directions, resulting in an arbitrarily-bad quantization of the corresponding layer. In practice, we observed that the probability of this happening increases with model size: concretely, it almost certainly occurs for at least a few layers on models that are larger than a few billion parameters. The main issue appears to be the repeated applications of Equation (5), which accumulate various numerical errors, especially through the additional matrix inversion.

For smaller models, applying dampening, that is adding a small constant λ (we always choose 1% of the average diagonal value) to the diagonal elements of $\mathbf{H}$ appears to be sufficient to avoid numerical issues. However, larger models require a more robust and general approach.

To address this, we begin by noting that the only information required from $\mathbf{H}^{-1}_{F_q}$, where F_q denotes the set of unquantized weights when quantizing weight q, is row q, or more precisely, the elements in this row starting with the diagonal. The consequence is that we could precompute all of these rows using a more numerically-stable method without any significant increase in memory consumption. Indeed, the row removal via (3) for our symmetric $\mathbf{H}^{-1}$ essentially corresponds to taking a Cholesky decomposition, except for the minor difference that the latter divides row q by $([\mathbf{H}^{-1}_{F_q}]_{qq})^{1/2}$. Hence, we can leverage state-of-the-art Cholesky kernels to compute all information we will need from $\mathbf{H}^{-1}$ upfront. In combination with mild dampening, the resulting method is robust enough to execute on huge models without issues. As a bonus, using a well-optimized Cholesky kernel also yields further speedup. We detail all small changes necessary for the Cholesky version of the algorithm next.

The Full Algorithm. Finally, we present the full pseudocode for GPTQ in Algorithm 1, including the optimizations discussed above.

Algorithm 1 Quantize $\mathbf{W}$ given inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$ and blocksize B.

$\mathbf{Q} \leftarrow \mathbf{0}_{d_{\text{row}} \times d_{\text{col}}}$ *// quantized output*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ *// block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})^\top$ *// Hessian inverse information*
for $i = 0, B, 2B, \ldots$ **do**
  for $j = i, \ldots, i + B - 1$ **do**
    $\mathbf{Q}_{:,j} \leftarrow \text{quant}(\mathbf{W}_{:,j})$ *// quantize column*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{Q}_{:,j}) \,/\, [\mathbf{H}^{-1}]_{jj}$ *// quantization error*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update weights in block*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update all remaining weights*
end for

Post-Training Piecewise Linear Quantization for Deep Neural Networks

3 Quantization Schemes

In this section, we review a uniform quantization scheme and discuss its limitations. We then present PWLQ, our piecewise linear quantization scheme and show that it has a stronger representational power (a smaller quantization error) compared to the uniform scheme.

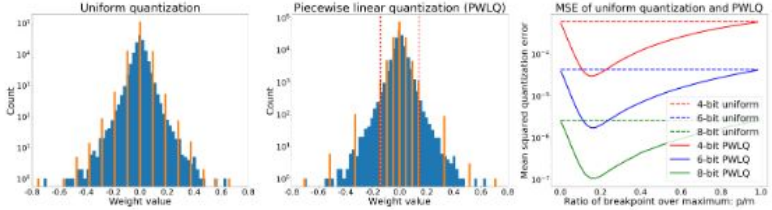


Fig. 1. Quantization of *conv*4 layer weights in a pre-trained Inception-v3. Left: uniform quantization. Middle: piecewise linear quantization (PWLQ) with one breakpoint, dotted line indicates the breakpoint. Right: Mean squared quantization error (MSE) for various bit-widths ($b = 4, 6, 8$). MSE of PWLQ is convex *w.r.t.* the breakpoint p, the b-bit PWLQ can achieve a smaller quantization error than the b-bit uniform scheme

3.1 Uniform Quantization

Uniform quantization (the left of Figure 1) linearly maps full-precision real numbers r into low-precision integer representations. From [25,7], the approximated version $\hat{r}$ from uniform quantization scheme at b-bit can be defined as:

$$\begin{aligned} \hat{r} &= \text{uni}(r; b, r_l, r_u, z) = s \times r_q + z, \\ r_q &= \left\lceil \frac{\text{clamp}(r; r_l, r_u) - z}{s} \right\rfloor_{\mathbb{Z}_b}, \\ \text{clamp}(r; r_l, r_u) &= \min(\max(r, r_u), r_l), \\ s &= \frac{\Delta}{N-1}, \quad \Delta = r_u - r_l, \quad N = 2^b, \end{aligned} \tag{1}$$

where $[r_l, r_u]$ is the quantization range, s is the scaling factor, z is the offset, N is the number of quantization levels, r_q is the quantized integer computed by a rounding function $\lceil \cdot \rfloor_{\mathbb{Z}_b}$ followed by saturation to the integer domain

$\mathbb{Z}_b$. We set the offset $z = 0$ for symmetric signed distributions combined with $\mathbb{Z}_b = \{-2^{b-1}, ..., 2^{b-1} - 1\}$ and $z = r_l$ for asymmetric unsigned distributions (e.g., ReLU-based activations) with $\mathbb{Z}_b = \{0, ..., 2^b - 1\}$. Since the scheme (1) introduces a quantization error defined as $\varepsilon_{uni} = \hat{r} - r$, the expected quantization error squared is given by:

$$\mathbb{E}(\varepsilon_{uni}^2; b, r_l, r_u) = \frac{s^2}{12} = C(b)\Delta^2, \tag{2}$$

with $C(b) = \frac{1}{12(2^b-1)^2}$ under uniform distributions [58].

From the above definition, uniform quantization divides the range evenly despite the distribution of r. Empirically, the distributions of weights and activations of pre-trained DNNs are similar to bell-shaped Gaussian or Laplacian [17,35]. Therefore, uniform quantization is not always able to achieve small enough approximation error to maintain model accuracy, especially in low-bit cases.

3.2 Piecewise Linear Quantization (PWLQ)

To improve model accuracy for quantized models, we need to approximate the original model as accurately as possible by minimizing the quantization error. We follow this natural criterion to investigate the quantization performance, even though no direct relationship can easily be established between the quantization error and the final model accuracy [7].

Inspired from [43,26] that takes advantage of bell-shaped distributions, our approach based on piecewise linear quantization is designed to minimize the quantization error. It breaks the quantization range into two non-overlapping regions: the dense, central region and the sparse, high-magnitude region. An equal number of quantization levels $N = 2^b$ is assigned to these two regions. We chose to use two regions with one breakpoint to maintain simplicity in the inference algorithm (Section 5.1) and the hardware implementation (Section 4). Multiple-region cases are discussed in Section 5.1.

Therefore, we only consider one breakpoint p to divide the quantization range[3] $[-m, m]$ $(m > 0)$ into two symmetric regions: the center region $R_1 = [-p, p]$ and the tail region $R_2 = [-m, -p) \cup (p, m]$. Each region consists of a negative piece and a positive piece. Within each of the four pieces, $(b-1)$-bit $(b \geq 2)$ uniform quantization (1) is applied such that including the sign every

inference algorithm (Section 5.1) and the hardware implementation (Section 4). Multiple-region cases are discussed in Section 5.1.

Therefore, we only consider one breakpoint p to divide the quantization range[3] $[-m, m]$ $(m > 0)$ into two symmetric regions: the center region $R_1 = [-p, p]$ and the tail region $R_2 = [-m, -p) \cup (p, m]$. Each region consists of a negative piece and a positive piece. Within each of the four pieces, $(b-1)$-bit $(b \geq 2)$ uniform quantization (1) is applied such that including the sign every value in the quantization range is being represented into b-bit. We define the b-bit piecewise linear quantization (denoted by PWLQ) scheme as:

$$\mathrm{pw}(r; b, m, p) = \begin{cases} \mathrm{sign}(r) \times \mathrm{uni}(|r|; b-1, 0, p, 0), r \in R_1 \\ \mathrm{sign}(r) \times \mathrm{uni}(|r|; b-1, p, m, p), r \in R_2 \end{cases}, \tag{3}$$

where the sign of full-precision real number r is denoted by $\mathrm{sign}(r)$. The associated quantization error is defined as $\varepsilon_{pw} = \mathrm{pw}(r; b, m, p) - r$.

[3] Here we consider symmetric quantization range $[-m, m]$ $(m > 0)$ for simplicity, it is extendable to asymmetric ranges $[m_1, m_2]$ for any real numbers $m_1 < m_2$.

Figure 1 shows the comparison between uniform quantization and PWLQ on the empirical distribution of the *conv*4 layer weights in a pre-trained Inception-v3 model [54]. We emphasize that b-bit PWLQ represents FP32 values into b-bit integers to support b-bit multiply-accumulate operations, even though in total, it has the same number of quantization levels as $(b+1)$-bit uniform quantization. The implications of this are further discussed in Section 4.