Model Compression: Quantization

This assignment focuses on different methods to reduce the precision of weights and activations of neural networks using uniform quantization. We start with the definition of our quantization function, and then we move to the questions. In the first set of questions, you will implement a quantization block based on different methods, which you will then use to quantize your neural network. For further reading, you can refer to Nagel *et al.* [1], Gholami *et al.* [2], and Li *et al.* [3]. First, read all of the assignment questions and, then, follow the instructions in the `main.ipynb` of the Colab notebook.

Uniform quantization

Uniform quantization is a method for mapping a real-valued vector $\boldsymbol{x}$ to a discrete grid of integers. The mapping can be either signed or unsigned, depending on the application. The mapping is done using the following transformation for each element:

$$x_{\text{int}} = \text{clip}\left(\left\lfloor \frac{x}{s} \right\rceil - z; L, U\right), \tag{1}$$

where s is the scaling factor, z is the zero point, and $\lfloor \cdot \rceil$ is the round-to-nearest integer operator. For unsigned quantization on b bits, $L = 0$ and $U = 2^b - 1$, whereas for signed quantization we use $L = -2^{b-1}$, and $U = 2^{b-1} - 1$. The function $\text{clip}(x; L, U)$ is defined as:

$$\text{clip}(x; L, U) = \begin{cases} L, & x < L, \\ x, & L \leq x \leq U, \\ U, & x > U\,. \end{cases} \tag{2}$$

This function ensures that the quantized value of x lies within the quantization range L, U. To recover the original real-valued vector $\boldsymbol{x}$ from its quantized representation $\boldsymbol{x}_{\textbf{int}}$, we apply the following de-quantization step:

$$\widehat{\boldsymbol{x}} = s \times (\boldsymbol{x}_{\textbf{int}} + z). \tag{3}$$

The quantization grid limits, denoted by $(q_{\min}, q_{\max})$, can be determined by the scaling factor and zero point used in the quantization process. These limits define the minimum and maximum values that an element of the recovered real-valued vector $\hat{\boldsymbol{x}}$ can have. Any elements in the original real-valued vector $\boldsymbol{x}$ that fall outside of this range will be clipped. The process of determining the appropriate clipping range is referred to as calibration. The ultimate goal of calibration is to ensure that the recovered vector $\hat{\boldsymbol{x}}$ is as close as possible to the original vector $\boldsymbol{x}$.

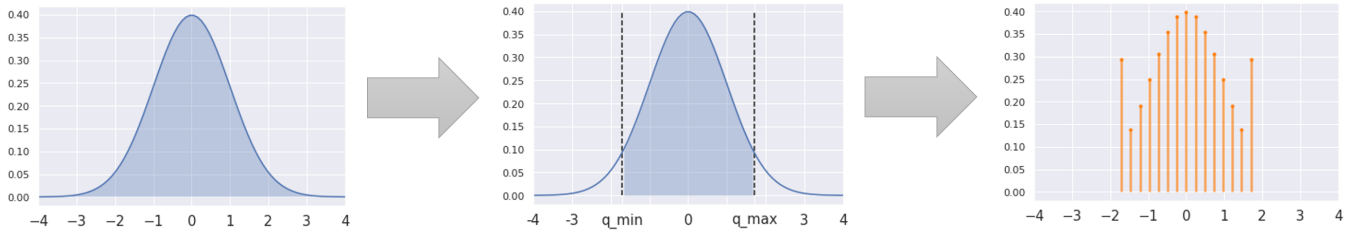

Figure 1: Distribution of vector $\boldsymbol{x}$ before and after quantization.

1. **Calibration (30 points)**

   Consider a signed vector $\boldsymbol{x} \in \mathbb{R}^n$ that needs to be quantized to $\boldsymbol{x}_{\text{int}}$, as in equation (1). In the case of symmetric quantization, the zero point is set to $z = 0$. Otherwise if $z \neq 0$, we say that the quantization is asymmetric. A typical objective when quantizing DNNs is to minimize the error between the original vector $\boldsymbol{x}$ and the recovered vector $\hat{\boldsymbol{x}}$, by considering the mean squared error (MSE), but this can be difficult to do in practice. A simple way to choose the scaling factor is to set $s = \max |\boldsymbol{x}|/2^{b-1}$, where b is the bit precision. This ensures that the largest absolute value of the vector is mapped to the largest representable integer value within the bit precision, but does not necessarily minimize the MSE. For an asymmetrical distribution, a simple method is to shift the distribution towards the positive values, then apply unsigned quantization.

   Many approaches have been proposed to analytically or empirically minimize the MSE, that is to find

   $$s^* = \arg\min_s \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2 \,. \tag{4}$$

   This may be solved heuristically by first arbitrarily restricting s to a small set of candidate values and then performing an exhaustive search over that set, or if assumptions about the distribution of $\boldsymbol{x}$ are made, a good approximate solution may be reached at a low computational cost. For instance, Choi *et al.* [4] propose an empirical solution, known as SAWB, for choosing the best scaling factor. They consider that the elements of $\boldsymbol{x}$ are realizations of a random variable X and they estimate the optimal quantization range based on the first and second moments of X for the symmetric quantization case ($z = 0$) as follows:

   $$\alpha^* = c_1 * \sqrt{\mathbb{E}(X^2)} - c_2 * \mathbb{E}(|X|)\,, \tag{5}$$

   where c_1 and c_2 are constants determined by the desired bitwidth, and the scaling factor is given by $s = \alpha^*/2^{b-1}$.

   (a) Complete `linear_quantize`, `linear_dequantize`, `reset_scale_and_zero_point`, and `get_scale` functions in `solution.py`. (18 points)

   (b) Choose scaling factor s using each of the three methods mentioned above, along with the asymmetric unsigned method for two different datasets (Dataset A and Dataset B). Explain the different approaches and report the measured MSE between the input vector $\boldsymbol{x}$ and the recovered vector $\hat{\boldsymbol{x}}$ in the table provided. (6 points)

   (c) Compare your results. Which method works better? Do you think the quantization error has a bias? Explain your observation. (6 points)

2. **PTQ -vs- QAT (30 points)**

Post-training quantization (PTQ) is a method for reducing the bitwidth of a pre-trained model and does not involve any fine-tuning. Due to the weights of a pre-trained neural network being fixed, you can directly apply weight quantization. On the other hand, since intermediate activations depend on the neural network input, implementing activation quantization is not as straightforward. One approach is to apply zero-shot-quantization (ZSQ) [5] by using a small set of synthetic data for calibration.

Quantization Aware Training (QAT) is a method that trains a model to handle quantization by inserting quantization and de-quantization steps into the training process. It uses the Straight-Through Estimator (STE) [6] to allow gradients to flow through quantization during training by treating it as an identity function.

In this assignment, a key focus is on implementing integer-only linear and conv2d layers. Instead of using the built-in linear and conv2d layers provided by Pytorch, you will be implementing your own versions that work with integers rather than floating-point numbers. Let's see how a convolution or fully-connected (FC) layer is quantized in signed symmetric for weights and unsigned symmetric for activations on a neural network.

$$\begin{aligned} y &= \sum xw + b \\ &= \sum \left(s_x x_{int}\right)\left(s_w w_{int}\right) + b_{int}(s_x s_w) \\ &= s_x s_w \left(\sum x_{int} w_{int} + b_{int}\right), \end{aligned} \tag{6}$$

where for simplicity we have set the scaling factor of the bias to $s_x s_w$. So by keeping track of the scaling factors we can implement these layers with integer operations.

(a) Complete `_quantize_func_STE` so that the quantization block (linear quantization and recovery functions together) follow the STE rule. (2 points)

(b) Complete `quantized_linear_function` and `quantized_conv2d_function` using only `integer_linear` and `integer_conv2d`. (10 points)

(c) Implemented functions are used in modified quantized *Conv2d* and *Linear* layers. Check the test accuracy of pre-trained ResNet32 and a Tiny ViT (Vision Transformer [7]) on CIFAR10 for `W8A8`, `W4A4`, `W2A2`, `W4A2`, and `W2A4` quantizations. (3 points)

Hint: For W4A4 you have to get at least 80% test accuracy for the ResNet32.

(d) Fine-tune the mentioned quantized models using any desired training method, and save the best performing model in the format of `ResNet32_W8A8.pt` for the ResNet32 and `TinyViT_W8A8.pt` for the Tiny ViT(e.g. for 8-bit weight and 8-bit activation quantization.) Evaluate the final performance, fill in Table 2, and compare the results with PTQ. You can use `plot_layers_histogram` function for the comparison. Compare the results

Dataset	(A)			(B)		
Bit width	8	4	2	8	4	2
Symmetric	??	??	??	??	??	??
Asymmetric	??	??	??	??	??	??
Heuristic method	??	??	??	??	??	??
SAWB	??	??	??	??	??	??

Table 1: Test accuracy of quantized ResNet32 on CIFAR10

	PTQ		QAT	
	SAWB	symmetric	SAWB	symmetric
W8A8	??	??	??	??
W4A4	??	??	??	??
W4A2	??	??	??	??
W2A4	??	??	??	??
W2A2	??	??	??	??

Table 2: Test accuracy of quantized Tiny ViT on CIFAR10

	PTQ	QAT
	symmetric	symmetric
W8A8	??	??
W4A4	??	??
W4A2	??	??
W2A4	??	??
W2A2	??	??

between the 2 architectures. Document the chosen training method and methodology in the report. (15 points)

Hint: After fine-tuning for W4A4 you have to get at least 90% test accuracy for ResNet32.

3. **Variable precision (20 points)**

   With variable precision, the weights in each layer of the neural network may be quantized to a different bit width. In this question, you will aim to find the best model size of a pre-trained ResNet32 and a Tiny ViT model trained on the CIFAR-10 dataset. Note that this question focuses only on weight quantization (with signed symmetric method) and we keep activations at 16 bits. For the Tiny ViT, due to the big number of layers, we ask you to quantize all the similar layers to the same quantization (as in the notebook).

   (a) Use a method of your choice to find the optimal quantized model size with a constraint on test accuracy above 85%. Any reasonable attempt at exploring the design space will give you the full marks. Please keep in mind that the computing resources you have access to within Colab limit the number of cases that can be evaluated. The best approaches will be considered for bonus points.

4. **Huffman Coding (20 points)**

   Huffman coding is a common method used in lossless data compression that assigns variable-length codes to input symbols based on their frequencies, with more frequent symbols receiving shorter codes. This approach results in efficient encoding and a straightforward decoding

method. In this context, the term "alphabet" refers to the set of symbols we encode, the size of the alphabet plays a crucial role in the compression process. Varying the alphabet size allows you to experiment with different levels of grouping and coding efficiency. Keep in mind that a well-chosen alphabet size can strike a balance between compression gains and decoding complexity.

In this question, you will apply Huffman coding to the quantized 2-bit weights of a ResNet32 model trained on CIFAR10.

(a) Encode the weights of the convolutional layer and the fully connected layer (classification layer) with different alphabets ranging from the scalar coding where we encode individual quantized weights to vector coding where we encode 1D or 2D tensors like rows or columns or even groups of rows or columns. (10 points)

(b) Compute the compression ratio for each alphabet. (5 points)

(c) Document your analysis in the report. (5 points)

Deliverables

The answers to all the questions should be presented in a single report. Please submit the code `solution.py` and the saved models for question 2-d.

Report and code are due on Sunday February 11, 23h59.

References

[1] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[2] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[3] Yuhang Li, Mingzhu Shen, Jian Ma, Yan Ren, Mingxin Zhao, Qi Zhang, Ruihao Gong, Fengwei Yu, and Junjie Yan. MQBench: Towards reproducible and deployable model quantization benchmark. In *Neural Information Processing Systems: Datasets and Benchmarks Track*, 2021.

[4] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. *Proceedings of Machine Learning and Systems*, 1:348–359, 2019.

[5] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *International Conference on Computer Vision*, 2019.

[6] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.