Reducing the size of LLMs Using Quantization
Brief Research Task

Abdelrahman Elaraby

September 2025

Contents

1 Introduction

Large Language Models (LLMs) like BERT or LLaMA introduced a turning point in the field of Artificial Intelligence (AI) by they remarkable effectiveness across various tasks like text classification, question answering, and text generation. However, these models are enormously large and contain hundreds of millions to billions of parameters, which makes them: **memory-intensive** requiring substantial RAM and storage, **compute-heavy** as in slow in inference and training, and **impractical to deploy** on hardware constrained environments like edge devices. To tackle these pain points, researchers proposed a technique called quantization, which in its essence, compresses the model size without significantly affecting performance.

2 What is Quantization?

Quantization can be described as a model compression technique that reduces the precision of the numbers that the model is composed of (its weights). The standard precision used in neural networks is 32-bit floating-point (FP32) numbers. In Figure 1 the encoding of two variables is shown in bits. in order to convert this bit format to the floating value we use the equation: $\textbf{value} = (-1)^S \times 1.M \times 2^{E-127}$; where S = sign bit, M = mantissa (fraction), and E = exponent (8-bit, biased by 127).

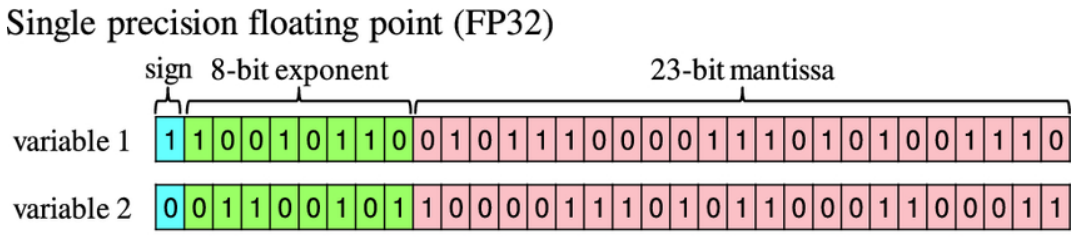


Figure 1: Visualization of encoding 2 variables in FP32 precision.

The quantization methods converts these FP32 numbers to lower-precision formats like 16-bit floating-point (FP16), 8-bit integers (INT8) or even 4-bit integers (INT4), thus using fewer number of bits. The integer based is more preferred practically because the arithmetic operations are much faster on hardware optimized for integer matrix multiplications.

To see how much memory is reduced by quantization, let's look at the LLaMA 7B model. This model has approximately $N = 7 \times 10^9$ parameters. If we store each parameter in FP32 format, which requires 32 bits (4 bytes), the total memory usage is $\text{Memory}_{\text{FP32}} = N \times 4 = 7 \times 10^9 \times 4 = 28$ GB. While by quantizing the same model to INT8, where each parameter is stored in 8 bits (1 byte), the memory usage becomes $\text{Memory}_{\text{INT8}} = N \times 1 = 7 \times 10^9 \times 1 = 7$ GB.

Therefore, INT8 quantization reduces the memory requirement by a factor of: $\frac{\text{Memory}_{\text{FP32}}}{\text{Memory}_{\text{INT8}}} = \frac{28}{7} = 4$, making it significantly more memory-efficient for inference on large language models.

The benefits of quantization can be summarized as:

- Smaller model size $\rightarrow$ reduced storage requirements.
- Faster inference $\rightarrow$ lower latency.
- Lower energy consumption $\rightarrow$ more efficient deployment.

Sometimes we need to convert the values from INT8 to FP32 in a process known as **Dequantization**. This is done because some operations in neural networks cannot be done in low precision without losing accuracy; For example, softmax, layer normalization, or certain activation functions often require FP32 computation. The computation frameworks store weights in INT8 but temporarily dequantize them during critical operations.
Another related concept is **Calibration**; it is the process of determining the scale and zero-point to map floating-point values to integers accurately in an attempt to reduce the quantization error by finding the optimal ranges of weights. It is usually used in Post-Training Quantization (PTQ) - a type that will be discussed in the next section - when the model is already trained. In this process we run representative data through the model to measure the min and max values of activations, then these statistics are then used to quantize activations properly.

3 Types of Quantization

Quantization can be divided into two main types: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). Each approach has its own advantages and trade-offs depending on the deployment scenario. Both of these techniques can be either **Symmetric** or **Asymmetric** quantization.

3.1 Symmetric vs Asymmetric Quantization

In symmetric quantization, the zero-point is fixed at zero after the mapping from float to integer, and the mapped integer range is symmetric around that zero. In asymmetric quantization, a non zero zero-point is introduced to better cover the range of values, which can improve representation efficiency for data with non-symmetric distributions. Figure 2 shows the visual difference between the two.

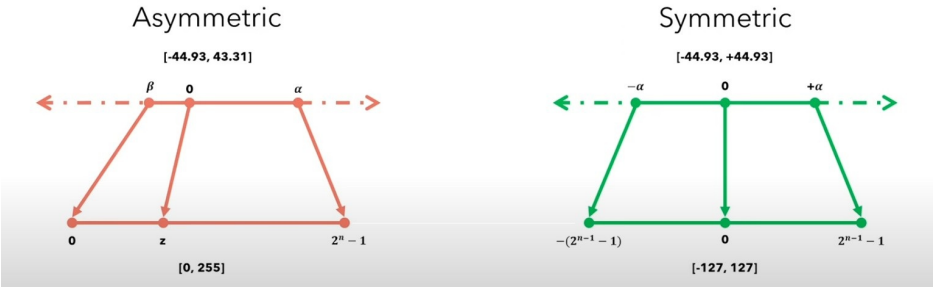


Figure 2: Illustration of symmetric (right) vs. asymmetric (left) quantization.

The equations used for each kind is shown below:
For Asymmetric Quantization:

$$x_q = \text{clamp}\Big(\Big\lfloor \frac{x_f}{s} \Big\rfloor + z;\ 0;\ 2^n - 1\Big), \quad s = \frac{\alpha - \beta}{2^n - 1}, \quad , \quad z = \Big\lfloor -\frac{\beta}{s} \Big\rfloor \tag{1}$$

and its Dequantization is done by:

$$x_f = s \cdot (x_q - z) \tag{2}$$

While for the Symmetric Quantization:

$$x_q = \text{clamp}\Big(\Big\lfloor \frac{x_f}{s} \Big\rfloor;\ -(2^{n-1} - 1);\ 2^{n-1} - 1\Big), \quad s = \frac{|\alpha|}{2^{n-1} - 1} \tag{3}$$

and its Dequantization is done by:

$$x_f = s \cdot x_q \tag{4}$$

Where each of the symbols mean:

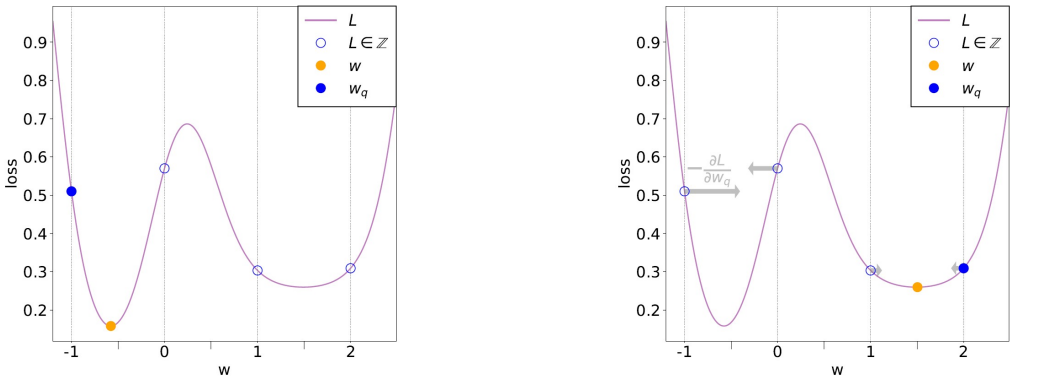
- x_f = original floating-point value
- x_q = quantized integer value
- s = scaling factor that maps floating-point values to integers
- z = zero-point (used only in asymmetric quantization to shift the range)
- α = maximum value of the floating-point range
- β = minimum value of the floating-point range
- n = number of bits used for quantization (e.g., $n = 8$ for INT8)
- clamp$(\cdot)$ = operation that restricts the value within the valid integer range
- $\lfloor \cdot \rfloor$ = rounding (floor) operation

3.2 Post-Training Quantization (PTQ)

Post-Training Quantization is applied to a model that is already pre-trained, and this is done without any additional training. This method is relatively straightforward and requires minimal effort. It is usually used when the deployment environment is computationally constrained and won't make it feasible to retrain. So it is quick to implement, however, a main disadvantage is that it slightly reduces model accuracy, especially for sensitive layers or small models.

3.3 Quantization-Aware Training (QAT)

The second type, Quantization-Aware Training is basically trying to simulate the effect of quantization during training, hoping that the model parameters will learn how to overcome quantization errors. The intuition behind QAT is that the model is encouraged to find a "wide minimum" along the loss curvature. Why wide minimum? so that small perturbations in weights—like those caused by quantization—do not lead to large increases in loss. This makes the model more robust to quantization, resulting in better performance. You can see this intuition highlighted in Figure 3.



(a) Post training quantization

(b) After quantization aware fine-tuning

Figure 3: Intuition behind QAT (b): the model aims to find a wide minimum along the loss curve so that small weight perturbations from quantization do not significantly increase the loss. *Image Source: Wu et al. [2020]*

4 Conclusion

To sum things up, Quantization is a very powerful technique proposed by the researchers to reduce the size and computational requirements of large deep neural network models like LLMs. It works by lowering the precision of the weights making them stored in less number of bits. And not just less number of bits, but also the integer representation (like INT8) is better for the hardware optimized for integer operations (much easier than floating point operations). Post-Training Quantization (PTQ) is done after the model is already trained, while Quantization-Aware Training (QAT) enables even better performance by making the model robust to quantization errors by incorporating a quantization and dequantization steps inside the training loop. Quantization methods make it feasible to bring large, state-of-the-art models into practical, real-world applications, from cloud servers to edge devices.

References

Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020. URL https://arxiv.org/abs/2004.09602. Accessed: 2025-09-27.