QUANTIZATION IN MACHINE LEARNING

A Comprehensive Guide to Model Optimization

By: Safia Tifour

Professional Technical Documentation

July 30, 2025

Contents

1 What is Quantization in ML?

Quantization is the process of reducing the precision of the numbers used to represent model parameters (weights, biases) and activations (intermediate outputs) from higher-precision formats (e.g., `float32`) to lower-precision formats (e.g., `int8` or `int16`).

1.1 Goal

The primary objectives of quantization are:

- **Make models smaller** (less memory)
- **Run faster** (especially on edge/mobile devices)
- **Reduce power consumption**

2 Mathematical Formulation of Quantization

For a floating point value $x \in \mathbb{R}$, quantization maps it to an integer $x_q \in \mathbb{Z}$:

2.1 Quantization Formula

$$x_q = \text{round}(x/S) + Z$$

Where:

- **S**: scale (a positive real number)
- **Z**: zero point (an integer that maps 0.0 in float to int)
- **round**: rounding to the nearest integer

This maps the real value x into a fixed integer range (e.g., `int8` $\in [-128, 127]$).

2.2 Dequantization Formula

$$x = S \times (x_q - Z)$$

3 Inference with Quantized Models

To avoid converting values back to float ahead of time (explicit dequantization), we perform inference directly in integer space using a mathematically equivalent formulation.

3.1 Inference Flow

Suppose we quantized:

- Input vector x_q with scale S_x, zero-point Z_x
- Weight matrix W_q with scale S_w, zero-point Z_w

Then matrix multiplication is computed as:

$$y_{\text{int32}} = (x_q - Z_x) \times (W_q - Z_w)$$

Then, the final float output is recovered as:

$$y_{\text{float}} = S_x \times S_w \times y_{\text{int32}}$$

Optionally, an output quantization step can be added:

$$y_q = \text{round}(y_{\text{float}}/S_y) + Z_y$$

Note: Intermediate results are often in `int32` to preserve precision, and only final results are optionally dequantized.

4 Types of Quantization

4.1 Post-Training Quantization (PTQ)

Overview: Done **after training** the model. No retraining needed.

Key Steps:
1. Weights quantized statically:

- During PTQ, weights are converted from `float32` to `int8`
- Compute min/max of each layer's weights
- Derive scale and zero-point:

$$\text{scale} = \frac{\text{max} - \text{min}}{q_{\text{max}} - q_{\text{min}}}, \quad \text{zero-point} = \text{round}\left(q_{\text{min}} - \frac{\text{min}}{\text{scale}}\right)$$

- Quantize each weight:

$$w_q = \text{round}\left(\frac{w_f}{\text{scale}} + \text{zero-point}\right)$$

2. Activations quantized using a calibration dataset:

- Run a small representative dataset through the model

- Record min/max of activations at each layer
- Use these to compute scale/zero-point for activations
- This is **static quantization of activations**

Math in inference:

$$Y_f = (\text{scale}_X \cdot \text{scale}_W) \cdot [(X_q - z_X) \cdot (W_q - z_W)] + \text{bias}$$

Pros:

- Simple to apply
- No retraining needed

Cons:

- Accuracy drop may be higher, especially for sensitive models

4.2 Dynamic Quantization

Overview: Done after training. **Only weights are statically quantized. Activations are quantized at runtime** — i.e., dynamically.

Key Steps:
1. Weights statically quantized (same as PTQ)
2. Activations quantized on the fly:

- At inference time, for each batch:
  - Compute min/max of the actual activation values
  - Compute scale and zero-point
  - Quantize the activation tensors just before use
- Matrix multiplications are in `int8`, but activations use per-batch scale

Pros:

- Easier to apply (no calibration dataset needed)
- Works well for NLP models (e.g., LSTM, Transformer)

Cons:

- Not as efficient as full PTQ on hardware
- Slightly more compute overhead for per-batch scale computation

4.3 Weight-only Quantization

Overview: **Only weights are quantized to int8. Activations remain float32.**

Key Steps:

1. Weights statically quantized same as PTQ
2. Activations are **not** quantized
3. Inference uses mixed precision:

$$Y_f = W_q \cdot X_f \quad (\text{convert } W_q \to W_f \text{ before mul})$$

Pros:

- Very minimal accuracy loss
- Easy to implement

Cons:

- Smaller memory savings (only weights)
- Limited acceleration

4.4 Quantization-Aware Training (QAT)

Overview: Model is **trained while simulating quantization noise**. Best accuracy among all quantization types.

Key Steps:
1. Fake quantization layers simulate quantization during forward pass:

$$x_q = \text{round}\left(\frac{x}{\text{scale}} + \text{zero-point}\right) \cdot \text{scale} - \text{zero-point}$$

2. Gradients flow through unquantized tensors (via Straight Through Estimator)
3. After training, convert real weights to `int8` using trained quantization params
Pros:

- Best accuracy retention
- Works well even for sensitive models

Cons:

- More training time
- More complex to implement

5 Accuracy Impact and Trade-Offs

- PTQ is simpler but may drop accuracy on sensitive models
- QAT is more complex but retains better accuracy
- Per-channel quantization mitigates large activation dynamic ranges
- Inference is efficient due to `int8`/`int32` math
- Output can remain quantized or dequantized depending on use case