TinyML Workshop

Efficient Machine Learning Inference and Deployment.

Obed Mogaka | obed.mogaka@mdu.se

HERO Lab, Malardalens University

November 4, 2025

Overview

1. **Why Quantization?**
2. **Fundamentals of Quantization**
   - Quantize to which data format?
   - How to quantize?
   - When to Quantize
3. **Extreme Quantization**
4. Advanced Quantization topics

Why Quantization?

Numbers in Neural Networks

The Core Problem:

- By default, NNs are trained and run using a number format called 32-bit Floating-Point (FP32).
- FP32 is precise but "**heavy**," leading to:
  - Large memory footprint: Models can be hundreds of MBs or even GBs.
  - High memory bandwidth usage: Moving all these numbers around is slow and costly.
  - High power consumption: Complex calculations consume more energy.

The Solution (and the point of this seminar):

- Use "cheaper" number formats to make models smaller, faster, and more energy-efficient.
- This process is called **quantization**.

So, Why Quantization?

Challenge 1: model size

- 4x decrease when using 8-bit parameters.

Challenge 2: Speed

- Hardware is less complex/less area → more throughput

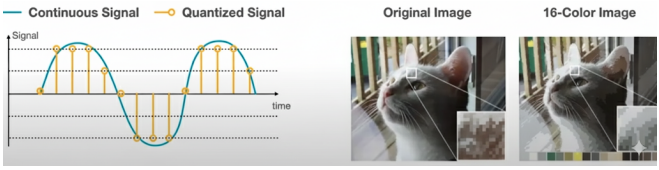
Challenge 3: Energy efficiency.

- Smaller model → less fetches from memory → less energy
- Hardware is less complex → less energy

Problem: Quantization is an irreversible process and causes information loss.

What is Quantization?

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



The Fundamental Trade-off

Precision & Range vs. Efficiency
Think of it like a seesaw. On one end, you have high-detail number formats; on the other, you have efficiency.

High-Precision Formats (like FP32):

- Can represent a vast range of numbers with great detail.
- Slow, require more memory and power.

Low-Precision Formats (like INT8):

- Extremely fast, require less memory and power.
- Can only represent a limited range of numbers with less detail.

Our goal in quantization is to find the sweet spot: get most of the efficiency benefits without losing too much model accuracy.

Number Systems Concepts

Dynamic Range: The span of numbers a format can represent, from the smallest to the largest.

- A number format with a high dynamic range can represent both 0.00001 and 100,000.
- A format with a low dynamic range might struggle with one or both of these extremes.
- In a CNN, dynamic range is crucial for representing both very small weight values (close to zero) and large activation outputs.

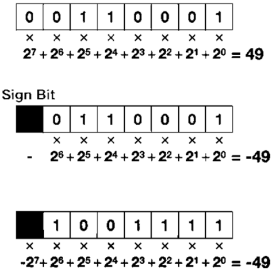
Precision: The level of detail or the gap between two representable numbers. Higher precision means smaller gaps.

- In a CNN, precision determines how closely we can represent the "true" value of a weight.
- The difference between 0.5001 and 0.5002 can be important.

Numeric Data Types

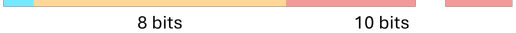
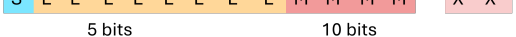
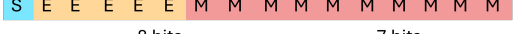
How is numeric data represented in modern computing systems?

- **Unsigned Integer**
  - N-bit Range: $[0, 2^n - 1]$
- **Signed Integer**
  - *Sign-Magnitude Representation*
    - N-bit Range: $[-(2^{n-1} - 1), 2^{n-1} - 1]$
    - Both `000...00` and `100...00` represent 0
  - *Two's Complement Representation*
    - N-bit Range: $[-2^{n-1}, 2^{n-1} - 1]$
    - `000...00` represents 0
    - `100...00` represents -2^{n-1}



Common Number Formats in Deep Learning

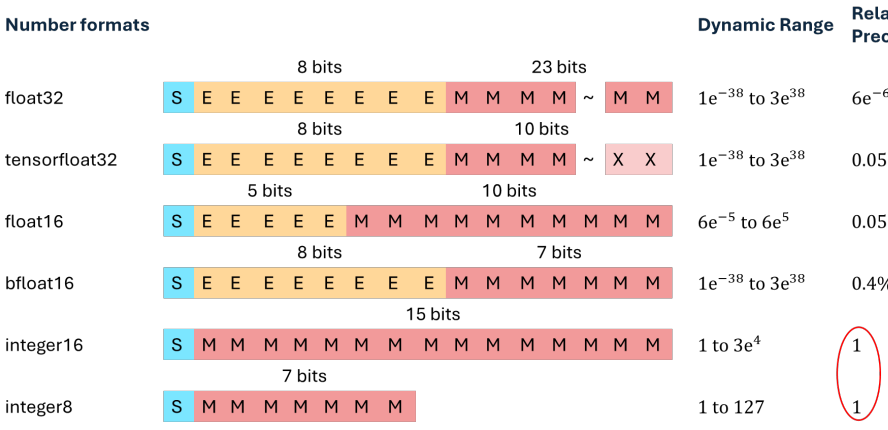
Typical data formats used in DNNs

Number formats		Dynamic Range	Relative Precision
float32		$1e^{-38}$ to $3e^{38}$	$6e^{-6}\%$
tensorfloat32		$1e^{-38}$ to $3e^{38}$	0.05%
float16		$6e^{-5}$ to $6e^{5}$	0.05%
bfloat16		$1e^{-38}$ to $3e^{38}$	0.4%

- **Nvidia** introduced the **Tensorfloat32** that has same 10-bit mantissa as IEEE float16, shown to be sufficient for DNNs.
- **Google** introduced **bfloat16** on the observation that “neural networks are far more sensitive to the size of the exponent than that of the mantissa”. Moreover: the physical size of a bfloat16 multiplier is roughly two times smaller than that of float16

Common Number Formats in Deep Learning

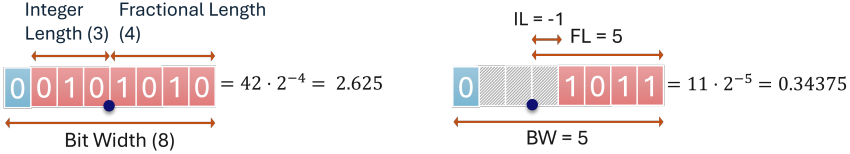
Typical data formats used in DNNs



- Integers are cheap, but their relative precision fluctuates
- Range problem of integers can be solved by the fixed-point number format

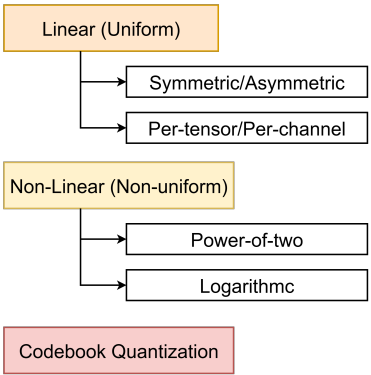
Fixed-point Number Format

- **Formula**: Bit-width $=$ Fractional Length $+$ Integer Length $+ 1$ (Sign)
- **Approximation**: To approximate a real value x, compute $\hat{x} = Round(x \cdot 2^{\text{FL}}) \cdot 2^{-\text{FL}}$
- **Range**: $-2^{\text{IL}} \leq \hat{x} \leq 2^{\text{IL}} - 2^{-\text{FL}}$ with a step size of $2^{-\text{FL}}$



How to Quantize?

How to Quantize?



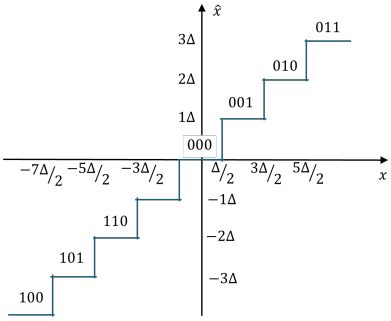
Linear Quantization

Its an affine mapping of integers to real numbers
Linear implies constant step size, Δ
The step size is determined by the range:

- Symmetric or asymmetric mode
- Per-tensor or per-channel

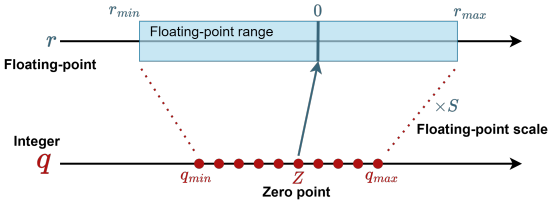
The equation for the quantized value $\hat{x}$ is:

$$\hat{x} = round(x/\Delta) \cdot \Delta$$



3 bits Quantization Example

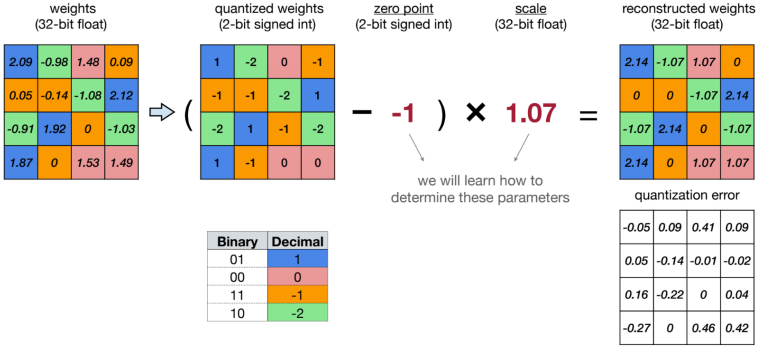
Linear Quantization : Scale



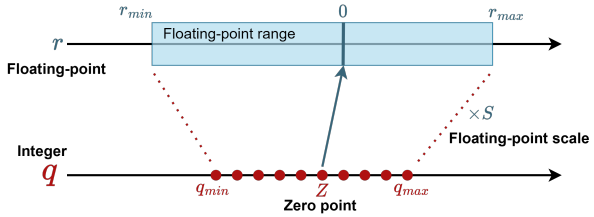
Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

$$r = S(q - Z)$$
$$r_{max} = S(q_{max} - Z)$$
$$r_{min} = S(q_{min} - Z)$$
$$r_{max} - r_{min} = S(q_{max} - q_{min})$$
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

Linear Quantization: Example



Linear Quantization : Zero-point



Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

$$r_{min} = S(q_{min} - Z)$$

$$Z = q_{min} - \frac{r_{min}}{S}$$

$$Z = round(q_{min} - \frac{r_{max}}{S})$$

Quantized Matrix Multiplication

Consider the following matrix multiplication:

$$Y = WX$$

$$S_Y(q_Y - Z_Y) = S_W(q_W - Z_W).S_X(q_X - Z_X)$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W - Z_W)(q_X - Z_X) + Z_Y$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Quantized Matrix Multiplication

Consider the following matrix multiplication:

$$Y = WX$$

$$S_Y(q_Y - Z_Y) = S_W(q_W - Z_W).S_X(q_X - Z_X)$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W - Z_W)(q_X - Z_X) + Z_Y$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_W q_X \underbrace{- Z_X q_W + Z_W Z_X}_{\text{Precompute}}) + Z_Y$$

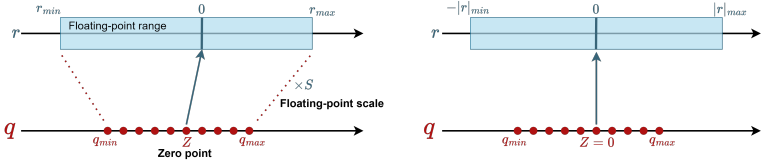
N-bit Integer Multiplication
32-bit Integer Addition/Substraction

N-bit Integer Addition

$$Z_W = 0?$$

Symmetric Linear Quantization

Zero point $Z = 0$ Symmetric floating-point range



Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

$$r_{min} = S(q_{min} - Z)$$

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

$$S = \frac{r_{min}}{q_{min} - Z} = \frac{-|r|_{max}}{q_{min}} = \frac{|r|_{max}}{2^{N-1}}$$

Symmetric Quantized Matrix Multiplication

Consider the matrix multiplication when $Z_W = 0$:

$$Y = WX$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Precompute

N-bit Integer Multiplication
32-bit Integer Addition/Substraction

N-bit Integer Addition

$Z_W = 0$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_X q_W) + Z_Y$$

Quantized FC Layer

So far, we ignore bias. Now we consider the following fully-connected layer with bias:

$$Y = WX + b$$

$$S_Y(q_Y - Z_Y) = S_W(q_W - Z_W).\, S_X(q_X - Z_X) + S_b(q_b - Z_b)$$

$\downarrow Z_W = 0$

$$S_Y(q_Y - Z_Y) = S_W S_X(q_W q_X - Z_X q_W) + S_b(q_b - Z_b)$$

$\downarrow Z_b = 0, S_b = S_W S_X$

$$S_Y(q_Y - Z_Y) = S_W S_X(q_W q_X - Z_X q_W + q_b)$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X + \underbrace{q_b - Z_X q_W}_{\text{Precompute}}) + Z_Y$$

$\downarrow q_{bias} = q_b - Z_X q_W$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X + q_{bias}) + Z_Y$$

Quantized FC Layer

So far, we ignore bias. Now we consider the following fully-connected layer with bias:

$$Y = WX + b$$

$$Z_W = 0$$
$$Z_b = 0, S_b = S_W S_X$$
$$q_{bias} = q_b - Z_X q_W$$

$$q_Y = \frac{S_W S_X}{S_Y}\left(q_W q_X + q_{bias}\right) + Z_Y$$

Rescale to N-bit Int

N-bit Int Mult. 32-bit Int Add.

N-bit Int Add

Quantized Convolution Layer

Consider the following convolution layer:

$$Y = Conv(W, X) + b$$

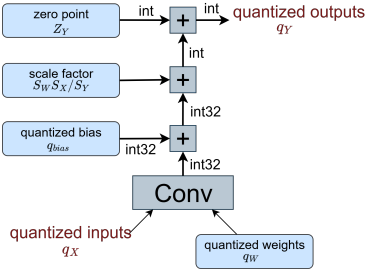
$$Z_W = 0$$
$$Z_b = 0, S_b = S_W S_X$$
$$q_{bias} = q_b - Conv(q_W, Z_X)$$

$$q_Y = \frac{S_W S_X}{S_Y}\left(Conv(q_W, q_X) + q_{bias}\right) + Z_Y$$

Rescale to N-bit Int

N-bit Int Mult. 32-bit Int Add.

N-bit Int Add



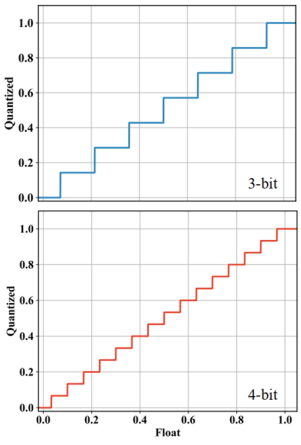
Uniform Quantization: Summary

Formally, uniform quantization maps a real value x to a quantized b-bit integer x_q via an affine transformation defined by the scale factor s and zero-point z.

Given a clipping range $[\alpha, \beta]$, these parameters are computed as:

$$s = \frac{2^b - 1}{\beta - \alpha}, \quad z = -round(\alpha \cdot s) - 2^{b-1},$$

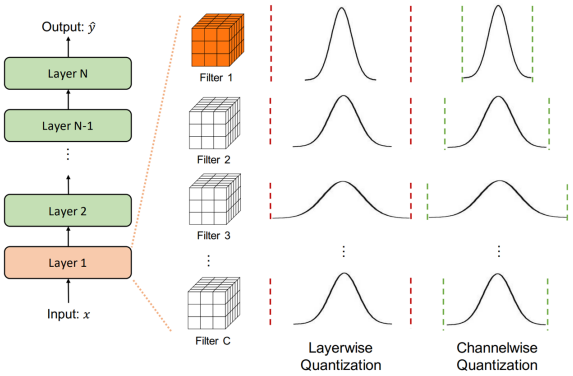
then $$x_q = clip\left(round(s \cdot x + z), -2^{b-1}, 2^{b-1} - 1\right).$$



Quantization Granularity

- Per-Tensor Quantization
- Per-Channel Quantization
- Group Quantization
  - Per-Vector Quantization
  - Shared Micro-exponent (MX) data type.

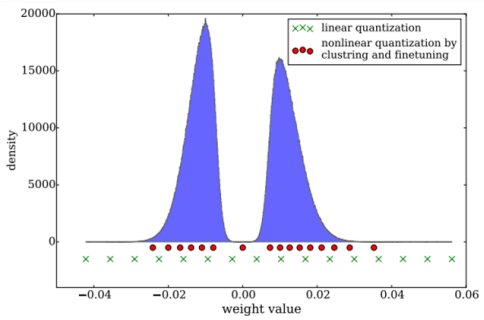
- Per-channel is useful when channels have different ranges



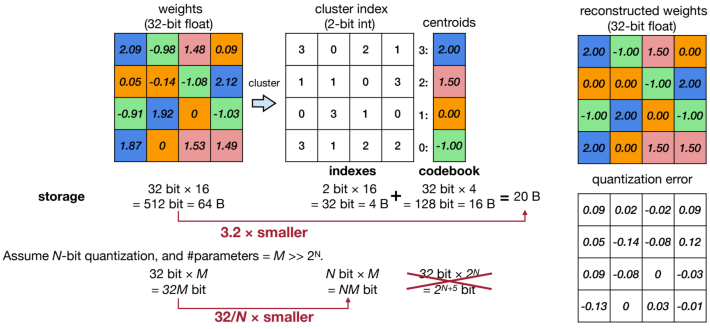
Codebook Quantization

- Useful when distribution of values is not normally distributed
- Quantized values (codewords) are represented by a codebook
- Codewords are selected using a clustering algorithm or training process

Note: requires hardware support for lookup

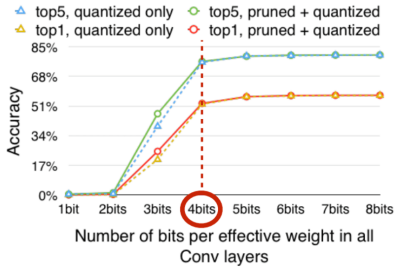


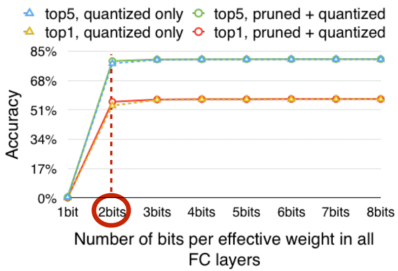
K-Means-based Weight Quantization



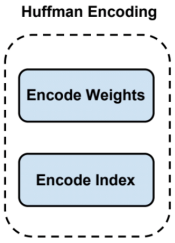
- The weights are decompressed using a lookup table (i.e., codebook) during runtime inference.
- K-Means-based Weight Quantization only saves storage cost of a neural network model.
- All the computation and memory access are still floating-point.

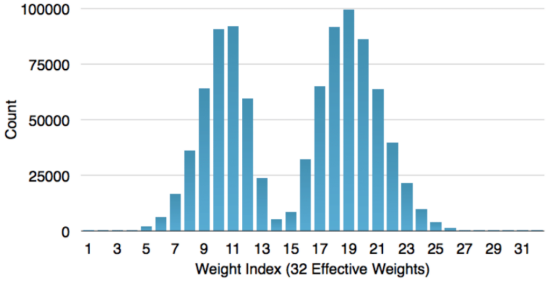
How many bits do we need?





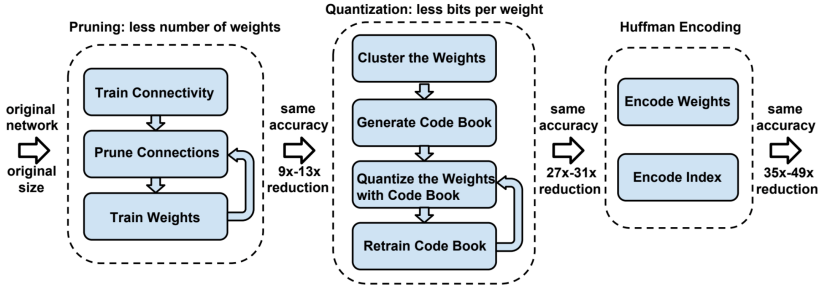
Huffman Coding





- In-frequent weights: use more bits to represent.
- Frequent weights: use less bits to represent.

Deep Compression [1]: Pruning+Quantization+Huffman

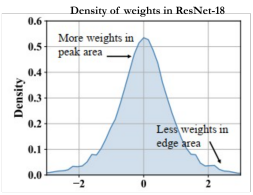


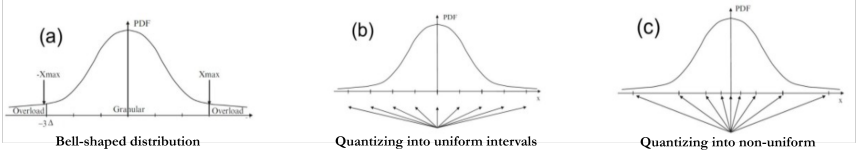
- On the ImageNet dataset, **Deep Compression** reduced the storage required by AlexNet by $35\times$, from 240MB to 6.9MB, VGG-16 by $49\times$ from 552MB to 11.3MB, without loss of accuracy.

[1] Deep Compression [Han et al., ICLR 2016]

Non-uniform Quantization

- In DNNs, weights and activations follow a bell-shaped distribution
- Nevertheless, uniform quantization does not match the distribution of weights (and activations), which is typically bell-shaped.
- When minimizing mean quantization errors, we should weight quantization errors by their probabilities. Nonuniform!

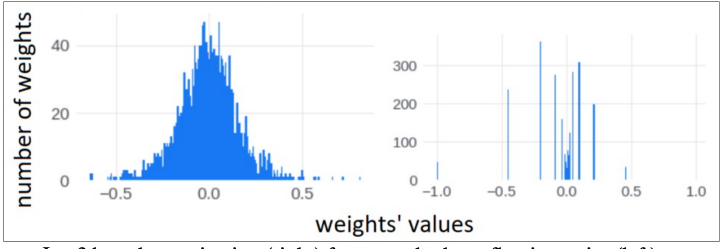




- However, it is difficult to implement the arithmetical operations for the non-uniform quantization levels efficiently.

Logarithmic Quantization

- Logarithmic representations can encode data with very large dynamic range in fewer bits than can fixed point representation (Gautschi et al., 2016).
- Data representation in log-domain is naturally encoded in digital hardware
- Using base-2 logarithmic representation allows optimizing the multiplication by replacing it with bit shifting
- Bit shifting based multiplication which vastly reduces the complexity and thus the power consumption for convolution operations in DNNs.



Log2 based quantization (right) for exemplar layer floating point (left)

$$2^3 \times 101_2 = 101000_2$$

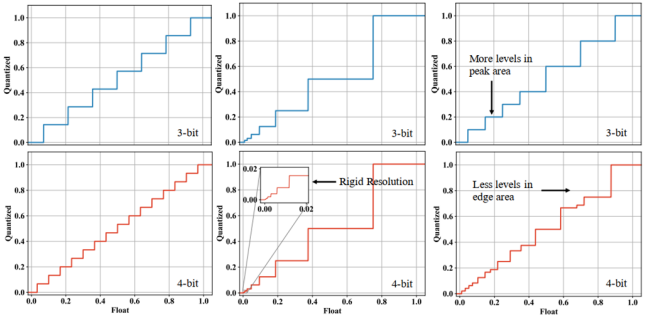
Power-of-Two Quantization

- PoT quantization constrains quantization levels to be powers of two values or zero.

$$Q^p(\alpha, b) = \alpha \times \{0, \pm 2^{2^{b-1}+1}, \pm 2^{2^{b-1}+2}, ..., \pm 2^{2^{-1}}, \pm 1\}$$

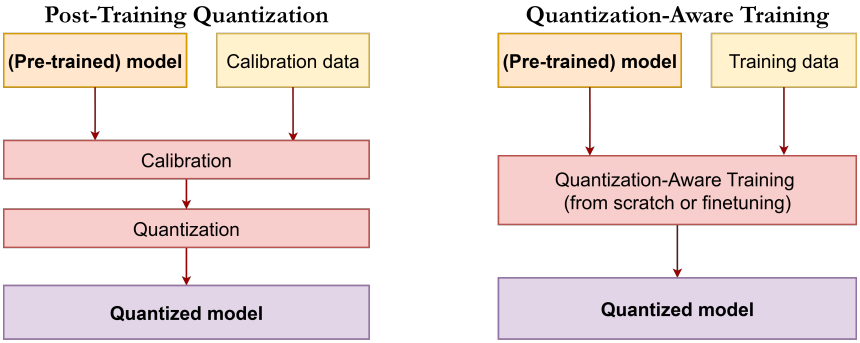
where α is the clipping threshold, and b is the bit-width

- PoT has a higher resolution for the value range with denser weights because of its exponential property.

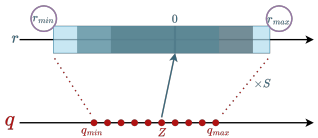


When to Quantize

When to Quantize



How should we get the optimal linear quantization parameters (S,Z)?



- Weight and Bias Quantization
- Activation Quantization

- Unlike weights, the activation range varies across inputs.
- To determine the floating-point range, the activations statistics are gathered before deploying the model.

- Different advanced techniques are used for the range analysis.

Post-Training Quantization

Typical PTQ procedure:

1. Range analysis using calibration data (subset of training data)
   - To determine step size for activations.
   - Step size for weights can be determined without any data.

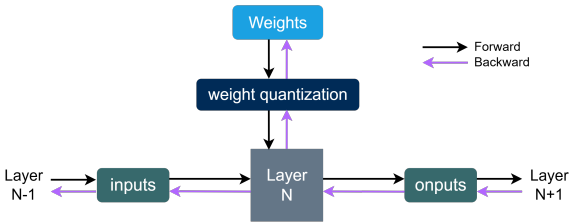
2. Check if model accuracy is satisfactory. If not, perform fine-tuning.

Note: Naive PTQ often results in significant loss of model accuracy.

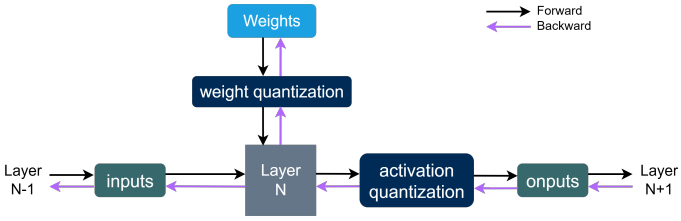
How should we improve performance of quantized models?

Train the model taking quantization into consideration

- To minimize the loss of accuracy, especially in aggressive quantization with lower bit width, the neural network will be trained/fine-tuned with quantized weights and activations.
- Usually, fine tuning a pretrained floating point model provides better accuracy than training from scratch.

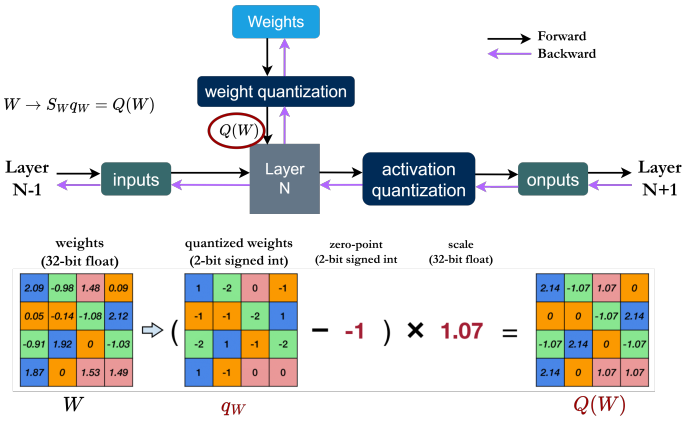


How should we improve performance of quantized models?



Fake Quantization

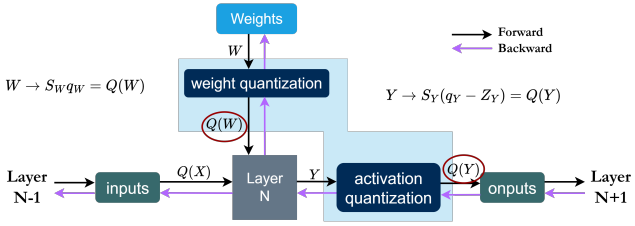
Simulated/Fake Quantization: the model learns from the effects of quantization, but the actual training computations (like gradient updates) are still performed in high precision (FP32).



Quantization-Aware Training

Train the model taking quantization into consideration.

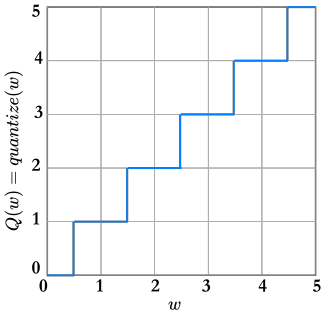
- A full precision copy of the weights W is maintained throughout the training.
- The small gradients are accumulated without loss of precision.
- Once the model is trained, only the quantized weights are used for inference.

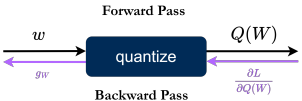


How should gradients back-propagate through the (simulated) quantization?

Quantizers

Quantization is discrete-valued, and thus the derivative is 0 almost everywhere.





$$g_W = \frac{\partial Q}{\partial W} = \frac{\partial L}{\partial Q(W)} \cdot \frac{\partial Q(W)}{\partial W}$$

$$\text{but } \frac{\partial Q(W)}{\partial W} = 0$$

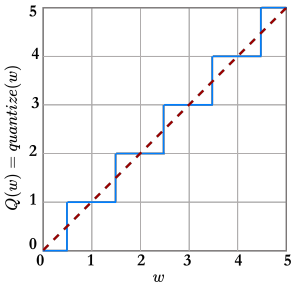
If the gradients become 0, then the weights wont get updated!

$$g_W = \frac{\partial Q}{\partial W} = \frac{\partial L}{\partial Q(W)} \cdot \frac{\partial Q(W)}{\partial W} = 0$$

Straight-Through Estimator

The Straight-Through Estimator (STE) simply passes the gradients through the quantization as if it had been the *identity* function.

$$g_W = \frac{\partial Q}{\partial W} = \frac{\partial L}{\partial Q(W)}$$



Quantization-Aware Training

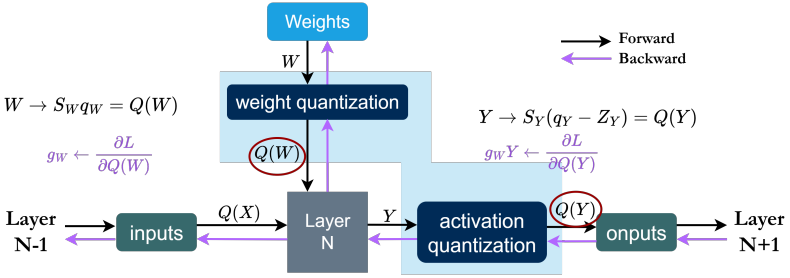


Table 1: Top-1 accuracy (%) comparison illustrating the effect of Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). [2]

Model	FP	Post-Training Quantization		Quantization-Aware Training	
		Asym. Per-Tensor	Sym. Per-Channel	Asym. Per-Tensor	Sym. Per-Channel
MobileNetV1	70.9%	0.1%	59.1%	70.0%	70.7%
MobileNetV2	71.9%	0.1%	69.8%	70.9%	71.1%
NASNet-Mobile	74.9%	72.2%	72.1%	73.0%	73.0%

QAT maintains accuracy closer to the floating-point baseline.

[2] Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." arXiv preprint arXiv:1806.08342 (2018).

Extreme Quantization

Extreme Quantization

Usually 8-bit quantization is sufficient for many applications.
Can we push the quantization precision to 1 bit.

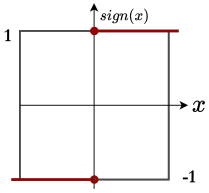
- Binary values $\{-1, +1\}$
  - Single bit per symbol
- Ternary values $\{-1, 0, +1\}$
  - 1.585 bits per symbol ($log_2 3 \approx 1.585$)

Binarization

Deterministic Binarization

- directly computes the bit value based on a threshold.

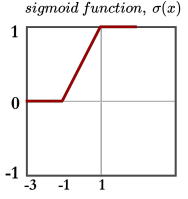
$$q = \text{sign}(x) = \begin{cases} +1, & x \geq 0, \\ -1, & x < 0 \end{cases}$$



Stochastic Binarization

- use global statistics or the value of input data to determine the probability of being -1 or +1.

$$q = \begin{cases} +1, & with\ probability\ p = \sigma(x), \\ -1, & with\ probability\ 1 - p \end{cases}, where \sigma(r) = min(max(\frac{x+1}{2}, 0), 1)$$

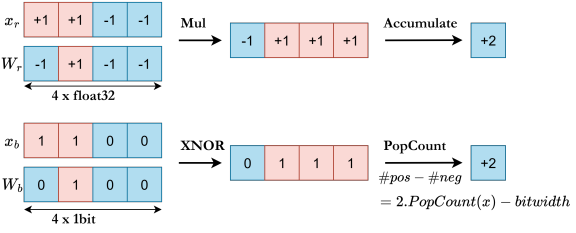


- harder to implement as it requires the hardware to generate random bits when quantizing.

Binary multiply-accumulate

Multiplication and addition are replaced by bitwise XNOR and PopCount.

- The bitwise XNOR (exclusive-NOR) operation results in a '1' if the corresponding bits are the same, '0' otherwise.
- A pop count, also known as Hamming weight simply counts the number of '1's in the binary string.
- This is incredibly fast on modern hardware.



General formulation of a binary MACC:

$$y_i = -n + 2 \cdot \sum_j (W_{ij}\ xnor\ x_j)$$

$$y_i = -n + popcount(W_i\ xnor\ x) \ll 1$$

Impact of Binarization

- Binarizing both weights and activations dramatically reduces computational and memory costs.

Neural Network	Quantization	Bit-Width		ImageNet Top-1 Accuracy Delta
		W	A	
AlexNet	BWN	1	32	0.2%
	BNN	1	1	-28.7%
	XNOR-Net	1	1	-12.4%

input	weight	operations	memory	computation
$\mathbb{R}$	$\mathbb{R}$	+ ×	1×	1×
$\mathbb{R}$	$\mathbb{B}$	+ -	~32× less	~2× less
$\mathbb{B}$	$\mathbb{B}$	xnor, popcount	~32× less	~58× less

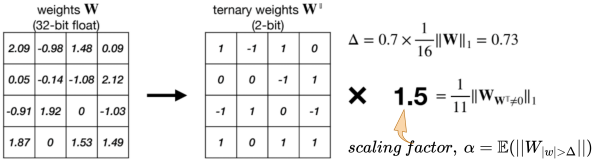
BWN: Binary Weight Network with scale for weight binarization
BNN: Binarized Neural Network without scale factors
XNOR-Net: scale factors for both activation and weight binarization

- **This efficiency comes at a price. Binarization can cause a substantial drop in accuracy.**

Ternary Quantization

Weights are quantized to $+1, -1$ *and* 0.
The inclusion of zero is crucial. It allows the network to effectively "prune" or ignore connections where the original weight was very small. This can save a lot of computation.

$$q = \begin{cases} r_t, & r > \Delta, \\ 0, & |r| \leq \Delta, \\ -r_t & r < -\Delta \end{cases} \quad where\ \Delta = 0.7 \times \mathbb{E}(|r|),\ r_t = \mathbb{E}_{|r|>\Delta}(|r|)$$



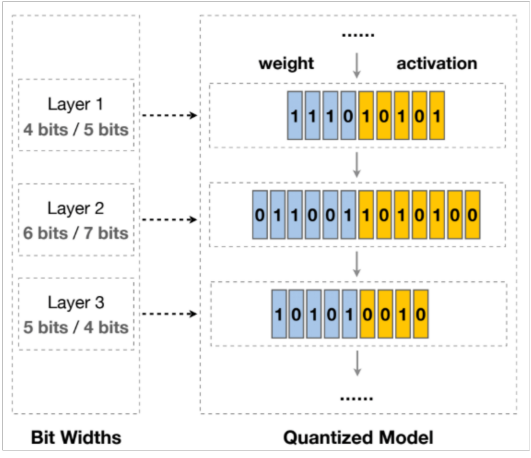
ImageNet Top-1 Accuracy	Full Precision	1 bit (BWN)	2 bit (TWN)
ResNet-18	69.6	60.8	65.3

[3] **TWN**

[3] Li, Fengfu, et al. "Ternary weight networks." arXiv preprint arXiv:1605.04711 (2016)

Advanced Quantization Topics

Mixed-Precision Quantization



Hands-on

“Talk is cheap. Show me the code.”

— Linus Torvalds