EE-508: Hardware Foundations for Machine Learning

Quantization

University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

Instructors:

Arash Saifhashemi

Quantization

Source: https://www.deviantart.com/dnobody/art/8-Bit-Last-Supper-176002023

Scientific Notation

$$+2.32\times 10^{4}$$

Scientific Notation

Decimal Point

Sign

Exponent

$$+2.32\times10^{4}$$

Significand (Mantissa)

Base

Scientific Notation

Sign

Exponent

$$\pm a \times 10^{n}$$

Significand (Mantissa)

Base

Scientific Notation (Non-Zero Values)

Sign

Exponent

$$\pm a \times 10^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 10)$$

$$n \in \mathbb{Z}$$

Scientific Notation (Non-Zero Values)

Sign

Exponent

$$\pm a \times 10^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 10)$$

$$n \in \mathbb{Z}$$

$$0.000{,}000{,}000{,}000{,}000{,}911$$

Scientific Notation (Non-Zero Values)

Sign

Exponent

$$\pm a \times 10^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 10)$$

$$n \in \mathbb{Z}$$

$$0.000,000,000,000,000,911$$

$$9.11 \times 10^{-16}$$

Scientific Notation (Non-Zero Values)

Sign

Exponent

$$\pm a \times 10^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 10) \qquad n \in \mathbb{Z}$$

$$0.000{,}000{,}000{,}000{,}000{,}911 \qquad 9.11 \times 10^{-16}$$

$$149{,}600{,}000$$

Scientific Notation (Non-Zero Values)

Sign

Exponent

$$\pm a \times 10^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 10) \qquad n \in \mathbb{Z}$$

$$0.000{,}000{,}000{,}000{,}000{,}911 \qquad 9.11 \times 10^{-16}$$

$$149{,}600{,}000 \qquad 1.496 \times 10^{8}$$

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm 1.1 \times 2^{2}$$

Significand (Mantissa)

Base

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2)$$

$$n \in \mathbb{Z}$$

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1 \times 2^{1}$

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1 \times 2^{1}$
0.1	0.5	

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1\times2^{1}$
0.1	0.5	$+1\times2^{-1}$

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1\times2^{1}$
0.1	0.5	$+1\times2^{-1}$
11.001		

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1 \times 2^{1}$
0.1	0.5	$+1 \times 2^{-1}$
11.001	3.125	

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2)$$

$$n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1 \times 2^{1}$
0.1	0.5	$+1 \times 2^{-1}$
~~11.001~~ 1.1001	3.125	

Binary Scientific Notation (Non-Zero Values)

Binary Point

Sign

Exponent

$$\pm a \times 2^{n}$$

Significand (Mantissa)

Base

$$a \in [1, 2) \qquad n \in \mathbb{Z}$$

Binary	Decimal	Representation
10	2	$+1\times 2^{1}$
0.1	0.5	$+1\times 2^{-1}$
~~11.001~~ 1.1001	3.125	$+1.1001\times 2^{1}$

Fixed Point vs. Floating Point

$$123.45678$$

123

Fixed point
0 digits after the decimal points

123.46

Fixed point
2 digits after the decimal points

1.2345678×10^{2}

Floating point

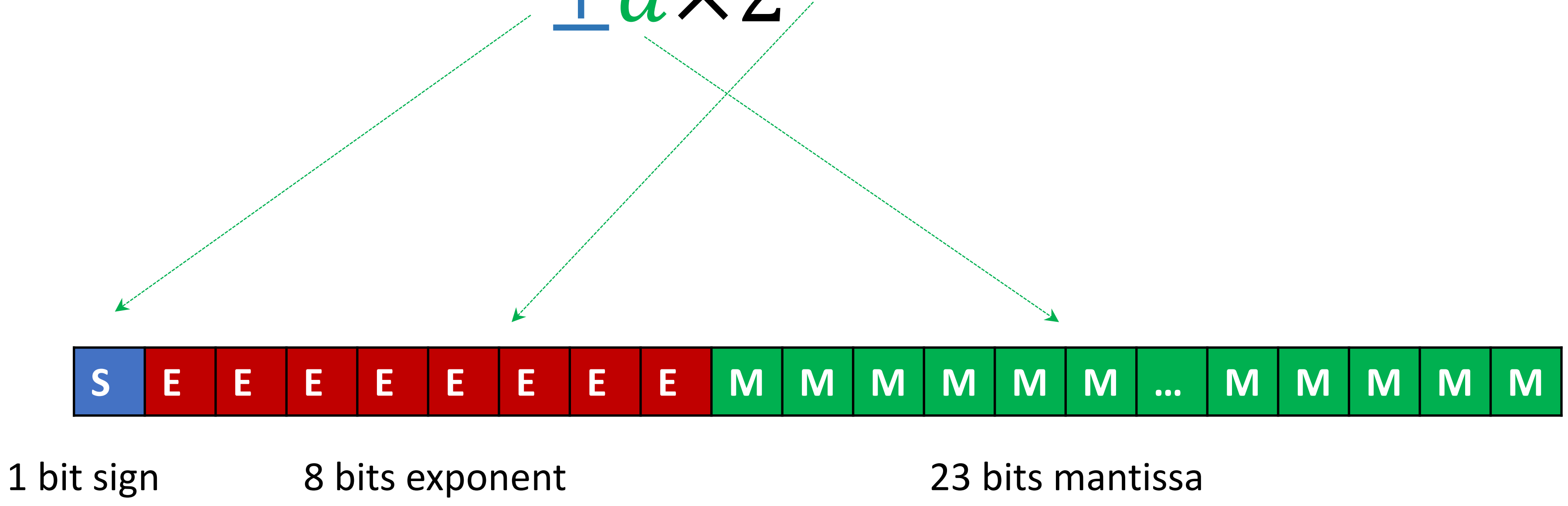
The decimal point shifts based on the exponent.

Floating Point Representation

$$\pm a \times 2^{n}$$

Floating Point Representation

$$\pm a \times 2^{n}$$



IEEE Floating Point Representation (fp32)

$$\pm a \times 2^{n-bias}$$

Bias = 127

Actual Exponent=Stored Exponent−127

Example:

•If the **stored exponent** is 150 (binary: 10010110),

- **Actual exponent** = 150 - 127 = 23

•If the **stored exponent** is 100 (binary: 01100100),

- **Actual exponent** = 100 - 127 = -27

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	…	M	M	M	M	M

1 bit sign

8 bits exponent

23 bits mantissa

Why Use a Bias?

- Biasing the exponent allows it to be stored as an **unsigned integer** (0 to 255) instead of a signed integer (-128 to 127).
- This helps in sorting floating-point numbers in a way that makes comparison operations simpler in hardware.

How Bias Affects Computation

- The biasing affects the range of representable exponents:
  - **Minimum exponent (nonzero values):** $1 - 127 = -126$
  - **Maximum exponent:** $254 - 127 = 127$
- A stored exponent of 0 and 255 are reserved for **special cases**:
  - 0 represents **denormals or zero**.
  - 255 represents **infinity (±∞) or NaN (Not a Number)**.
- Why bias?
  - Simplify comparisons and arithmetic operations in hardware.
  - Instead of storing the exponent as a signed integer (which requires handling negative values separately), it is stored as an unsigned integer with a bias.

IEEE Floating Point Representation (fp32)

$$\pm a \times 2^{n-bias}$$

Bias = 127

Actual Exponent=Stored Exponent−127

Example:
•If the **stored exponent** is 150 (binary: 10010110),
- **Actual exponent** = 150 - 127 = 23

•If the **stored exponent** is 100 (binary: 01100100),
- **Actual exponent** = 100 - 127 = -27

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	…	M	M	M	M	M

1 bit sign

8 bits exponent

23 bits mantissa

Range:

IEEE Floating Point Representation (fp32)

$$\pm a \times 2^{n-bias}$$

Bias = 127

Actual Exponent=Stored Exponent−127

Example:

- If the **stored exponent** is 150 (binary: 10010110),
  - **Actual exponent** = 150 - 127 = 23
- If the **stored exponent** is 100 (binary: 01100100),
  - **Actual exponent** = 100 - 127 = -27

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	…	M	M	M	M	M

1 bit sign

8 bits exponent

There are special values for denoting zero, infinities, and NaN (Not a Number).

23 bits mantissa

Range: $[1 \times 2^{-126},$ $]$

IEEE Floating Point Representation (fp32)

$$\pm a \times 2^{n-bias}$$

Bias = 127

Actual Exponent=Stored Exponent−127

Example:
•If the **stored exponent** is 150 (binary: 10010110),
- **Actual exponent** = 150 - 127 = 23

•If the **stored exponent** is 100 (binary: 01100100),
- **Actual exponent** = 100 - 127 = -27

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	…	M	M	M	M	M

1 bit sign

8 bits exponent

23 bits mantissa

- A stored exponent of 0 and 255 are reserved for **special cases**:
  - 0 represents **denormals or zero**.
  - 255 represents **infinity (±∞) or NaN (Not a Number)**.
- **Minimum exponent (nonzero values):** 1−127=−126
- **Maximum exponent:** 254−127=127

Range: $[1 \times 2^{-126}, (2-2^{-23}) \times 2^{127}]$

IEEE Floating Point Representation (fp32)

$$\pm a \times 2^{n-bias}$$

Bias = 127

Actual Exponent=Stored Exponent−127

Example:

•If the **stored exponent** is 150 (binary: 10010110),
- **Actual exponent** = 150 - 127 = 23

•If the **stored exponent** is 100 (binary: 01100100),
- **Actual exponent** = 100 - 127 = -27

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	…	M	M	M	M	M

1 bit sign   8 bits exponent   23 bits mantissa

- A stored exponent of 0 and 255 are reserved for **special cases**:
  - 0 represents **denormals or zero**.
  - 255 represents **infinity (±∞) or NaN (Not a Number)**.
- **Minimum exponent (nonzero values):** 1−127=−126
- **Maximum exponent:** 254−127=127

Range: $[1\times2^{-126}, (2-2^{-23})\times2^{127}]$ $\cong [1.18\times10^{-38}, 3.4\times2^{38}]$

$1.11111111111111111111111_2$

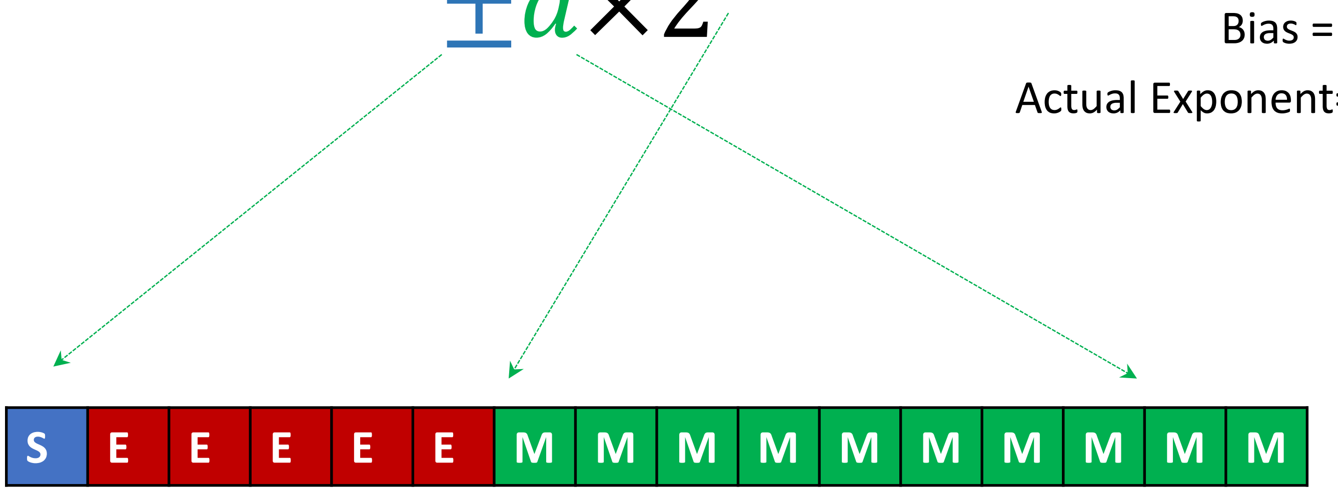
$$1+\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots+\frac{1}{2^{23}}\right)$$

IEEE Floating Point Representation (fp16)

$$\pm a \times 2^{n-bias}$$

Bias = 15

Actual Exponent=Stored Exponent−15



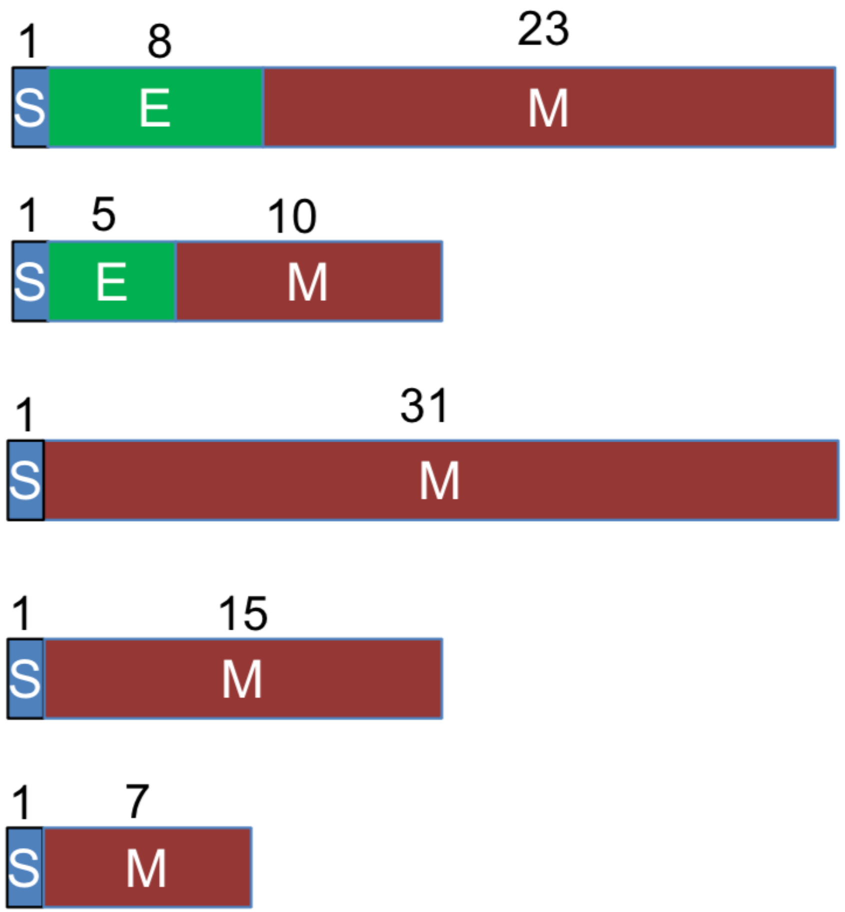
1 bit sign

5 bits exponent

10 bits mantissa

Range: $[1\times 2^{-14}, (2-2^{-10})\times 2^{15}]$ $\cong [6.1035\times 10^{-5}, 6.5504\times 10^{4}]$

Summary of Number Representations

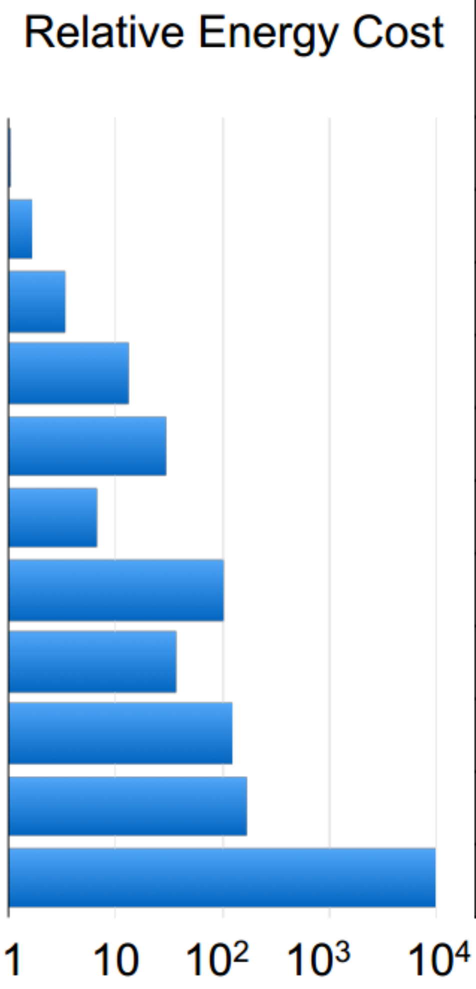


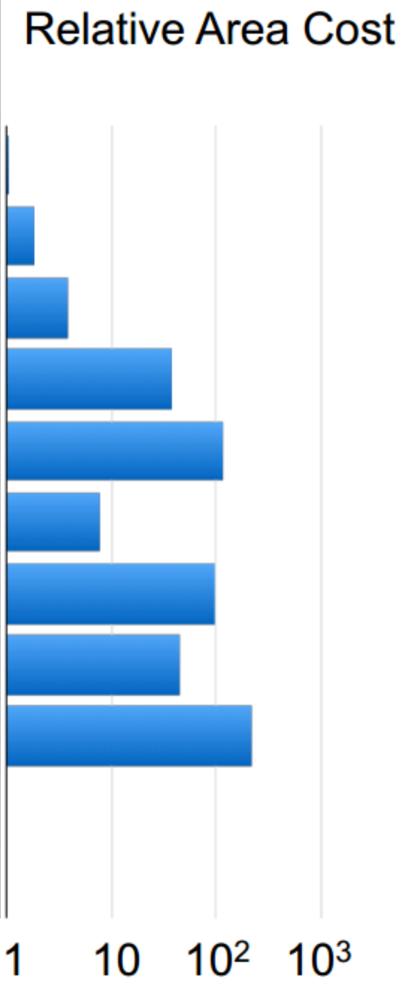
	Range	Accuracy
FP32	$10^{-38} - 10^{38}$	.000006%
FP16	$6x10^{-5}$ - $6x10^{4}$	.05%
Int32	$0 - 2x10^{9}$	½
Int16	$0 - 6x10^{4}$	½
Int8	0 – 127	½

Source: [Horowitz, “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014]

Energy Cost

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





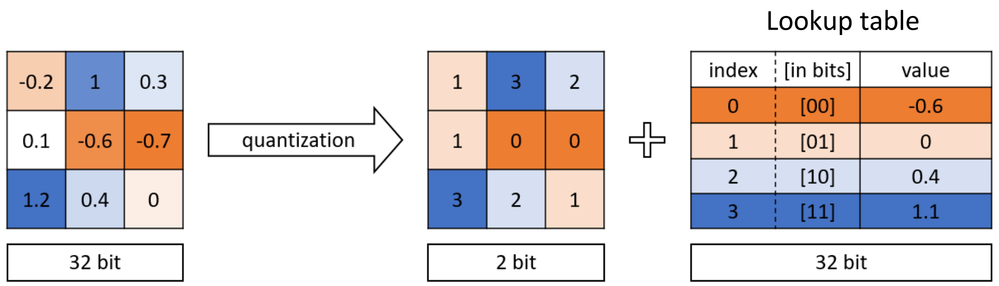
Source: [Horowitz, "Computing's Energy Problem (and what we can do about it)", ISSCC 2014]

Why Quantization?

- **1. Memory Efficiency**:
  - Reduce model and data size
  - Minimize storage requirements, especially for edge devices

Floating point   Integer

32.54 ⟶ 33



Source: https://medium.com/@kaustavtamuly/compressing-and-accelerating-high-dimensional-neural-networks-6b501983c0c8

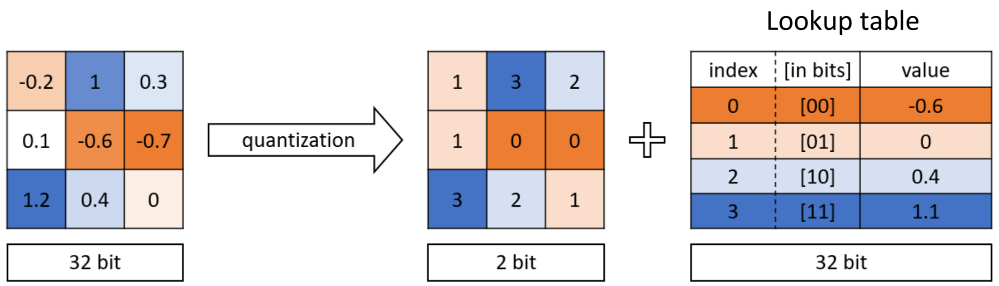
Quantization: Striking the balance between precision and efficiency.

Why Quantization?

- **1. Memory Efficiency**:
  - Reduce model and data size
  - Minimize storage requirements, especially for edge devices
- **2. Latency**:
  - Accelerate inference times

Floating point Integer

32.54 ⟶ 33



Source: https://medium.com/@kaustavtamuly/compressing-and-accelerating-high-dimensional-neural-networks-6b501983c0c8

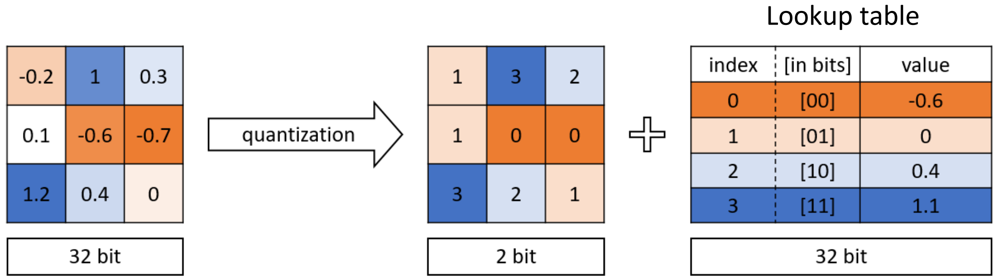
Quantization: Striking the balance between precision and efficiency.

Why Quantization?

- **1. Memory Efficiency**:
  - Reduce model and data size
  - Minimize storage requirements, especially for edge devices
- **2. Latency**:
  - Accelerate inference times
- **3. Energy Savings**:
  - Decrease power consumption, vital for battery-operated devices
  - Reduced heat generation

Floating point Integer

32.54 ⟶ 33



Source: https://medium.com/@kaustavtamuly/compressing-and-accelerating-high-dimensional-neural-networks-6b501983c0c8

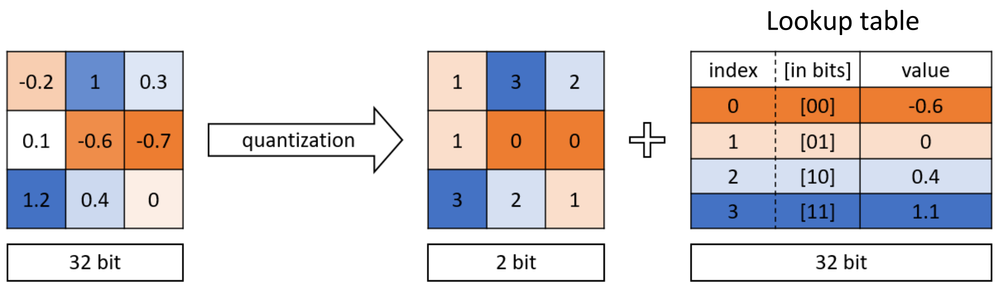
Quantization: Striking the balance between precision and efficiency.

Why Quantization?

- **1. Memory Efficiency**:
  - Reduce model and data size
  - Minimize storage requirements, especially for edge devices
- **2. Latency**:
  - Accelerate inference times
- **3. Energy Savings**:
  - Decrease power consumption, vital for battery-operated devices
  - Reduced heat generation
- **4. Hardware Compatibility**:
  - Adapt models for devices with limited precision capability
  - Enable deployment on specialized hardware accelerators

Floating point Integer

32.54 ⟶ 33



Source: https://medium.com/@kaustavtamuly/compressing-and-accelerating-high-dimensional-neural-networks-6b501983c0c8

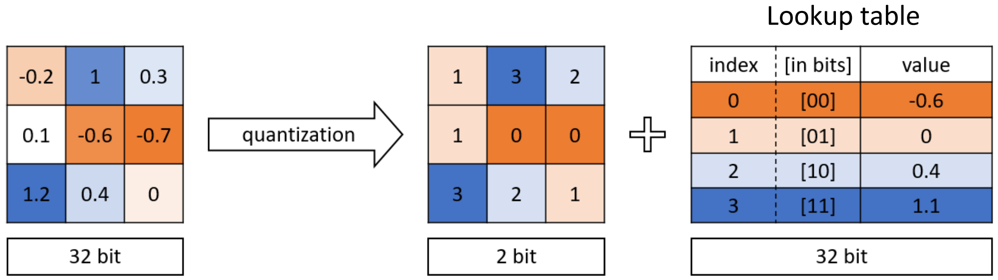
Quantization: Striking the balance between precision and efficiency.

Why Quantization?

- **1. Memory Efficiency**:
  - Reduce model and data size
  - Minimize storage requirements, especially for edge devices
- **2. Latency**:
  - Accelerate inference times
- **3. Energy Savings**:
  - Decrease power consumption, vital for battery-operated devices
  - Reduced heat generation
- **4. Hardware Compatibility**:
  - Adapt models for devices with limited precision capability
  - Enable deployment on specialized hardware accelerators
- **5. Bandwidth Conservation**:
  - Faster data transmission for cloud-edge architectures
  - Reduced data transfer costs

Floating point Integer

$32.54 \longrightarrow 33$



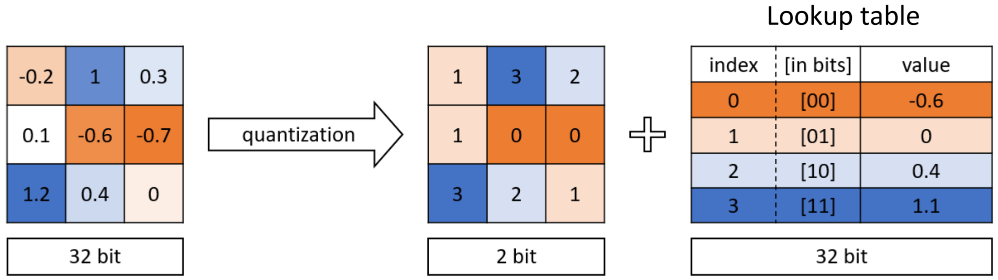
Source: https://medium.com/@kaustavtamuly/compressing-and-accelerating-high-dimensional-neural-networks-6b501983c0c8

Quantization: Striking the balance between precision and efficiency.

Why Quantization?

- **1. Memory Efficiency**:
  - Reduce model and data size
  - Minimize storage requirements, especially for edge devices
- **2. Latency**:
  - Accelerate inference times
- **3. Energy Savings**:
  - Decrease power consumption, vital for battery-operated devices
  - Reduced heat generation
- **4. Hardware Compatibility**:
  - Adapt models for devices with limited precision capability
  - Enable deployment on specialized hardware accelerators
- **5. Bandwidth Conservation**:
  - Faster data transmission for cloud-edge architectures
  - Reduced data transfer costs
- **6. Noise and Redundancy Reduction**:
  - Mitigate overfitting by discarding insignificant parameters
  - Improve model robustness

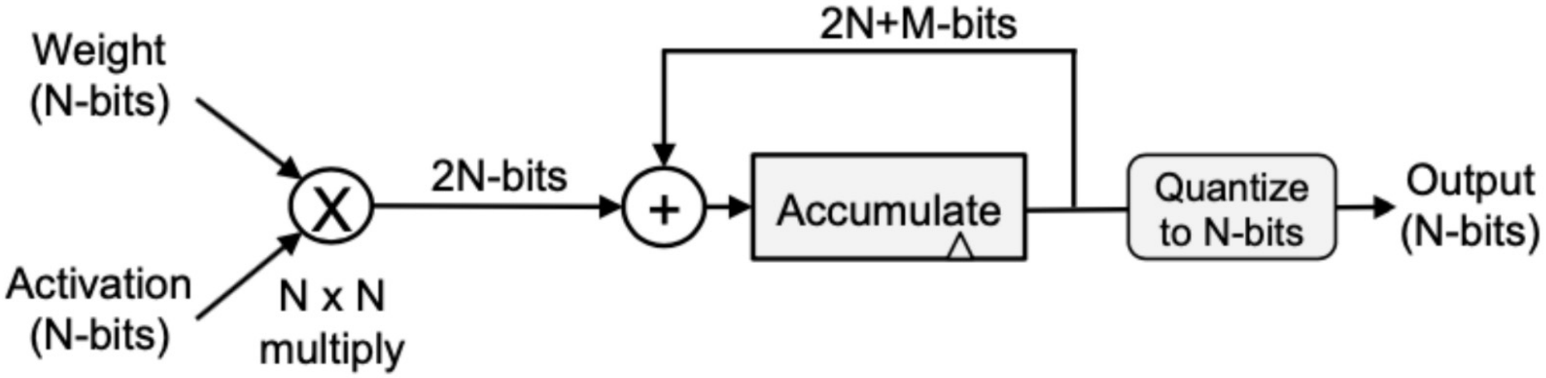
Floating point		Integer
32.54	⟶	33



Source: https://medium.com/@kaustavtamuly/compressing-and-accelerating-high-dimensional-neural-networks-6b501983c0c8

Quantization: Striking the balance between precision and efficiency.

Various Bit Widths Within a MAC



- **Internal precision is higher than inputs and outputs** to maintain accuracy.
- **Accumulation bit width (2N + M) is greater** than the multiplication bit width (2N) to prevent loss of precision.
- **Quantization reduces bit width** to fit within hardware constraints and improve efficiency.

What is Quantization?

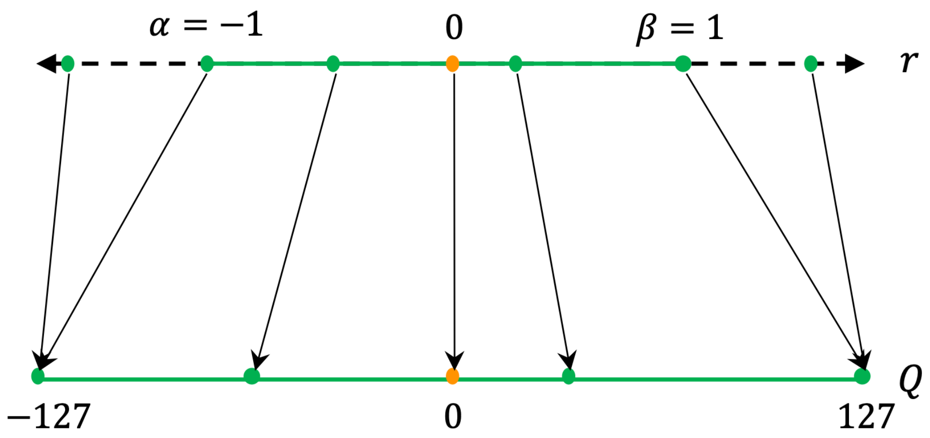
The process of mapping input values from a **large set (r)** to output values in a **smaller set (q)**.

$$r \rightarrow q$$

What is Quantization?

The process of mapping input values from a **large set (r)** to output values in a **smaller set (q)**.

$$r \rightarrow q$$



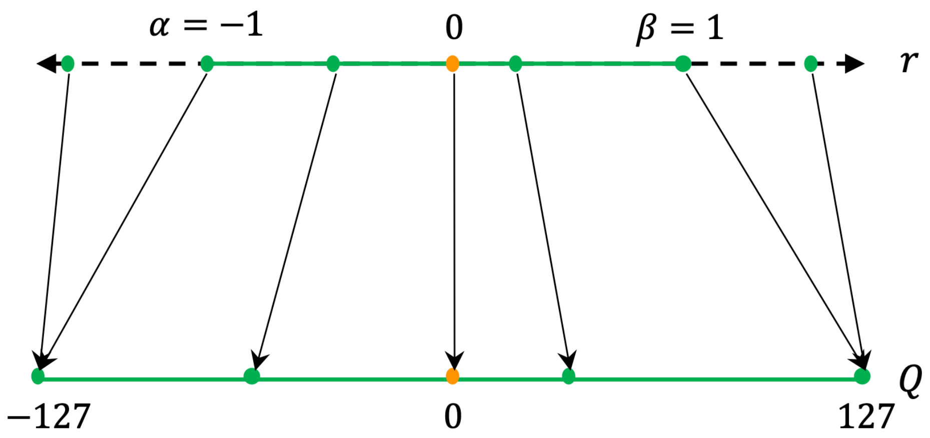
Mapping r to Q

Source: Gholami et al. A Survey of Quantization Methods for Efficient Neural Network Inference

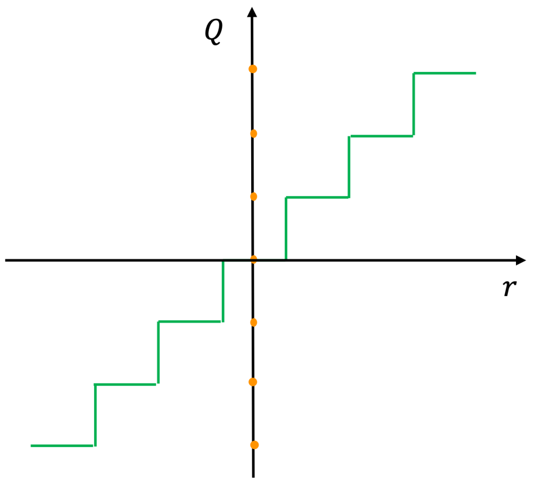
What is Quantization?

The process of mapping input values from a **large set (r)** to output values in a **smaller set (q)**.

$$r \rightarrow q$$



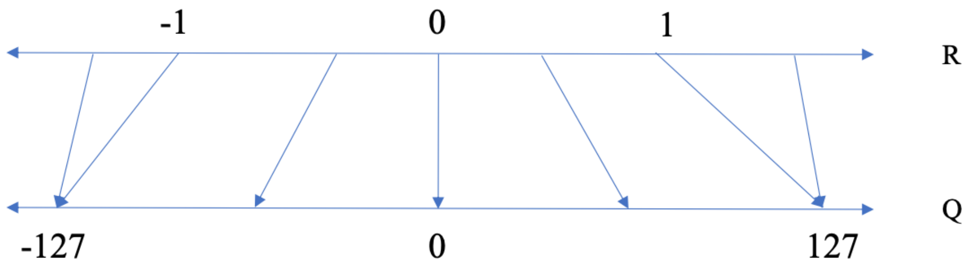
Mapping r to Q



Uniform Quantization

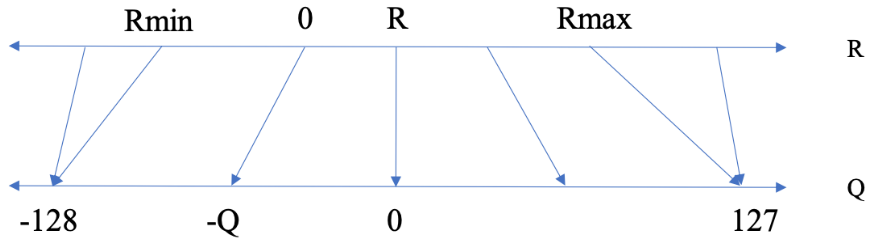
Source: Gholami et al. A Survey of Quantization Methods for Efficient Neural Network Inference

Symmetric vs. Asymmetric



(Symmetric)
sets the boundaries of parameter values to an equal range (from −1 to 1) and maps them over the range of [−127, 127]

Usually used for **weights**, Since weights can have both positive and negative values, having a symmetric range ensures that zero remains zero after quantization, which can simplify certain computations and storage



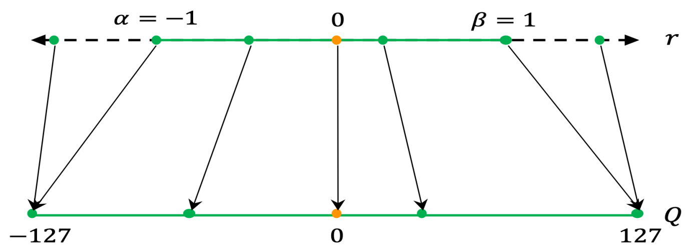
(Asymmetric)
sets the boundaries of parameter values to a range (from Rmin to Rmax) and maps them over the range of [−128, 127]

Usually used for **activations**

Source: [Garifulla et. al, "A Case Study of Quantizing Convolutional Neural Networks for Fast Disease Diagnosis on Portable Medical Devices", Sensors, 2022]

What is Quantization?

We are ignoring -128
to be symmetrical



The process of mapping input values from a **large set (r)** to output values in a **smaller set (q)**.

Scale factor

Zero point: **the quantized value** that corresponds to the real number 0

$$r = S(q - Z)$$

Real value (floating point)

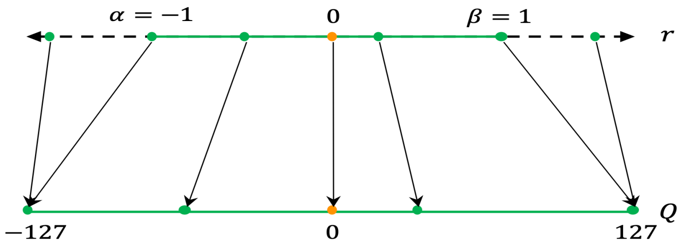
Quantized value (fixed point, b bits)

$$r \in [r_{min}, r_{max}]$$

$$q \in [0, 2^b - 1]$$ Unsigned

$$q \in [-2^{b-1}, 2^{b-1} - 1]$$ Signed

Formulation of Quantization



$$r = S(q - Z)$$

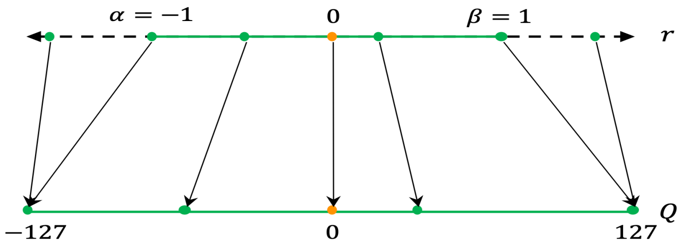
$$q \in [0, 2^b - 1]$$

$$r \in [r_{min}, r_{max}]$$

$$\Longrightarrow$$

$$S = \frac{r_{max} - r_{min}}{2^b - 1}$$

Formulation of Quantization



$$r = S(q - Z)$$

$$q \in [0, 2^b - 1]$$

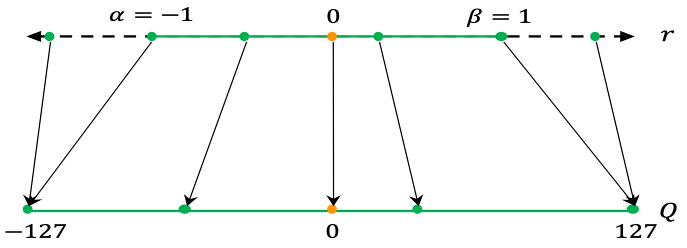
$$r \in [r_{min}, r_{max}]$$

$$S = \frac{r_{max} - r_{min}}{2^b - 1}$$

$$r = S(q - Z)$$

Quantization

Formulation of Quantization



$$r = S(q - Z)$$

$q \in [0, 2^b - 1]$

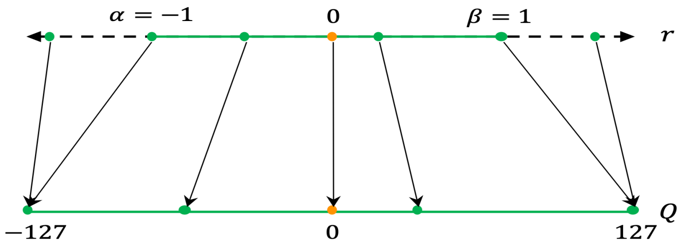
$r \in [r_{min}, r_{max}]$

$$S = \frac{r_{max} - r_{min}}{2^b - 1}$$

$$r = S(q - Z)$$

$$q = \left(\frac{r}{S}\right) + Z$$

Formulation of Quantization



$$r = S(q - Z)$$

$q \in [0, 2^b - 1]$

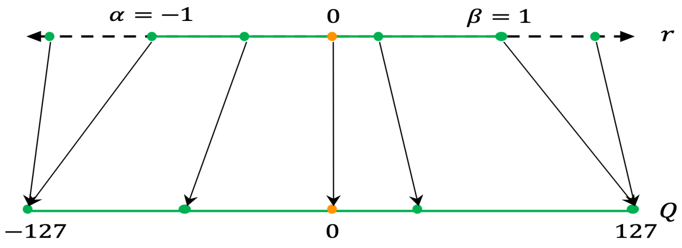
$r \in [r_{min}, r_{max}]$

$$S = \frac{r_{max} - r_{min}}{2^b - 1}$$

$$r = S(q - Z)$$

$$q = \mathrm{round}\left(\frac{r}{S}\right) + Z$$

Formulation of Quantization



$$r = S(q - Z)$$

$$q \in [0, 2^b - 1]$$
$$r \in [r_{min}, r_{max}]$$

$$S = \frac{r_{max} - r_{min}}{2^b - 1}$$

$$r = S(q - Z)$$

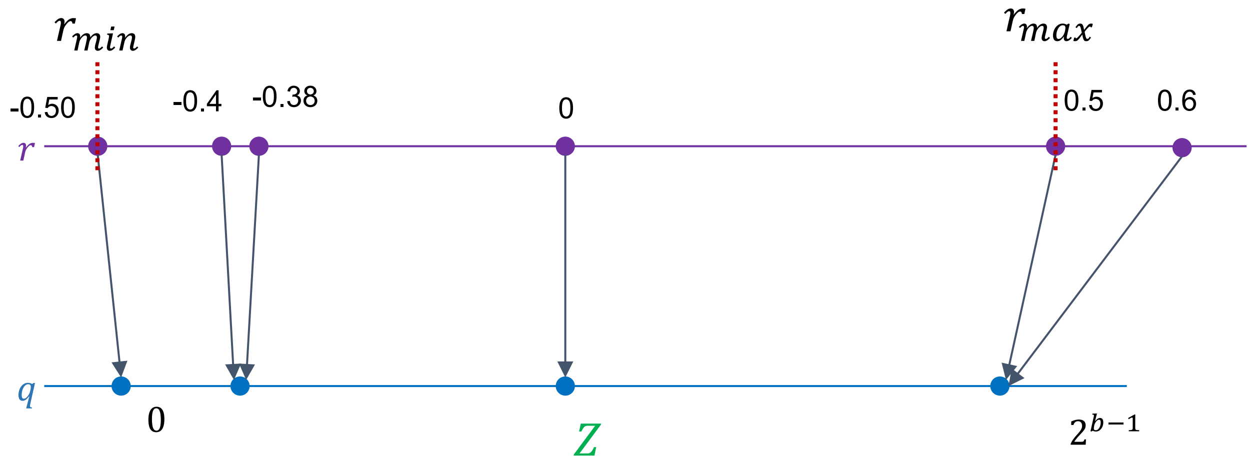
$$q = \text{clip}\left(\text{round}\left(\frac{r}{S}\right) + Z, 0, 2^b - 1\right)$$

Formulation of Quantization

$$q \in [0, 2^b - 1] \quad \Rightarrow \quad S = \frac{r_{max} - r_{min}}{2^b - 1}$$
$$r \in [r_{min}, r_{max}]$$

$$r = S(q - Z)$$

$$q = \text{clip}\left(\text{round}\left(\frac{r}{S}\right) + Z, 0, 2^b - 1\right)$$



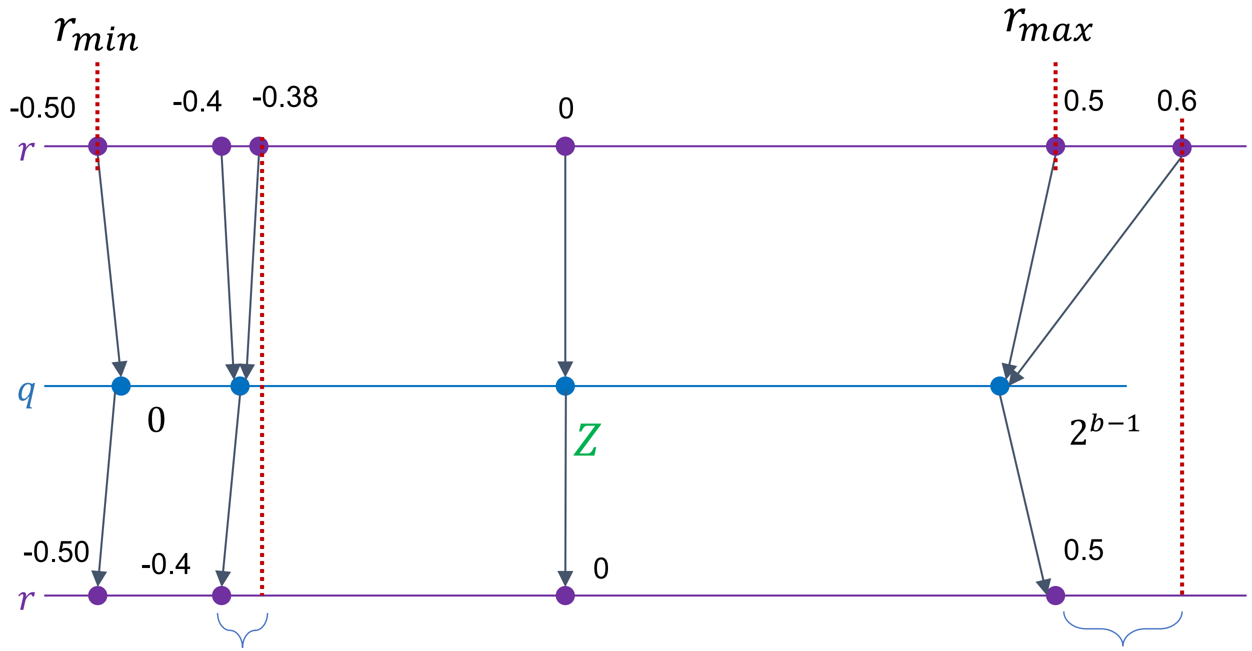
Formulation of Quantization

$$q \in [0, 2^b - 1]$$
$$r \in [r_{min}, r_{max}]$$

$$\Longrightarrow \quad S = \frac{r_{max} - r_{min}}{2^b - 1}$$

$$r = S(q - Z)$$

$$q = \text{clip}\left(\text{round}\left(\frac{r}{S}\right) + Z, 0, 2^b - 1\right)$$



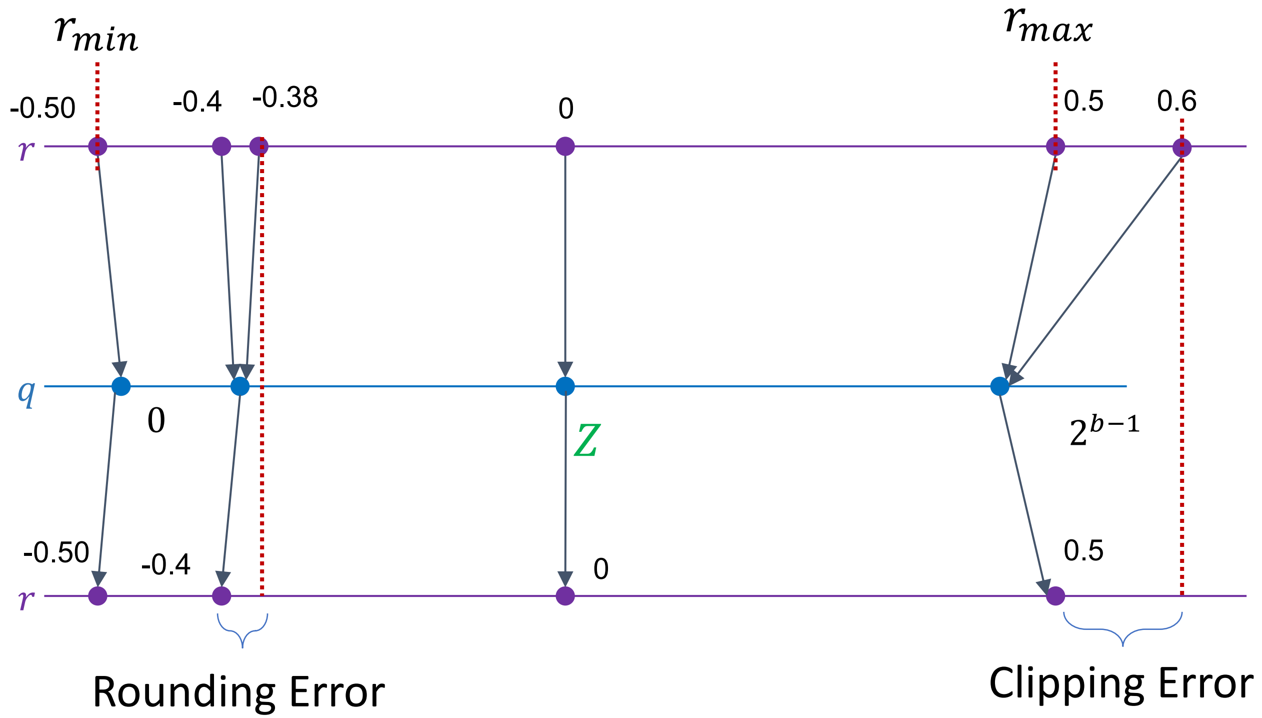
Formulation of Quantization

$$q \in [0, 2^b - 1]$$
$$r \in [r_{min}, r_{max}]$$

$$S = \frac{r_{max} - r_{min}}{2^b - 1}$$

$$r = S(q - Z)$$

$$q = \text{clip}\left(\text{round}\left(\frac{r}{S}\right) + Z, 0, 2^b - 1\right)$$



Quantizing Matrix Multiplication

$$r = S(q - Z)$$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

Quantizing Matrix Multiplication

$$r = S(q - Z)$$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

Quantizing Matrix Multiplication

$$r = S(q - Z)$$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)(x_{kj}^q - Z_x)$$

Quantizing Matrix Multiplication

$$r = S(q - Z)$$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)(x_{kj}^q - Z_x)$$

If all conversions are symmetrical

$$y_{ij}^q = \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^q \cdot x_{kj}^q$$

Quantizing Matrix Multiplication

$$r = S(q - Z)$$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \qquad y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)(x_{kj}^q - Z_x)$$

If all conversions are symmetrical

$$y_{ij}^q = \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^q \cdot x_{kj}^q$$

Integer matrix multiplication 😀

Quantizing Matrix Multiplication

$r = S(q - Z)$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)\,(x_{kj}^q - Z_x)$$

If conversions are not symmetrical

Quantizing Matrix Multiplication

$$r = S(q - Z)$$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)\,(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^q \cdot x_{kj}^q - W_{ik}^q \cdot Z_x - x_{kj}^q \cdot Z_W + Z_W Z_x$$

If conversions are not symmetrical

Quantizing Matrix Multiplication

$r = S(q - Z)$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^q \cdot x_{kj}^q - W_{ik}^q \cdot Z_x - x_{kj}^q \cdot Z_W + Z_W Z_x$$

If conversions are not symmetrical

Constant! 😀

Quantizing Matrix Multiplication

$r = S(q - Z)$

(Quantize each matrix separately)

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$

$$y_{ij} = \sum_{k=1}^{N} W_{ik} x_{kj}$$

$$S_y(y_{ij}^q - Z_y) = \sum_{k=1}^{N} S_W(W_{ik}^q - Z_W) S_x(x_{kj}^q - Z_x)$$

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} (W_{ik}^q - Z_W)(x_{kj}^q - Z_x)$$

Integer matrix multiplication 😀

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^q \cdot x_{kj}^q - W_{ik}^q \cdot Z_x - x_{kj}^q \cdot Z_W + Z_W Z_x$$

If conversions are not symmetrical

Constant! 😀

Method 1: Using Power of 2 Approximation

$$M := \frac{S_1 S_2}{S_3}.$$

Round to the nearest power of 2

Method 1: Using Power of 2 Approximation

$$M := \frac{S_1 S_2}{S_3}.$$

Round to the nearest power of 2

Example:

Suppose $M = 0.01$

Method 1: Using Power of 2 Approximation

$$M := \frac{S_1 S_2}{S_3}.$$

Round to the nearest power of 2

Example:

Suppose $M = 0.01 \approx \frac{1}{128} = 2^{-7}$

Now as an example, let's multiply number 1234 by M

Method 1: Using Power of 2 Approximation

$$M := \frac{S_1 S_2}{S_3}.$$

Round to the nearest power of 2

Example:

Suppose $M = 0.01 \approx \frac{1}{128} = 2^{-7}$

Now as an example, let's multiply number 1234 by M

$\mathbf{1234}$/128 = 9.64

This can be done by shifting 😀

Method 2: Using Fixed Point Arithmetic

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 **is in the interval** $[0.5, 1)$

This can be done using fixed point arithmetic

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$\Rightarrow$

$$M = 2^{-n} M_0$$

M_0 **is in the interval** $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

Suppose $M = 0.01$

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 is in the interval $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

Suppose $M = 0.01$

$$M = 2^{-6} \times 0.64$$

$$M_0 = 0.64$$

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 is in the interval $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

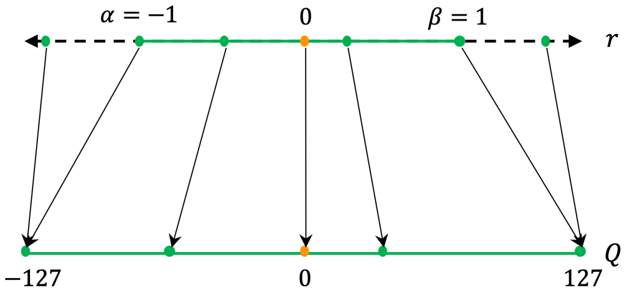
Suppose $M = 0.01$

$M = 2^{-6} \times 0.64$

$M_0 = 0.64$

$M_0(Scaled) = 0.64{\times}2^{31} = 1374389534$

We scale M_0 to fit inside a 32-bit integer.



Mapping r to Q

This allows it to be stored and used as an **integer**, enabling **fast integer multiplication** instead of floating-point operations.

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 **is in the interval** $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

Suppose $M = 0.01$

$$M = 2^{-6} \times 0.64$$

$$M_0 = 0.64$$

$M_0(Scaled) = 0.64 \times 2^{31} = 1374389534$

We scale M_0 to fit inside a 32-bit integer (1 bit is for sign).

Now as an example, let's multiply number **1234** by M

This allows it to be stored and used as an **integer**, enabling **fast integer multiplication** instead of floating-point operations.

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 is in the interval $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

Suppose $M = 0.01$

$M = 2^{-6} \times 0.64$

$M_0 = 0.64$

$M_0(Scaled) = 0.64 \times 2^{31} = 1374389534$

We scale M_0 to fit inside a 32-bit integer (1 bit is for sign).

Now as an example, let's multiply number **1234** by M

$\mathbf{1234} \times 1374389534 = 1695996684956$

Multiply by M_0

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 is in the interval $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

Suppose $M = 0.01$

$$M = 2^{-6} \times 0.64$$

$$M_0 = 0.64$$

$$M_0(Scaled) = 0.64{\times}2^{31} = 1374389534$$

We scale M_0 to fit inside a 32-bit integer (1 bit is for sign).

Now as an example, let's multiply number **1234** by M

1234×1374389534 = 1695996684956 — Multiply by M_0

$1695996684956{\times}2^{-6} = 26499948202$ — Shift the result by 6 bits

Method 2: Using Fixed Point Arithmetic for scaling factor

$$M := \frac{S_1 S_2}{S_3}.$$

$$M = 2^{-n} M_0$$

M_0 **is in the interval** $[0.5, 1)$

This can be done using fixed point arithmetic

Example:

Suppose $M = 0.01$

$$M = 2^{-6} \times 0.64$$

$$M_0 = 0.64$$

$$M_0(Scaled) = 0.64 \times 2^{31} = 1374389534$$

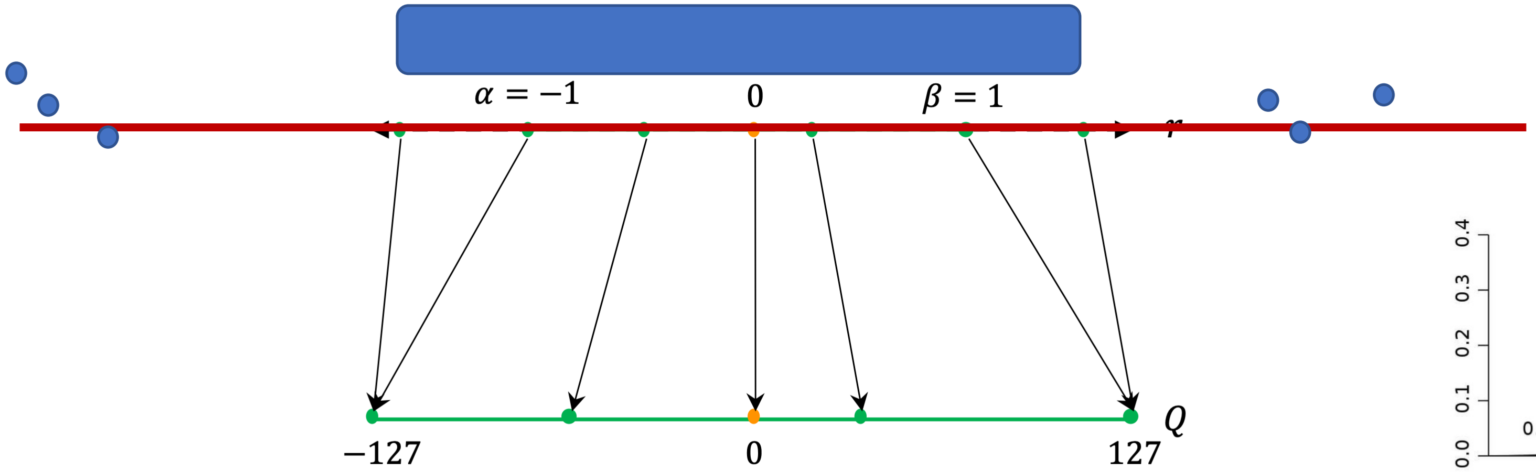
We scale M_0 to fit inside a 32-bit integer (1 bit is for sign).

Now as an example, let's multiply number **1234** by M

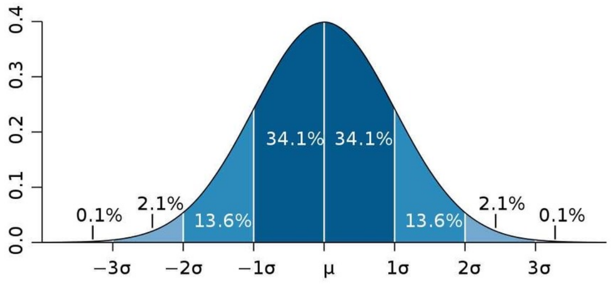
1234×1374389534 = 1695996684956 — Multiply by M_0

$1695996684956 \times 2^{-6} = 26499948202$ — Shift the result by 6 bits

$26499948202/2^{31} =$ **12.3399999333173** — Scale it back by shifting 31 bits



Mapping r to Q



Finding Scaling Ranges and Zero Points

$$y_{ij}^{q} = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^{q} \cdot x_{kj}^{q} - W_{ik}^{q} \cdot Z_x - x_{kj}^{q} \cdot Z_W + Z_W Z_x$$

Static vs Dynamic Quantization

Static

- Weights and activations are statically quantized
  - **before** deployment
    - **Use Case**: Edge devices (e.g., mobile inference with TensorFlow Lite or ONNX Runtime).

Dynamic

- Weights are statically quantized
- Quantization parameters for activations are found
  - **after** deployment (during inference)
  - **Use Case**: CPU-based inference (e.g., PyTorch dynamic quantization for LSTMs, Transformers).

Weight only

- Only weights are statically quantized.
- **Use Case**: Large-scale deep learning models where activation quantization significantly affects accuracy.

$$y_{ij}^q = Z_y + \frac{S_W S_x}{S_y} \sum_{k=1}^{N} W_{ik}^q \cdot x_{kj}^q - W_{ik}^q \cdot Z_x - x_{kj}^q \cdot Z_W + Z_W Z_x$$

Static Quantization

- Definition
  - **Scales** and **zero points** for activation tensors are **pre-computed**.
  - Less **memory usage**
  - Eliminates **overhead** of computing scales and zero points on-the-fly.
- Method:
  - Run the floating-point neural network with **representative unlabeled data**.
  - Collect distribution statistics for all activation layers.
  - Use statistics to compute scales and zero points.
- Inference
  - All computations use integer ops for highest performance.
  - Requires representative unlabeled data.

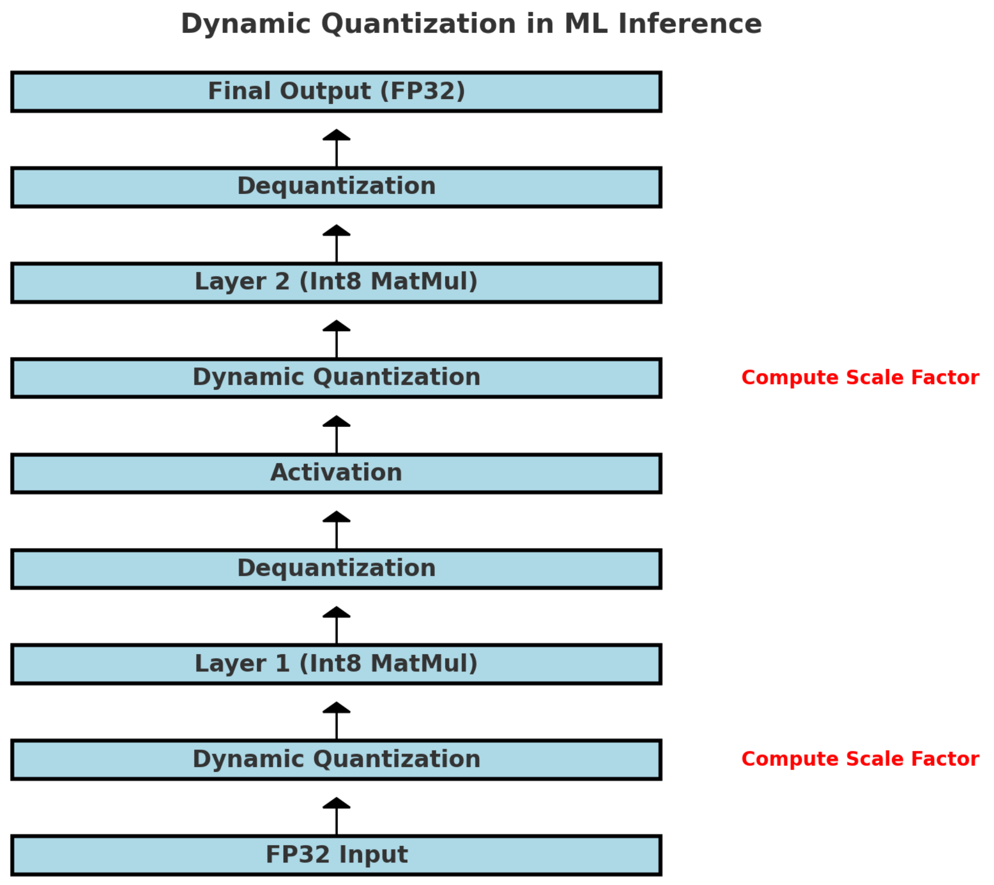
Dynamic Quantization

- **Weights** are **pre-quantized** to integers before inference.
- **Scales** and **zero** points for **output/activation** are found **dynamically**.
  - Calculated for each activation map during runtime.
  - Activation ranges are determined **on the fly** at runtime.
  - The model receives activations in full precision, then quantizes them dynamically
- **How?**
  - Collect Activation Statistics (During Inference)
  - When an **input batch** is processed during inference, we track:
    - **The minimum activation value**: x_{min}
    - **The maximum activation value**: x_{max}
- **2. Compute the Scale Factor and Zero**

$$S_X = \frac{x_{\max} - x_{\min}}{2^b - 1} \qquad Z_X = \text{round}\left(-\frac{x_{\min}}{S_X}\right)$$

Dynamic Quantization

- After performing integer matrix multiplications, the outputs are dequantized back to floating-point format before applying activation functions.
- The scaling factors used for dequantization are derived from the quantization process.



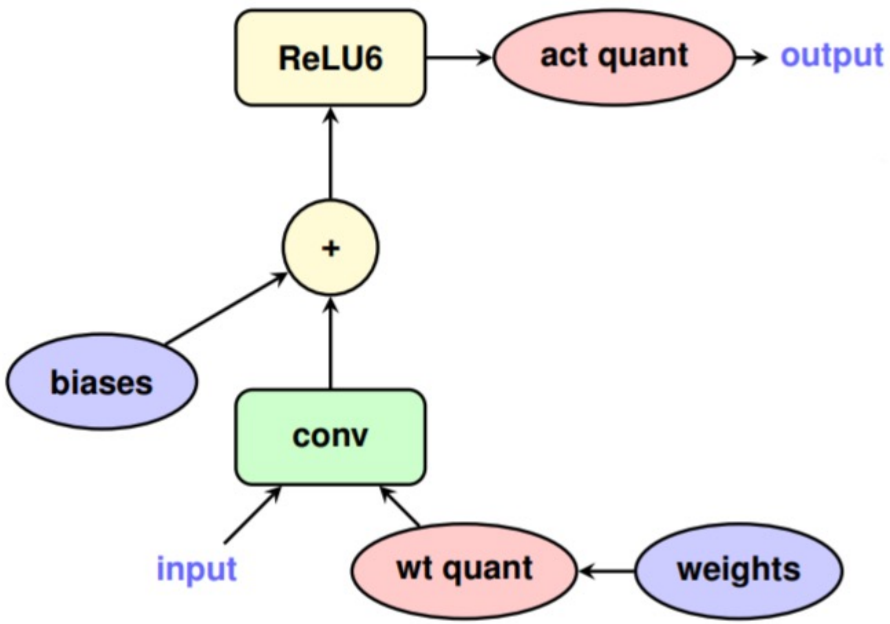
$$Y_{fp} = \text{scale}_{\text{input}} \times \text{scale}_{\text{weight}} \times Y_q$$

Quantization Aware Training

- Standard quantization methods, such as:
  - **Post-Training Quantization (PTQ)** (where a trained model is quantized after training)
  - **Dynamic Quantization** (which quantizes weights but applies activation quantization at runtime)
- Both suffer from **accuracy degradation**, especially in models that rely on fine numerical precision (e.g., CNNs, Transformers).
  - **Rounding errors** from converting floating-point weights/activations to lower precision (e.g., INT8).
  - **Non-uniform distributions** of activations can cause larger quantization errors.

Quantization Aware Training

- **Definition**
  - **Simulates the effects** of quantization **during the training** process.
  - Makes the model robust to quantization effects, minimizing accuracy loss.

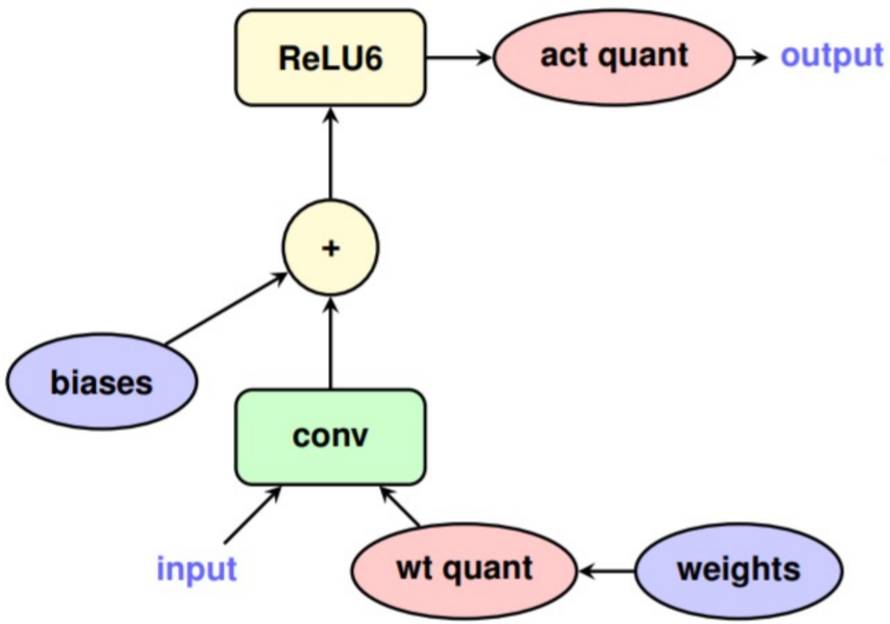


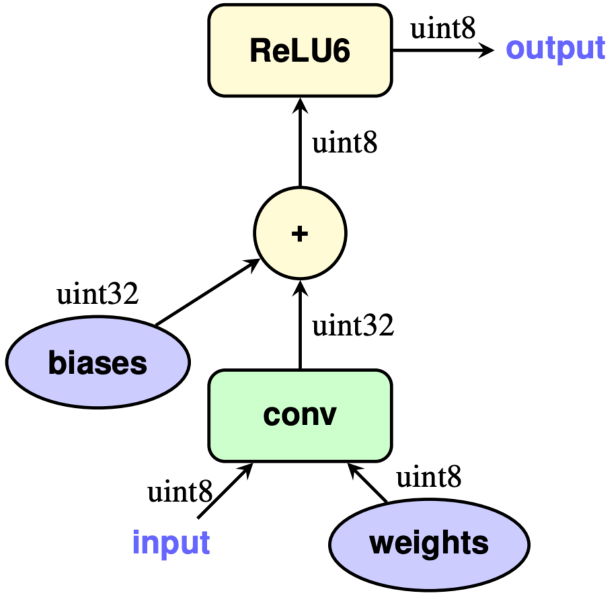
The ‘**wt quant**’ and ‘**act quant**’ ops introduce losses in the forward pass of the model to simulate actual quantization loss during inference.

Source: https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html

Quantization Aware Training

- **Definition**
  - **Simulates the effects** of quantization **during the training** process.
  - Makes the model robust to quantization effects, minimizing accuracy loss.





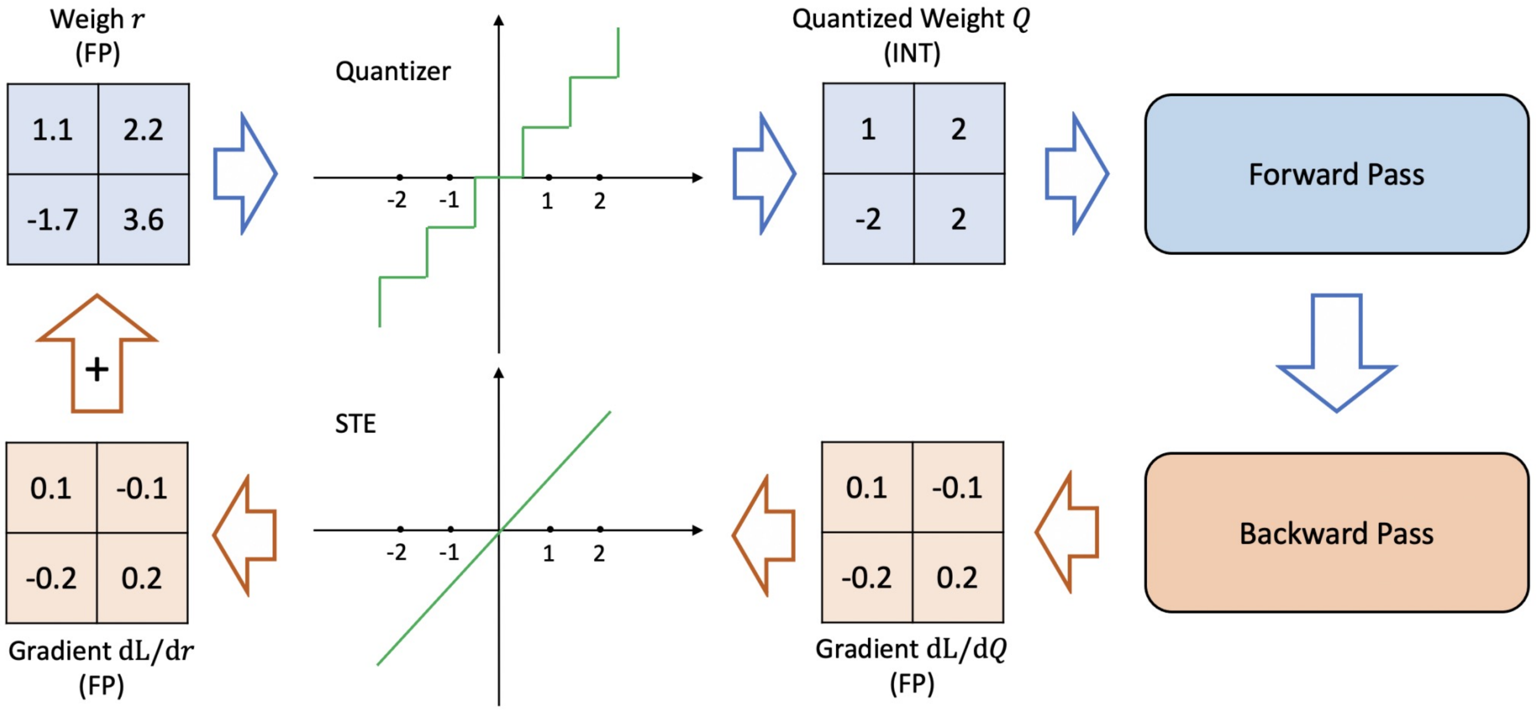
(a) Integer-arithmetic-only inference

The ‘**wt quant**’ and ‘**act quant**’ ops introduce losses in the forward pass of the model to simulate actual quantization loss during inference.

Source: https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html

Quantization Aware Training

- Instead of training in **full precision (FP32) and then quantizing**, QAT **simulates quantization (INT8) during training**.
- How is it done?
  - **Start with a Pretrained Model (FP32)**
    - A floating-point model is first trained normally.
  - **Introduce Fake Quantization (Simulating INT8)**
    - During forward propagation, **fake quantization layers** simulate the rounding and scaling effects of quantization (without actually reducing precision).
    - Weights and activations are **clamped** and **quantized to INT8**, but stored as FP32 to allow gradient updates.
  - **Backward Pass and Gradient Updates (In FP32)**
    - Despite simulating INT8, gradients remain in FP32.
    - The model adjusts itself to be **more robust to quantization errors**.
  - **Deploy the Final INT8 Model**
    - After training, the model is converted into a **true INT8 model**, ready for optimized inference on edge devices, CPUs, or TPUs.



Quantization-Aware Training procedure, with Straight Through Estimator

- **Forward Pass:** The network works with **quantized values**.
- **Backward Pass:** The network behaves **as if no quantization exists**, allowing gradients to update normally.
- **Effect:** The model **learns** how to adjust its floating-point weights so that after quantization, they still produce good results.

Source: Gholami et al. A Survey of Quantization Methods for Efficient Neural Network Inference

Straight-Through Estimator (STE)

- Quantization involves a **step function**, which is **non-differentiable** (it has zero gradient almost everywhere and is undefined at step points).
  - Gradient descent requires gradients, this poses a challenge.
  - STE provides an approximation by treating the quantization function as the **identity function** in the backward pass, allowing gradient-based optimization to continue.
- Suppose a floating-point weight ***r*** is quantized into a discrete value ***Q*** using a function such as:

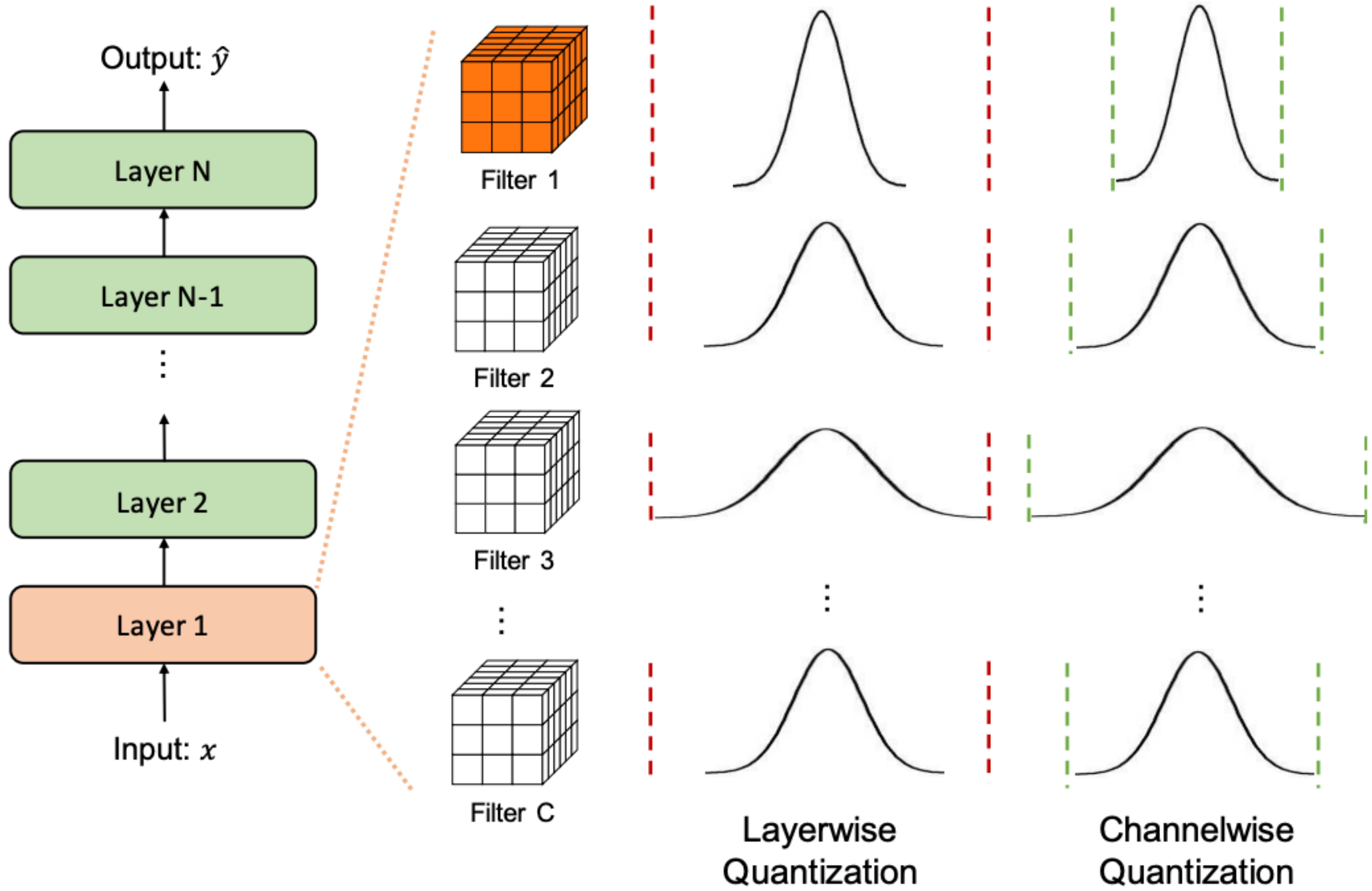
$$Q = \mathrm{round}(r)$$

The gradient of the loss function is:

zero!

$$\frac{dL}{dr} = \frac{dL}{dQ} \cdot \frac{dQ}{dr}$$

STE Approximate $\frac{dQ}{dr} \approx 1$ $\longrightarrow$ $\frac{dL}{dr} \approx \frac{dL}{dQ}$

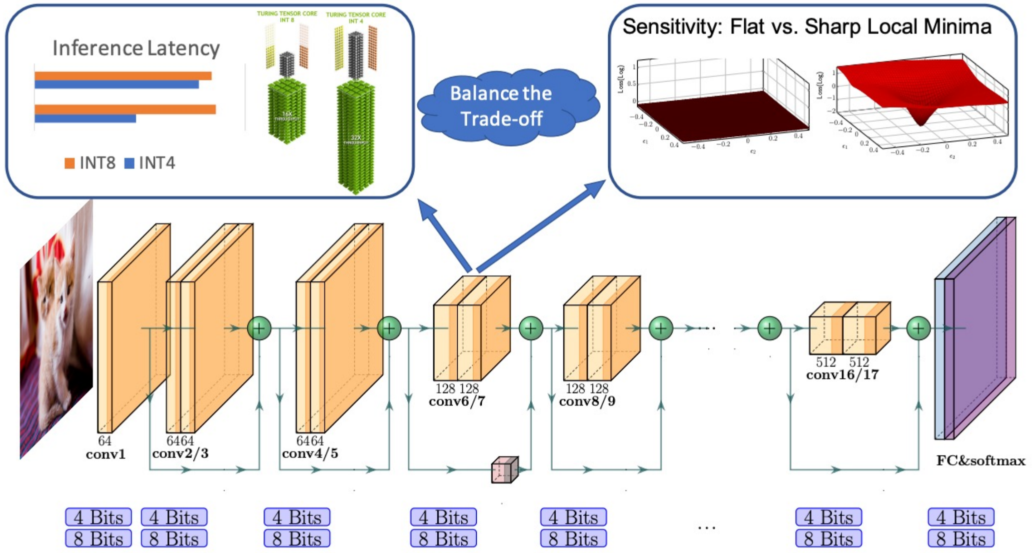


- Layer wise Quantization
  - The same clipping range is applied to all the filters in the same layer.
  - Bad quantization resolution for the channels that have narrow distributions (e.g., Filter 1).
- Channel wise Quantization
  - Dedicates different clipping ranges to different channels

Source: Gholami et al. A Survey of Quantization Methods for Efficient Neural Network Inference

Mixed-precision quantization

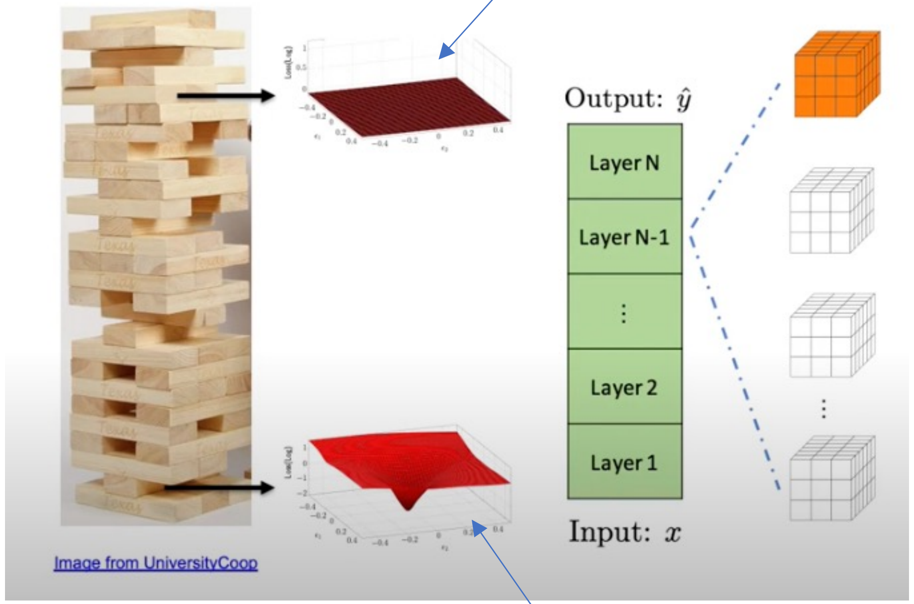
Some layers have a **flat loss surface**, meaning they are **robust to quantization** and can be quantized to lower precision without significantly affecting accuracy.



Each layer can be quantized to either 4-bit or 8-bit precision

Not sensitive

Less sensitive layers (top) can tolerate lower precision.



Sensitive layers (at the bottom) require careful handling (higher precision) to avoid collapsing accuracy.

Sensitive

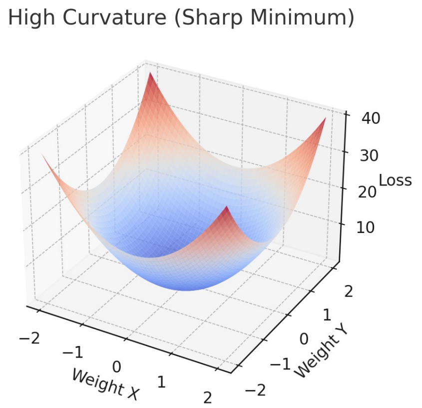
- Sensitive and efficient layers in higher precision and only apply low-precision quantization to insensitive and inefficient layers. The efficiency metric is hardware dependent, and it could be latency or energy consumption.
- For each layer we have two choices. So, this is an exponential search.

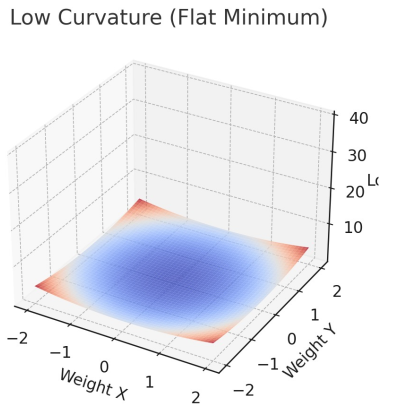
Source: Gholami et al. A Survey of Quantization Methods for Efficient Neural Network Inference

How Do We Determine Which Layers Are Sensitive?

1.Loss Landscape Analysis

1. The **sensitivity of a layer** is related to how weight perturbations affect the loss function.
2. If the loss function has **sharp minima**, **Layer is sensitive**.
3. If the loss function has **flat minima**, **Layer is not sensitive**.





2.Quantization Sensitivity Testing

1. Apply different quantization bit-widths (e.g., 4-bit, 8-bit) to each layer **one at a time**.
2. Observe the accuracy drop when using lower precision.
3. Layers that cause a **large drop in accuracy** when quantized are **more sensitive** and should use higher precision.

3.Gradient-based Sensitivity Measurement

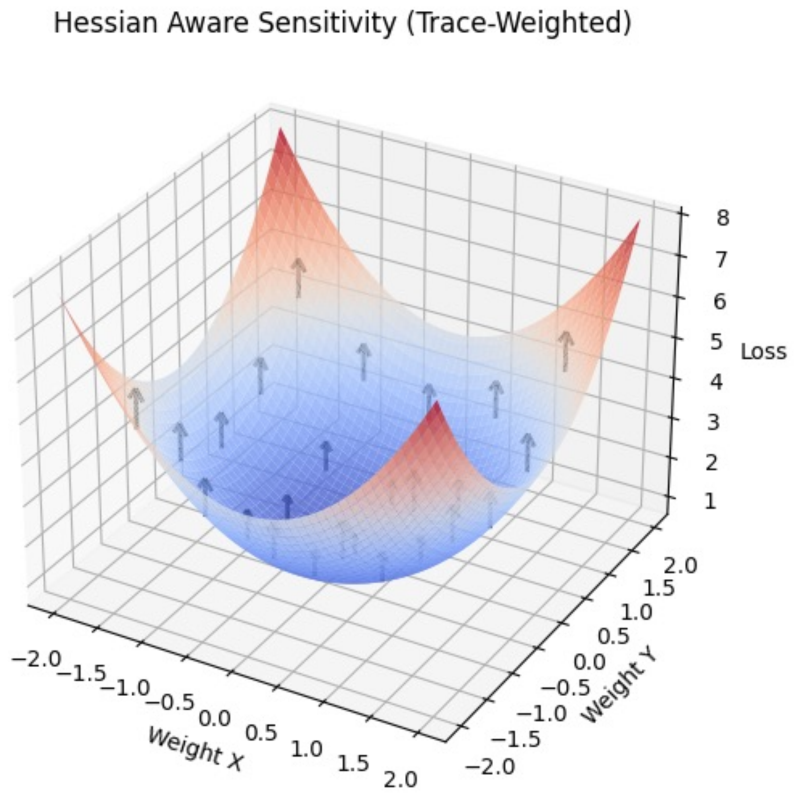
1. Some techniques use the **gradient magnitude** or **Hessian eigenvalues** to estimate sensitivity.
2. High curvature (high second derivative) indicates a **sensitive layer**.

4.Hardware and Latency Constraints

1. Some layers may need higher precision due to hardware efficiency constraints (e.g., certain hardware accelerators perform better with 8-bit over 4-bit.

Hessian-Aware Trace-Weighted Quantization

- The **3D surface plot** represents a **loss function** with respect to **weight parameters (X, Y)**.
- The **curvature (sharpness) of the loss surface** is linked to the Hessian matrix.
- **Arrows represent Hessian-based sensitivity**:
  - **Larger arrows** indicate **higher curvature**, meaning the layer is **more sensitive** to quantization.
  - **Smaller arrows** indicate a **flatter loss**, meaning the layer is **less sensitive** and can be quantized more aggressively.



- $f''(x) > 0$ → Function is convex (curving **upwards**).
- $f''(x) < 0$ → Function is concave (curving **downwards**).

Hessian Aware trace-Weighted Quantization

Results: ResNet50

(b) *ResNet50*

Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
Baseline	✓	✓	77.72	W32A32	97.8	3951	77.72
Integer Only (Jacob et al., 2018)	✓	✓	76.40	W8A8	24.5	247	74.90
RVQuant (Park et al., 2018)	✗	✗	75.92	W8A8	24.5	247	75.67
HAWQV3	✓	✓	77.72	W8A8	24.5	247	**77.58**
PACT (Choi et al., 2018)	✗	✓	76.90	W5A5	16.0	101	76.70
LQ-Nets (Zhang et al., 2018)	✗	✗	76.50	W4A32	13.1	486	76.40
RVQuant (Park et al., 2018)	✗	✗	75.92	W5A5	16.0	101	75.60
HAQ (Wang et al., 2019)	✗	✗	76.15	WMPA32	9.62	520	75.48
OneBitwidth (Chin et al., 2020)	✗	✓	76.70	W1*A8	12.3	494	76.70
HAWQV3	✓	✓	77.72	W4/8A4/8	18.7	154	75.39
HAWQV3+DIST	✓	✓	77.72	W4/8A4/8	18.7	154	**76.73**

8 bit

Mixed precision

- **BOPS (Bit Operations Per Second)** (computational cost)
- **Top-1 Accuracy (%)** (classification performance)

Source: HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks Zhen Dong et al.

Example Results

Source: [Garifulla et. al, “A Case Study of Quantizing Convolutional Neural Networks for Fast Disease Diagnosis on Portable Medical Devices”, Sensors, 2022]

$f(x) = min(max(0, x), 6)$

Act.	type	accuracy		recall 5	
		mean	std. dev.	mean	std.dev.
ReLU6	floats	78.4%	0.1%	94.1%	0.1%
	8 bits	75.4%	0.1%	92.5%	0.1%
	7 bits	75.0%	0.3%	92.4%	0.2%
ReLU	floats	78.3%	0.1%	94.2%	0.1%
	8 bits	74.2%	0.2%	92.2%	0.1%
	7 bits	73.7%	0.3%	92.0%	0.1%

Table 4.3: Inception v3 on ImageNet: Accuracy and recall 5 comparison of floating point and quantized models.

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, Google, 2017

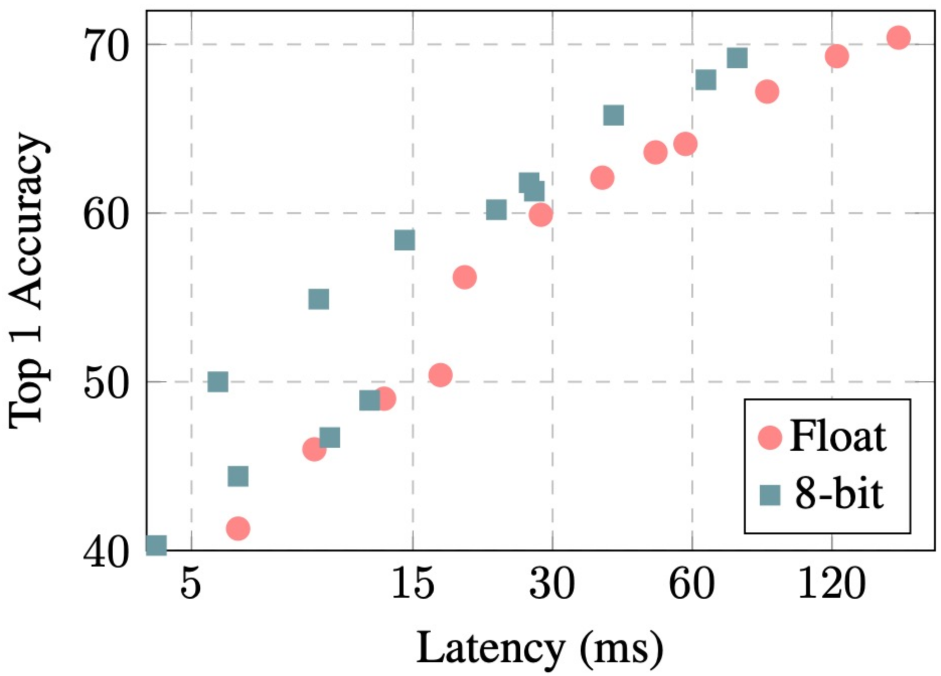
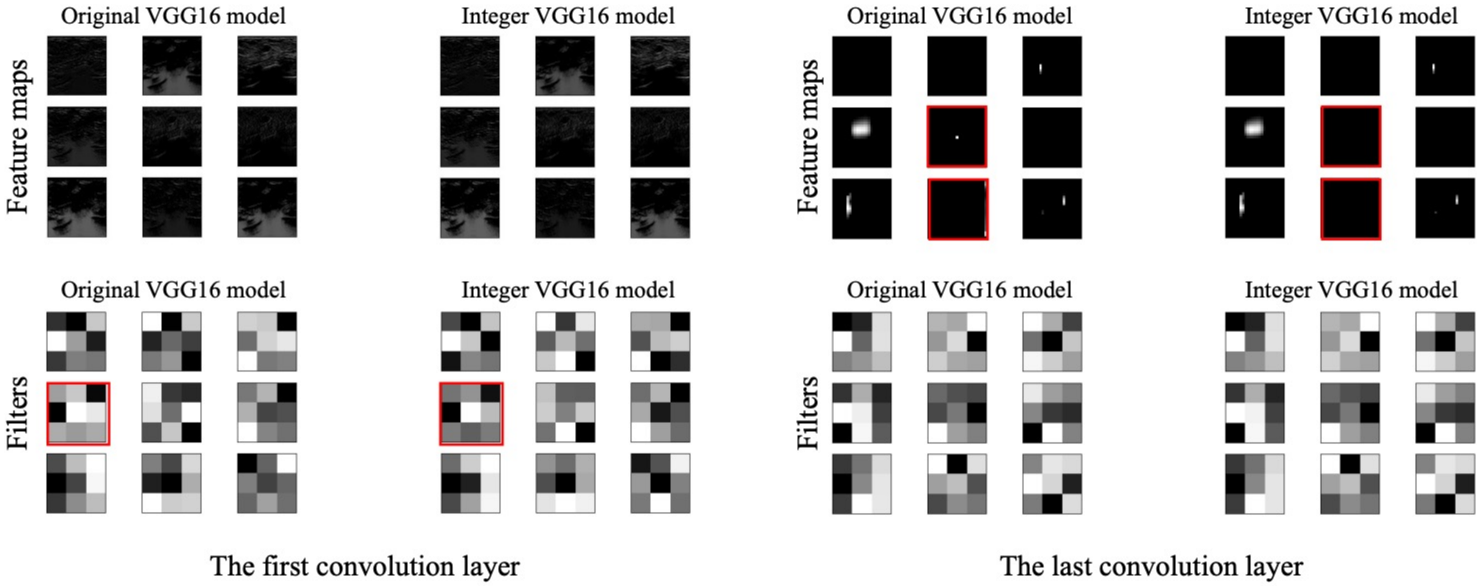


Figure 4.1: ImageNet classifier on Qualcomm Snapdragon 835 big cores: Latency-vs-accuracy tradeoff of floating-point and integer-only MobileNets.

Floating-point models generally have higher accuracy but come with higher latency. **8-bit models have lower latency** but can suffer from an accuracy drop due to quantization.

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, Google, 2017

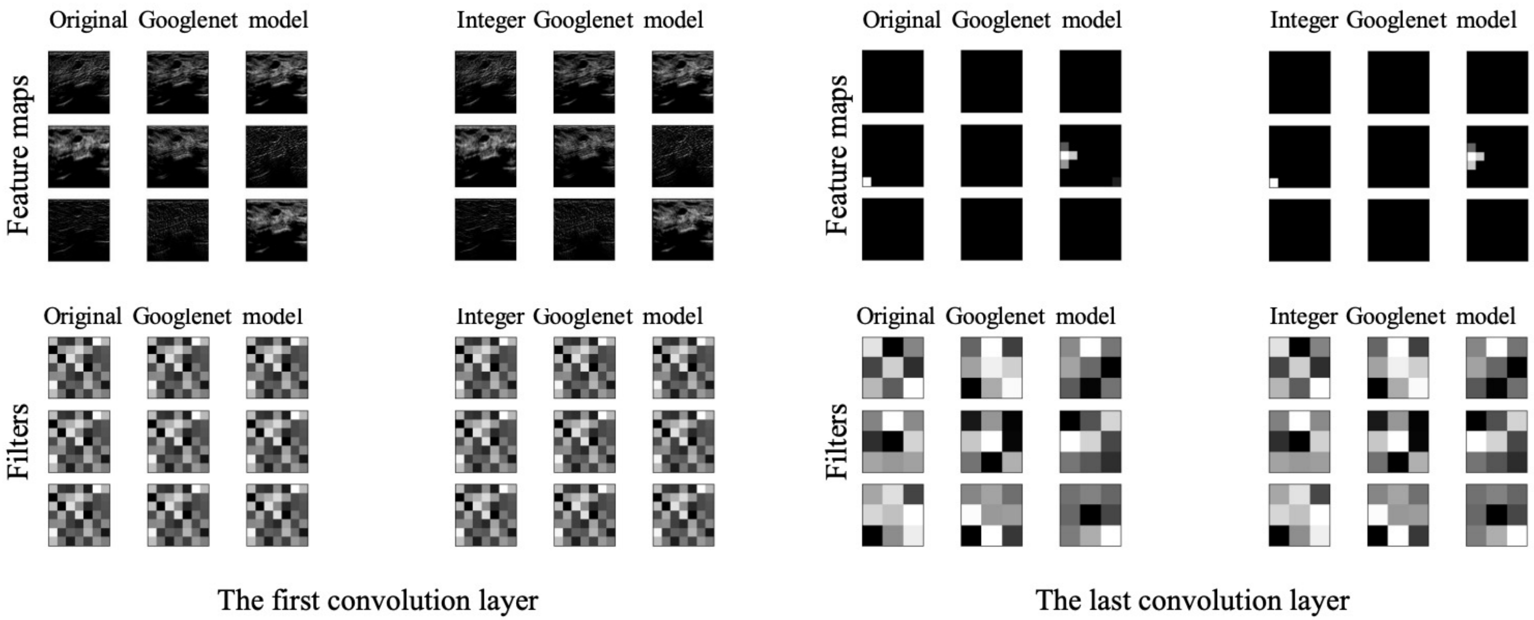
- Feature maps: No significant differences between the feature maps of the original model and those of the quantized model
- Filters: some filters in the quantized model have slightly different weight values from the original model



VGG16

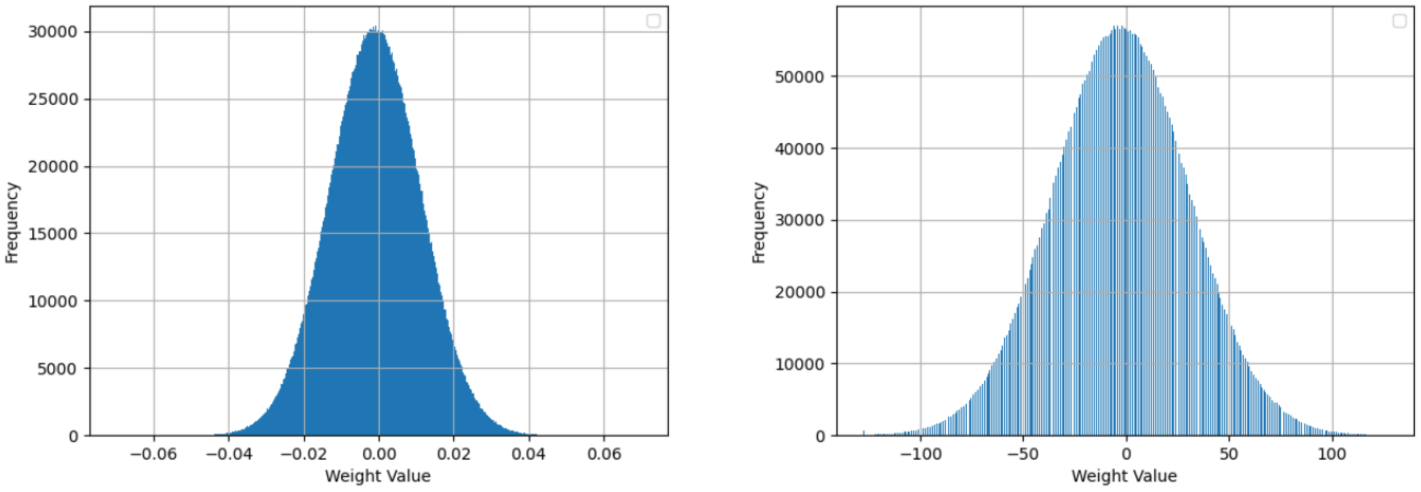
Source: [Garifulla et. al, "A Case Study of Quantizing Convolutional Neural Networks for Fast Disease Diagnosis on Portable Medical Devices", Sensors, 2022]

- Feature maps: No significant differences between the feature maps of the original model and those of the quantized model
- Filters: some filters in the quantized model have slightly different weight values from the original model

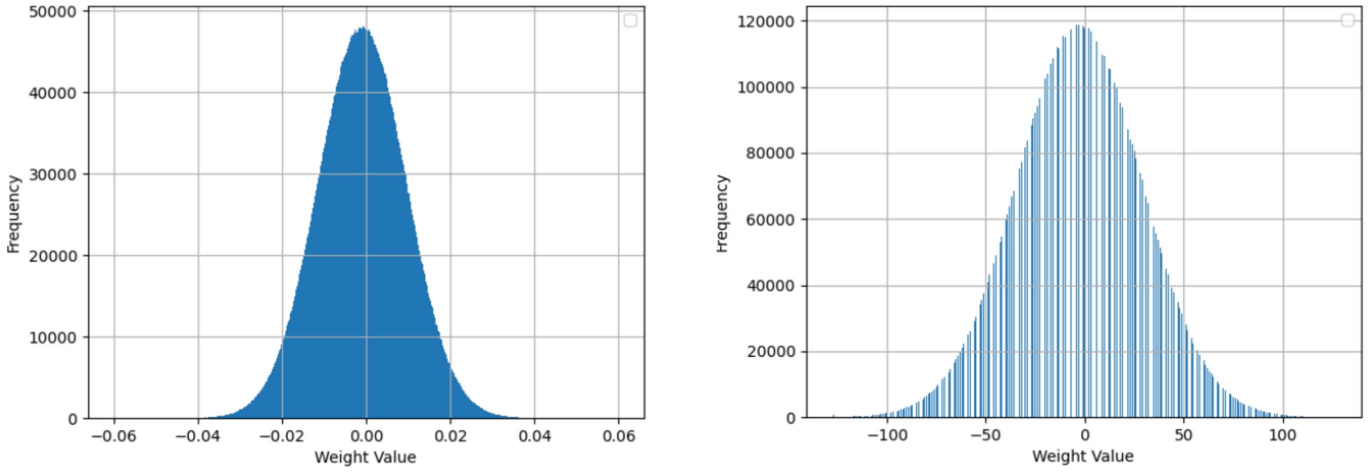


GoogLeNet

Source: [Garifulla et. al, "A Case Study of Quantizing Convolutional Neural Networks for Fast Disease Diagnosis on Portable Medical Devices", Sensors, 2022]



(**a**) VGG16



(**b**) GoogLeNet

The frequency of the weight values in the full integer model increases due to the reassignment of float-precision values onto a limited range of integer values

Source: [Garifulla et. al, "A Case Study of Quantizing Convolutional Neural Networks for Fast Disease Diagnosis on Portable Medical Devices", Sensors, 2022]