



Lecture 05: Quantization Strategies for Efficient DNN Implementation

Notes

- Please send email to efficientaiaccelerator@gmail.com
- Lab1 has been released.
- Start considering the project topic, teaming.
- In-course quiz today, covering materials of DNN pruning.

Recap

- Why pruning?
 - Reduce running cost
 - Reduce storage
- General pruning techniques
- Transformer pruning

Topics

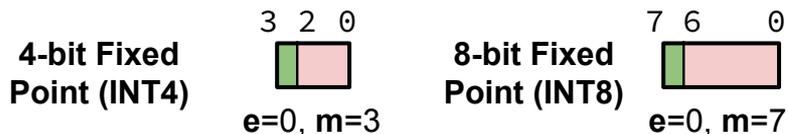
- Basic Data Formats
 - Fixed point (INT)
 - Floating point (FP)
 - Block floating point (BFP)
- Quantization methods
 - Taxonomy of Quantization
 - Learnable adaptive quantization scheme

Topics

- **Basic Data Formats**
 - Fixed point (INT)
 - Floating point (FP)
 - Block floating point (BFP)
- **Quantization methods**
 - Taxonomy of Quantization
 - Learnable adaptive quantization scheme

Fixed-Point Arithmetic (INT)

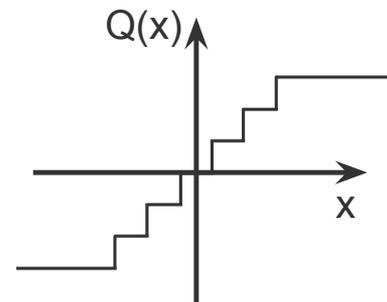
Fixed Point Formats



- Hyperparameter associated with the fixed-point format:
 - Clipping range $(-L, L)$: usually symmetrical around 0
 - Bitwidth (b)
- Quantization with Fixed-point format is called **Fixed point quantization** or **INT quantization**.

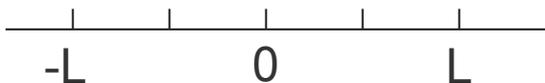
Fixed-Point Format (Symmetrical)

- How to convert a number x to INT representation?
 - Set the clipping range: $(-L, L)$, bitwidth: b
 - Compute the scale: $s = 2L / (2^b - 2)$
 - Clip the input x : $x_c = \text{Clip}(x, L, -L)$
 - Calculate the INT representation: $x_{int} = \text{round}(x_c / s)$
 - Rescale: $x_q = Sx_{int}$
- Have a uniform representation power within the clipping range.
- All the computations can be performed using x_{int}



Fixed-Point Format (Symmetrical)

- Have a uniform representation power within the clipping range.
- All the computations can be performed using x_{int}

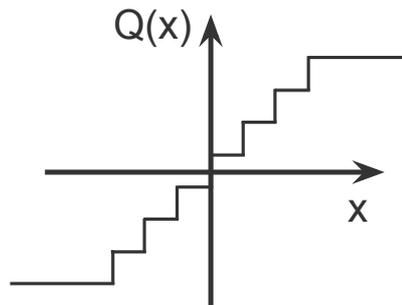
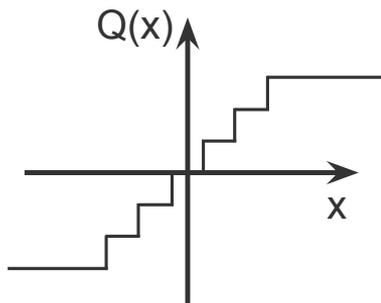


$$s=2L/(2^b-2)$$



$$s=2L/(2^b-1)$$

- With $s=2L/(2^b-2)$, zero can be represented using quantized number



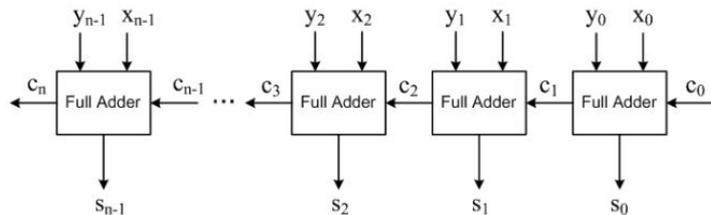
Example

- $X = [1.1, 2.4, -0.3, 0.8]$, bitwidth = 3, $L = 2$
- How to convert a number x to INT representation?
 - Set the clipping range: $(-L, L)$, bitwidth: b $b=3, L=2$
 - Compute the scale: $s = 2L / (2^b - 2)$ $s = 4/6 = 2/3$
 - Clip the input x : $x_c = \text{Clip}(x, L, -L)$ $x_c = [1.1, 2, -0.3, 0.8]$
 - Calculate the INT representation: $x_{int} = \text{round}(x_c / s)$ $x_{int} = [2, 3, 0, 1]$
 - Rescale: $x_q = s x_{int}$ $X_q = [1.33, 2.0, 0.0, 0.67]$

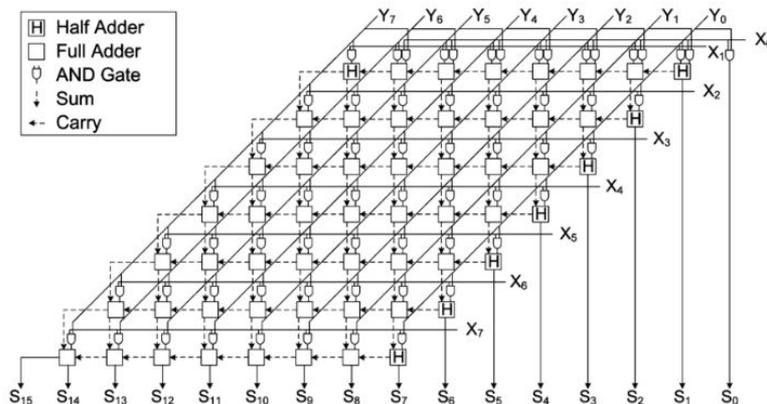
Computation with Fixed-Point Format

- Addition/Subtraction: $x_q \pm y_q = s(x_{int} \pm y_{int})$
- Multiplication: $x_q \times y_q = s^2(x_{int} \times y_{int})$
- Division: $x_q/y_q = x_{int}/y_{int}$

If the scales are the same



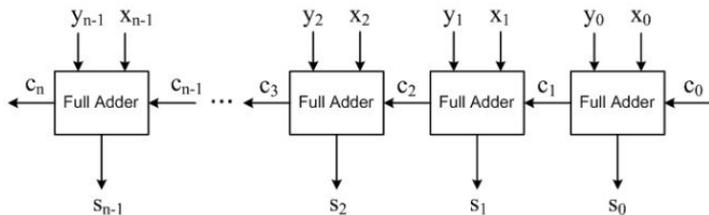
Fixed-point adder



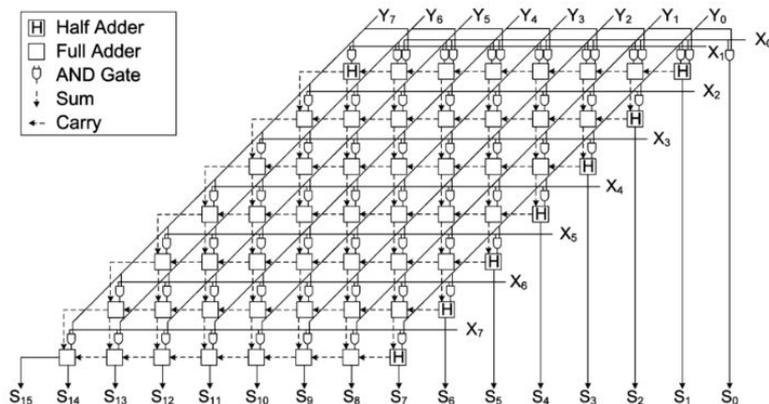
Fixed-point multiplier

Computation with Fixed-Point Format

- Addition/Subtraction: Hard to compute
 - Multiplication: $x_q \times y_q = s_x s_y (x_{int} \times y_{int})$
 - Division: $x_q / y_q = (s_x / s_y) \times (x_{int} / y_{int})$
- If the scales are not the same



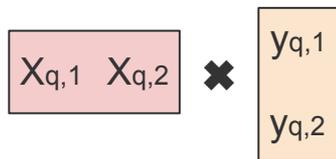
Fixed-point adder



Fixed-point multiplier

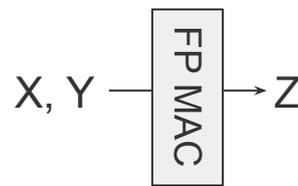
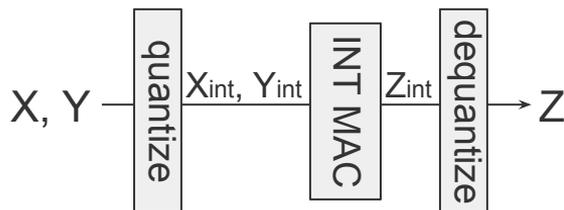
Computation with Fixed-Point Format

- If we try to compute the dot product between X and Y:



All elements within the tensors are quantized using the same scale, but the scale across the tensors can be different

$$x_{q,1} \times y_{q,1} + x_{q,2} \times y_{q,2} = s_x s_y (x_{int,1} \times y_{int,1} + x_{int,2} \times y_{int,2})$$



Computation with Fixed-Point Format

- INT can be applied to a block of numbers, with the block size defined in a customizable manner.

1	2
10	11

1.1	2.4
10.5	11.8

Per tensor
quantization

1.1	2.4
10.5	11.8

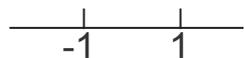
Row-wise
quantization
(low error)

1.1	2.4
10.5	11.8

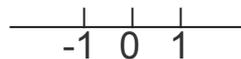
Column-wise
quantization
(high error)

- However, a higher quantization granularity will also incur a conversion overhead.

Computation with Fixed-Point Format



Binary quantization

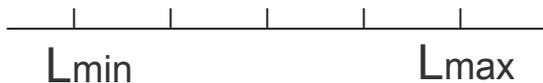


Ternary quantization

- Binary and Ternary neural networks are both multiplication-free DNN.

Fixed Point Format (Unsymmetrical)

- How to convert a number to INT8 representation?
 - Set the clipping range: (L_{min}, L_{max}) , bitwidth: b
 - Compute the scale: $s = (L_{max} - L_{min}) / (2^b - 1)$
 - Clip the input x : $x_c = Clip(x, L_{min}, L_{max})$
 - Calculate the fixed-point representation:
$$x_{int} = round((x_c - L_{min}) / s)$$
 - Rescale: $x_q = sx_{int} + L_{min}$



Example

- $X = [1.1, 2.4, -0.3, 0.8]$, bitwidth = 3, $L = 2$
- How to convert a number to INT8 representation?
 - Set the clipping range: (L_{min}, L_{max}) , bitwidth: b $b=3, L_{max}=2, L_{min}=-0.5$
 - Compute the scale: $s = (L_{max} - L_{min}) / (2^b - 1)$ $s = 0.357$
 - Clip the input x : $x_c = Clip(x, L_{min}, L_{max})$ $X_c = [1.1, 2, -0.3, 0.8]$
 - Calculate the fixed-point representation:
 $x_{int} = round((x_c - L_{min}) / s)$ $X_{int} = [4, 7, 1, 4]$
 - Rescale: $x_q = sx_{int} + L_{min}$ $X_q = [0.93, 2.0, -0.14, 0.93]$

Computation with Fixed-Point Format

- Addition/Subtraction:

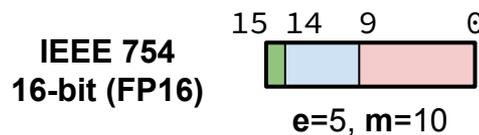
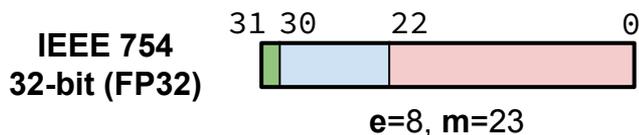
$$x_q + y_q = s(x_{int} + y_{int}) + 2L_{min} \quad x_q - y_q = s(x_{int} - y_{int})$$

- Multiplication (needs additional computation):

$$x_q \times y_q = s_x s_y (x_{int} \times y_{int}) + L_{min,x} y_q s_y + L_{min,y} x_q s_x + L_{min,x} L_{min,y}$$

- Division: hard to implement

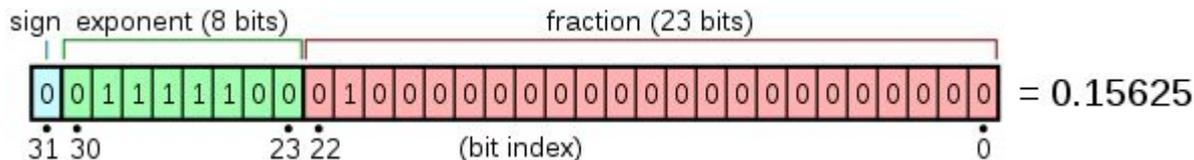
Floating-Point Arithmetic



■ Sign field ■ Exponent (e) ■ Mantissa (m)

- The floating-point number has three fields:
 - Sign (s)
 - Exponent (e)
 - Mantissa (m)

Floating-Point Arithmetic



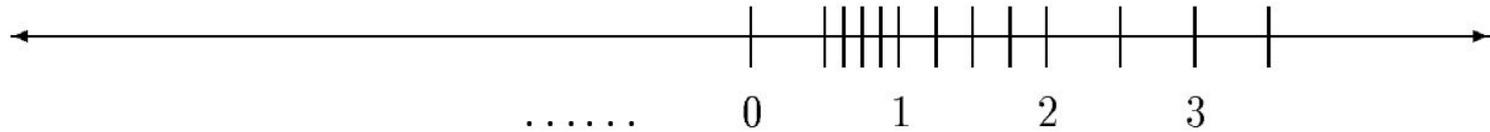
- IEEE-754 standard:

$$x = (-1)^s \times 2^{e-bias} \times (1 + m)$$

$$m = (0.b_0b_1b_2\dots b_{22})_2$$

- The exponent field is unsigned.
- We need some special representation:
 - A bit stream of all zeros represents 0

Floating Point Arithmetic



- Have better representation power for values with small magnitudes.
- How to convert a real number x to FP representation?

$$x = |x| \quad s = \text{sign}(x)$$

$$a = \lfloor \log_2 x \rfloor \quad e = a + \text{bias} \quad m = \frac{x}{2^a} - 1$$

Example

$x = -13.24$, bias=127

$x = |x|$ $s = \text{sign}(x)$

$$a = \lfloor \log_2 x \rfloor \quad e = a + \text{bias} \quad m = \frac{x}{2^a} - 1$$

$a = 3$, $e = 130$, $m = 0.655$

$s = (1)_2$, $e = (10000010)_2$, $m = (10100111101011100001000)_2$

Difference in Representation Power Between INT and FP

- FP provides relative precision that scales with magnitude. Small numbers near zero have finer granularity, while very large numbers have coarser steps.

- For FP, the as the magnitude getting larger, the granularity will also decrease.

- $x = (-1)^s \times 2^{e-bias} \times (1 + m)$

Under the fixed exponent, mantissa fills the gap evenly

As exponent increase, the granularity will decrease exponentially

- INT provides uniform precision. Each step between representable values is exactly the same.

Computation with FP Representation

- Addition/Subtraction:

- Need to align the exponent

$$\begin{array}{ccccccc}
 011010 & + & 001111 & = & 011010 & + & 011011 & = & 011101 \\
 \underbrace{\quad\quad\quad} & & \underbrace{\quad\quad\quad} & & \underbrace{\quad\quad\quad} & & \underbrace{\quad\quad\quad} & & \\
 s_1 e_1 m_1 & & s_2 e_2 m_2 & & & & & &
 \end{array}$$

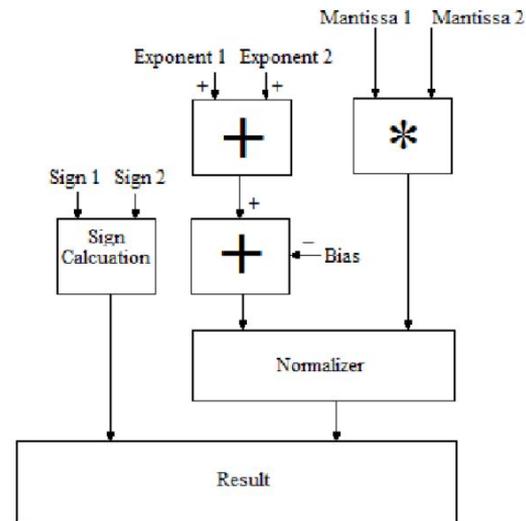
Alignment

- Multiplication/Subtraction:

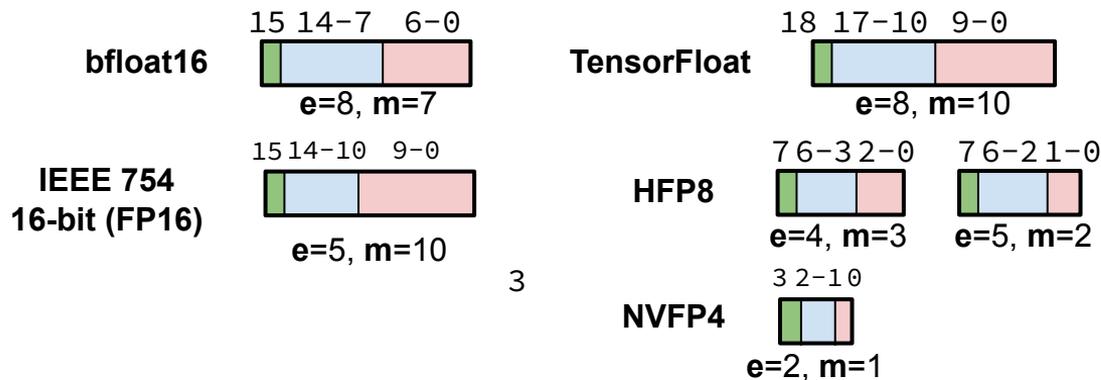
- Sum the exponent, multiply the mantissa

$$\begin{array}{ccccccc}
 011010 & \times & 001111 & & e = e_1 + e_2 \\
 \underbrace{\quad\quad\quad} & & \underbrace{\quad\quad\quad} & & 1 + m = \text{normalizer}(1.m_1 \times \\
 s_1 e_1 m_1 & & s_2 e_2 m_2 & & 1.m_2)
 \end{array}$$

- Addition and subtraction is expensive for FP.

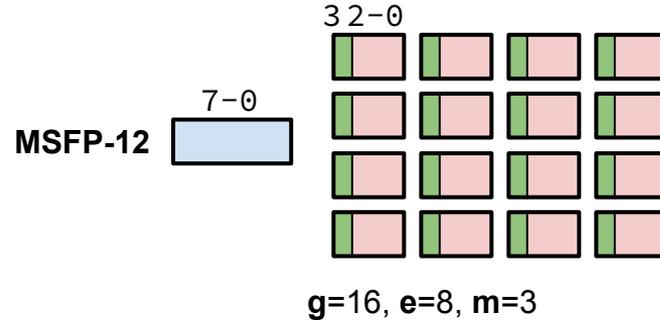
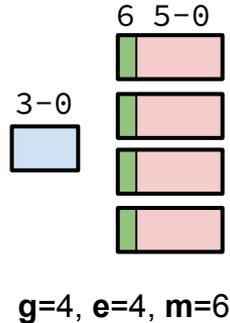
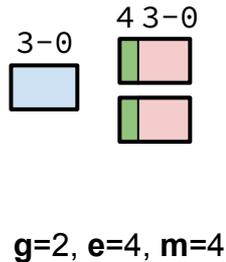


Customized FP Representation



- Numerous customized FP representations have been developed to facilitate DNN execution.
- FP can be applied to a block of numbers, with the block size defined in a customizable manner.

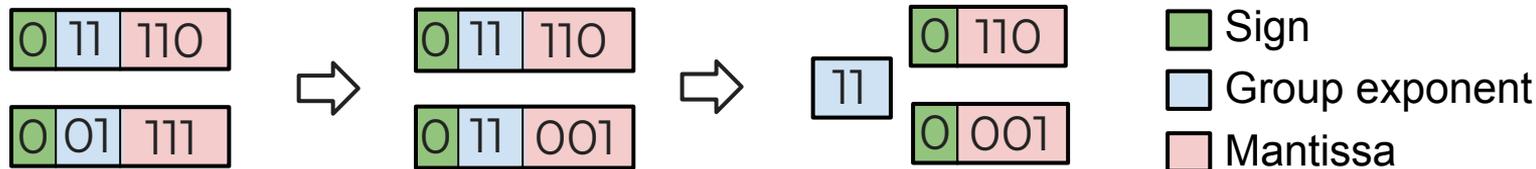
Block Floating Point (BFP)



■ Sign field ■ Exponent (e) ■ Mantissa (m)

- BFP formats offer a middle ground between FP and INT formats, by enforcing that a group of values share a common exponent while maintaining individual mantissas.

Block-Floating Arithmetics (BFP)



- Block floating point (BFP) is a numerical representation method that applies a shared exponent to a block of fixed-point values, balancing precision and dynamic range while reducing computational complexity compared to full floating-point arithmetic.
- There is no “leading 1”.

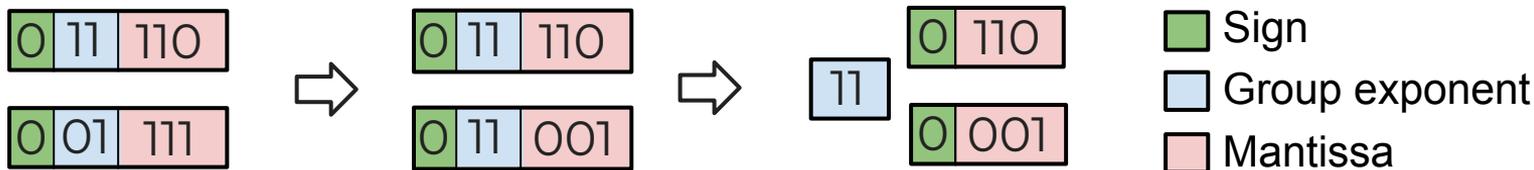
$$\begin{aligned}
 x &= (-1)^s \times 2^{e-bias} \times (1 + m) \\
 m &= (0.b_0b_1b_2\dots b_{22})_2
 \end{aligned}$$

FP

$$\begin{aligned}
 x &= (-1)^s \times 2^{e-bias} \times m \\
 m &= (b_0.b_1b_2b_3\dots b_{22})_2
 \end{aligned}$$

BFP

Block-Floating Arithmetics (BFP)



- Inner-group operations are performed using fixed-point arithmetic.
- Cross-group operations are performed using floating-point arithmetic.
- Each group exponent also includes a bias, which is shared across all the groups.

$$x = (-1)^s \times 2^{e-bias} \times (1 + m)$$

$$m = (0.b_0b_1b_2 \dots b_{22})_2$$

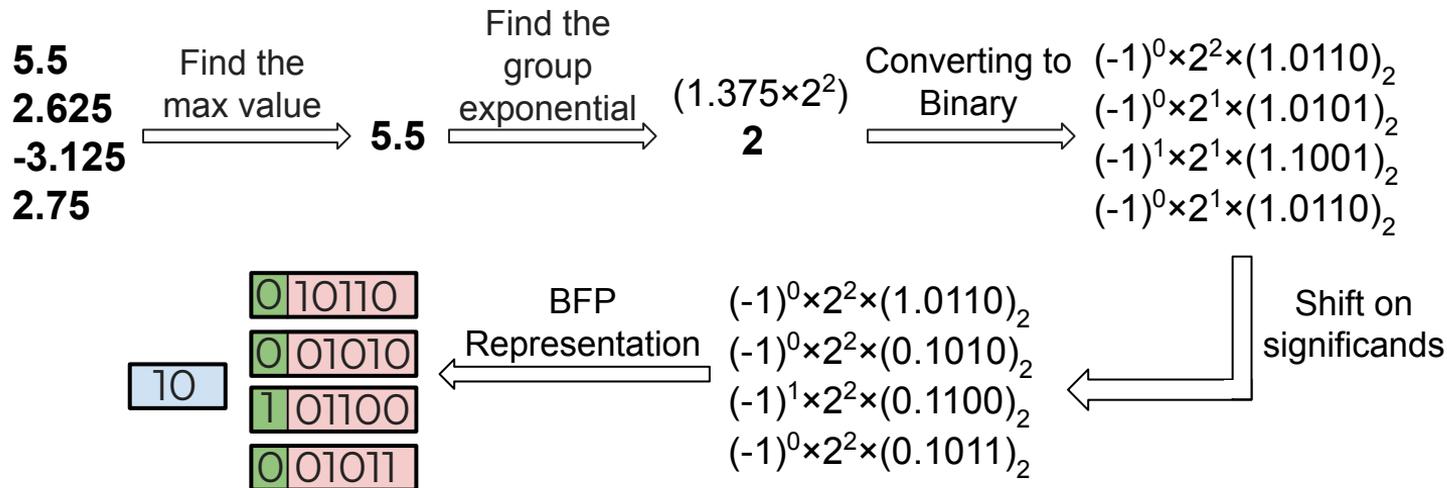
FP

$$x = (-1)^s \times 2^{e-bias} \times m$$

$$m = (b_0.b_1b_2b_3 \dots b_{22})_2$$

BFP

Example

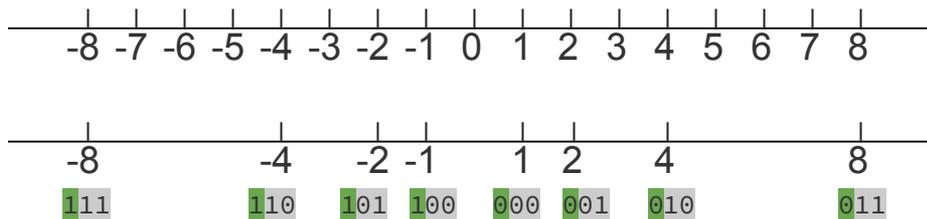


Assume the bias is 0

- Sign
- Group exponent
- Mantissa

Logarithm Arithmetics

- A specialized form of integer (INT) quantization
- Utilizes only power-of-two integer values, making hardware multiplication more efficient and cost-effective.



- Each INT number can be represented by its exponent = $\log(\text{INT})$.
- A total of 8 numbers, 3 bits are needed to encode the bits.

$$a = (1100)_2 \quad a \times 2 = (11000)_2 \quad a \times 8 = (1100000)_2$$

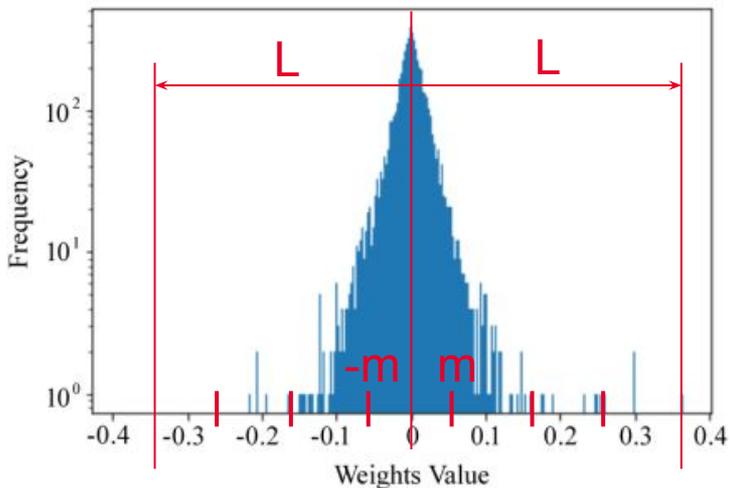
Topics

- Basic Data Formats
 - Fixed point (INT)
 - Floating point (FP)
 - Block floating point (BFP)
- Quantization methods
 - Taxonomy of Quantization
 - Learnable adaptive quantization scheme

Taxonomy of Quantization

- Quantization techniques can be classified from different perspectives:
 - **Weight quantization, activation quantization**
 - Quantization aware training, post training quantization
 - Tensor-based quantization, vector-based quantization, group-based quantization
 - Quantization for inference/training
 - Deterministic quantization, stochastic quantization

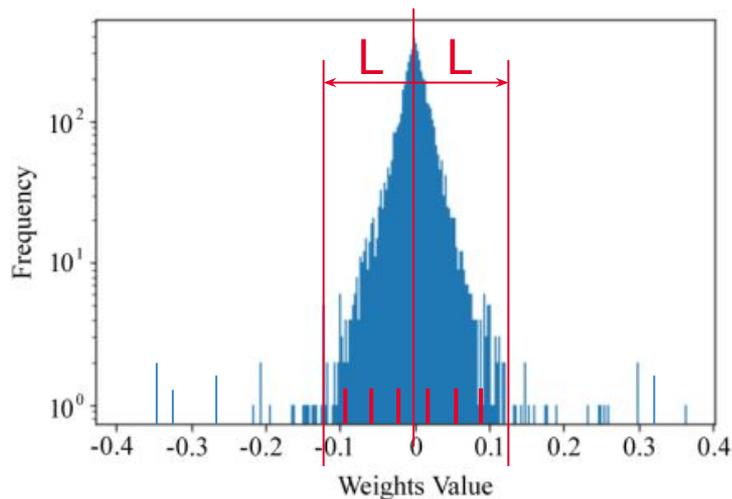
Weight Quantization



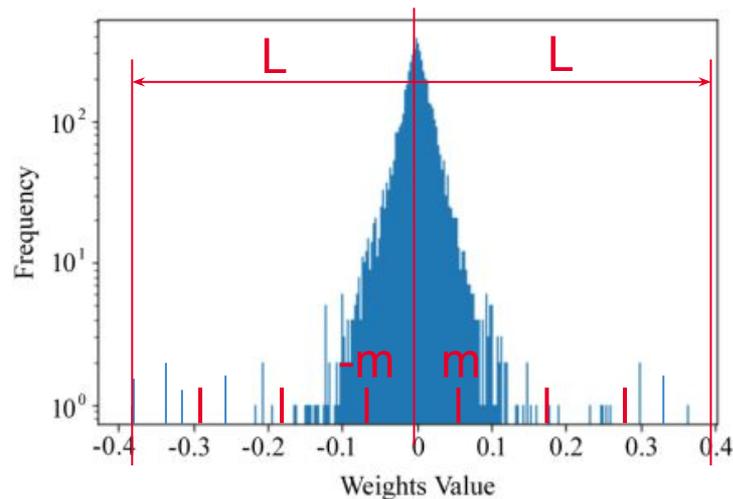
Weight distribution in ResNet

- The weight distribution follows a gaussian-like distribution.
- The outlier will lead to large quantization error.
- A good selection on the clip range L is critical for accuracy performance.

Weight Quantization



- Large truncation error
- Low quantization error for small values

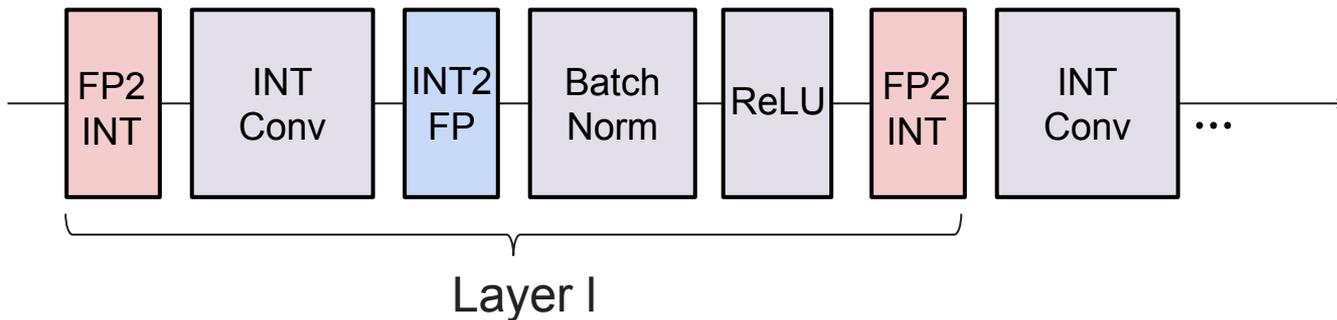


- Small truncation error
- Large quantization error for small values

- $L = 0.9 \times \max(|W|)$, $L = 0.95 \times \max(|W|)$, 0.9 and 0.95 are chosen by experience.

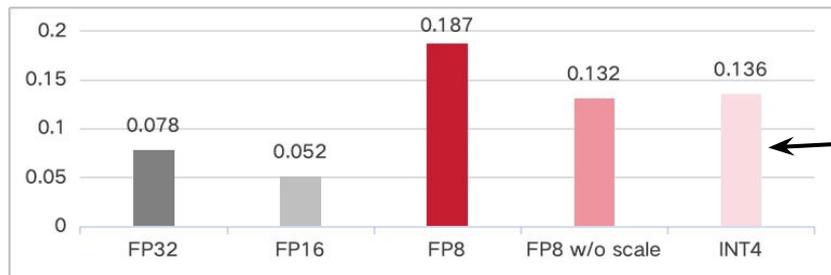
Activation Quantization

- Quantization on activation needs to be performed dynamically. This will introduce additional compute overhead.
- Also the activation will pass the nonlinear functions, which are usually very sensitive to quantization error, so dequantization is required to convert back to FP 16/32.



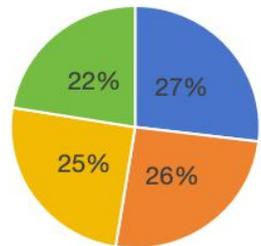
Activation Quantization

$(577 \times 1024) \times$
 (1024×1024)
Projection Layer:
Input: 577×1024
Weight: 4096×1024



On 4090 GPU

MatMul (yellow) Cal_scale (blue)
Dequantize (green) Quant (orange)



- For low-precision quantization, the quantization process may cause more computation than the computational savings achieved by using low-precision quantization.

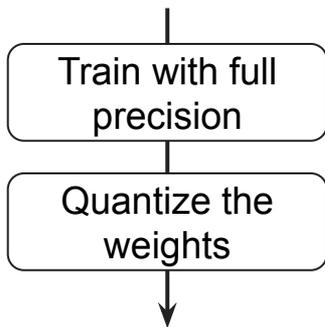
- Set the clipping range: $(-L, L)$, bit width: b
- Compute the scale: $s = 2L / (2^b - 2)$
- Clip the input x : $x_c = \text{Clip}(x, L, -L)$
- Calculate the INT representation: $x_{int} = \text{round}(x_c / s)$
- Rescale: $x_q = s x_{int}$

Taxonomy of Quantization

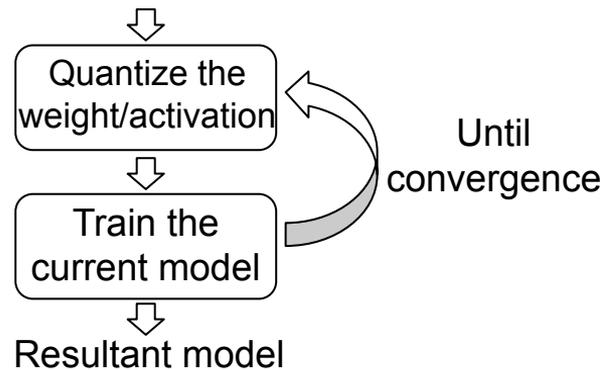
- Quantization techniques can be classified from different perspectives
 - Weight quantization, activation quantization
 - **Post training quantization, quantization aware training**
 - Tensor-based quantization, vector-based quantization, group-based quantization
 - Quantization for inference/training
 - Deterministic quantization, stochastic quantization

When to Quantize?

Post-training quantization (PTQ)

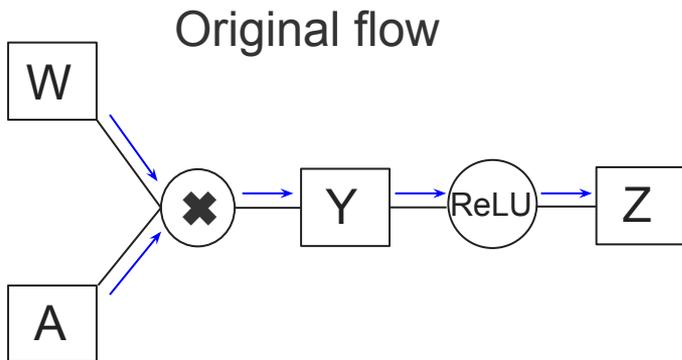


Quantization-aware Training (QAT)



- PTQ has lower computational cost, but accuracy is also lower.
- For the model which is expensive to train (LLM), PTQ is applied to facilitate their implementations.

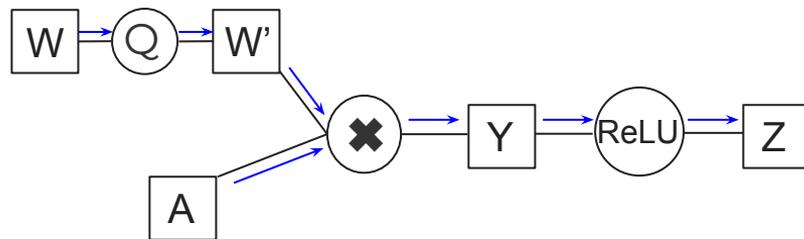
Another Way to Look at Quantization



$$Y = WA, Z = \text{ReLU}(Y)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial W}$$

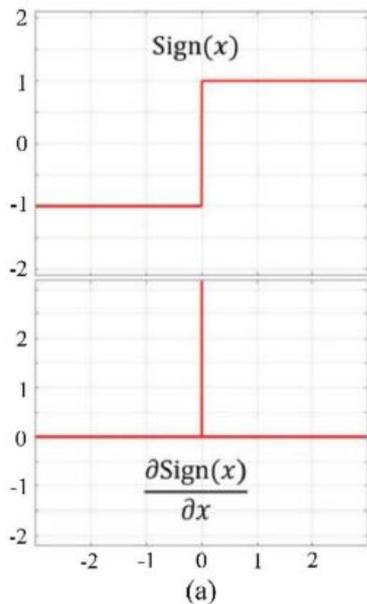
Flow with quantization



$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial W'} \frac{\partial W'}{\partial W}$$

How to compute $\frac{\partial W'}{\partial W}$?

Straight Through Estimator (STE)



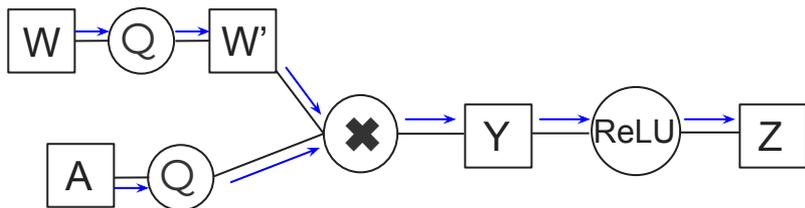
- Staircase function has a derivative of 0 at most of the values. This will make the DNN not trainable.
- We instead use STE to estimate the gradient of a non-differentiable quantized function in the backward pass.

$$\frac{\partial W'}{\partial W} = 1$$

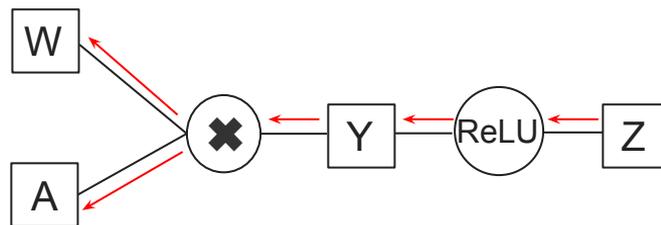
- During the forward pass, apply quantization, for backprop, ignore it.

Straight Through Estimator (STE)

Forward pass

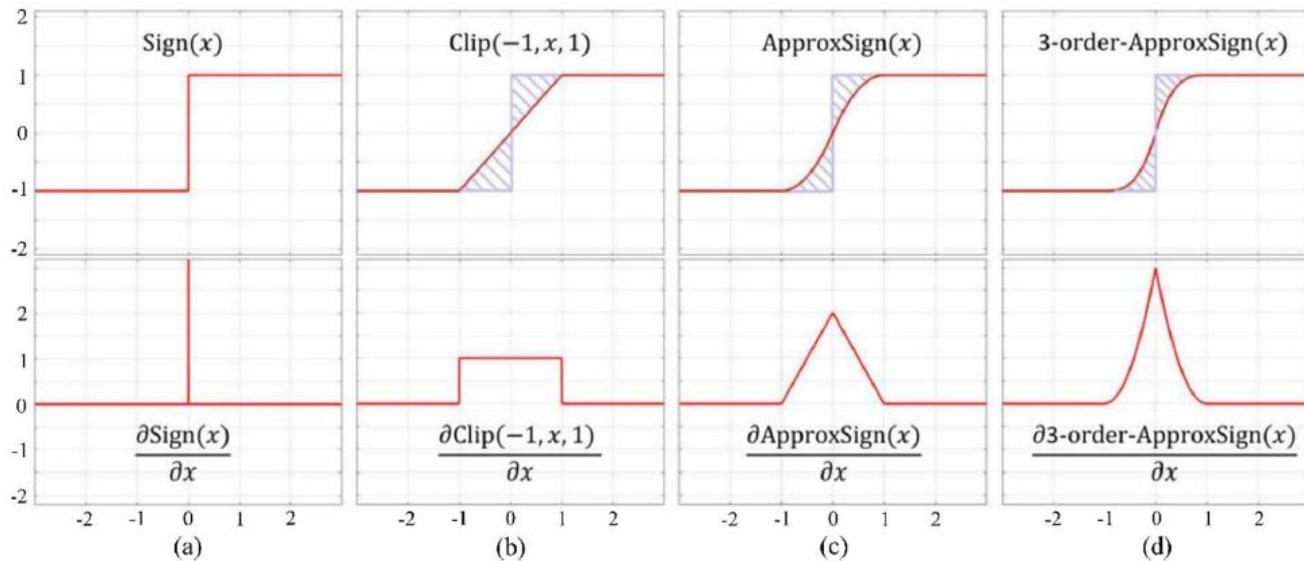


Backward pass



- During the forward pass, apply quantization, for backprop, ignore it.

Other Ways to Approximate Quantization



Pytorch Implementation of Quantization

```
def forward(self, x):  
    y = F.conv2d(self.w, x)  
    return y
```

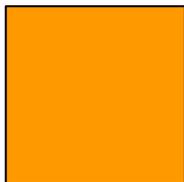
```
def forward(self, x, b, L):  
    self.quantized_w = Q(self.w, b, L)  
    y = F.conv2d(self.quantized_w, x)  
    return y  
def Q(w, b, L):  
    L = 0.9 * w.abs().max()  
    w = torch.clip(w, min=-L, max=L)  
    scale = 2L / (2**b - 2)  
    wq = (w / scale).round() * scale  
    return wq
```

Taxonomy of Quantization

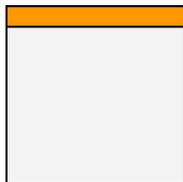
- Quantization techniques can be classified from different perspectives
 - Weight quantization, activation quantization
 - Post training quantization, quantization aware training
 - **Tensor-based quantization, vector-based quantization, group-based quantization**
 - Quantization for inference/training
 - Deterministic quantization, stochastic quantization

Granularity of Quantization

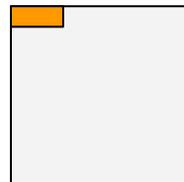
- The weight can be quantized with different granularity:
 - Tensor-based quantization
 - Vector-based quantization
 - Group-based quantization
- A higher quantization granularity will lead to a lower quantization error and a higher hardware implementation cost.



Tensor-based
quantization



Vector-based
quantization

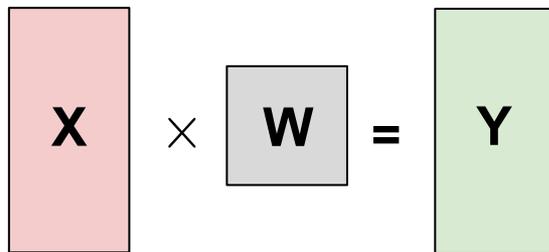


Group-based
quantization

Taxonomy of Quantization

- Quantization techniques can be classified from different perspectives
 - Weight quantization, activation quantization
 - Post training quantization, quantization aware training
 - Tensor-based quantization, vector-based quantization, group-based quantization
 - **Quantization for inference/training**
 - Deterministic quantization, stochastic quantization

Quantization During Training



X: input

W: weight filters

Y: output

- The forward propagation is very similar to the inference operation, where the input **X** is multiplied by weight **W**, generating the output **Y**.

Quantization During Training

Data gradient
Computation

$$\nabla Y \times W^T = \nabla X$$

X: input
 ∇X : input gradient

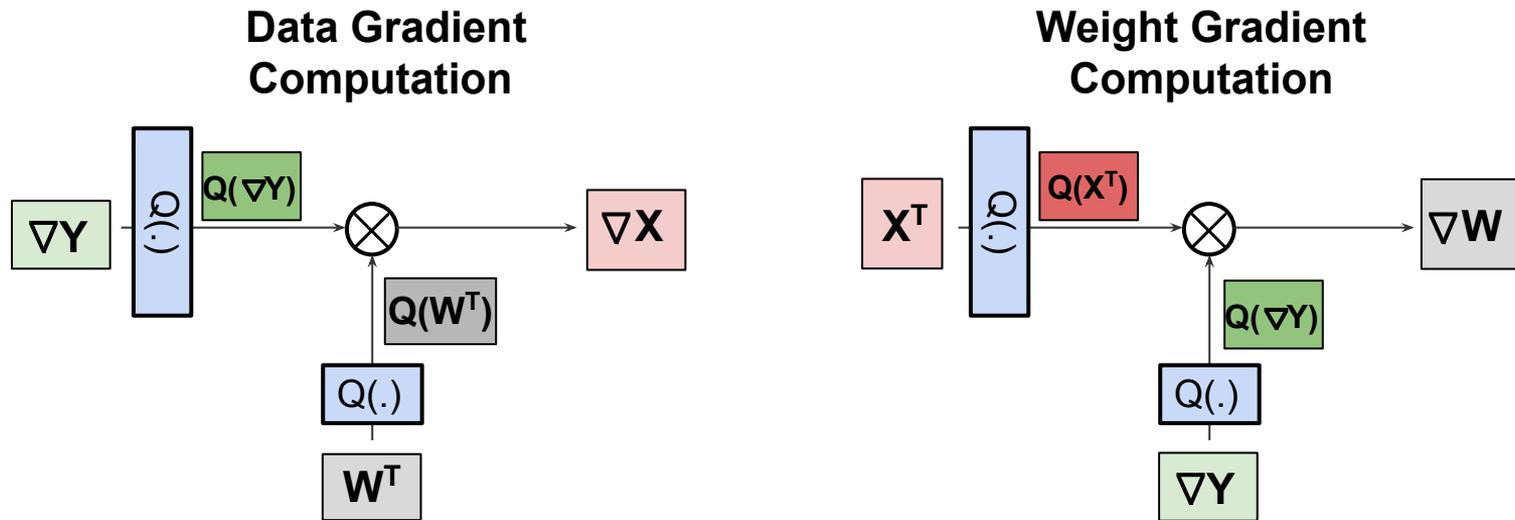
Weight gradient
Computation

$$X^T \times \nabla Y = \nabla W$$

W: weight filters
 ∇W : weight gradient

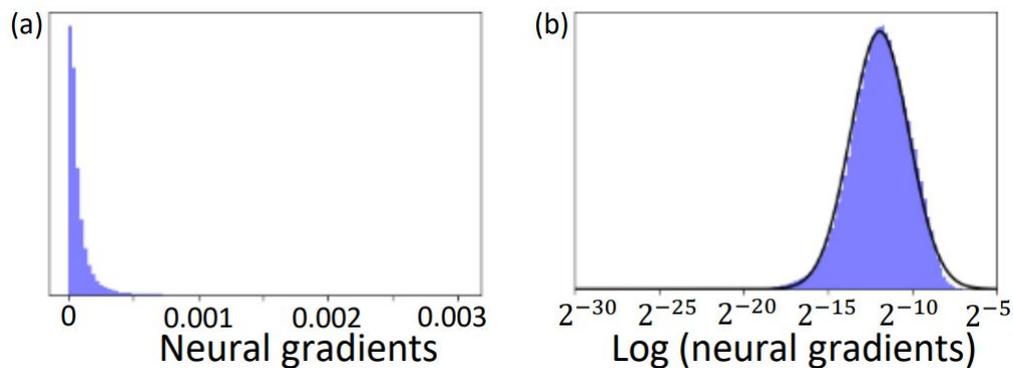
Y: output
 ∇Y : output gradient

Quantization During Training



- Gradient is much more sensitive to quantization error.

DNN Gradient Distribution

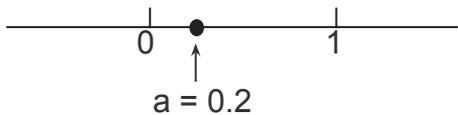


- DNN gradient is much hard to quantize and very sensitive to quantization error.

Taxonomy of Quantization

- Quantization techniques can be classified from different perspectives
 - Weight quantization, activation quantization
 - Post training quantization, quantization aware training
 - Tensor-based quantization, vector-based quantization, group-based quantization
 - Quantization for inference/training
 - **Deterministic quantization, stochastic quantization**

Deterministic and Stochastic Quantization

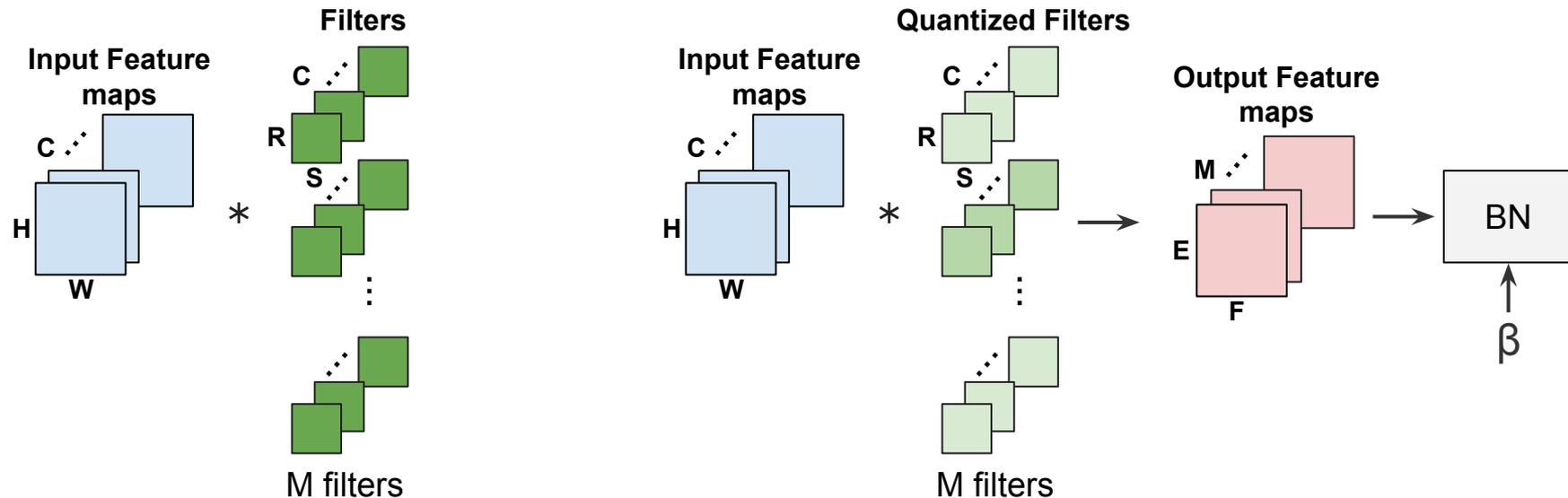


- To quantize a , conventional linear quantization will make $q(a) = 0$. However, this will cause a bias.
- With stochastic quantization:

$$q(a) = \begin{cases} 1 & \text{for } p = 0.2 \\ 0 & \text{for } p = 0.8 \end{cases}$$

- For QAT, the bias will not cause any problem, due to the existence of bias in BN.
- Stochastic quantization is extremely useful when applying quantization to accelerate DNN training.

Deterministic and Stochastic Quantization

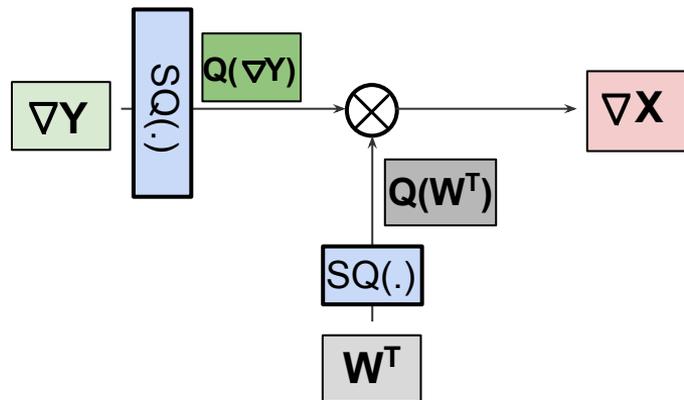


FP weights

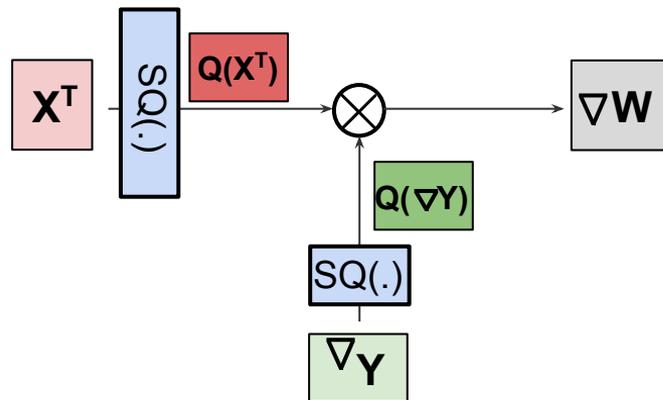
Quantized weights

Quantization During Training

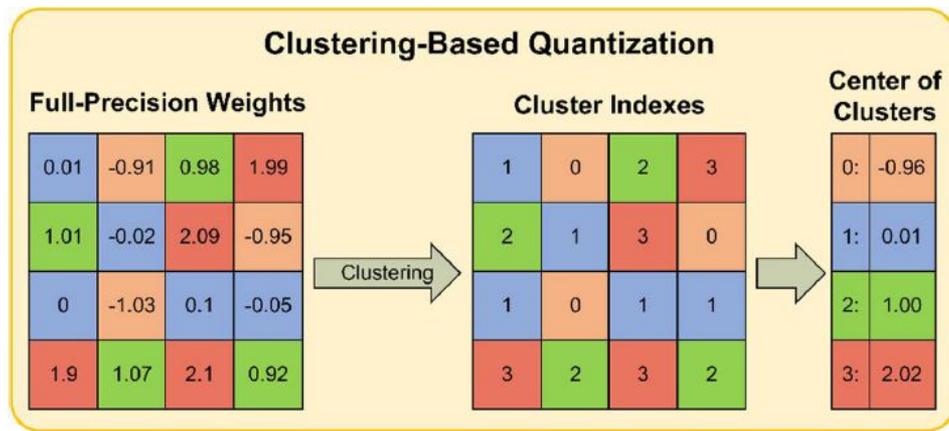
Data Gradient Computation



Weight Gradient Computation



Clustering-Based Quantization



- Quantization and clustering share similarities. In clustering, each value is assigned to a centroid, while in quantization, each full-precision value is mapped to one of the predefined quantization levels.
- Clustering is usually used to compress the weight matrix and efficient storage, but it is hard for accelerating computations.
- However, due to the flexibility of selecting the centroid, clustering usually achieves a better accuracy.

Topics

- Basic Data Formats
 - Fixed point (INT)
 - Floating point (FP)
 - Block floating point (BFP)
- Quantization methods
 - Taxonomy of Quantization
 - Learnable adaptive quantization scheme

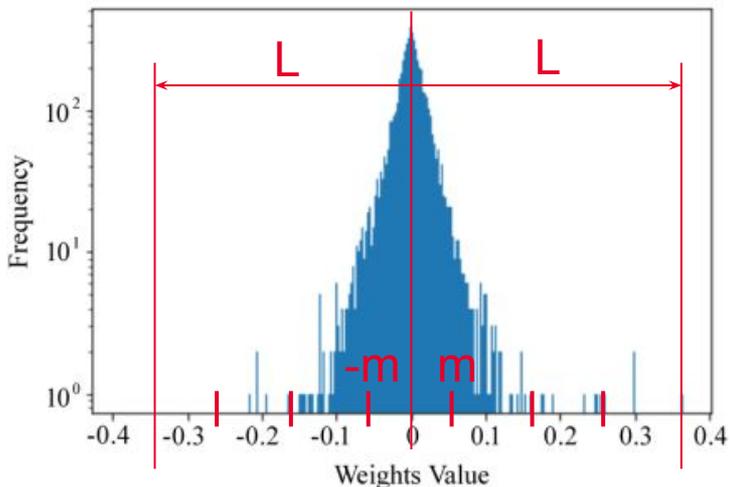
Learnable Quantization

- Multiple methods have been proposed to learn the quantization hyperparameters:
 - PACT
 - QIL
 - Quantization network

Learnable Quantization

- How to convert a number to INT8 representation?
 - Set the clipping range: $(-L_{min}, L_{max})$, bitwidth: b
 - Compute the scale: $s = (L_{max} - L_{min}) / (2^b - 1)$
 - Clip the input x : $x_c = Clip(x, L_{min}, L_{max})$
 - Calculate the fixed-point representation:
$$x_{int} = round((x_c - L_{min}) / s)$$
 - Rescale: $x_q = sx_{int} + L_{min}$

Learnable Quantization



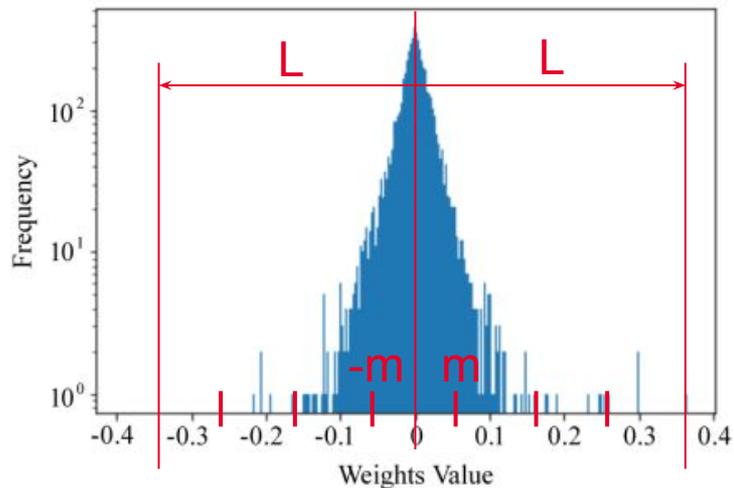
Weight distribution in ResNet

- How to convert a number to INT8 representation?
 - Set the clipping range: $(-l, l)$, bitwidth: b
 - Compute the scale: $s = (2l)/(2^b - 1)$
 - Clip the input x : $x_c = \text{Clip}(x, l, -l)$
 - Calculate the fixed-point representation:
$$x_{\text{int}} = \text{round}(x_c/s)$$
 - Rescale: $x_q = s x_{\text{int}}$

$$l = 0.9 \times \max(|W|), \quad l = 0.95 \times \max(|W|)$$

Can learn by learnt during training?

Learnable Quantization



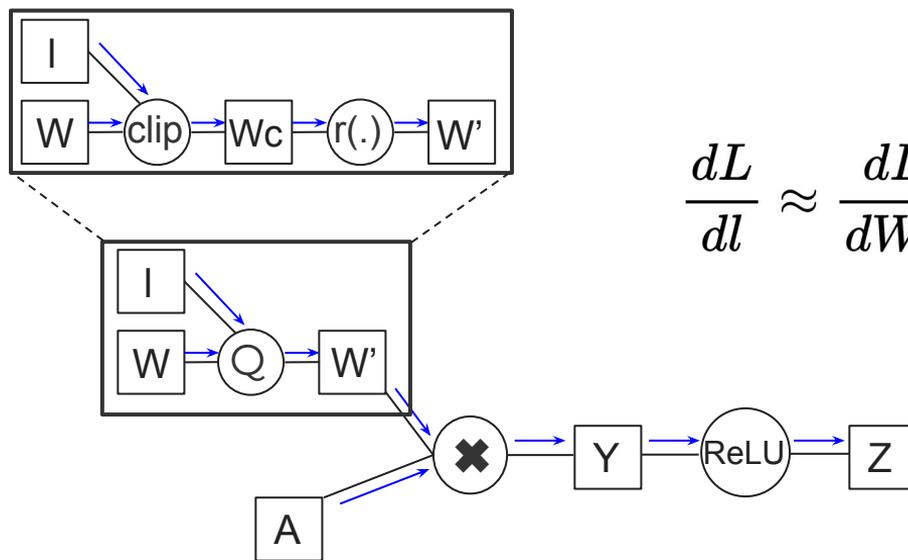
- First we need to apply CLIP function to the input x , where the clip function has a range of $(-l, l)$.

- $$x_c = \text{Clip}(x, l) = \begin{cases} l, & \text{if } x \geq l \\ x, & -l \leq x \leq l \\ -l, & x \leq -l \end{cases}$$

$$x_q = \text{round}\left(\frac{x_c}{s}\right) \times s$$

- Can we learn l ?
$$\frac{dL}{dl} = \frac{dL}{dx_q} \frac{dx_q}{dx_c} \frac{dx_c}{dl} \approx \frac{dL}{dx_q} \frac{dx_c}{dl}$$

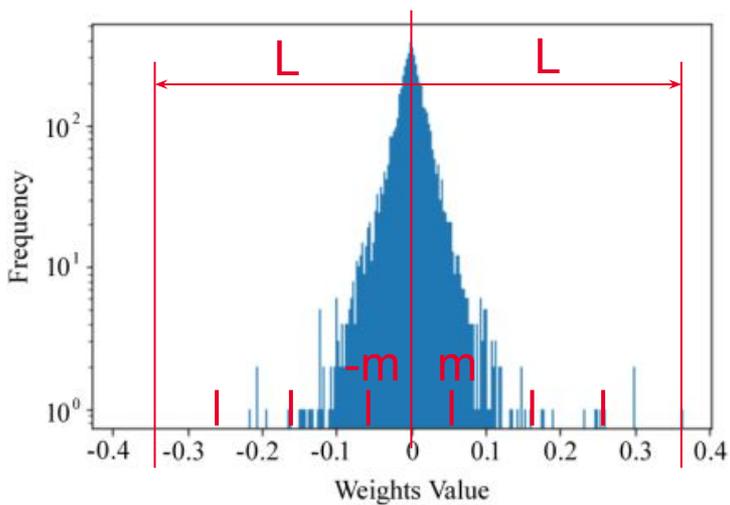
Learnable Quantization



$$\frac{dL}{dl} \approx \frac{dL}{dW'} \frac{dW'}{dW_c} \frac{dW_c}{dl}$$

1

Learnable Quantization



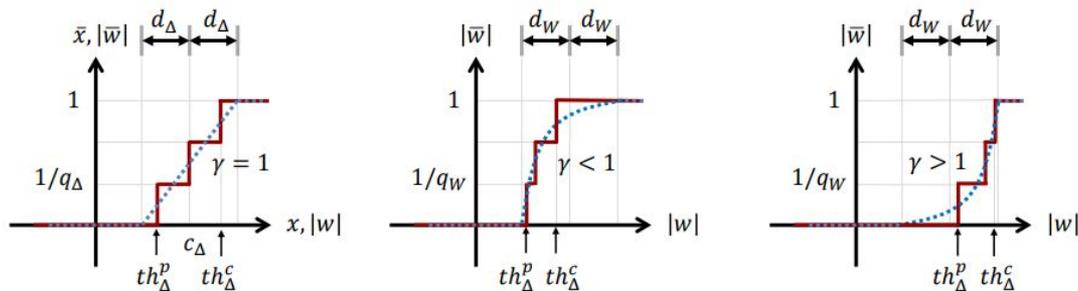
$$\text{Clip}(x, l) = \begin{cases} l, & \text{if } x \geq l \\ x, & -l \leq x \leq l \\ -l, & x \leq -l \end{cases}$$

$$\frac{d\text{Clip}(x, l)}{dx} = \begin{cases} 0, & \text{if } x \geq l \\ 1, & -l \leq x \leq l \\ 0, & x \leq -l \end{cases}$$

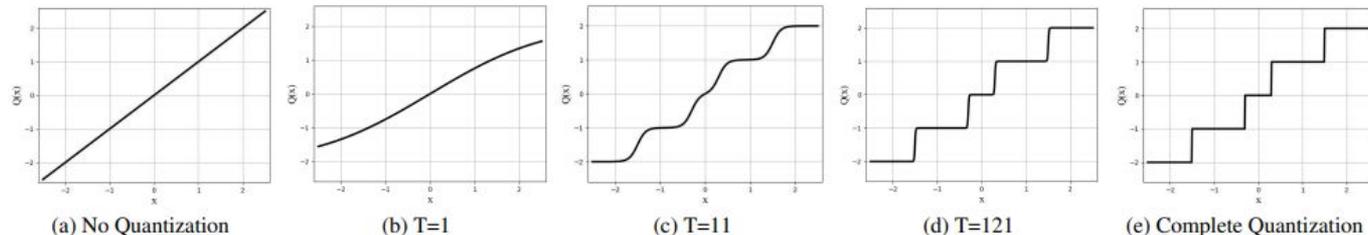
$$\frac{d\text{Clip}(x, l)}{dl} = \begin{cases} 1, & \text{if } x \geq l \\ 0, & -l \leq x \leq l \\ -1, & x \leq -l \end{cases}$$

L can be learnable

Learnable Quantization



QIL



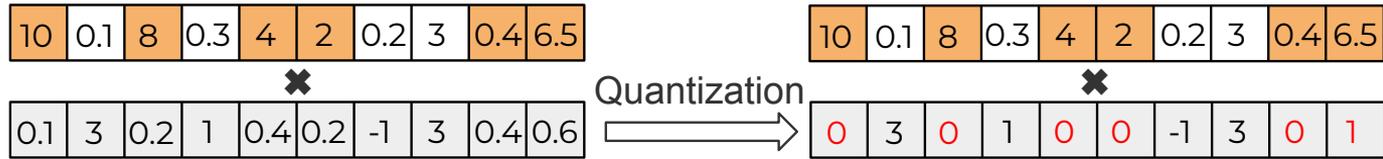
QN

Jung, Sangil, et al. "Learning to quantize deep networks by optimizing quantization intervals with task loss." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

Yang, Jiwei, et al. "Quantization networks." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

Problem of Quantization

- The major drawback of quantization is that it does not consider the impact of the input when making the quantization decision.



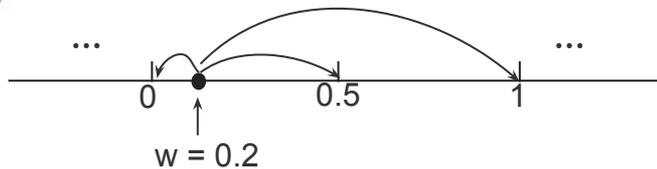
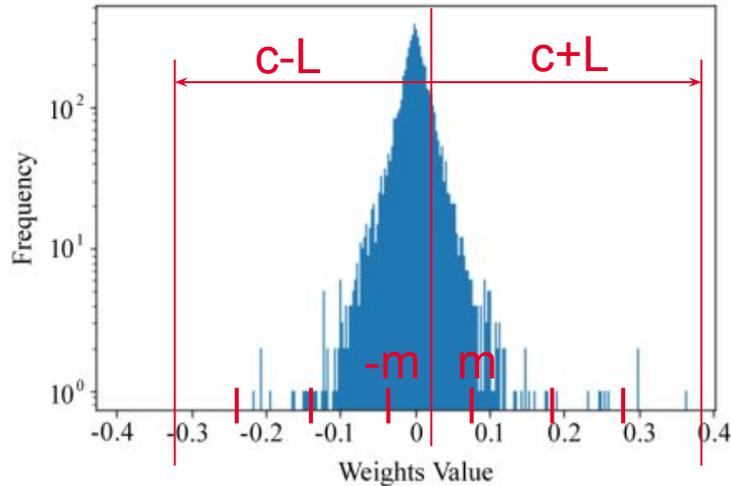
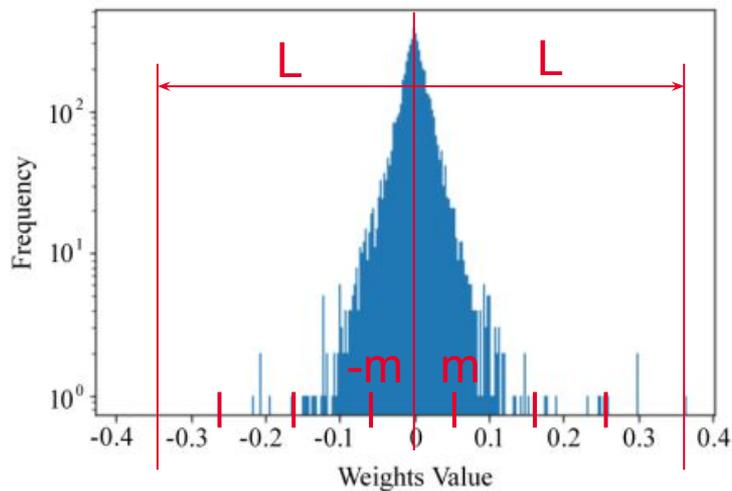
Problem of Quantization



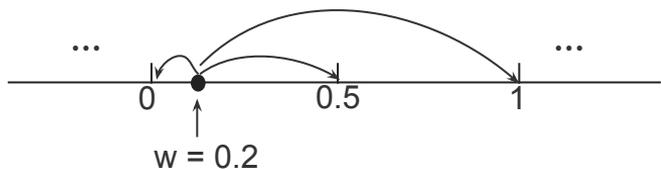
- We use a calibration dataset and profile some data x_l, y_l across each layer.
- After that, we use these data to train the optimal quantized weights.

$$\min_{W_l} \|Y_l - X_l Q(W_l)\|^2 \quad \text{For each } l$$

Quantization Interval Learning (QIL)



Quantization Interval Learning (QIL)



- To achieve this rounding flexibility, we combine a learnable function with quantization.

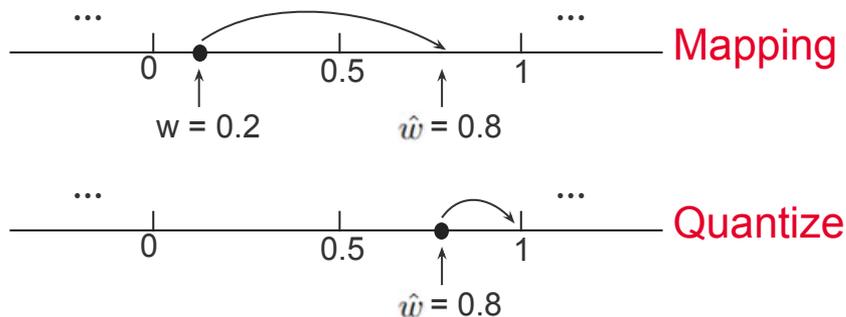
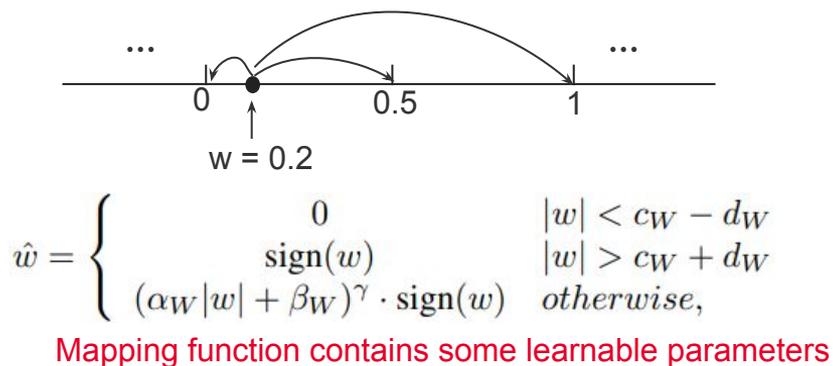
$$w_q = Q(w) \longrightarrow w_q = Q(F(w))$$

- $F(\cdot)$ is a function which contains learnable hyperparameters.

$$\hat{w} = \begin{cases} 0 & |w| < c_W - d_W \\ \text{sign}(w) & |w| > c_W + d_W \\ (\alpha_W |w| + \beta_W)^\gamma \cdot \text{sign}(w) & \text{otherwise,} \end{cases}$$

Quantization Interval Learning (QIL)

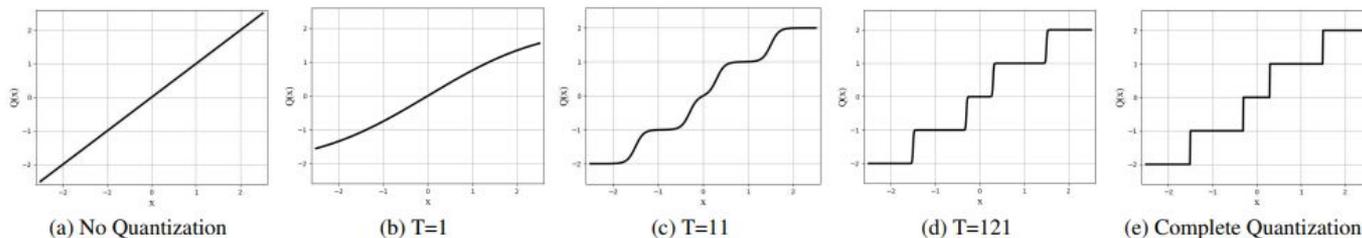
- QIL offers flexibility to round the FP weights.



- $w_q = Q(F(w))$ are stored for inference after the training process finished.
- We can not apply this techniques over the activation, due to its large computational overhead.

Quantization Networks

- We propose a novel perspective of interpreting and implementing neural network quantization by formulating low-bit quantization as a differentiable non-linear function.



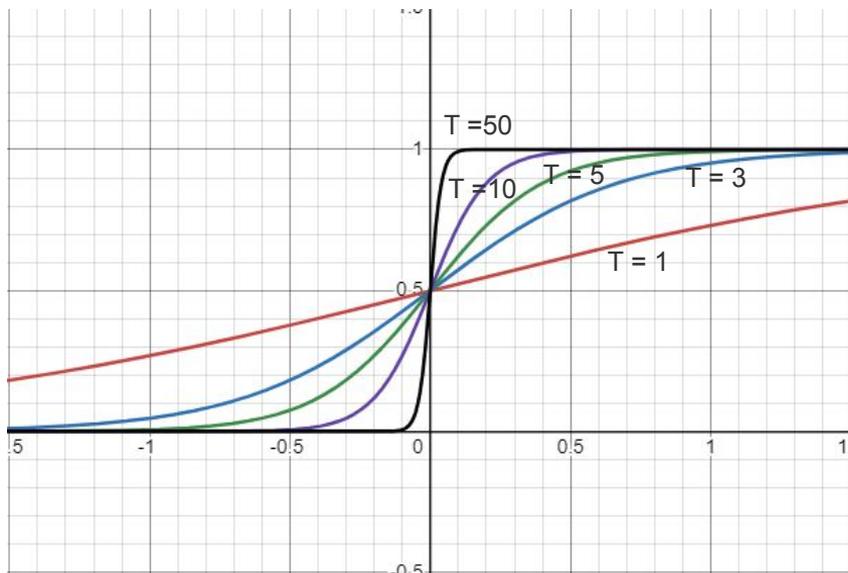
$$y = \alpha \left(\sum_{i=1}^n s_i \mathcal{A}(\beta x - b_i) - o \right)$$

$$\mathcal{A}(x) = \begin{cases} 1 & x \geq 0, \\ 0 & x < 0. \end{cases}$$

- $n + 1$ is the number of quantization intervals
- β is the scale factor of inputs
- s_i and b_i are the scales and biases for the unit step functions

Quantization Networks

$$\mathcal{A}(x) = \begin{cases} 1 & x \geq 0, \\ 0 & x < 0. \end{cases} \quad \sigma(Tx) = \frac{1}{1 + \exp(-Tx)}$$



- We can replace the staircase function with a sigmoid function.
- We can progressively increase T during the training process.

