Quantization – 2

EECE695D: Efficient ML Systems

Spring 2025

Agenda

- Linear quantization
- Various issues
  - Granularity
  - Rescaling
  - Clipping
  - Rounding
  - QAT

Linear Quantization

Idea

- Represent each weight as a **scaled-and-shifted integers**
- **Advantage.** Less compute, easy encoding/decoding
  - Matmuls can be done in integer, and scaled-and-shifted back

$$[1.2, 2.4] \begin{bmatrix} -1.1 \\ 3.3 \end{bmatrix} = (1.2) \cdot (1.1) \cdot [1, 2] \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

4 FLOPs

1 FLOP + 4 Integer Op

Formalization

- Represent each weight as **scaled-and-shifted integers**
  - That is, our decoder is:

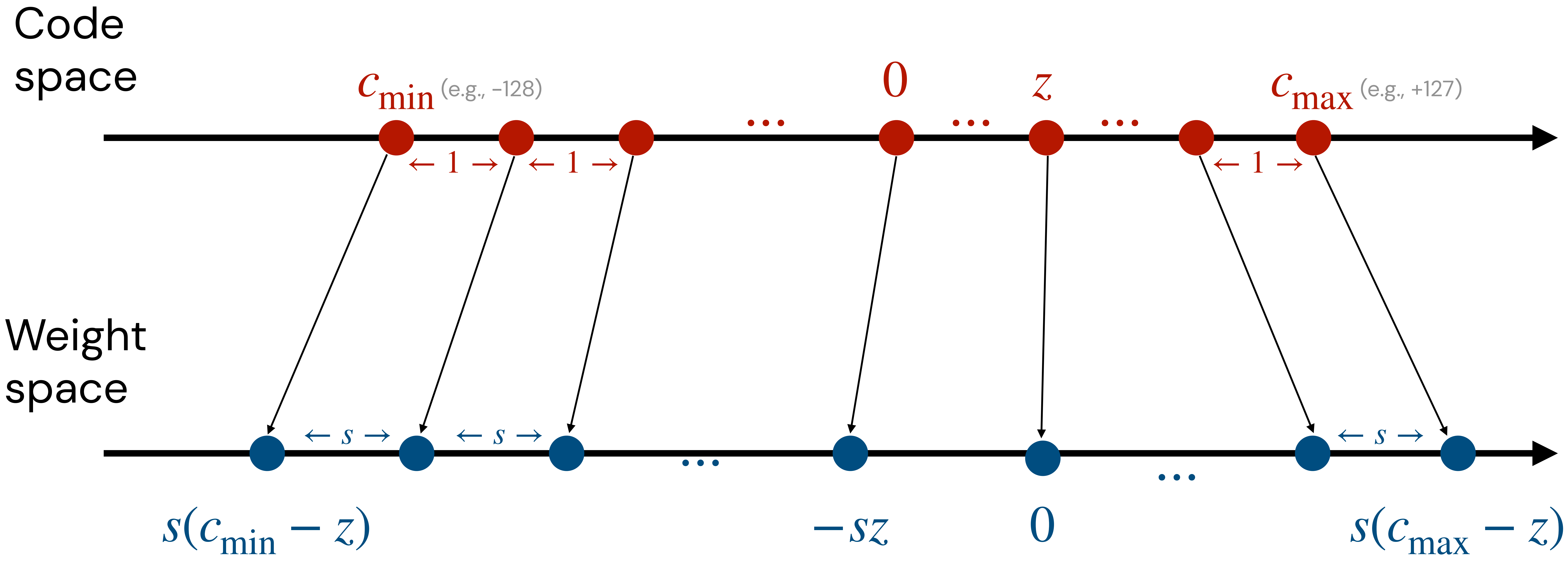
$$\hat{\mathbf{w}} = s \cdot (\mathbf{c} - z\mathbf{1})$$

  - $\hat{\mathbf{w}}$: reconstructed weight
  - $\mathbf{c}$: code (e.g., INT8 $\in \{-128, \ldots, 127\}$)
  - s: scaling factor (e.g., FP32)
  - z: zero point (e.g., INT)

Formalization

$$\hat{\mathbf{w}} = s \cdot (\mathbf{c} - z\mathbf{1})$$

- Visually, what this decoder does is as follows:



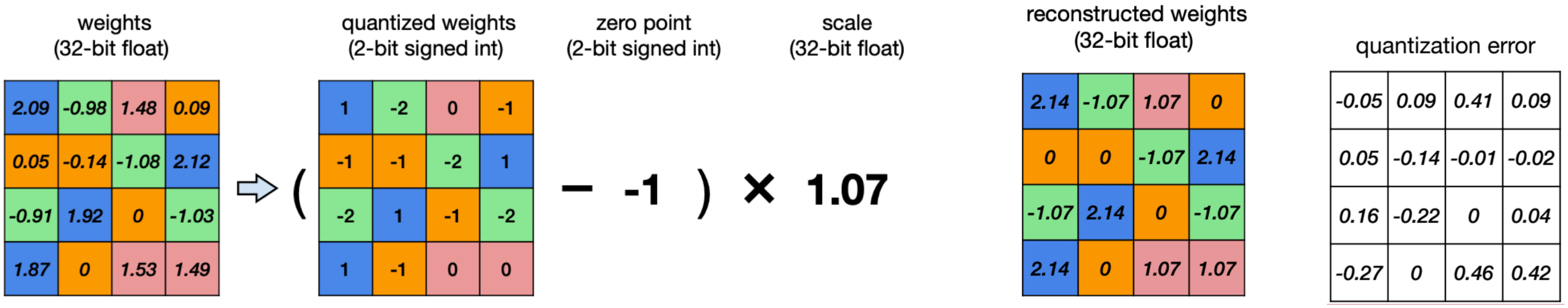
Formalization

$$\hat{\mathbf{w}} = s \cdot (\mathbf{c} - z\mathbf{1})$$

- Given this decoder, we encode to the nearest neighbor

$$\mathbf{c} = \mathrm{round}(\mathbf{w}/s + z\mathbf{1})$$

  - $\mathrm{round}(\,\cdot\,)$: mapping to the nearest integer inside the range



Formalization

- Now we have the encoder and decoder
    - Encoder: $f(\mathbf{w}) = \mathrm{round}(\mathbf{w}/s + z\mathbf{1})$
    - Decoder: $q(\mathbf{c}) = s \cdot (\mathbf{c} - z\mathbf{1})$

- **Want-to-do.** Given the weights $\mathbf{w}$, select the parameters s, z so that it solves

$$\min_{s,z} \hat{L}(g(f(\mathbf{w})))$$

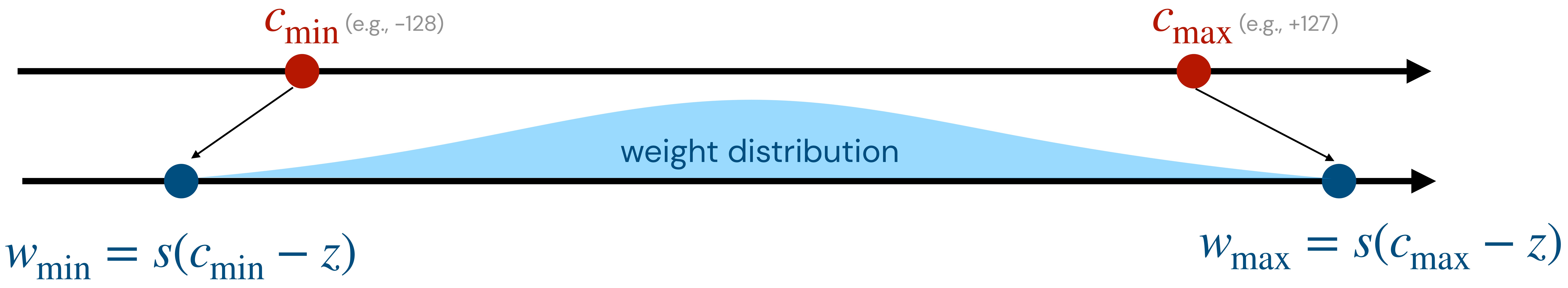
- Of course, this is difficult; thus we use heuristic methods

Minmax Quantization

- A crude but working way: match the range!
  - That is, determine s, z so that

$$\min_i w_i = s(c_{\min} - z), \qquad \max_i w_i = s(c_{\max} - z)$$

  - Philosophy. Range that is just enough to capture the largest weights



Minmax Quantization

$$\min_i w_i = s(c_{\min} - z), \qquad \max_i w_i = s(c_{\max} - z)$$

- Solving this, we get:

$$s = \frac{w_{\max} - w_{\min}}{c_{\max} - c_{\min}}, \qquad z = c_{\min} - \text{round}(w_{\min}/s)$$

- One can do a similar thing to quantize activations:
  - Requires some “calibration data” to compute $x_{\max}, x_{\min}$
    - Quite brittle; often needs some clipping
- <u>Note</u>. It is also popular (and often better) to simply do “grid search”

Minmax Quantization

- **Brain teaser.** Suppose that we want to choose $x_{\max}, x_{\min}$
    - If data and model parameters are sharded over many servers, how much communication cost would we need?

(will be problematic for "dynamic quantization")

Linear Quantization: Matmuls

Matmuls

- Consider the matmul:

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

- Suppose that we have good quantizers for $\mathbf{W}, \mathbf{X}, \mathbf{Y}$:
  - That is, we have $s_{\mathbf{W}}, z_{\mathbf{W}}, s_{\mathbf{X}}, z_{\mathbf{X}}, s_{\mathbf{Y}}, w_{\mathbf{Y}}$

- Then, we get:

$$\begin{aligned} s_{\mathbf{Y}}(\mathbf{c}_{\mathbf{Y}} - z_{\mathbf{Y}}) &= s_{\mathbf{W}}(\mathbf{c}_{\mathbf{W}} - z_{\mathbf{W}}) \cdot s_{\mathbf{X}}(\mathbf{c}_{\mathbf{X}} - z_{\mathbf{X}}) \\ &= s_{\mathbf{W}} s_{\mathbf{X}}(\mathbf{c}_{\mathbf{W}}\mathbf{c}_{\mathbf{X}} - z_{\mathbf{W}}\mathbf{c}_{\mathbf{X}} - z_{\mathbf{X}}\mathbf{c}_{\mathbf{W}} + z_{\mathbf{W}} z_{\mathbf{X}}) \end{aligned}$$

Matmuls

- Rewriting, we have a formula for computing the codes of Y

$$\mathbf{c_Y} = \frac{s_\mathbf{W} s_\mathbf{X}}{s_\mathbf{Y}} (\mathbf{c_W c_X} - z_\mathbf{W} \mathbf{c_X} - z_\mathbf{X} \mathbf{c_W} + z_\mathbf{W} z_\mathbf{X}) + z_\mathbf{Y}$$

  - We have separated out FP ops from INT ops
  - **Problem.** To compute $\mathbf{c_Y}$, we need (FP) * (INT) operation
    - Empirically, $s_\mathbf{W} s_\mathbf{X} / s_\mathbf{Y} \in (0,1)$

      $\Rightarrow$ Write it as $2^{-n} \times M_0$, with M_0 being an INT
      (bit shift)

Matmuls

- Some INT ops can be pre-computed, reducing inference-time compute

$$\mathbf{c}_\mathbf{Y} = \frac{s_\mathbf{W} s_\mathbf{X}}{s_\mathbf{Y}}(\mathbf{c}_\mathbf{W}\mathbf{c}_\mathbf{X} - z_\mathbf{W}\mathbf{c}_\mathbf{X} - z_\mathbf{X}\mathbf{c}_\mathbf{W} + z_\mathbf{W} z_\mathbf{X}) + z_\mathbf{Y}$$

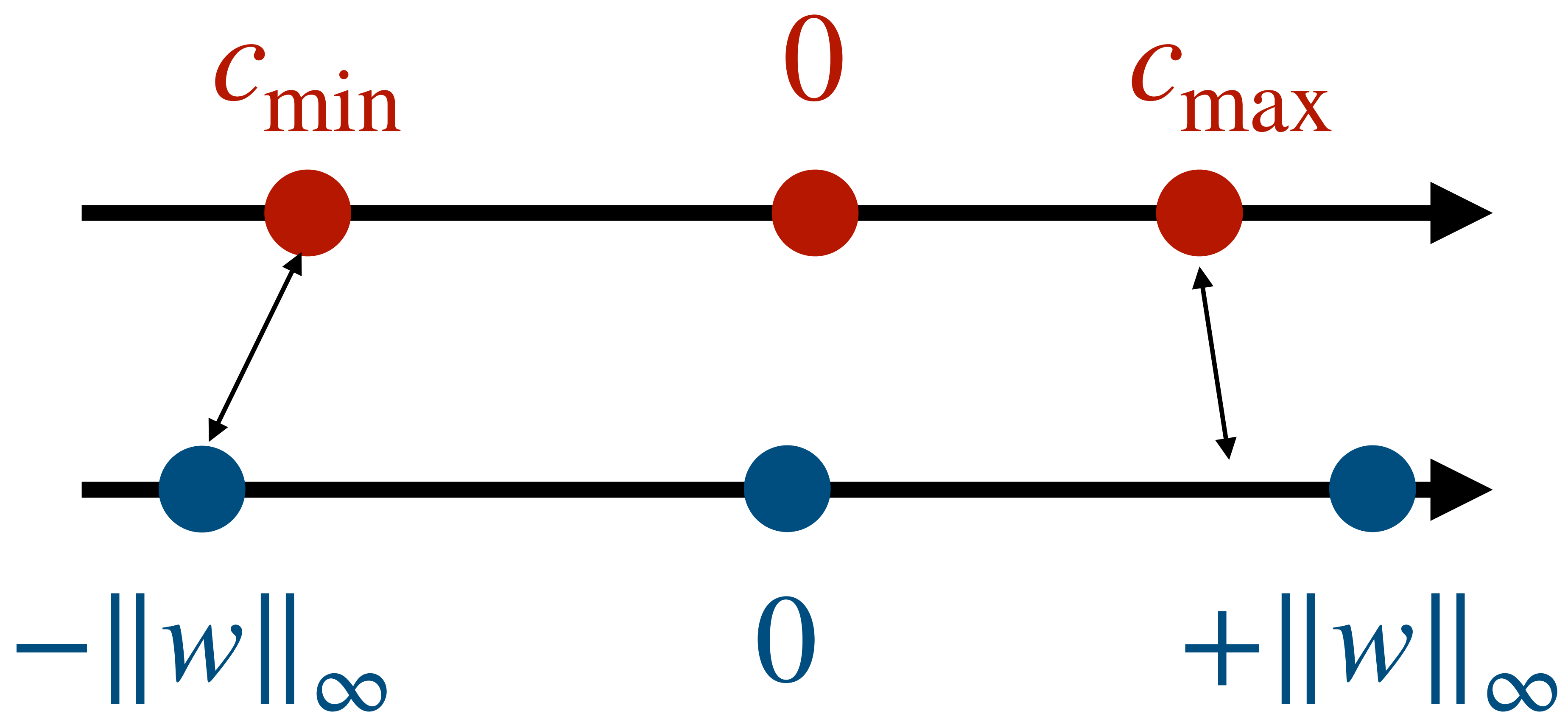
- Also, many weight distributions are nearly symmetric:
  - If we let $z_\mathbf{W} = 0$,

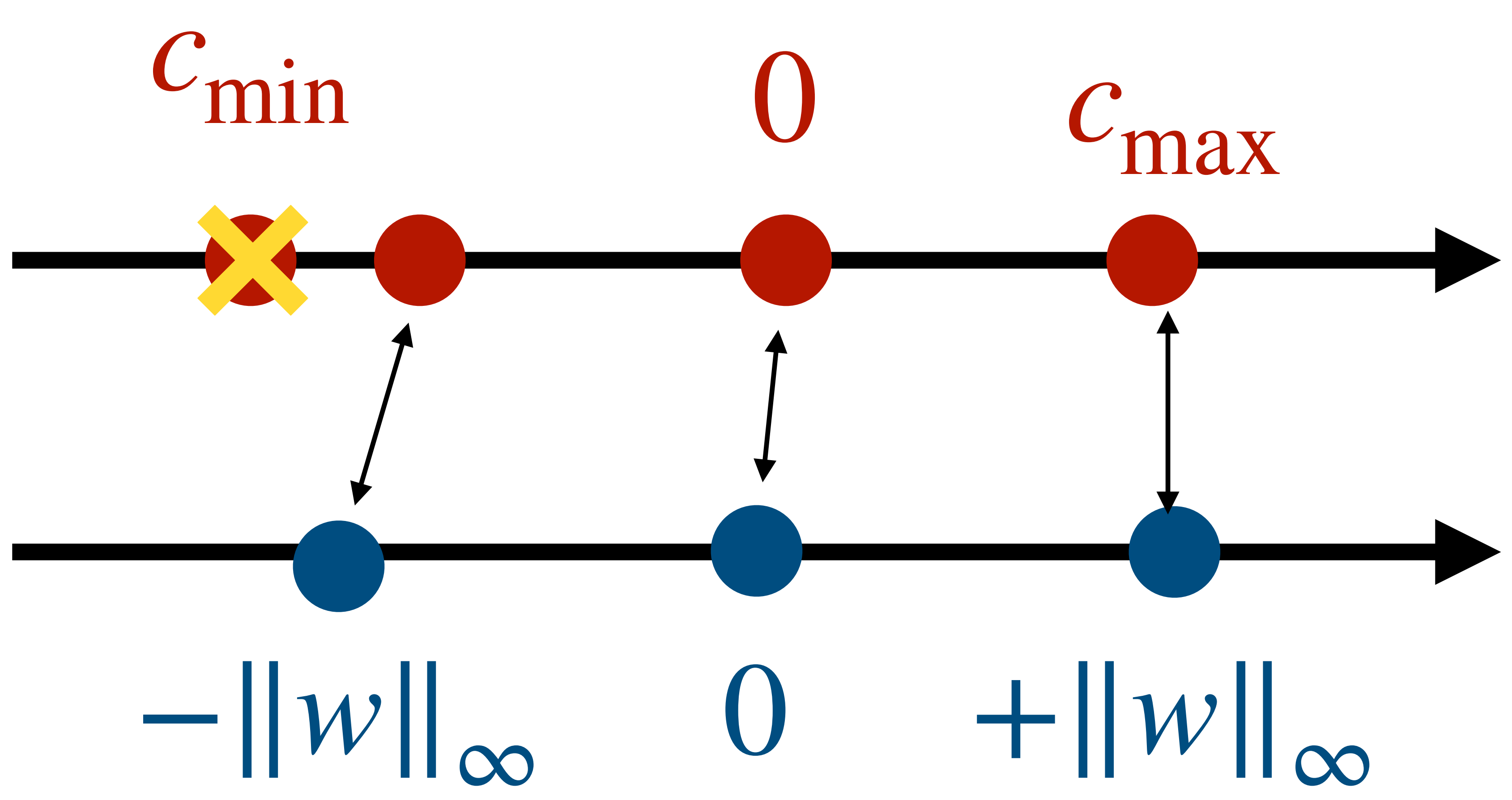
$$\mathbf{c}_\mathbf{Y} = \frac{s_\mathbf{W} s_\mathbf{X}}{s_\mathbf{Y}}(\mathbf{c}_\mathbf{W}\mathbf{c}_\mathbf{X} - z_\mathbf{X}\mathbf{c}_\mathbf{W}) + z_\mathbf{Y}$$

  - Not doable for activations, usually.

Symmetric Quantization

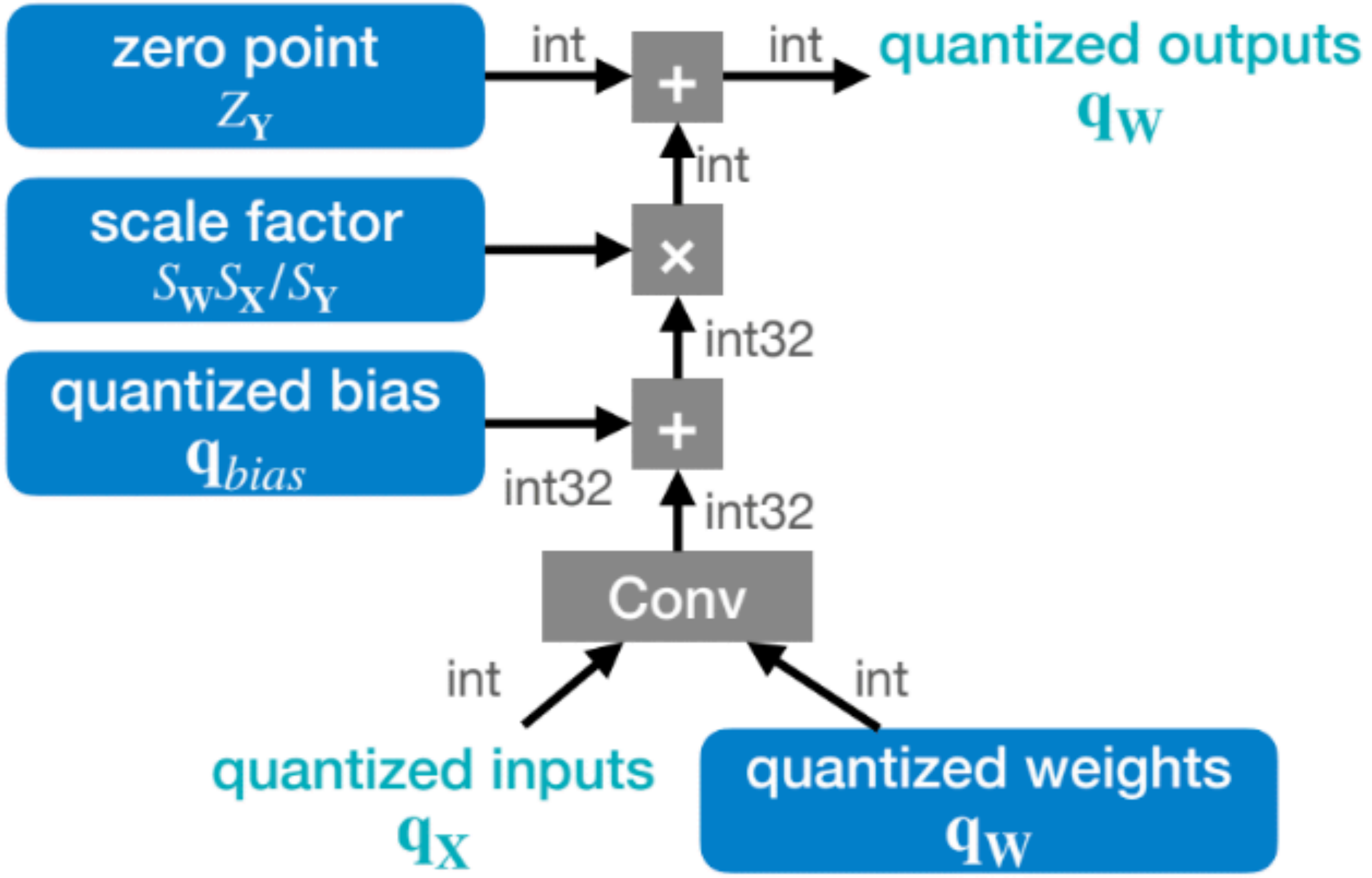
- **Problem.** INT is not symmetric! (e.g., $\{-128,\ldots,127\}$)
  - Two different ways to do symmetric quantization:
    - **Full-range ($\Leftarrow$).** ONNX, PyTorch, …
    - **Restricted-Range ($\Rightarrow$).** TensorFlow, TensorRT, …





Symmetric Quantization

- **Note.** Accumulation can take place in high-bits (e.g., INT32)



Symmetric Quantization

- Integer-only ops can dramatically reduce the latency

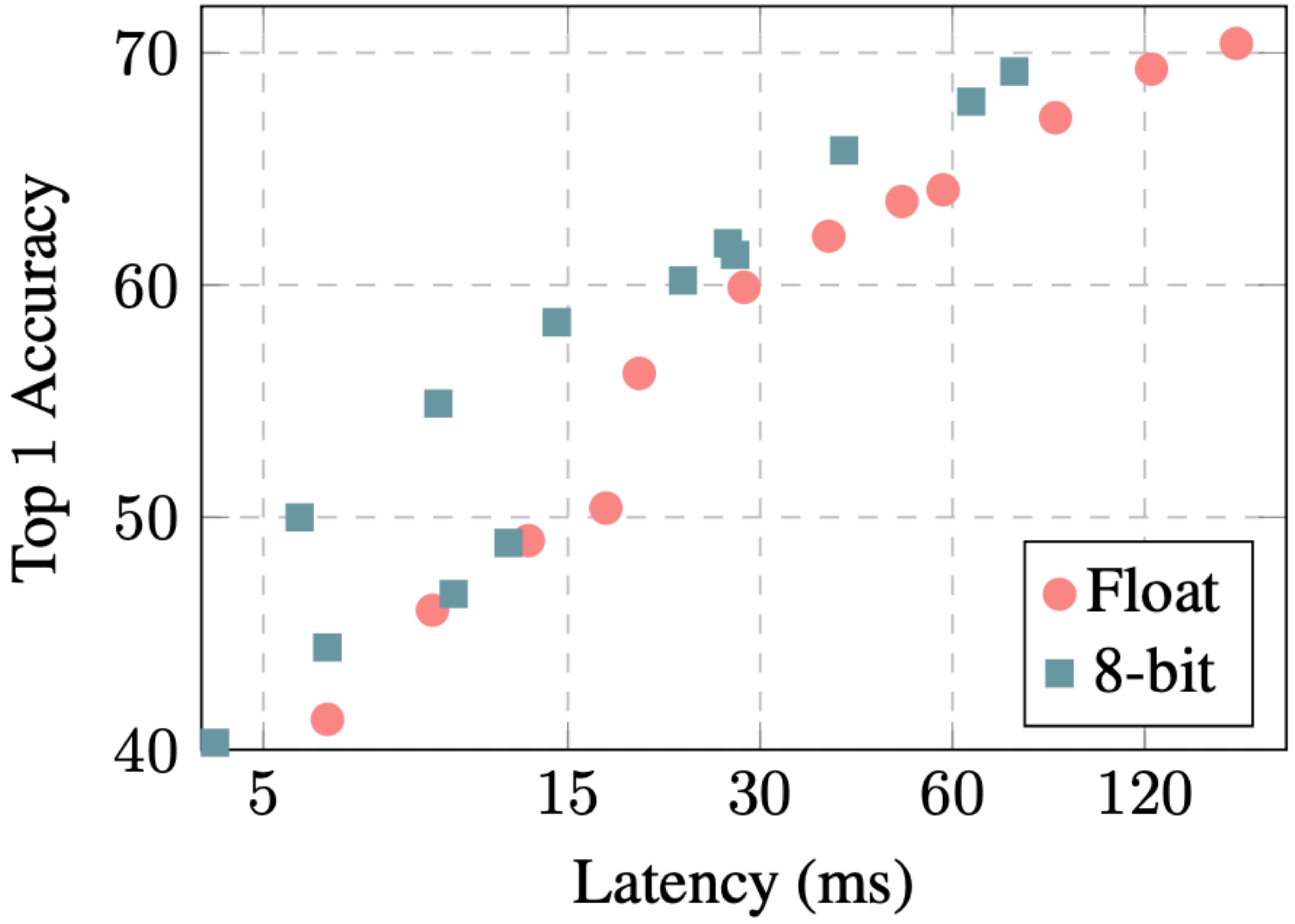


Figure 4.1: ImageNet classifier on Qualcomm Snapdragon 835 big cores: Latency-vs-accuracy tradeoff of floating-point and integer-only MobileNets.

Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," CVPR 2018

Advanced ideas for PTQ

Agenda

- In most PTQ algorithms, we adopt more ideas:
    - Finer granularity
    - Weight rescaling
    - Activation clipping
    - Adaptive rounding

Granularity

- **Motivation.** Weight ranges are quite dissimilar in different dimensions

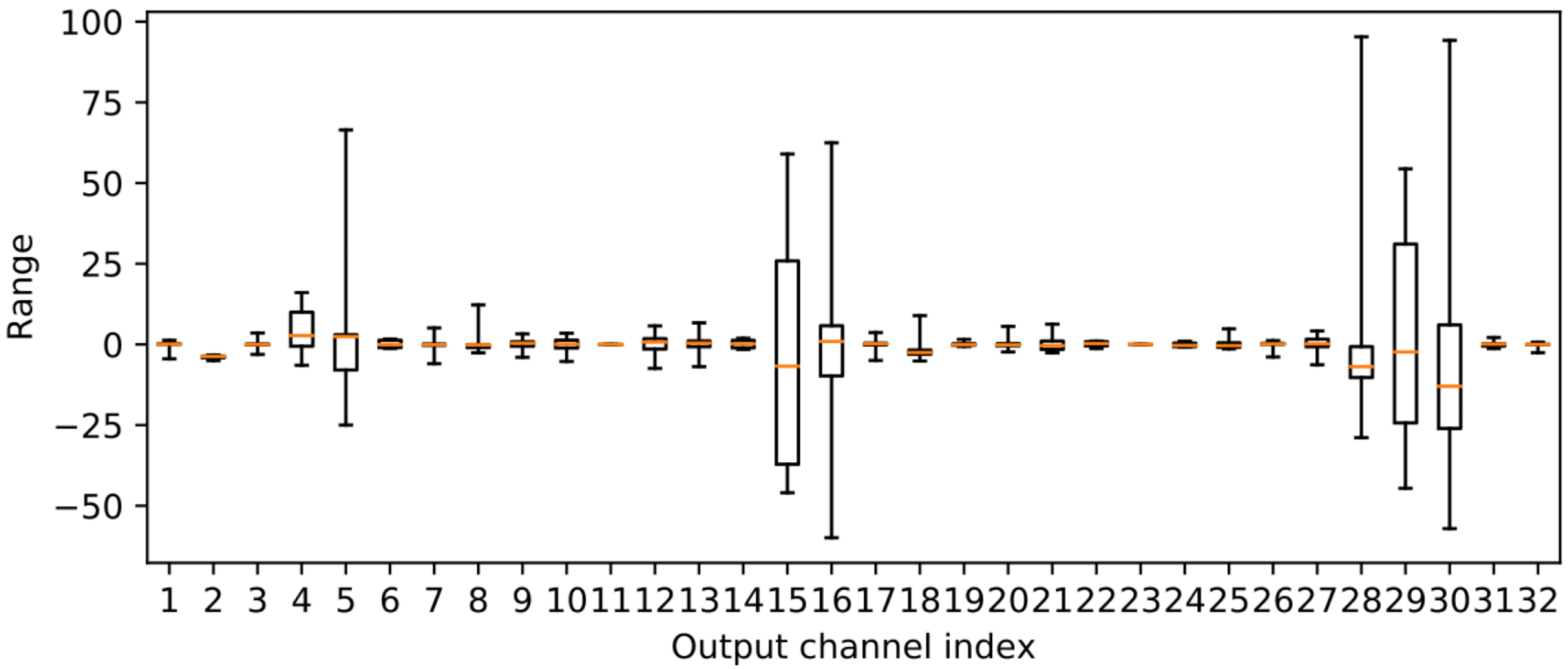


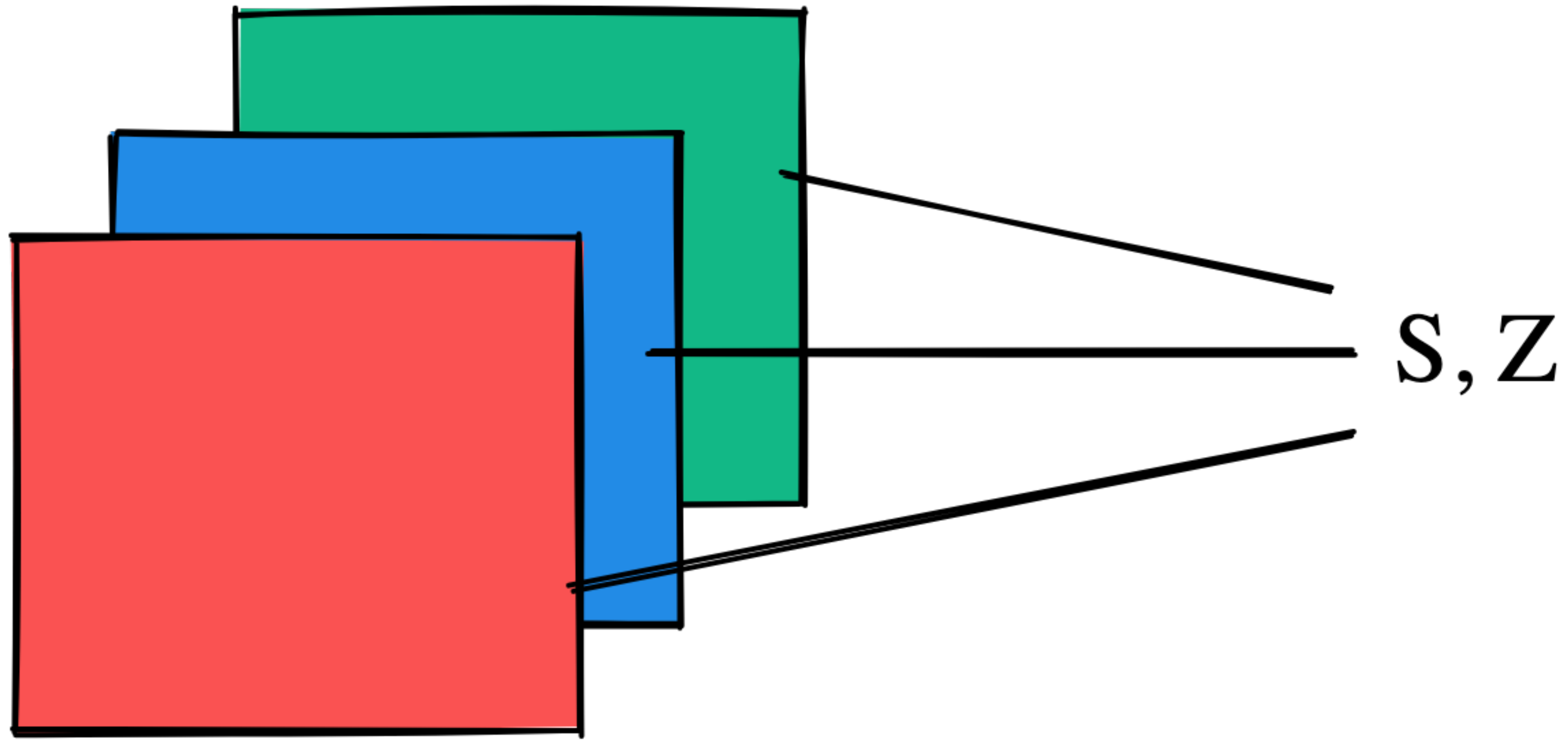
Figure 2. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. This layer exhibits strong differences between channel weight ranges.

Nagel et al., "Data-free quantization through weight equalization and bias correction," ICCV 2019

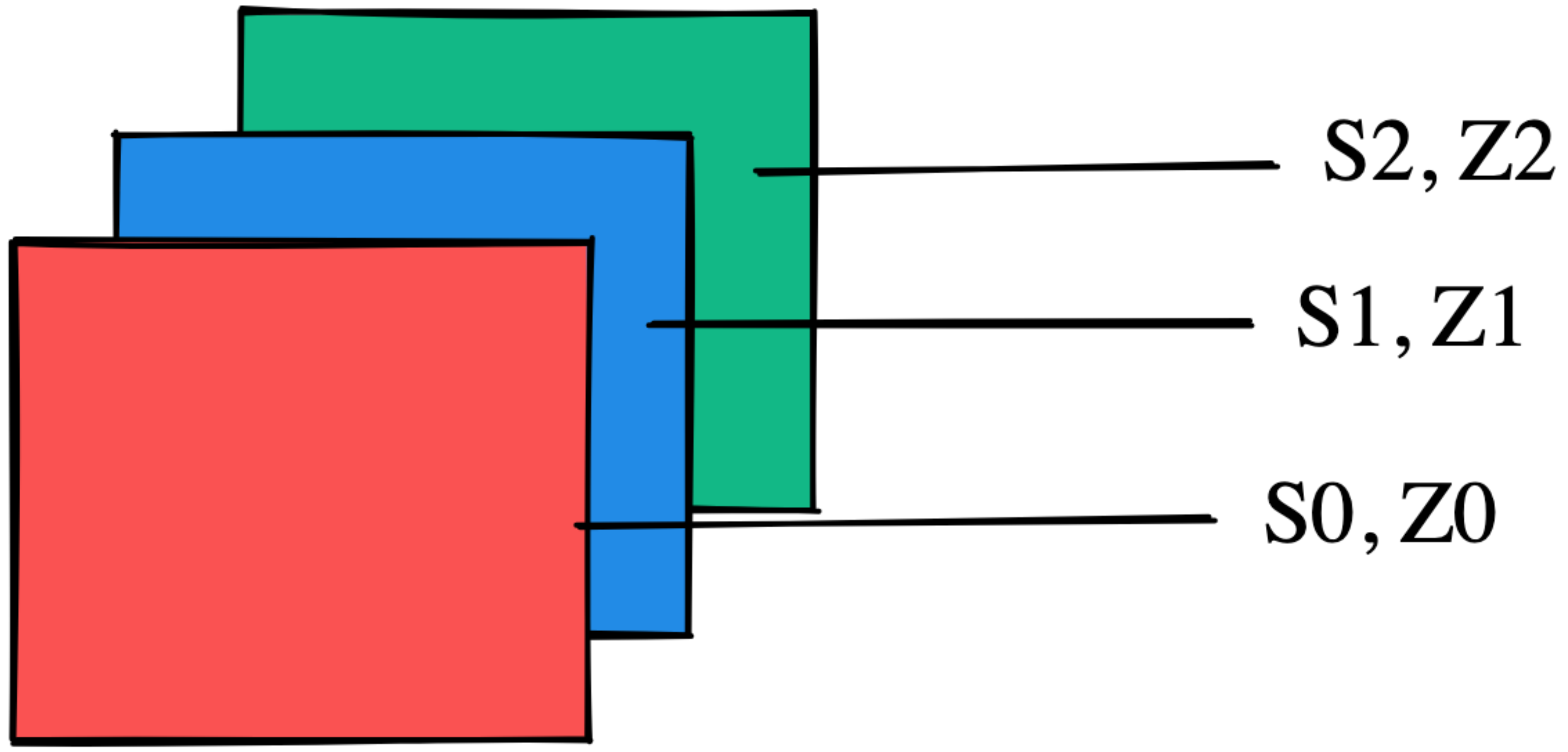
Granularity

- **Idea.** Apply different (s, z) for different group of weights (and/or activations)
  - <u>Example</u>. Per-Channel Quantization

per-tensor quantization



per-channel quantization



Nagel et al., "Data-free quantization through weight equalization and bias correction," ICCV 2019

Granularity

- <u>Example</u>. Per-Vector Quantization
    - Not much degradation in speed, if computation is done in the units of fixed-length vectors

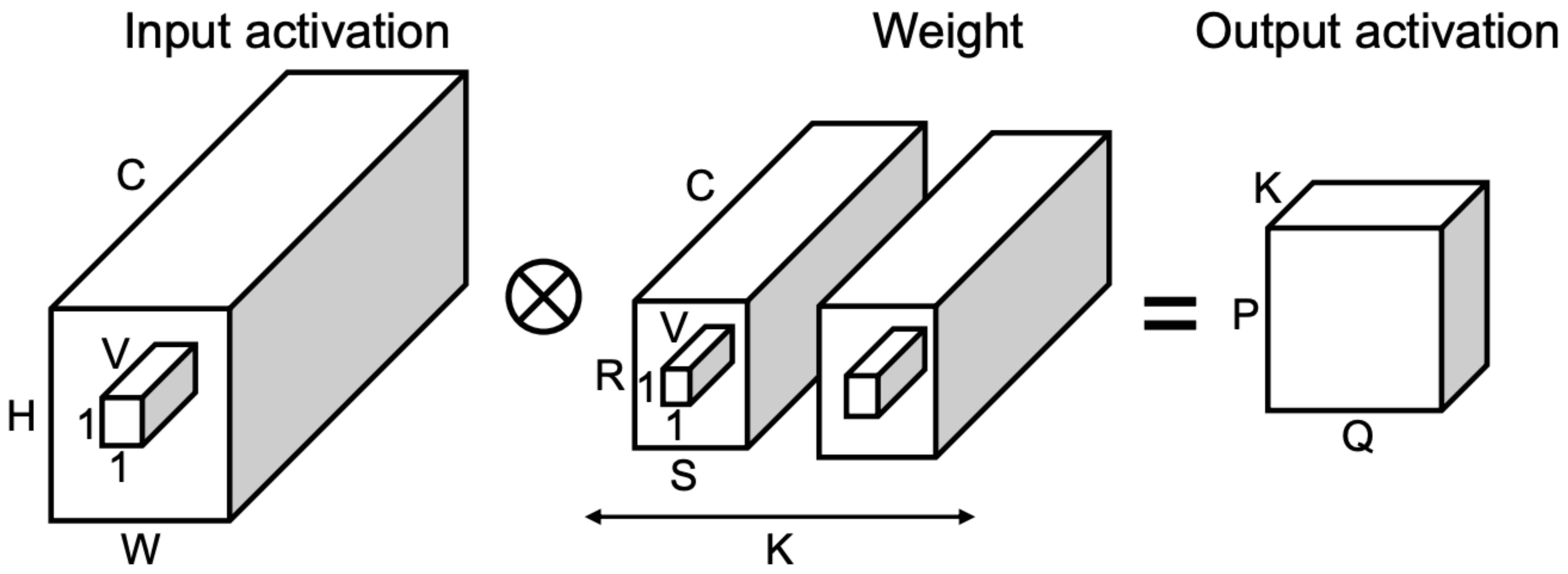


Figure 1. **Convolution** — Comparison between per-layer/per-output-channel scaling and per-vector scaling.

Dai et al., "VS-Quant: Per-Vector Scaled Quantization for Accurate Low-Precision Neural Network Inference," arXiv 2021

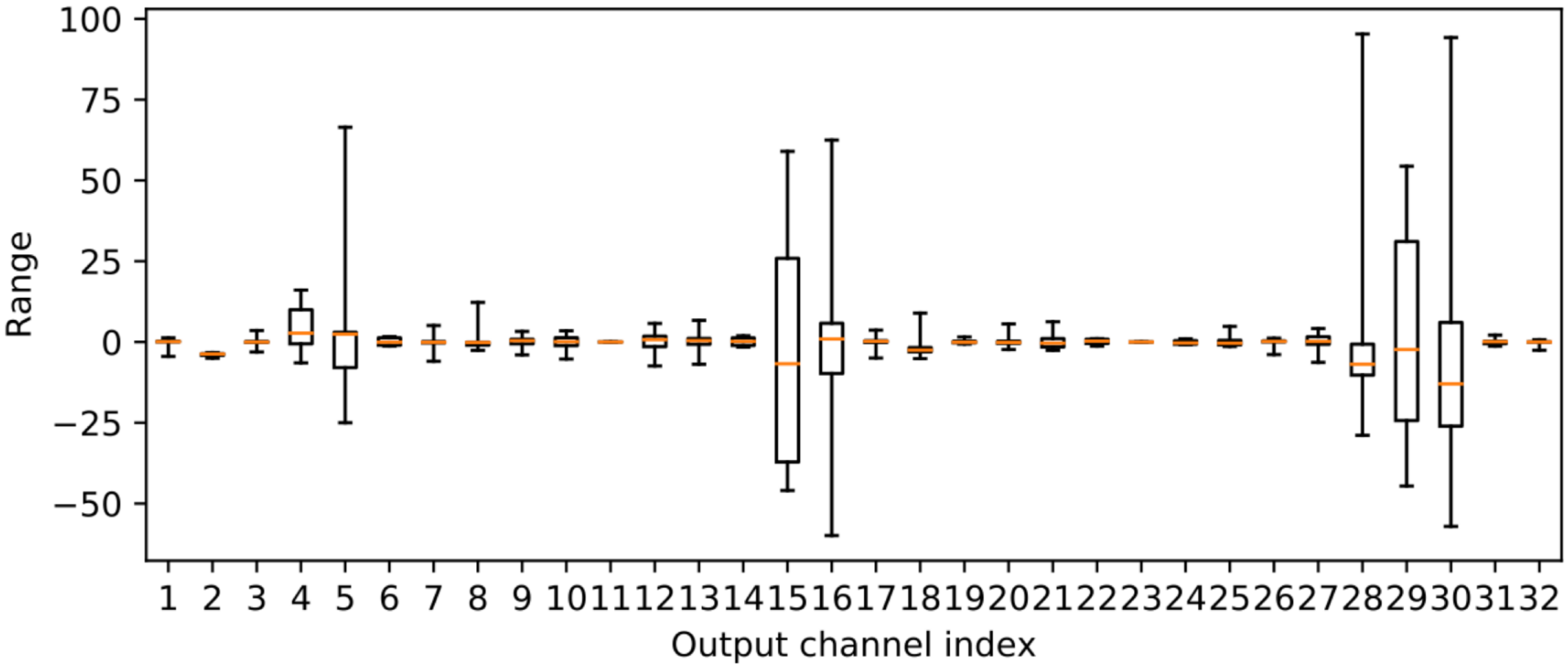
Granularity

- **Further readings**
  - Use two-stage scaling factors (link)
  - Sharing micro-exponents (link)

Dai et al., “VS-Quant: Per-Vector Scaled Quantization for Accurate Low-Precision Neural Network Inference,” arXiv 2021

Rescaling

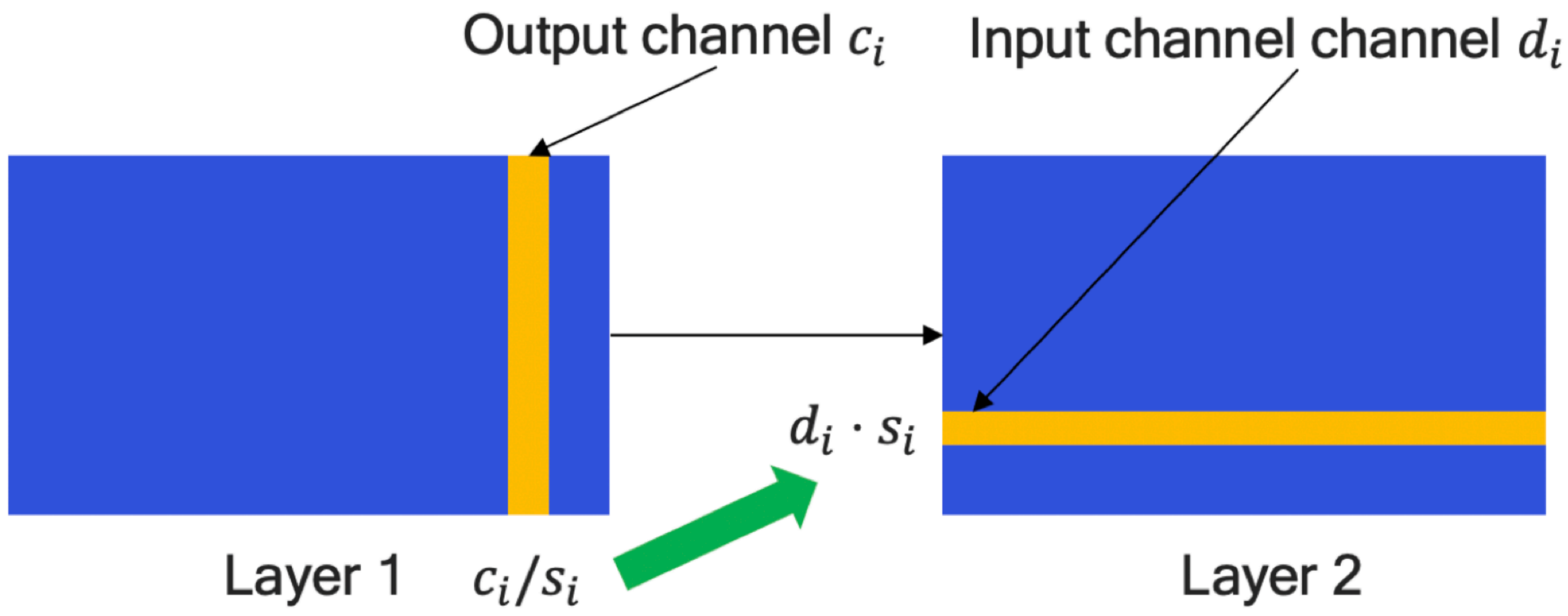
- **Idea.** Tackle the same problem, but use the positive homogeneity of ReLU.
  - i.e., $\sigma(cx) = c\sigma(x), \quad \forall c > 0$



Nagel et al., "Data-free quantization through weight equalization and bias correction," ICCV 2019

Rescaling

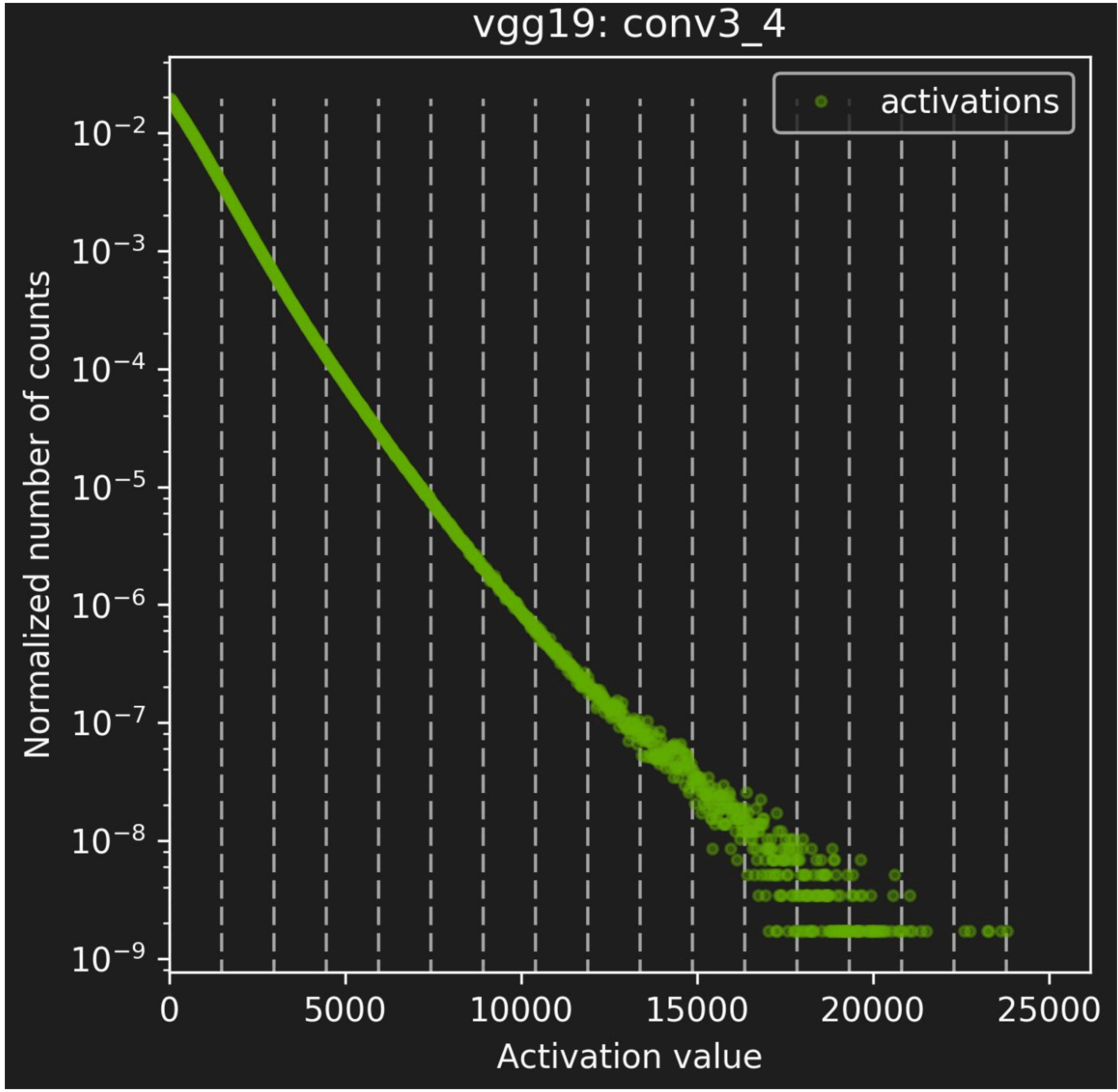
- Take a ReLU neural net
    - Multiply x100 to all weights in jth output channel of layer i
    - Multiply x0.01 to all weights in jth input channel of layer i+1
        - Identical function, with different weights
- Do this many times to match the weight range



Nagel et al., "Data-free quantization through weight equalization and bias correction," ICCV 2019

Clipping

- **Motivation.**
- For activation, $s_{\mathbf{X}}$ is determined via:
  - <u>During training.</u> Take an exponential moving average
  - <u>After training</u>. Use calibration batches
- However, many **outliers** appear



Mygasz, "8-bit inference with TensorRT," 2021

Clipping

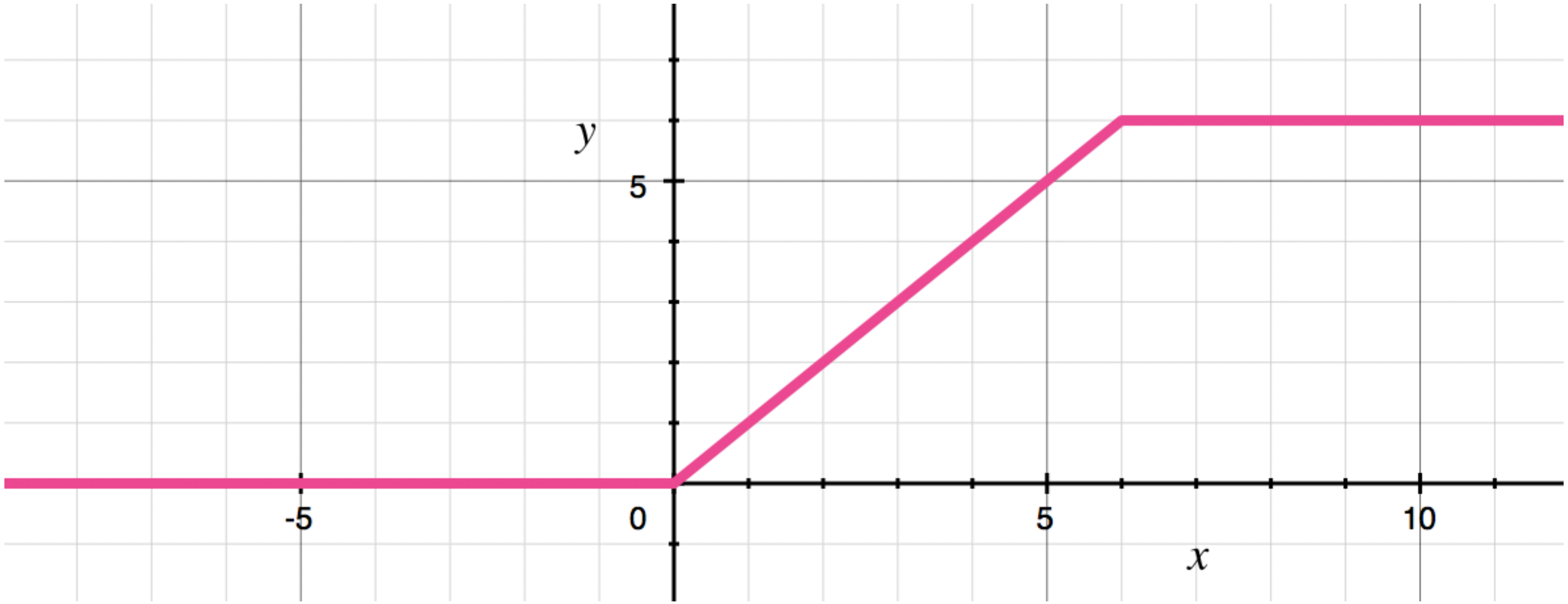
- **Idea.** Clip the activations

- **Question.** Where do we clip?
  - Explicit optimization.
    - Approximates w/ Gaussian/Laplace and minimize ℓ^2 (e.g., ACIQ)
    - Minimize ℓ^2 via Newton-Raphson (e.g., OCTAV)
    - Minimize the KL-divergence b/w quantized & reference dist (link)

Mygasz, "8-bit inference with TensorRT," 2021

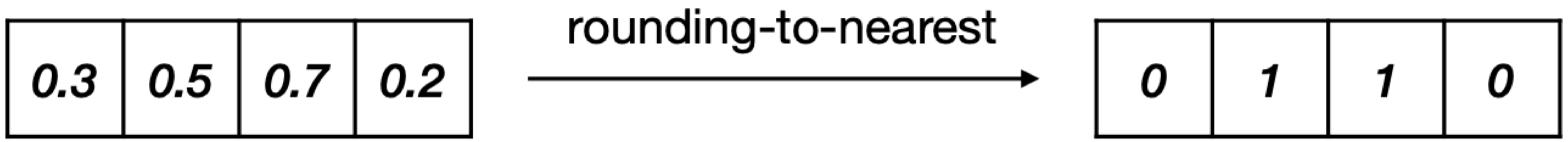
Clipping

- Use clipping activation.
    - ReLU6 (brainteaser: why 6?)
    - ReLU with learnable clipping range (e.g., PACT)



Rounding

- **Motivation.** Turns out that round-to-nearest (RTN) is suboptimal
  - In fact, stochastic rounding gives better options than RTN



Rounding scheme	Acc(%)
Nearest	52.29
Ceil	0.10
Floor	0.10
Stochastic	52.06±5.52
Stochastic (best)	63.06

Nagel et al., "Up or Down? Adaptive Rounding for Post-Training Quantization," ICML 2020

Rounding

- **Idea.** Round up-or-down in a way that minimizes the activation distortion
  - More concretely, AdaRound solves

$$\min_{\mathbf{V}: v_i \in [0,1]} \|\mathbf{W}\mathbf{X} - \tilde{\mathbf{W}}\mathbf{X}\|_F^2 + \lambda \cdot f_{\text{reg}}(\mathbf{V})$$

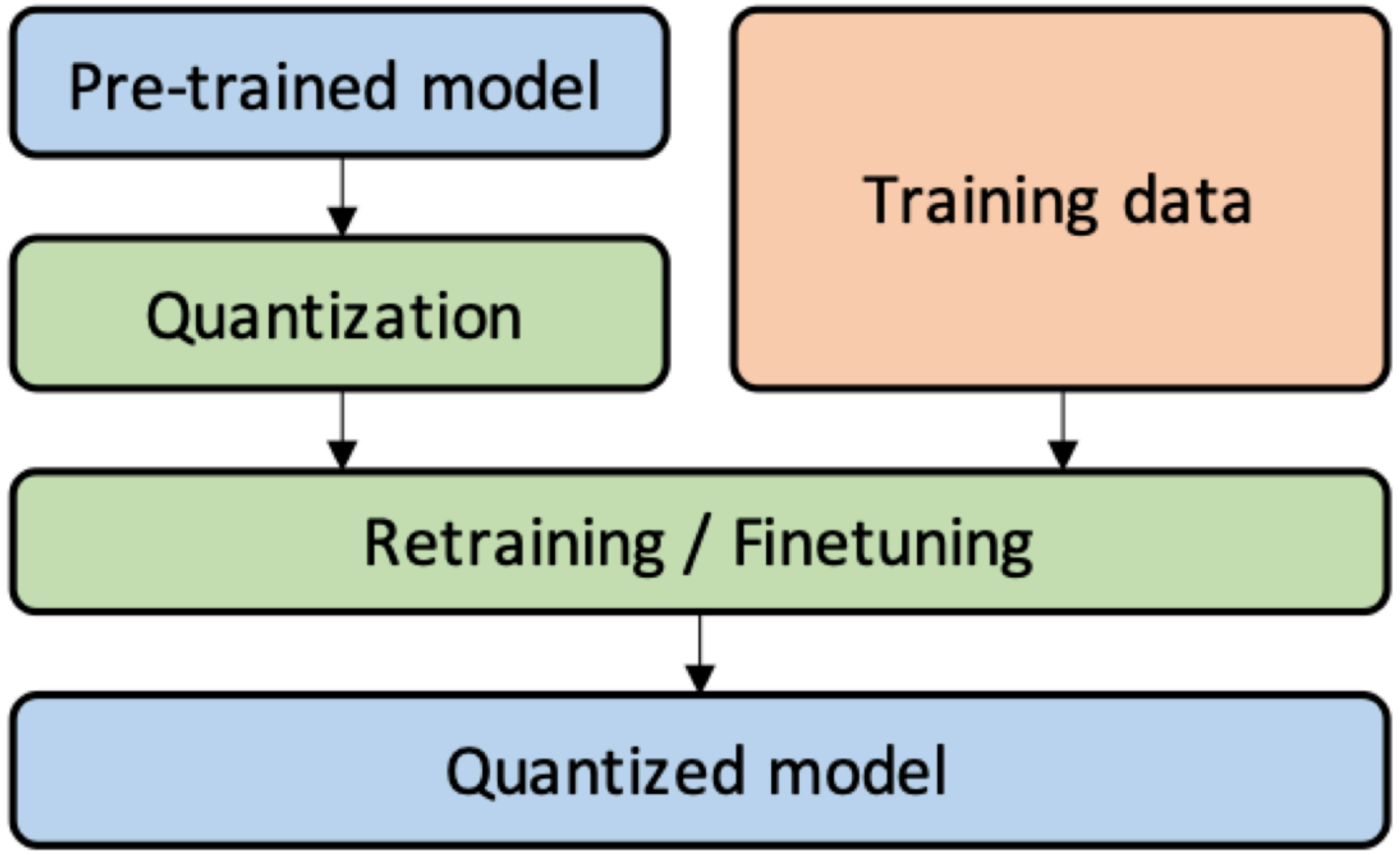
    - $\mathbf{V}$: weight shift for round up/down
    - $\tilde{\mathbf{W}} = s \cdot (\lfloor \mathbf{W}/s \rfloor + \mathbf{V})$: quantized weight
    - $f_{\text{reg}}(\mathbf{V}) = \sum_i 1 - |2\mathbf{V}_i - 1|^\beta$: binary-forcing regularizer
  - See also: AdaQuant, FlexRound

Nagel et al., "Up or Down? Adaptive Rounding for Post-Training Quantization," ICML 2020

QAT

Quantization-aware training

- After quantization, fine-tune the weight
    - Recovers much of the lost accuracy

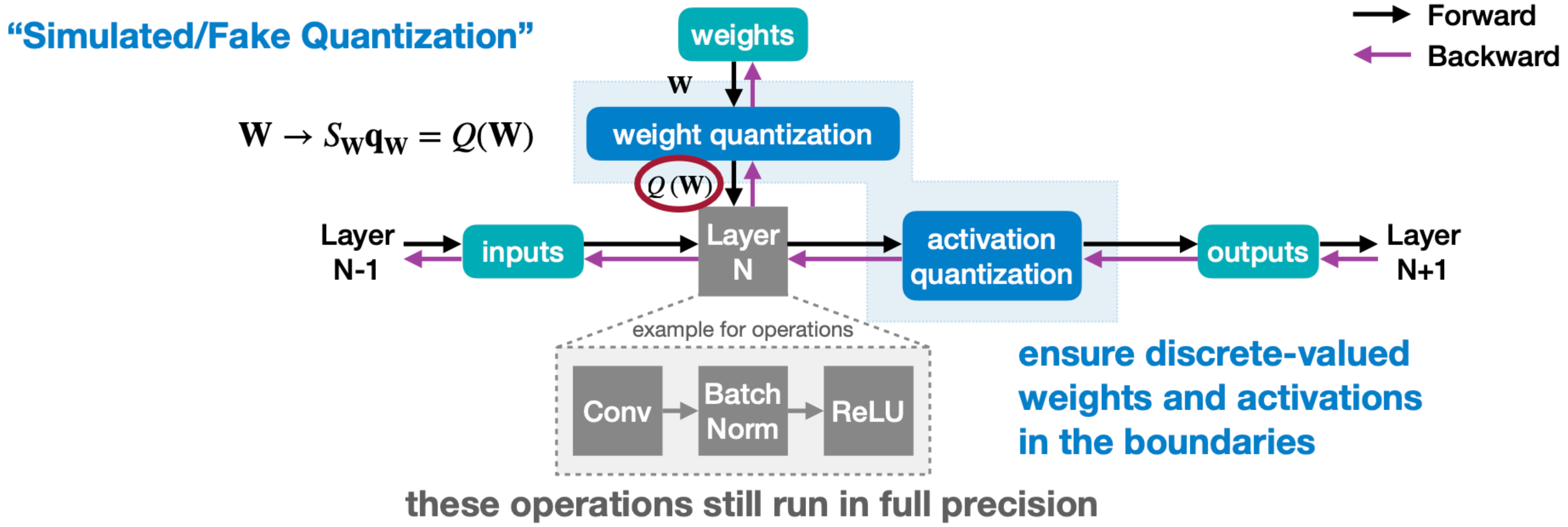


Neural Network	Floating-Point	Post-Training Quantization		Quantization-Aware Training	
		Asymmetric	Symmetric	Asymmetric	Symmetric
		Per-Tensor	Per-Channel	Per-Tensor	Per-Channel
MobileNetV1	70.9%	0.1%	59.1%	70.0%	70.7%
MobileNetV2	71.9%	0.1%	69.8%	70.9%	71.1%
NASNet-Mobile	74.9%	72.2%	72.1%	73.0%	73.0%

Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," arXiv 2018

Quantization-aware training

- (1) Keep full-precision weight, and simulate quantization at forward
- (2) Compute gradients in full-precision
- (3) Update full-precision weights



Quantization–aware training

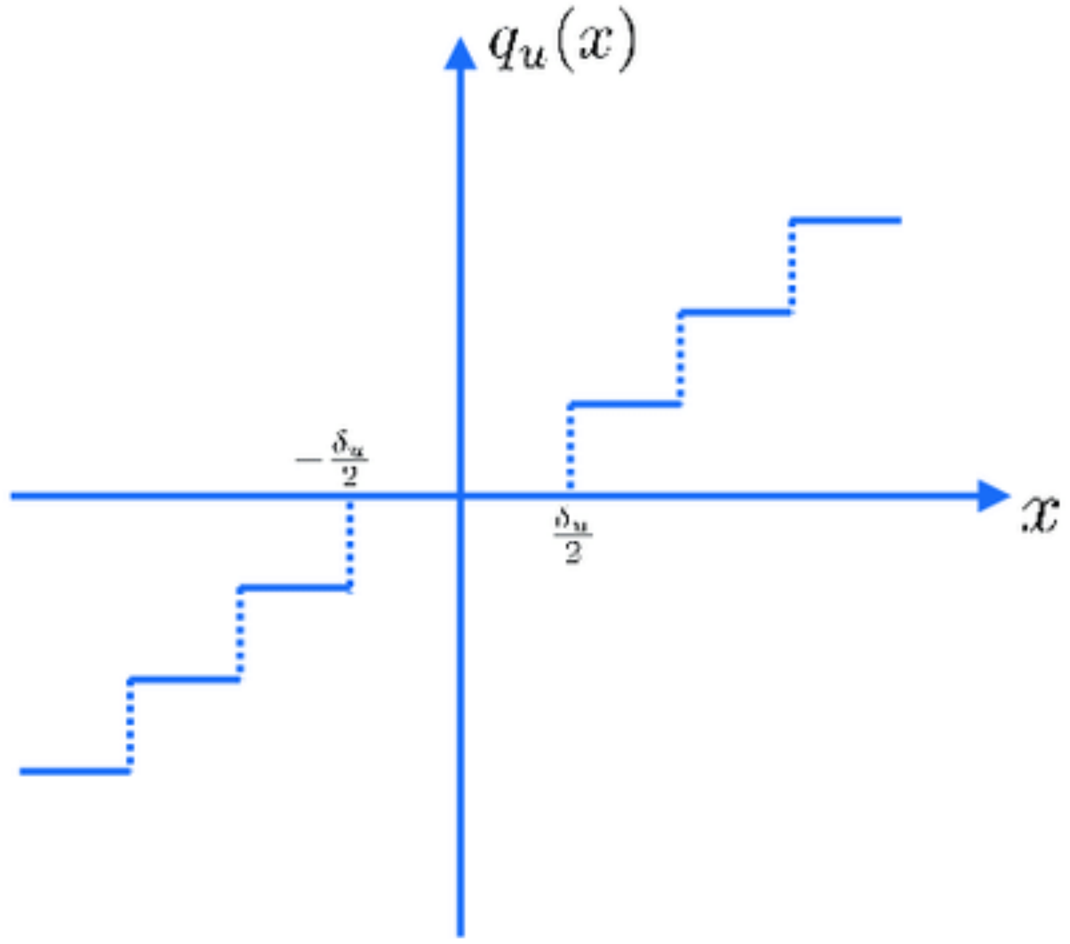
- Again, we are solving $\min_{\mathbf{w}} \hat{L}(q(\mathbf{w}))$
- **Challenge.** Computing gradients through discretizing operation $q(\,\cdot\,)$
    - <u>Example</u>. Consider a linear regression

$$\min_{\mathbf{w}} (y - q(\mathbf{w})^\top \mathbf{x})^2$$

- The gradient is

$$2(q(\mathbf{w})^\top \mathbf{x} - y) \cdot \mathbf{x}^\top \nabla_{\mathbf{w}} q(\mathbf{w})$$

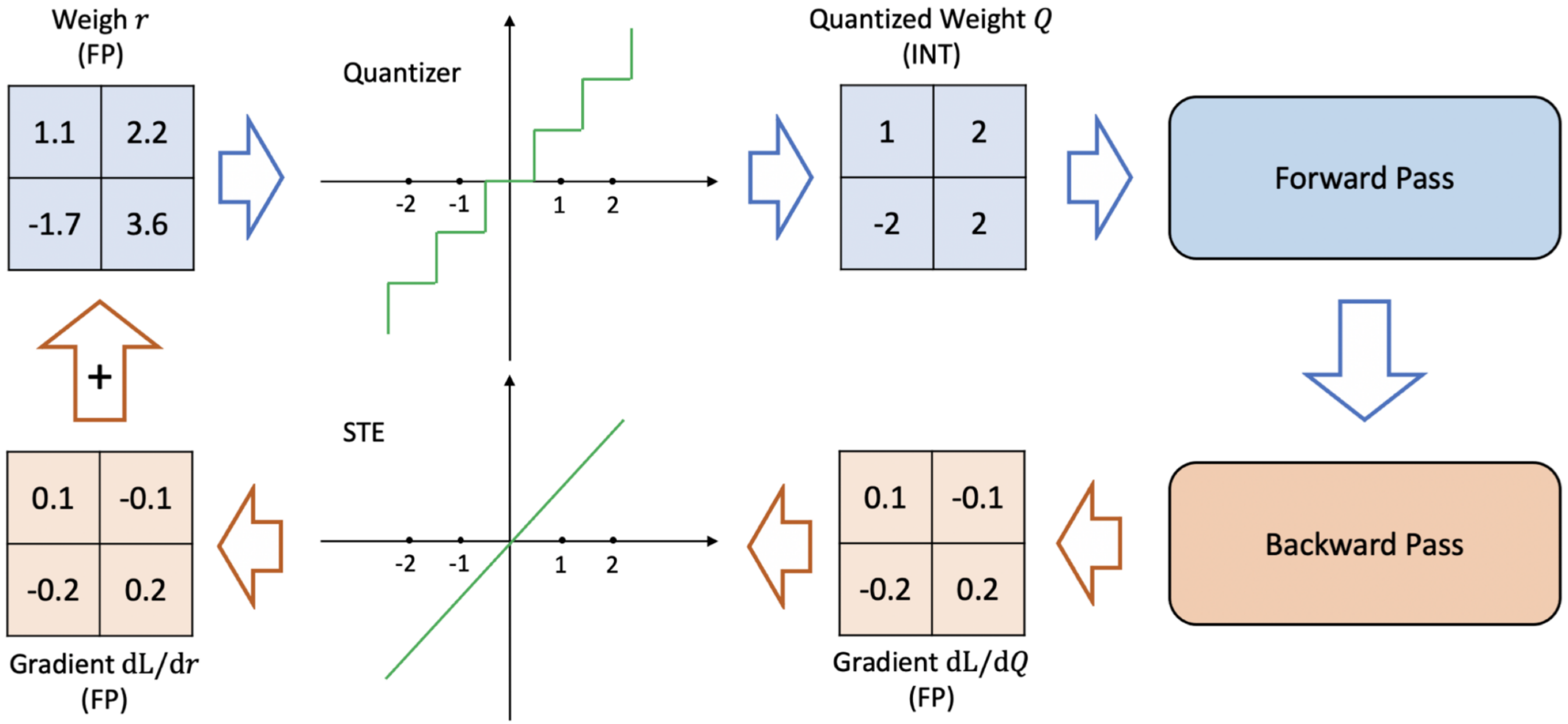
- The red term is always zero-or-infinity!



Straight-through estimator

- Again, we use STE
  - i.e., ignore quantization during backward

$$2(q(\mathbf{w})^\top \mathbf{x} - y) \cdot \mathbf{x}^\top \nabla_{\mathbf{w}} q(\mathbf{w}) \quad \Rightarrow \quad 2(q(\mathbf{w})^\top \mathbf{x} - y) \cdot \mathbf{x}^\top \nabla_{\mathbf{w}} (\mathbf{w})$$



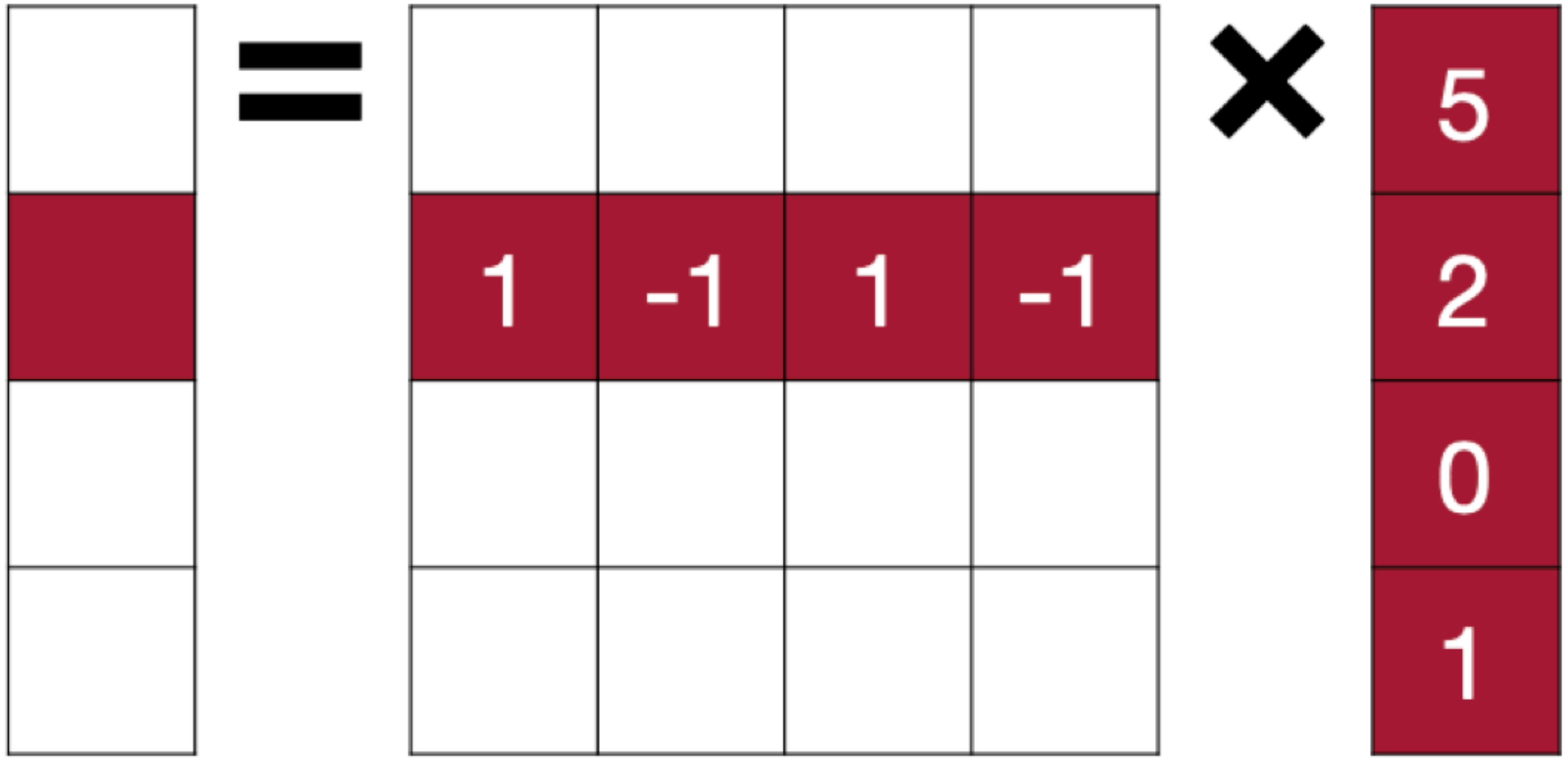
Further reading

- Some theoretical arguments suggest that using (clipped) ReLU, instead of identity function, is a better choice for STE
    - https://arxiv.org/abs/1903.05662

Other topics

Binary nets

- If all weights are binary, then we need no multiplications

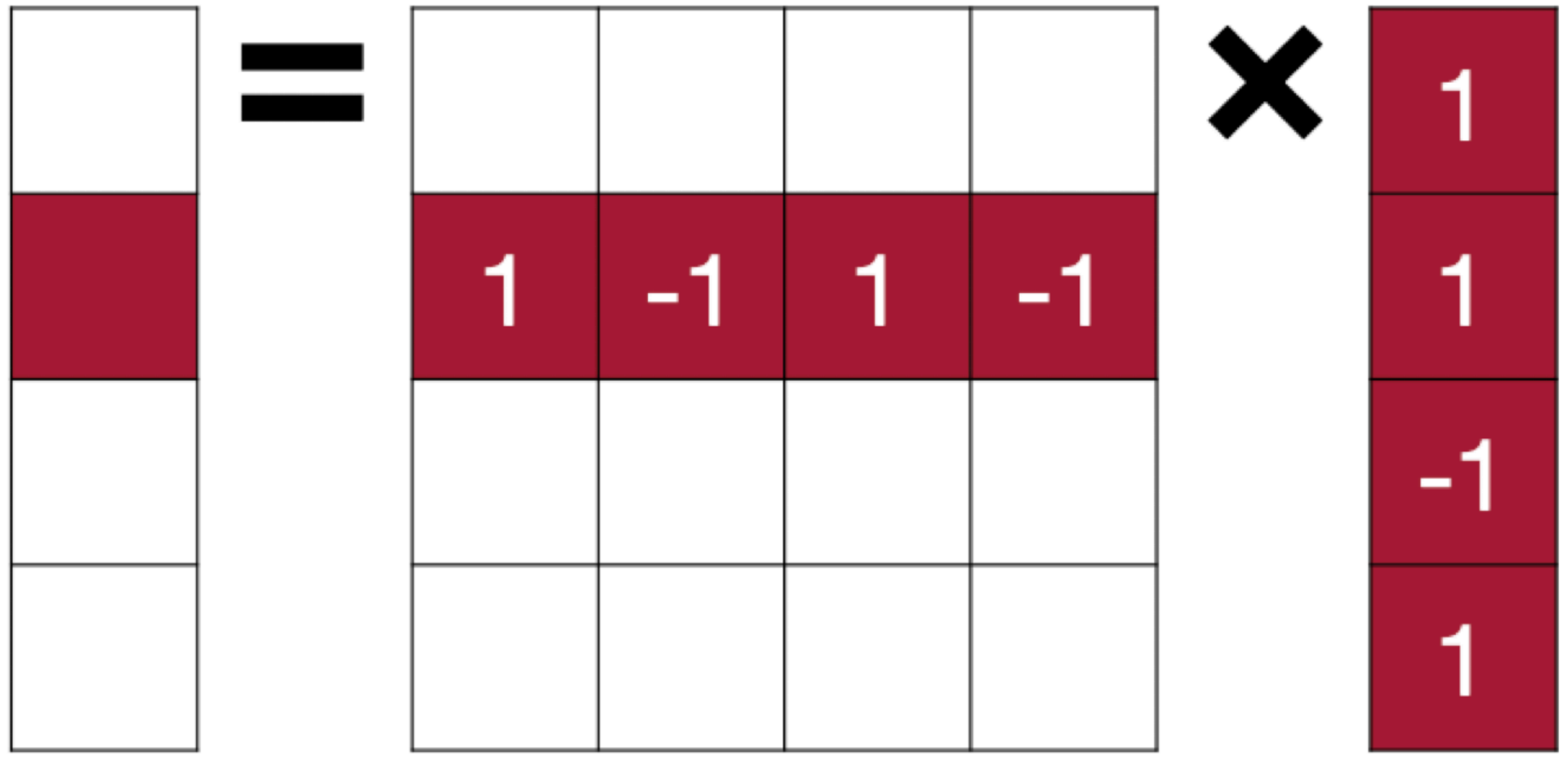


$$y = 5 + (-2) + 0 + (-1)$$

Coubarieaux et al., "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," NeurIPS 2015

Binary nets

- If all activations are binary as well, then we only need XNOR + Counting



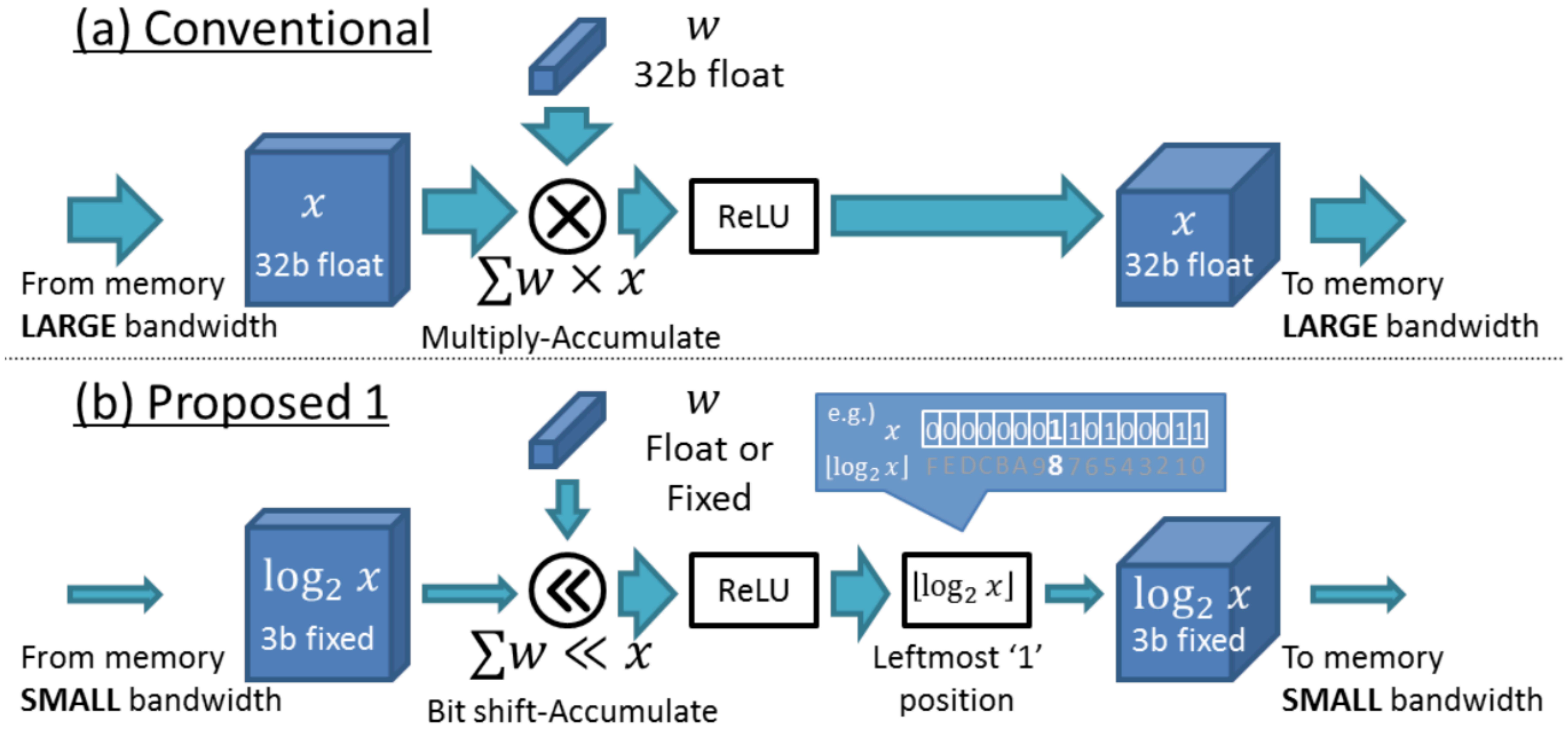
W	X	Y=WX
1	1	1
1	-1	-1
-1	-1	1
-1	1	-1

bw	bx	XNOR(bw, bx)
1	1	1
1	0	0
0	0	1
0	1	0

Rastegari et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," ECCV 2016

LogQuant

- Use logarithmically quantized activations
  - Multiplications are simply shifting bits



Miyashita et al., "Convolutional Neural Networks using Logarithmic Data Representation," arXiv 2016

Next Class

- Knowledge distillation

That’s it for today 🙌