EECE695D: Efficient ML Systems

Quantization

(pt 2)

Last Class

Linear quantization is beneficial in both storage & compute # Nonlinear is usually beneficial only for storage

Idea. Given some weight $\mathbf{r}$, express the weight using the formula:

$$\mathbf{r} = S \cdot (\mathbf{q} - Z)$$

$$\mathbf{q} = \text{Clip}\left(\text{Round}\left(\frac{\mathbf{r}}{S}\right) + Z \right)$$

where $\mathbf{r}$ is in FP32 and $\mathbf{q}$ is in low-bit format, e.g., INT8.
Biases/Activations are still high-bit (e.g., INT32)

Symmetric Q. Using the symmetric quantization for weights saves computation:

$$\mathbf{q}_Y = \frac{S_W S_X}{S_Y} \left(\mathbf{q}_W \mathbf{q}_X + \mathbf{q}_{\text{bias}} \right) + Z_Y$$

Today

PTQ. Post-Training Quantization

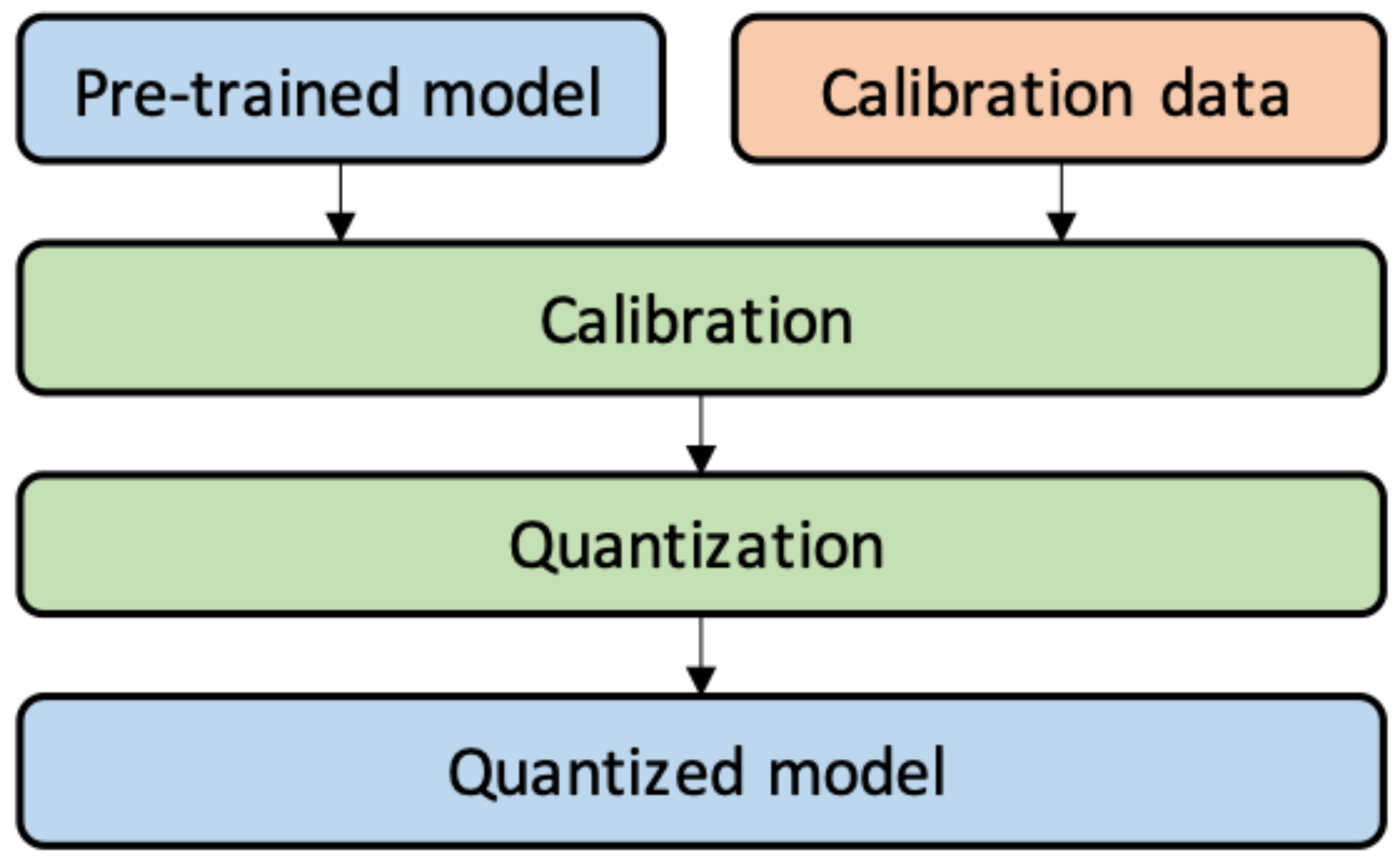
Given a well-trained model parameters, find the right S and Z.

Importantly, do it for both weight and activation!

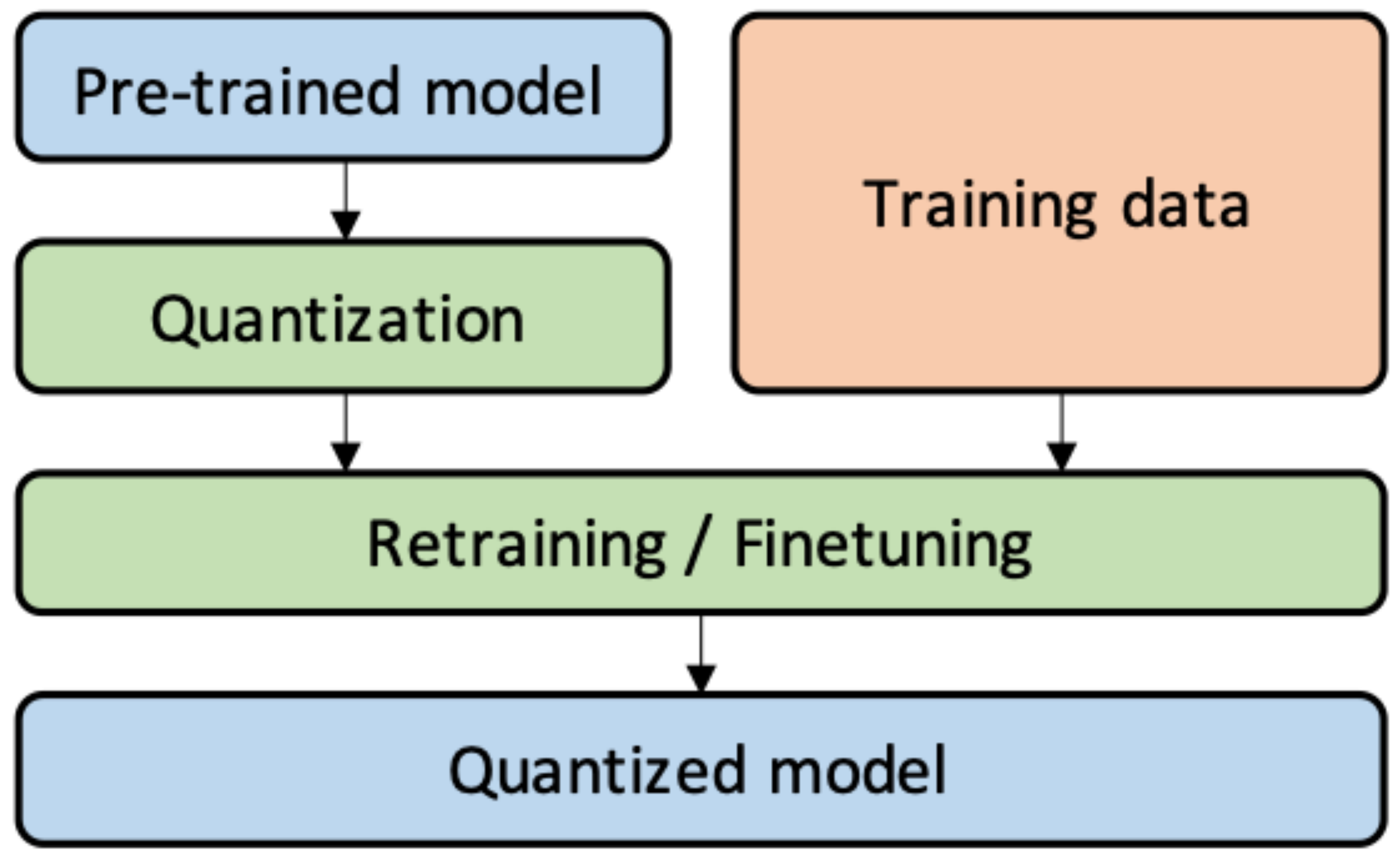
QAT. Quantization-Aware Training

Train a neural network so that the model can be quantized well.

How should we simulate the effect of quantization?



PTQ



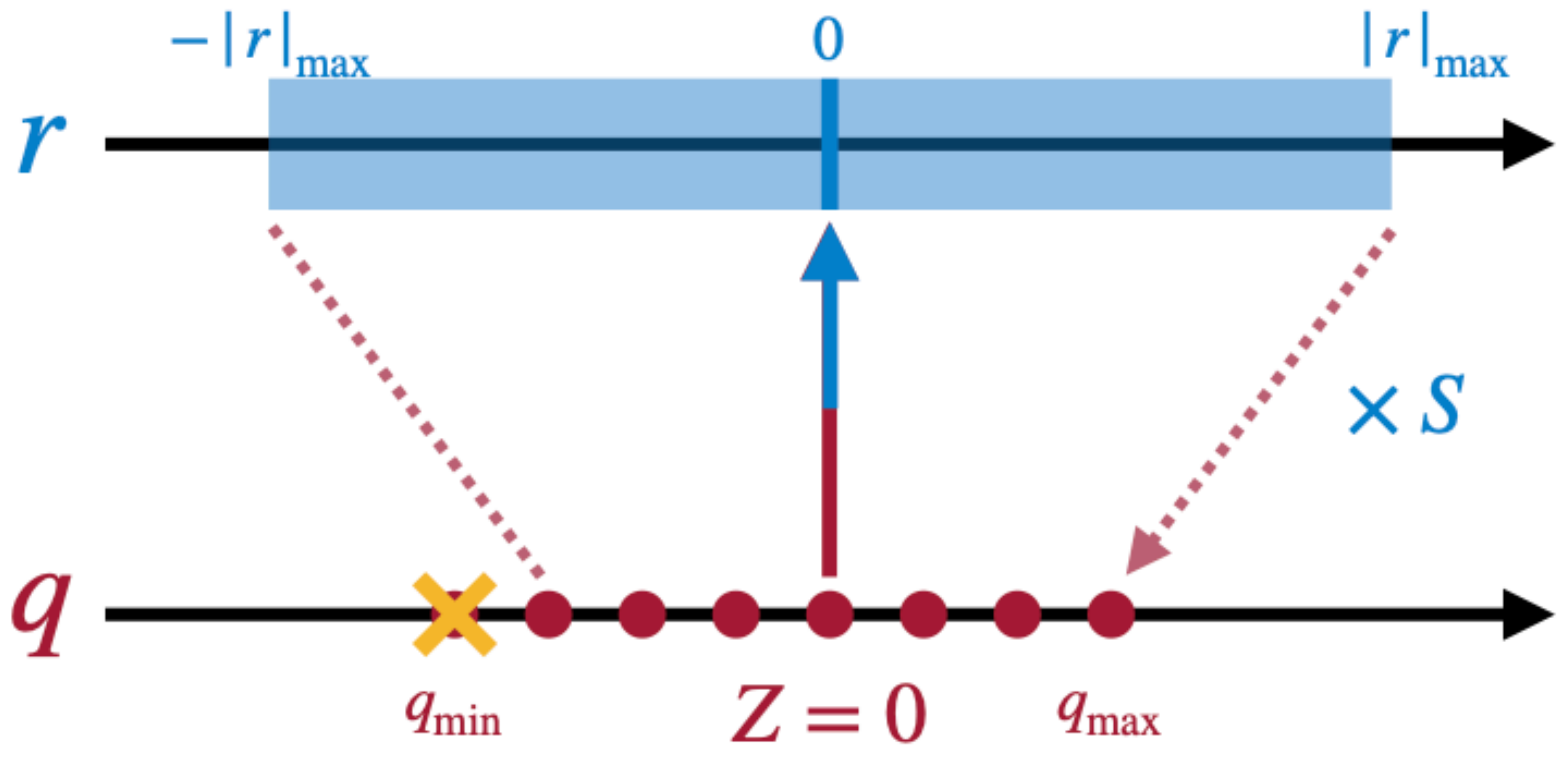
QAT

PTQ: Weight Quantization

Weights. Quantized symmetrically, mapping the range $[-\|\mathbf{W}\|_\infty, +\|\mathbf{W}\|_\infty]$ to low-bit integers.
Last time, we saw that we could set the scale parameter as

$$S = \frac{\|\mathbf{W}\|_\infty}{2^{N-1} - 1}$$

(Z determined automatically, as it is symmetric)



Problem. Channel of convolutional layers has disparate values of $||W||_\infty$
In other words, many of the channels will be zero-ed out.

Question. How to fix this?

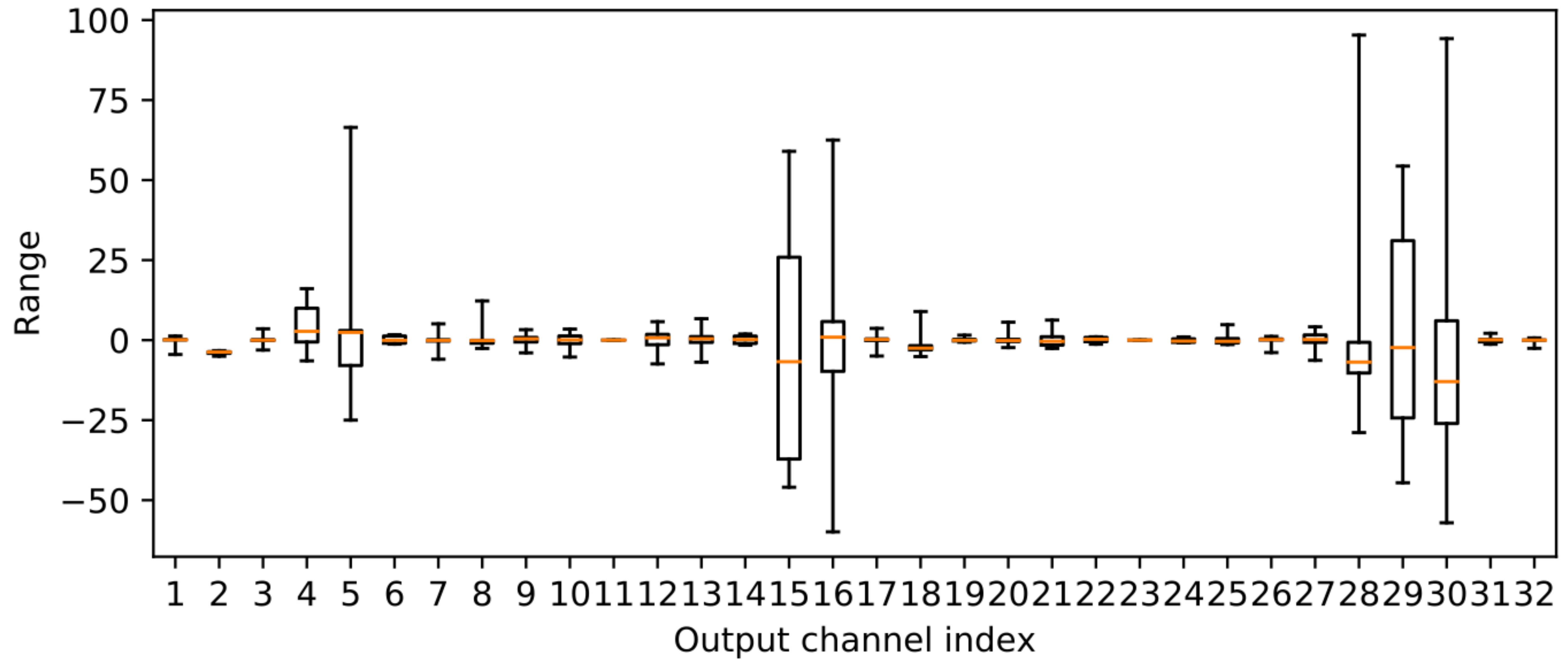


Figure 2. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. This layer exhibits strong differences between channel weight ranges.

Solution#1. Use a smaller granularity; have a separate S for each channel!

(a.k.a. per-channel quantization)

ic

oc				
	2.09	-0.98	1.48	0.09
	0.05	-0.14	-1.08	2.12
	-0.91	1.92	0	-1.03
	1.87	0	1.53	1.49

Per-Channel Quantization

$$|r|_{\max} = 2.09 \qquad S_0 = 2.09$$
$$|r|_{\max} = 2.12 \qquad S_1 = 2.12$$
$$|r|_{\max} = 1.92 \qquad S_2 = 1.92$$
$$|r|_{\max} = 1.87 \qquad S_3 = 1.87$$

Quantized

1	0	1	0
0	0	-1	1
0	1	0	-1
1	0	1	1

Reconstructed

2.09	0	2.09	0
0	0	-2.12	2.12
0	1.92	0	-1.92
1.87	0	1.87	1.87

$$\|\mathbf{W} - \mathbf{S} \odot \mathbf{q_W}\|_F = 2.08$$

$<$

Per-Tensor Quantization

$$|r|_{\max} = 2.12$$

$$S = \frac{|r|_{\max}}{q_{\max}} = \frac{2.12}{2^{2-1} - 1} = 2.12$$

Quantized

1	0	1	0
0	0	-1	1
0	1	0	0
1	0	1	1

Reconstructed

2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	0
2.12	0	2.12	2.12

$$\|\mathbf{W} - S\mathbf{q_W}\|_F = 2.28$$

Problem. Requires a special hardware support for a real speed-up.
Typically slower than per-tensor quantization.

Solution#2. Equalize the weights; use the positive homogeneity of ReLU-like activations!

Definition. We say that an activation function $\sigma(\,\cdot\,) : \mathbb{R} \to \mathbb{R}$ is positive homogeneous, when

$$\sigma(c \cdot x) = c \cdot \sigma(x) \qquad \forall c > 0.$$

Observation. If we scale the weight connected to the jth output channel at layer i by $1/s$, and scale the weight connected to the jth input channel at layer $i+1$ by s, the model is the same!

Idea. Do this scaling many times to match the values of $\|\mathbf{W}\|_\infty$ uniform over channels.

Question. If we have two layers, how should we do this simultaneously for both tensors?

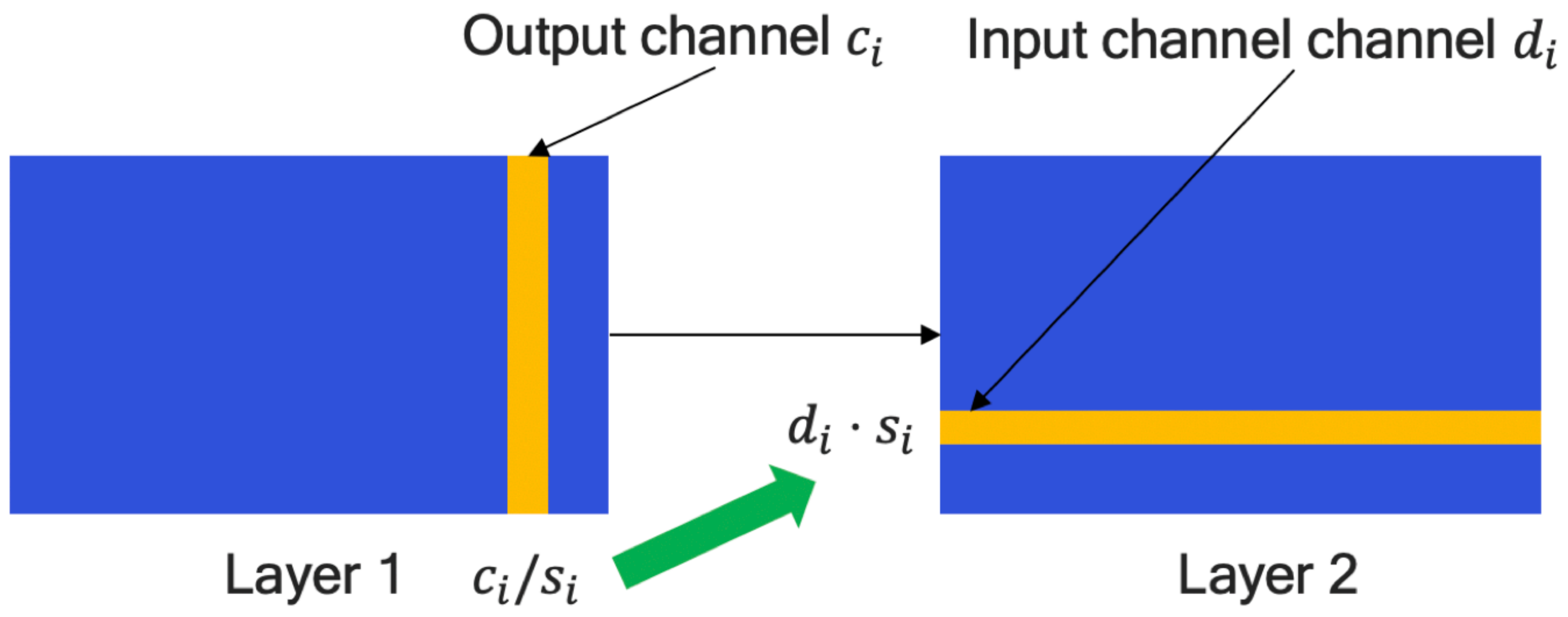


Figure 5. Illustration of the rescaling for a single channel. If scaling factor s_i scales c_i in layer 1; we can instead factor it out and multiply d_i in layer 2.

Question. If we have two layers, how should we do this simultaneously for both tensors?

Formulation. Define the "precision" of the ith channel of weight $\mathbf{W}$ as:

$$\hat{\mathbf{p}}_i = \frac{\|\mathbf{w}_i\|_\infty}{\|\mathbf{W}\|_\infty}.$$

Optimize the scaling factors of each channel to solve

$$\text{maximize} \qquad \sum_i \hat{\mathbf{p}}_i^{(1)} \hat{\mathbf{p}}_i^{(2)}$$

(abuse of notation; superscripts means layer, and subscript means output channel for layer 1 and input for layer 2)

Solution. For each channel i, divide the 1st layer weights on the ith output channel by

$$\mathbf{s}_i = \sqrt{\frac{\|\mathbf{w}_i^{(1)}\|_\infty}{\|\mathbf{w}_i^{(2)}\|_\infty}}$$

and multiply the same quantity for 2nd layer weights on ith input channel.

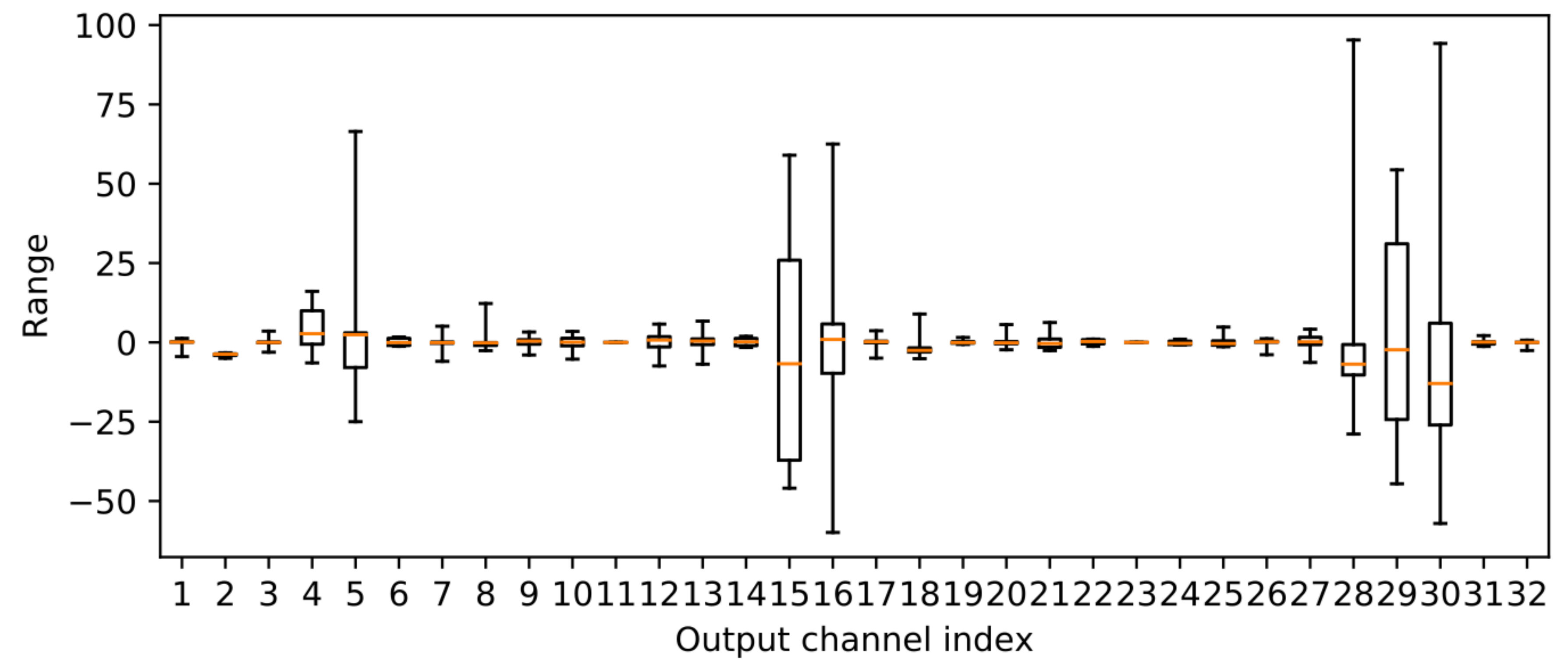


Figure 2. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. This layer exhibits strong differences between channel weight ranges.

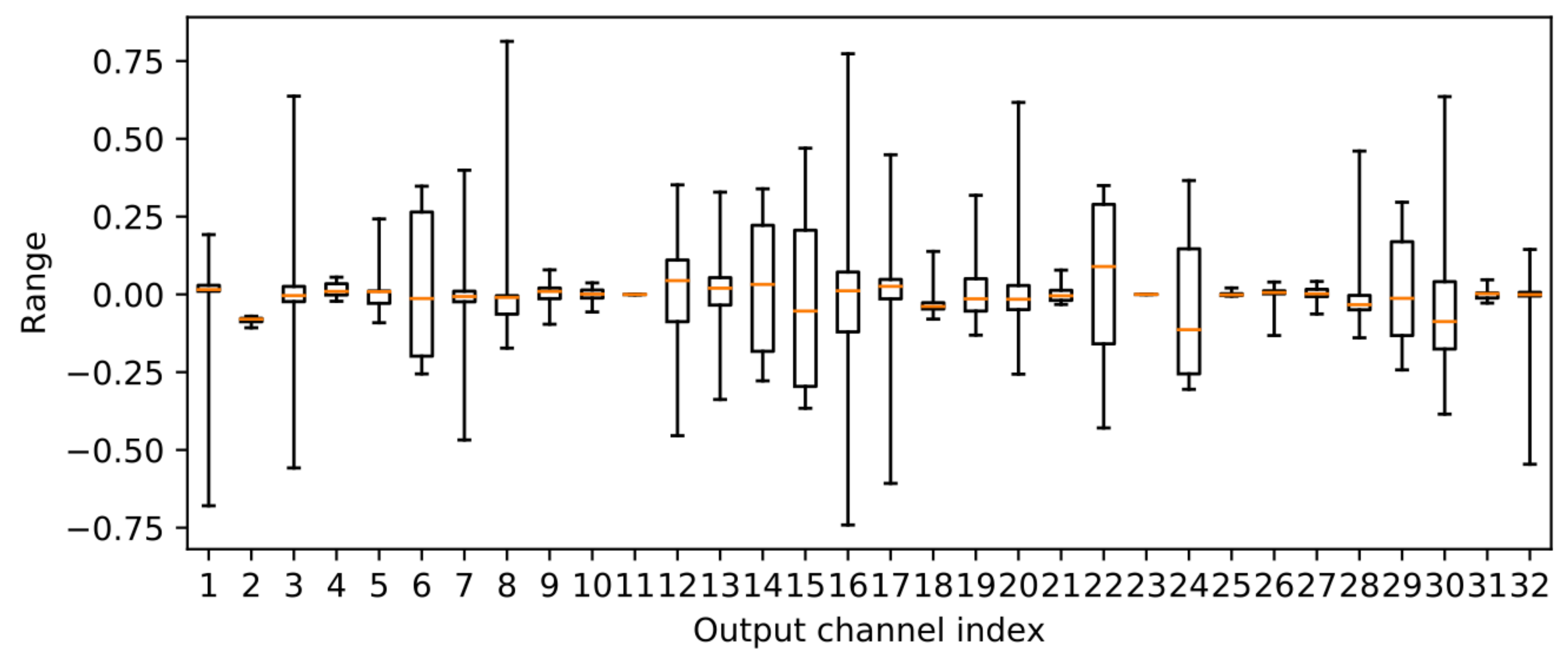


Figure 6. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2 after equalization. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. Most channels in this layer are now within similar ranges.

Note#1. There are more than two layers!
What people do is to do this sufficiently many times for adjacent connections until convergence.

Note#2. There are some theoretical results stating that things are not too pessimistic,
as SGD tend to automatically balance the input/output ratio for each neuron.
See Du et al., (2018) if interested.

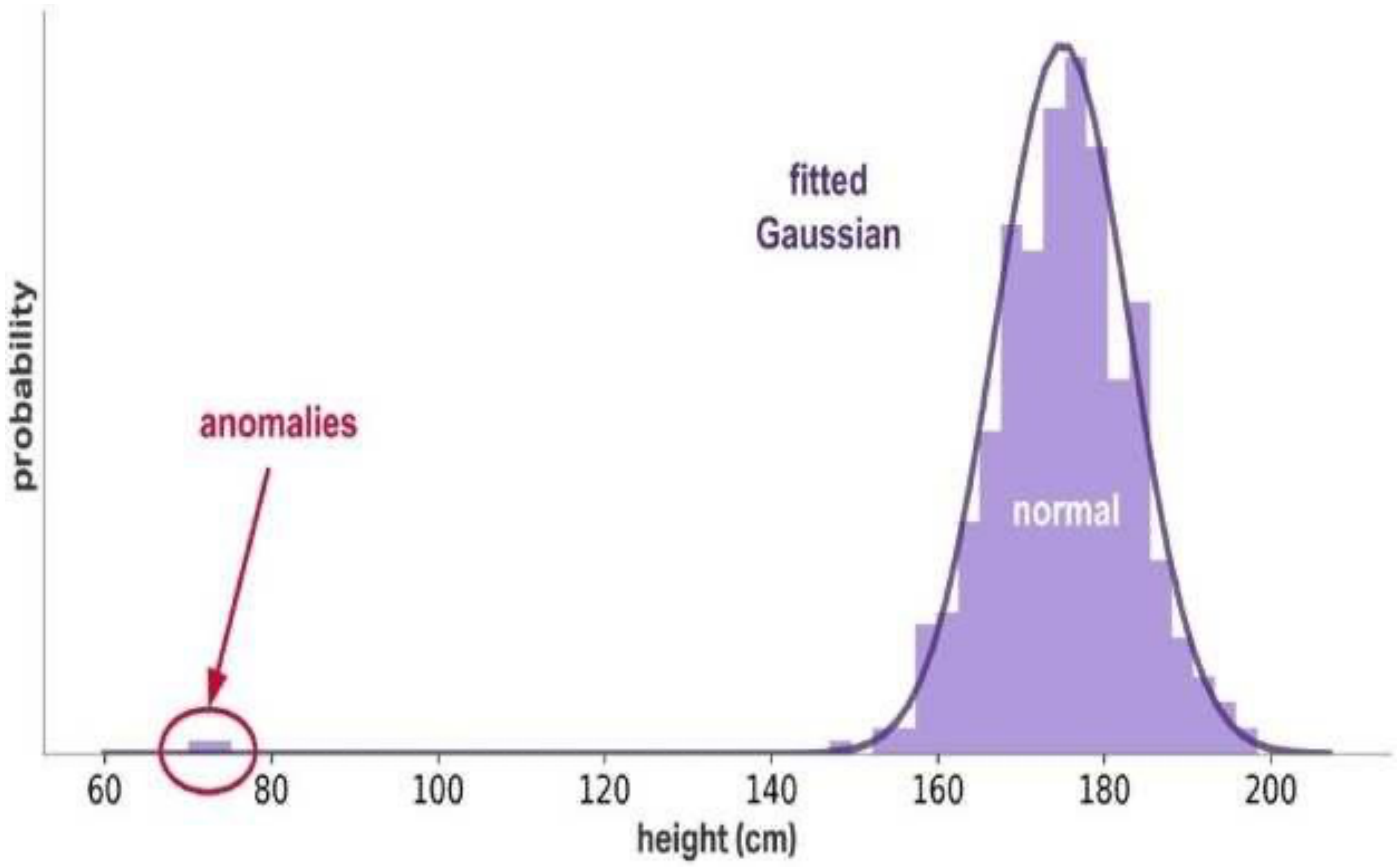
Theorem 2.1 (Balanced incoming and outgoing weights at every neuron). *For any $h \in [N-1]$ and $i \in [n_h]$, we have*

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\|\boldsymbol{W}^{(h)}[i,:]\|^2 - \|\boldsymbol{W}^{(h+1)}[:,i]\|^2\right) = 0. \tag{6}$$

Du et al., "Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced," NeurIPS 2018

PTQ: Activation Quantization

Activations. Depend on the data, so finding the $||\mathbf{A}||_\infty$ for tuning (S, Z) could...
(1) require some data statistics recorded, or fresh "calibration" batches
(2) be somewhat overly pessimistic (e.g., outliers).



Jacob et al., "Quantization and training of NNs for efficient integer-arithmetic-only inference," CVPR 2018

Banner et al., "Post-training 4-bit quantization of convolutional networks for rapid-deployment" NeurIPS 2019

Methods. Many methods, including…

(1) During the training, take an exponential moving average (Jacob et al., 2018)

$$x \leftarrow \lambda \cdot \|\mathbf{A}\|_\infty^{(\text{batch})} + (1-\lambda)\cdot x$$

(2) Use the fresh calibration batch to take max (Jacob et al., 2018)

$$\|\mathbf{A}\|_\infty^{(\text{batch})}$$

(3) Use the fresh calibration batch to find the MMSE solution (Banner et al., 2019)

$$\min \|\mathbf{A} - q(\mathbf{A})\|^2$$

(4) Info-theoretic methods; minimize the KL-divergence of activation distribution
(see "8-bit inference with TensorRT" by NVIDIA)

$$\min D_{\text{KL}}(\mu(\mathbf{A})\|\mu(\mathbf{q}(\mathbf{A})))$$

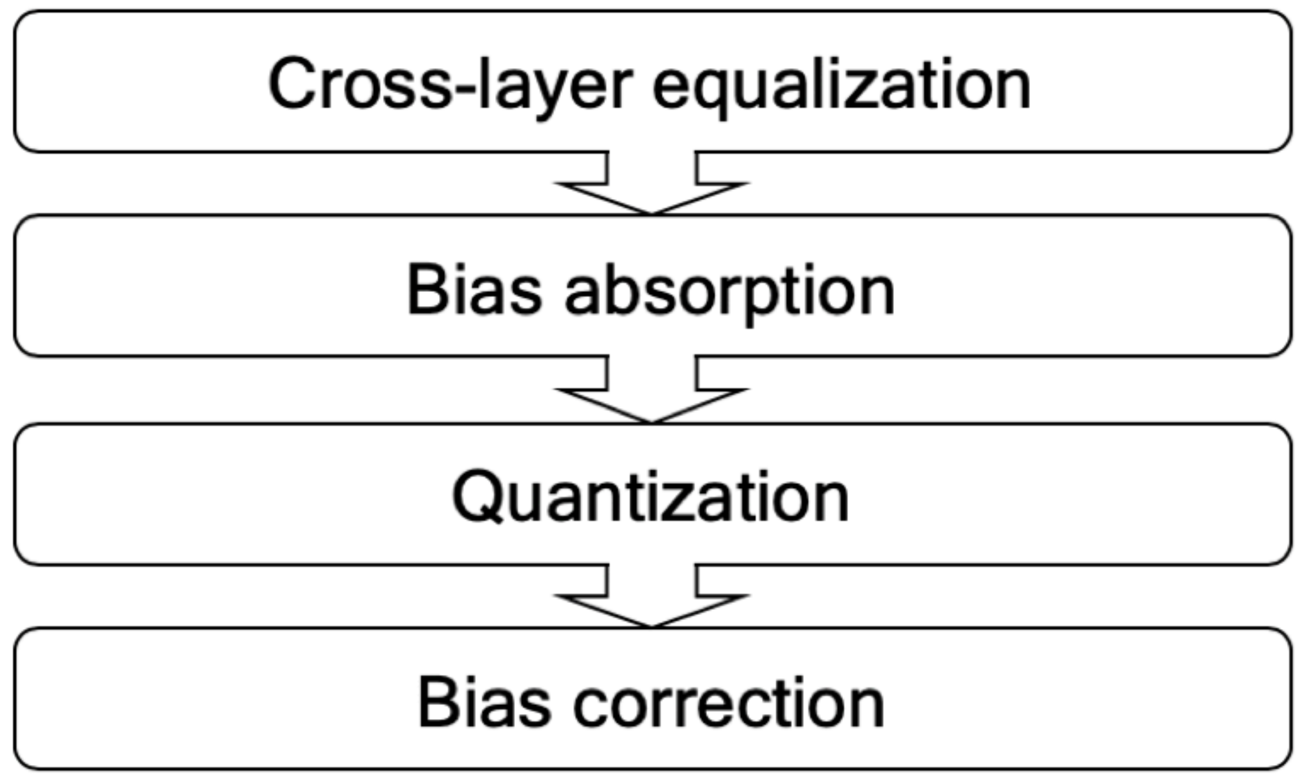


Figure 4. Flow diagram of the proposed DFQ algorithm.

Note. Per-channel issues should be resolved in activations; see "bias absorption" in Nagel et al. (2019).
Note. For activations, training with ReLU6 reduces much headache.

Jacob et al., "Quantization and training of NNs for efficient integer-arithmetic-only inference," CVPR 2018
Banner et al., "Post-training 4-bit quantization of convolutional networks for rapid-deployment" NeurIPS 2019
Nagel et al., "Data-free quantization through weight equalization and bias correction," ICCV 2019

QAT: Quantization-Aware Training

Idea. Train a full-precision model to optimize the "performance of its quantized version," by simulating the quantization during training.

$$\text{minimize} \qquad \mathbb{E}\ell(f(x; \mathbf{q}(\theta)), y)$$

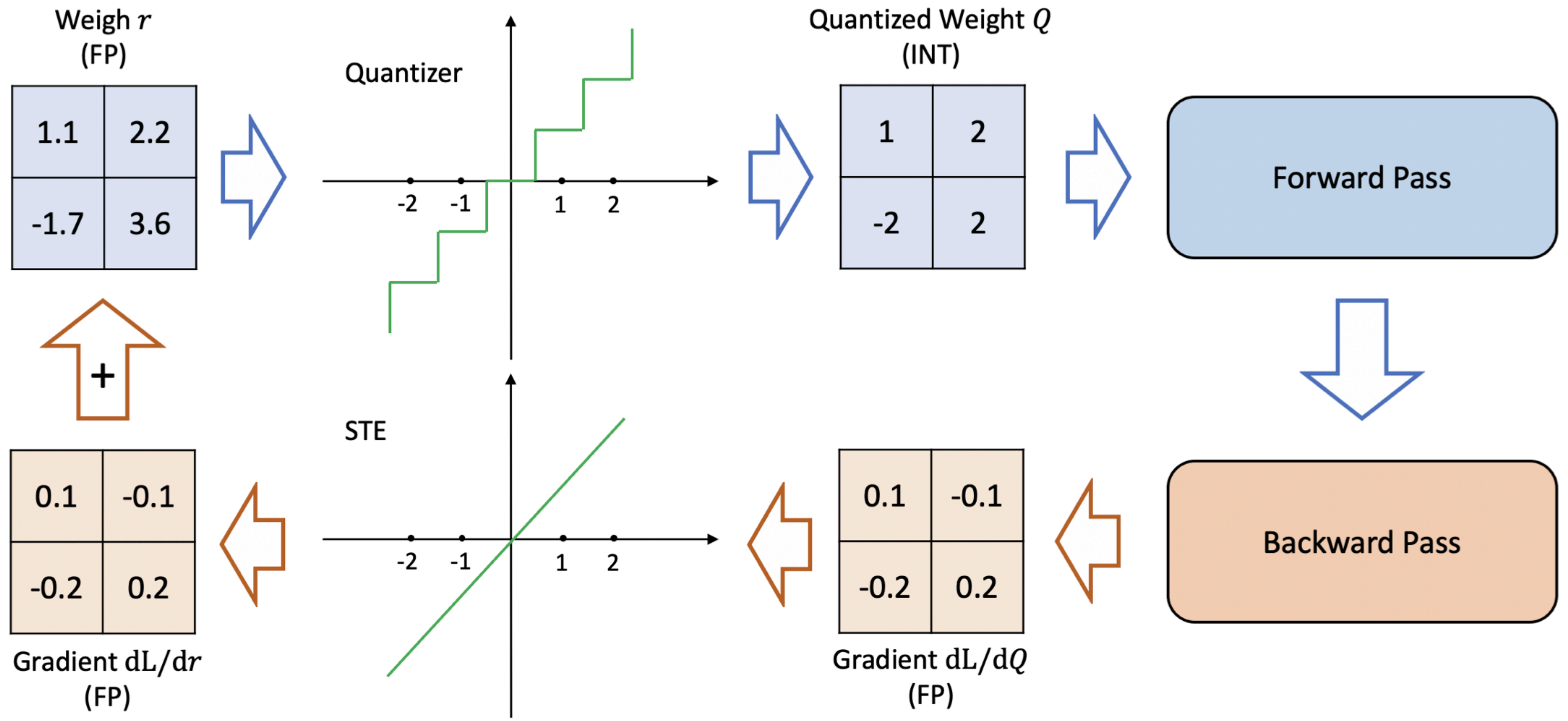
Challenge. How do we propagate gradient signals through the quantization operator?

Example. Let us consider a quantized linear regression $\min_{\mathbf{w}} (y - \mathbf{q}(\mathbf{w})^\top \mathbf{x})^2$.

Then, the gradient for $\mathbf{w}$ should be...

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

Solution. Simply ignore the quantization during the backpropagation.
(a.k.a. Straight-Through Estimator)



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

Note. Replacing quantization with identity… theoretically more valid ones are replacing with (clipped) ReLU.

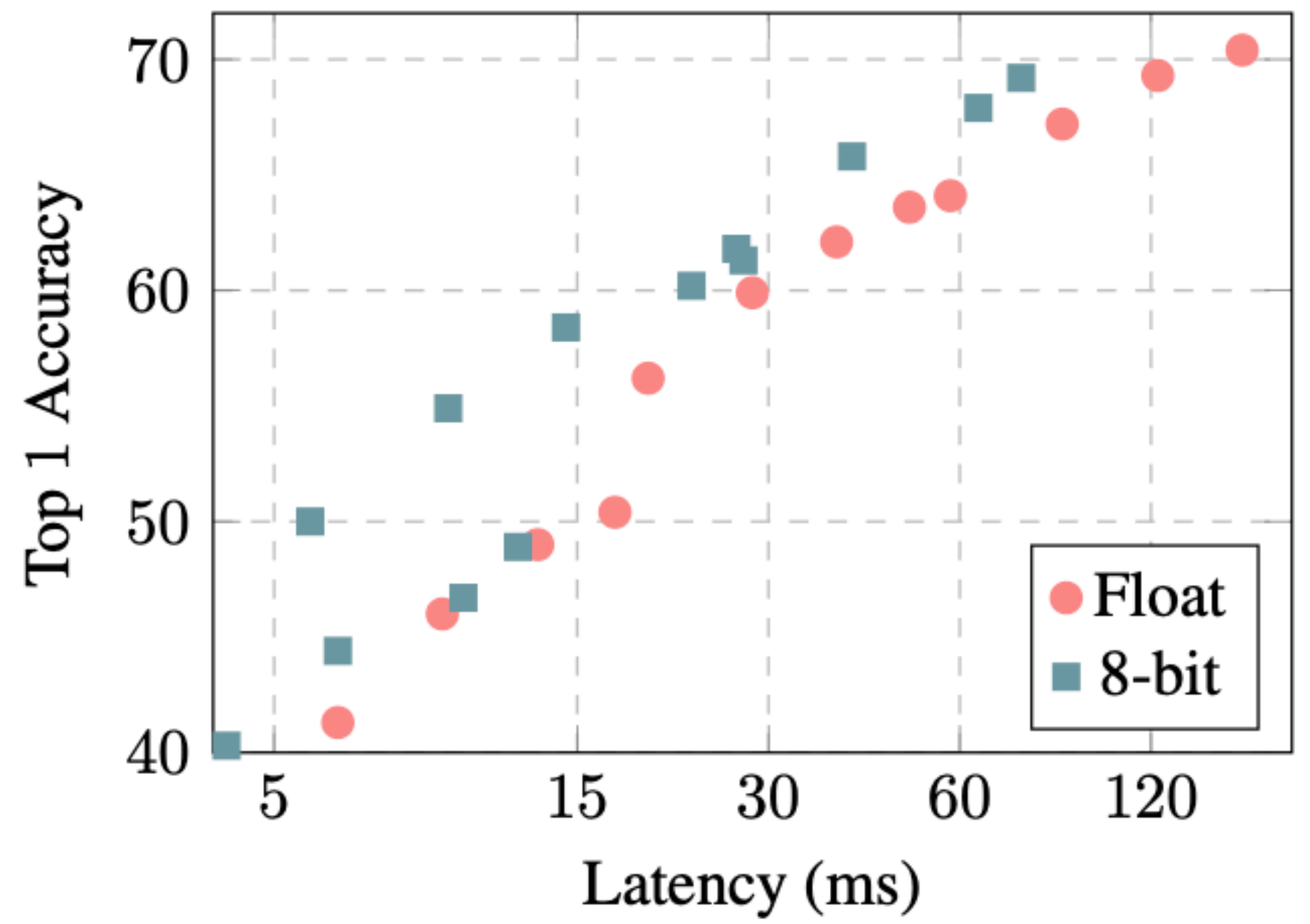
Yin et al., "Understanding STE in training activation-quantized neural nets," ICLR 2019

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%

INT8 Result

Krishnamoorthi et al., "Deep convolutional networks for efficient inference: A whitepaper," arXiv 2018

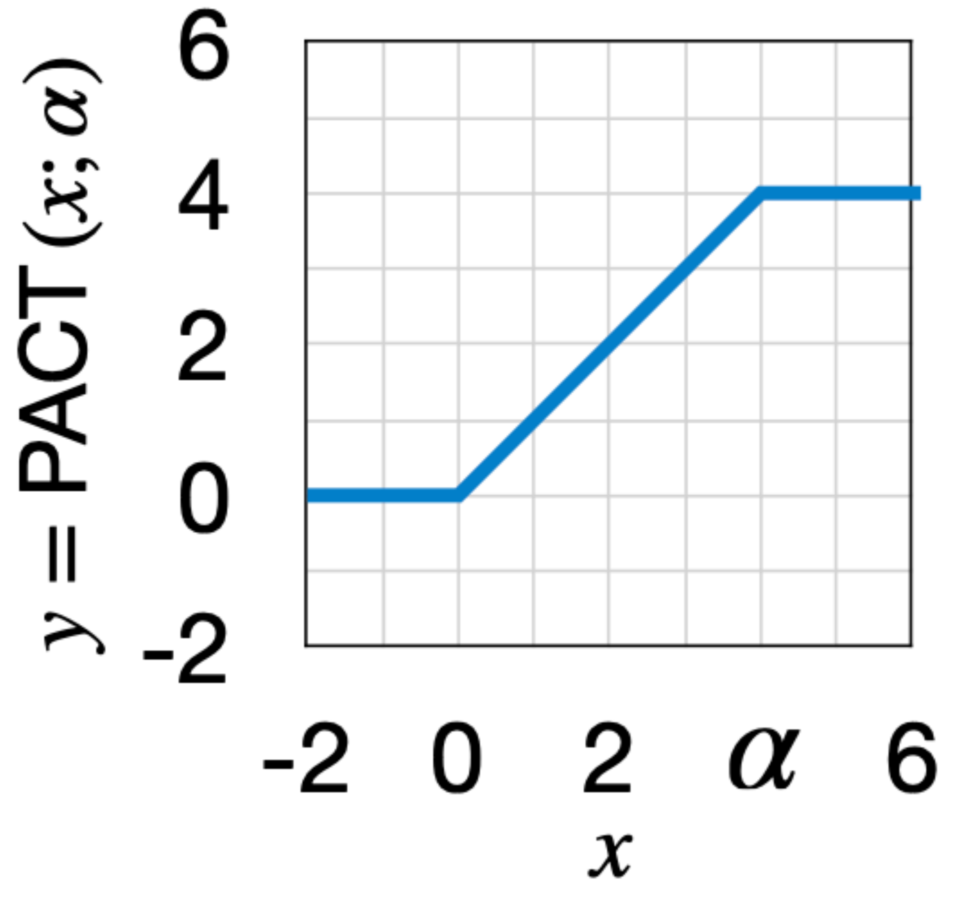
Neural Network	ResNet-50	Inception-V3
Floating-point Accuracy	76.4%	78.4%
8-bit Integer-quantized Acurracy	74.9%	75.4%



Latency-vs-accuracy tradeoff of float vs. integer-only MobileNets on ImageNet using Snapdragon 835 big cores.

PACT. We discussed using ReLU6 instead of ReLU to ensure that activations are in $[0,6]$.
There is an alternative with range $[0,\alpha]$, with a trainable $\alpha \in \mathbb{R}$.

$$y = \text{PACT}(x;\alpha) = 0.5\left(|x| - |x-\alpha| + \alpha\right) = \begin{cases} 0, & x \in [-\infty, 0) \\ x, & x \in [0,\alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$



Training α. If we look at the formula, we get

$$\frac{\partial\, \text{PACT}(x;\alpha)}{\partial \alpha} = \begin{cases} 1 & \cdots & x \geq \alpha \\ 0 & \cdots & x < \alpha \end{cases}$$

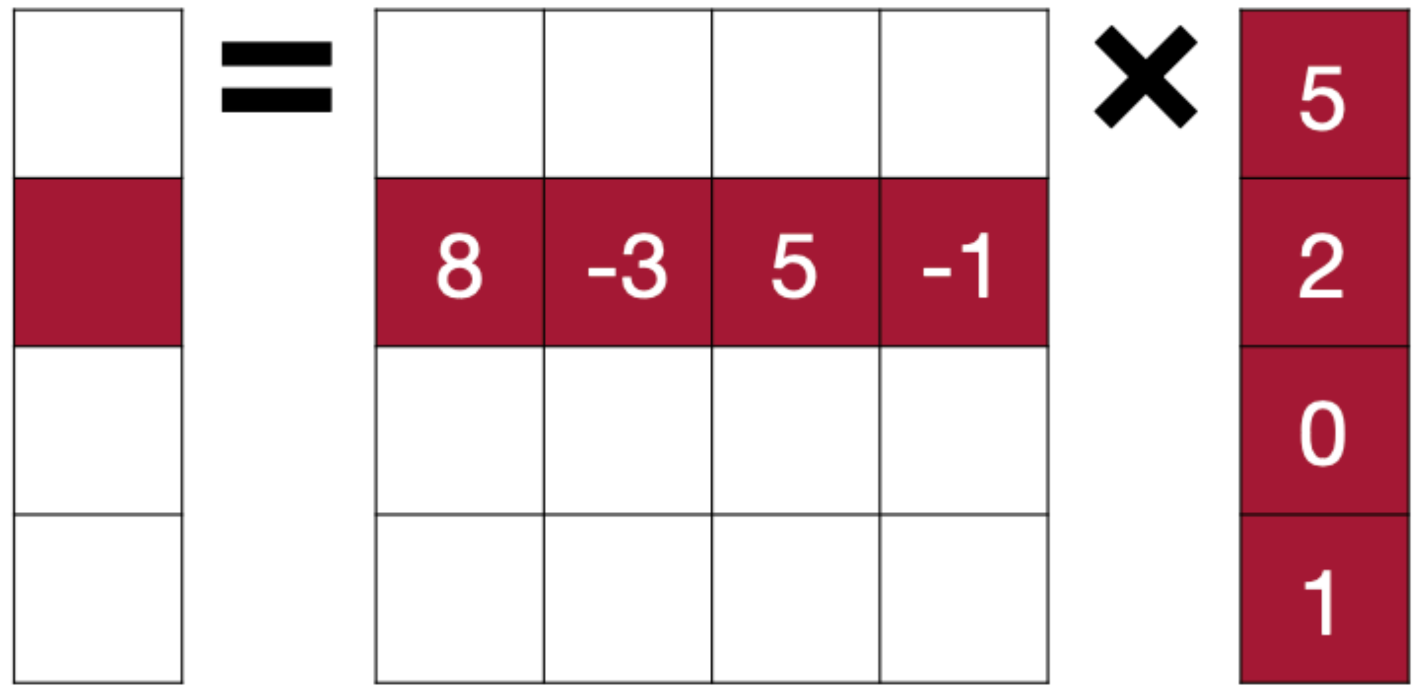
Use ℓ_2 regularization on α to encourage the small range.

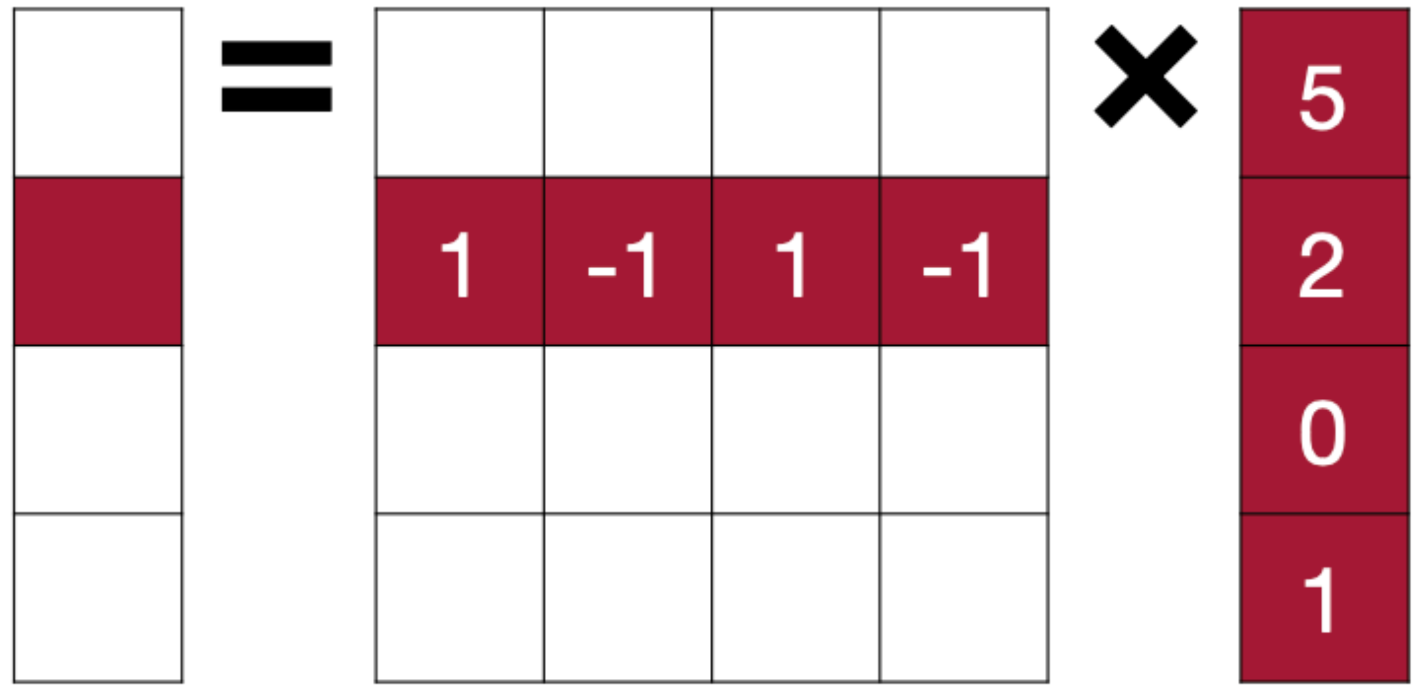
Neural Network	ImageNet Top-1 Accuracy								
	Floating-Point	DoReFa				PACT			
	32 bits	2 bits	3 bits	4 bits	5 bits	2 bits	3 bits	4 bits	5 bits
AlexNet	55.1%	53.6%	55.0%	54.9%	54.9%	55.0%	55.6%	55.7%	55.7%
ResNet-18	70.2%	62.6%	67.5%	68.1%	68.4%	64.4%	68.1%	69.2%	69.8%
ResNet-50	76.9%	67.1%	69.9%	71.4%	71.4%	72.2%	75.3%	76.5%	76.7%

Compute Alternatives. For very low-bit models or specially quantized models, it is possible to replace the multiply-accumulate operation with something else…

Example. Binary neural network; i.e., weights are either $+1$ or -1.
Then, there is no multiplication!

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

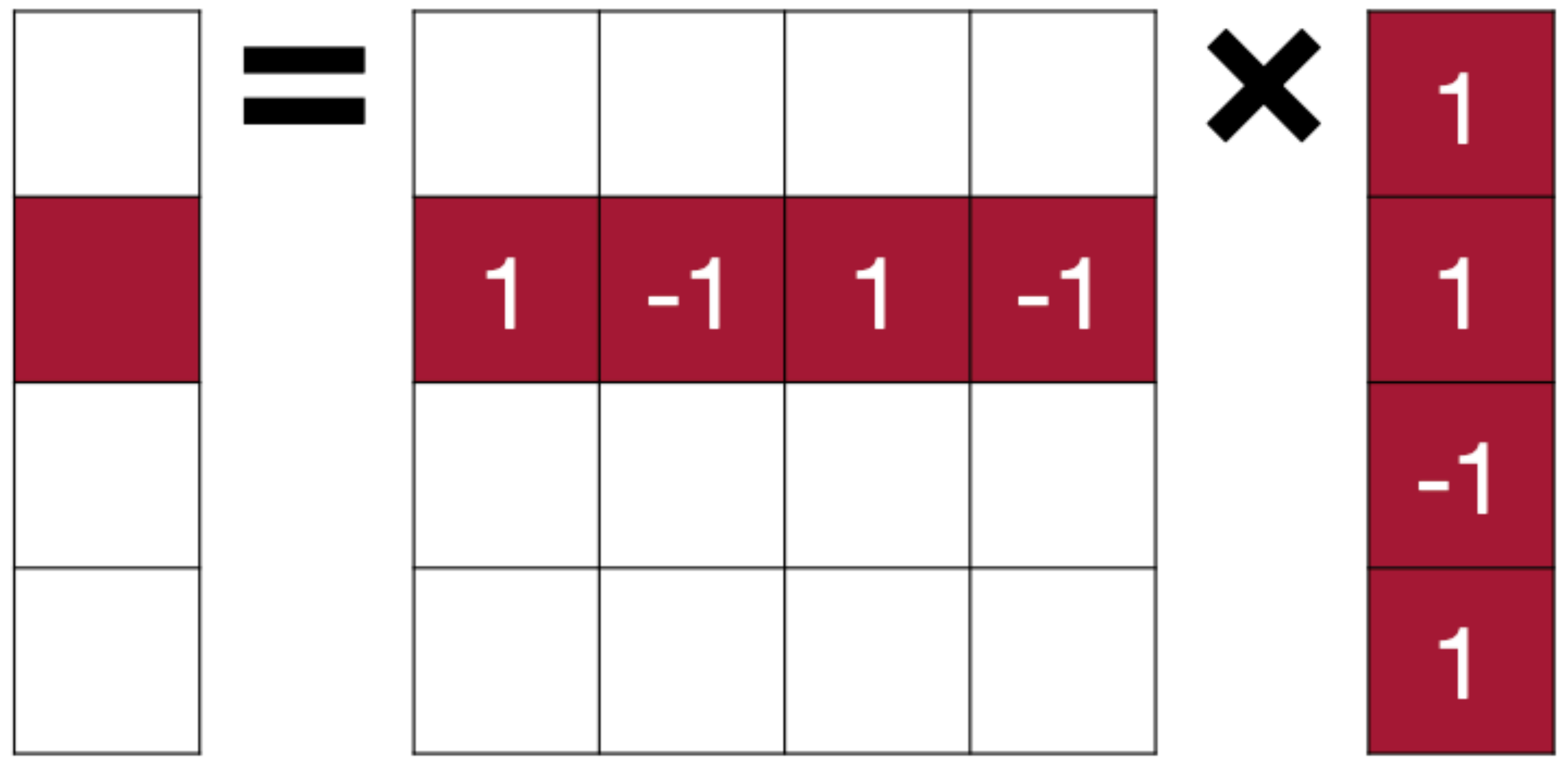




Courbariaux et al., "BinaryConnect: Training DNNs with binary weights during propagations," NeurIPS 2015

Rastegari et al., "XNOR-Net: ImageNet classification using binary CNNs," ECCV 2016

Example. If weights AND activations are binary,
then things can be computed via XNORs:



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

$$y_i = \sum_j W_{ij} \cdot x_j$$

$= 1×1 + (-1)×1 + 1×(-1) + (-1)×1$

$= 1 + (-1) + (-1) + (-1) = -2$

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

$= 1 \text{ xnor } 1 + 0 \text{ xnor } 1 + 1 \text{ xnor } 0 + 0 \text{ xnor } 1$

$= \underset{\uparrow +2}{1} + 0 + 0 + 0 = \boxed{\begin{array}{l} 1 \times 2 \\ + \\ -4 \end{array}} = -2$

Assuming $-1 \quad -1 \quad -1 \quad -1 \rightarrow -4$

Note. Number of 1s can be counted with "popcount" operation.

Note. See also LogQuant (Miyashita et al., 2016)
which uses shifting instead of multiplying

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

Courbariaux et al., "BinaryConnect: Training DNNs with binary weights during propagations," NeurIPS 2015
Rastegari et al., "XNOR-Net: ImageNet classification using binary CNNs," ECCV 2016
Miyashita et al., "CNNs using logarithmic data representation," arXiv 2016