MIT HAN LAB

Lecture 06 Quantization

Part II

Song Han
songhan@mit.edu

Lecture Plan

Today we will:

1. Review *Linear Quantization.*

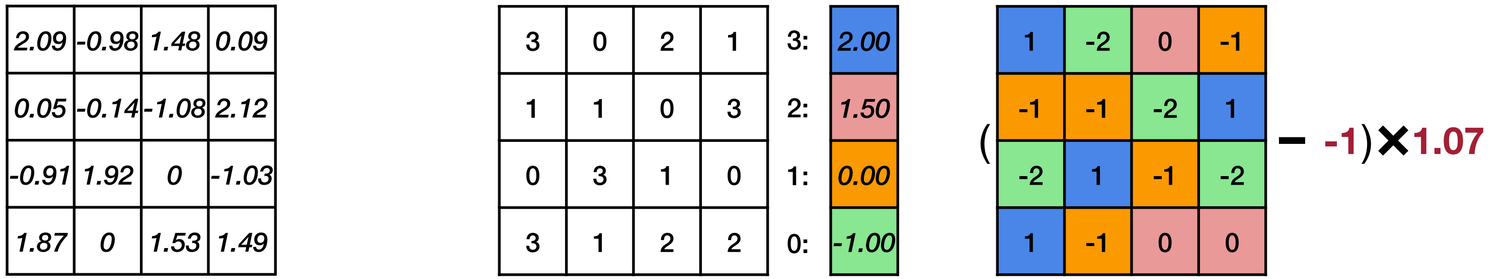
2. Introduce **Post-Training Quantization (PTQ)** that quantizes a floating-point neural network model, including: weight quantization, activation quantization, and bias quantization.

3. Introduce **Quantization-Aware Training (QAT)** that emulates inference-time quantization during the training/fine-tuning and recover the accuracy.

4. Introduce **binary and ternary** quantization.

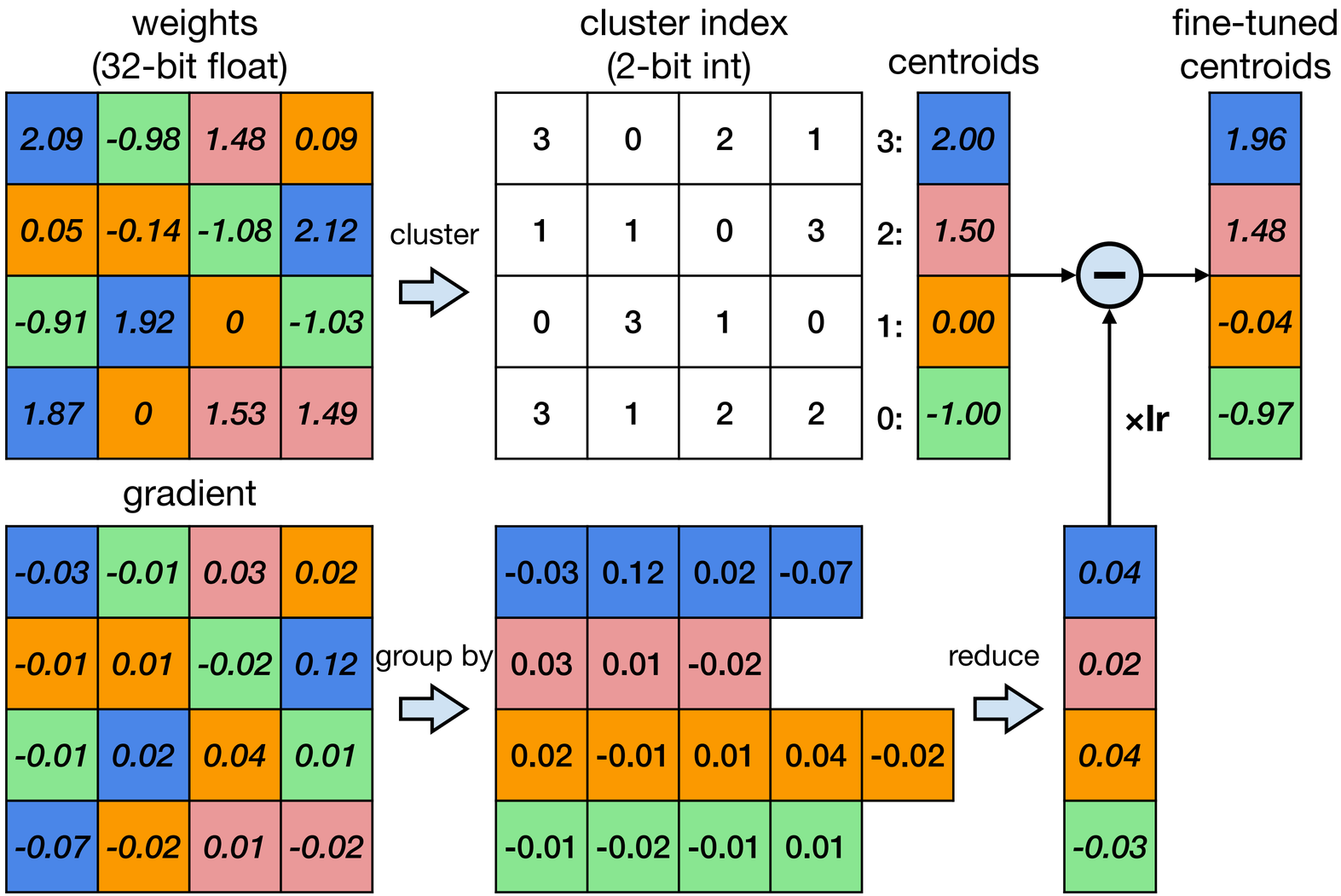
5. Introduce automatic **mixed-precision** quantization.

Neural Network Quantization



		K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

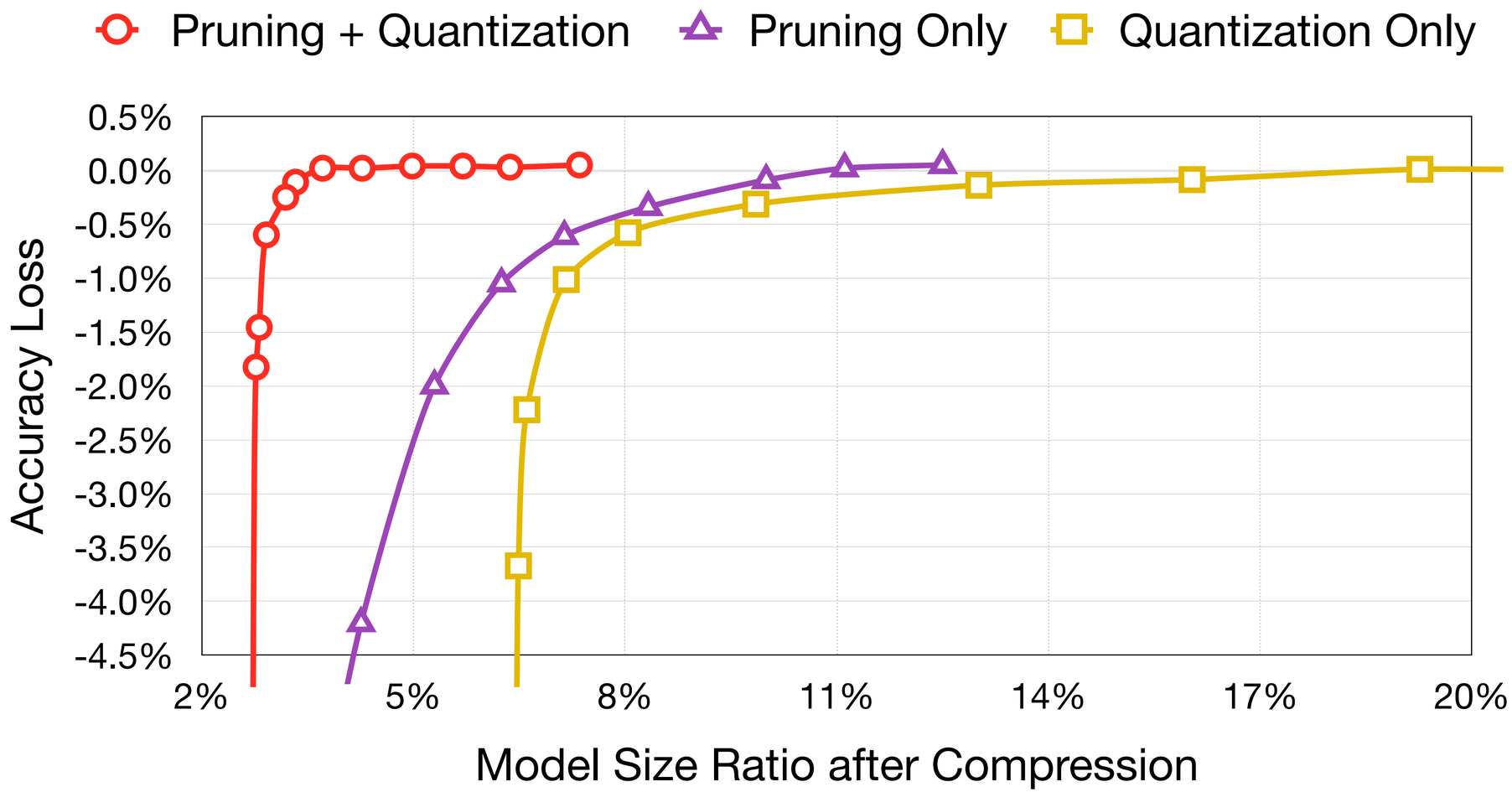
K-Means-based Weight Quantization



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

K-Means-based Weight Quantization

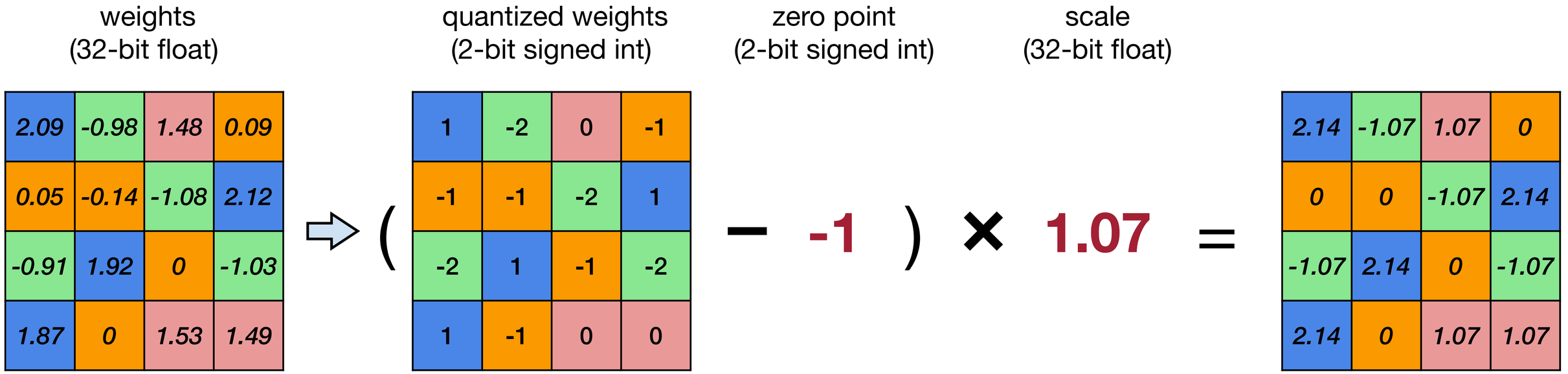
Accuracy vs. compression rate for AlexNet on ImageNet dataset



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

Linear Quantization

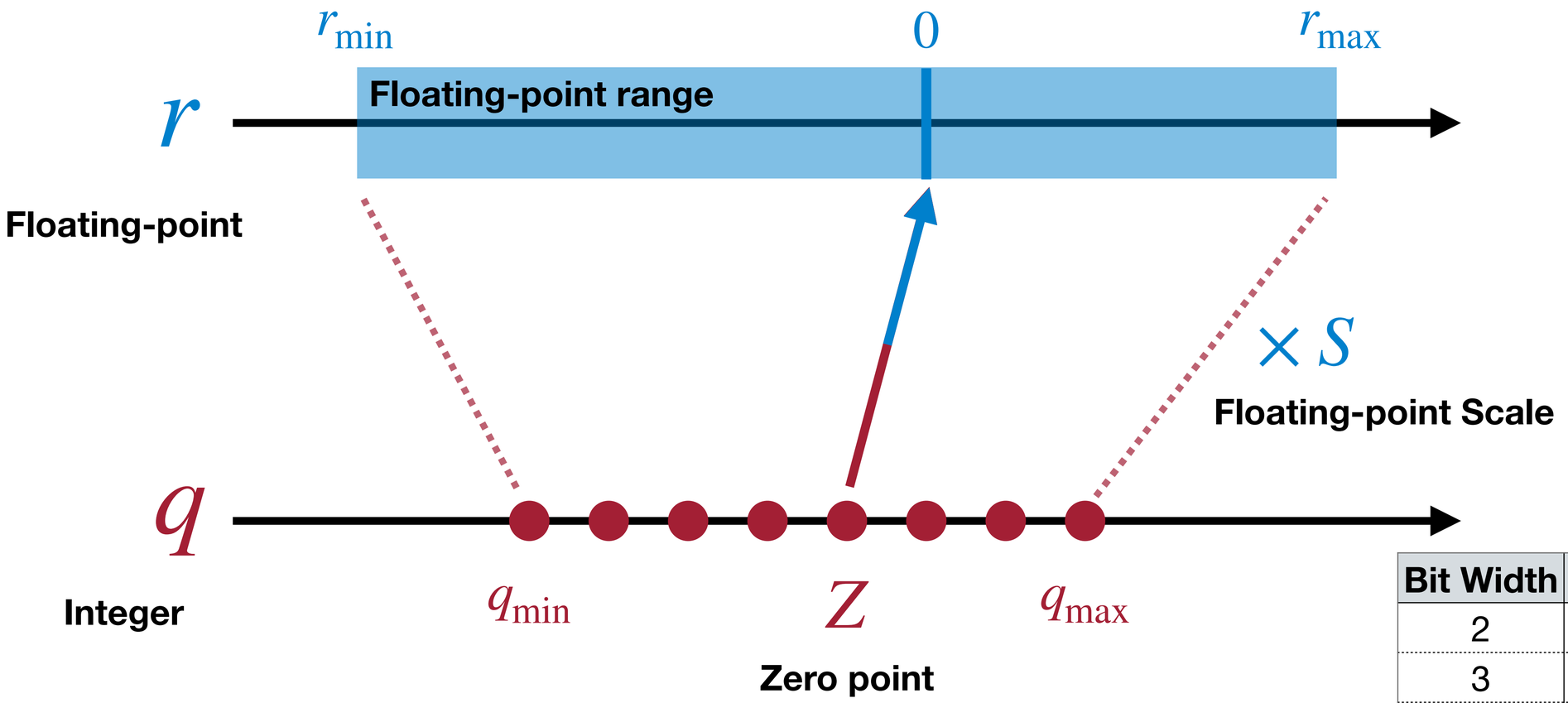
An affine mapping of integers to real numbers $r = S(q - Z)$



Binary	Decimal
01	1
00	0
11	-1
10	-2

Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$



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

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

Linear Quantized Fully-Connected Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following fully-connected layer.

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

$$Z_{\mathbf{W}} = 0$$

$$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}}$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}} \left(\mathbf{q}_{\mathbf{W}} \mathbf{q}_{\mathbf{X}} + \mathbf{q}_{bias} \right) + Z_{\mathbf{Y}}$$

Rescale to *N*-bit Int | **N-bit Int Mult. 32-bit Int Add.** | **N-bit Int Add**

Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.

Linear Quantized Convolution Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following convolution layer.

$$\mathbf{Y} = \text{Conv}(\mathbf{W}, \mathbf{X}) + \mathbf{b}$$

$$Z_{\mathbf{W}} = 0$$

$$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - \text{Conv}(\mathbf{q}_{\mathbf{W}}, Z_{\mathbf{X}})$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}} \left(\text{Conv}(\mathbf{q}_{\mathbf{W}}, \mathbf{q}_{\mathbf{X}}) + \mathbf{q}_{bias} \right) + Z_{\mathbf{Y}}$$

Rescale to *N*-bit Int

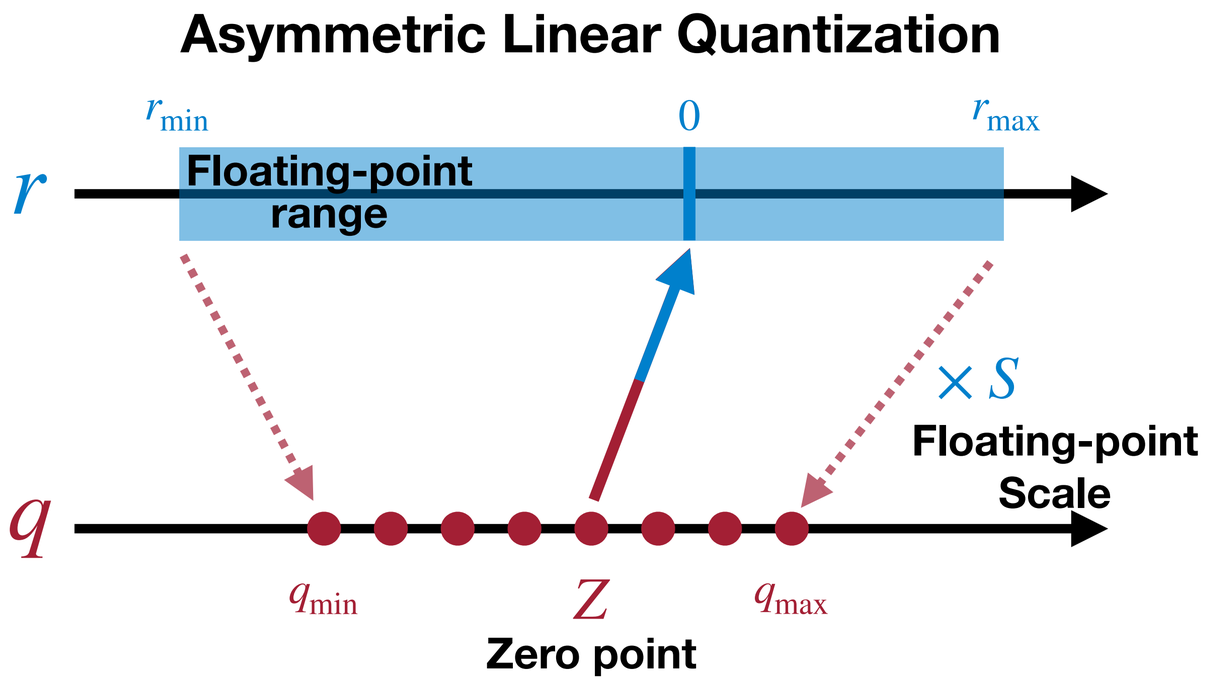
***N*-bit Int Mult. 32-bit Int Add.**

***N*-bit Int Add**

Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.

Scale and Zero Point of Linear Quantization

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



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

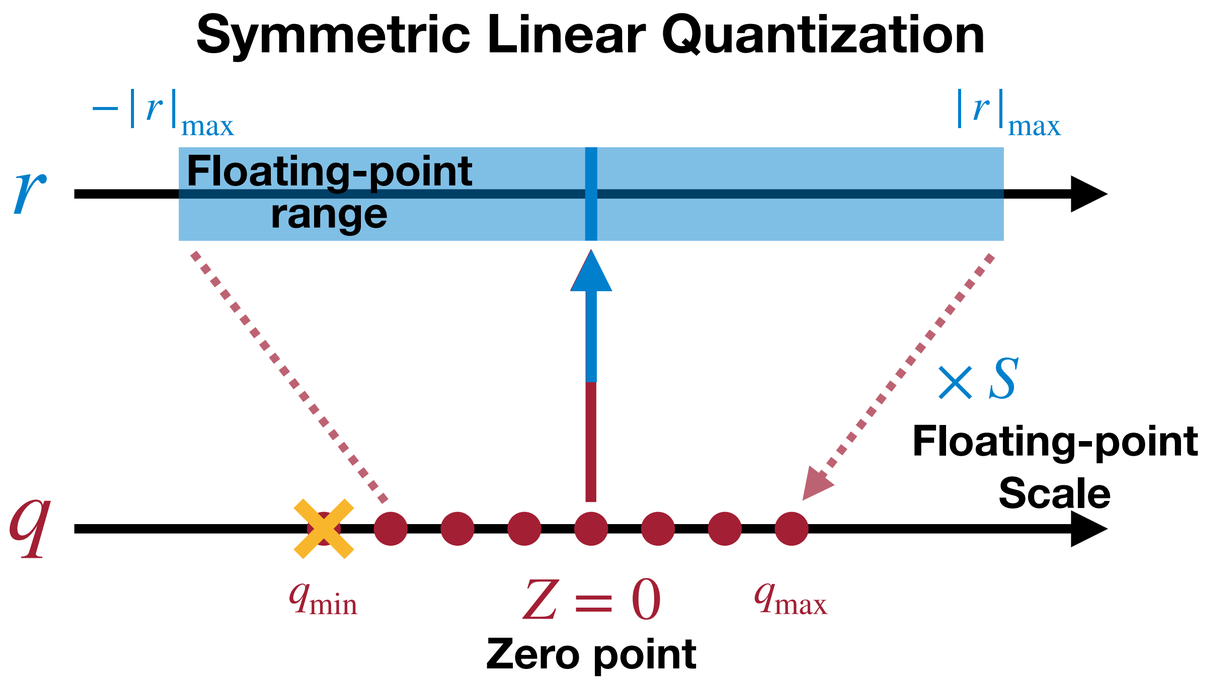
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} \qquad Z = q_{min} - \frac{r_{min}}{S}$$

$$= \frac{2.12 - (-1.08)}{1 - (-2)} \qquad = \text{round}(-2 - \frac{-1.08}{1.07})$$

$$= 1.07 \qquad = -1$$

Scale and Zero Point of Linear Quantization

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



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

$$S = \frac{|r|_{\max}}{q_{\max}} = \frac{2.12}{1} = 2.12 \qquad Z = 0$$

Post-Training Quantization

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

Topic I: Weight Quantization
Topic II: Activation Quantization
Topic III: Bias Quantization

Post-Training Quantization

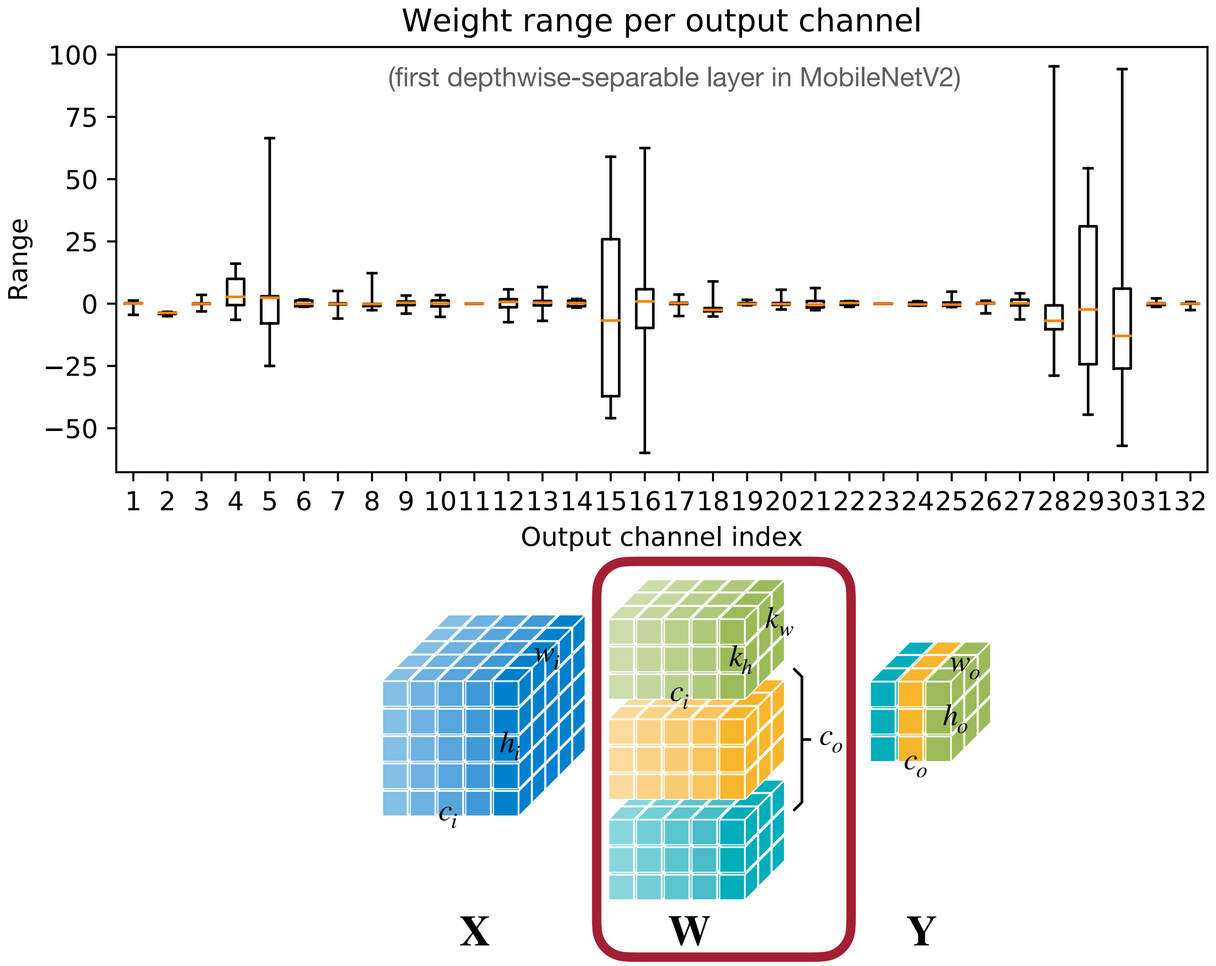
How should we get the optimal linear quantization parameters (S, Z)?

Topic I: Weight Quantization

Topic II: Activation Quantization

Topic III: Bias Quantization

Symmetric Linear Quantization on Weights



- $|r|_{\max} = |\mathbf{W}|_{\max}$
- Using *single* scale S for whole weight tensor (**Per-Tensor Quantization**)
  - works sufficiently well for large models
  - leads to significant accuracy drops for small models
- Common failure results from
  - large differences (more than 100×) in ranges of weights for different output channels
  - outlier weight values that make all remaining weights less precise after quantization
- Solution: **Per-Channel Quantization**

Data-Free Quantization Through Weight Equalization and Bias Correction [Markus *et al.*, ICCV 2019]
Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Per-Channel Weight Quantization

Example: 2-bit linear quantization

Per-Channel Quantization

oc \ ic				
	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-Tensor Quantization

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

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

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

Quantized

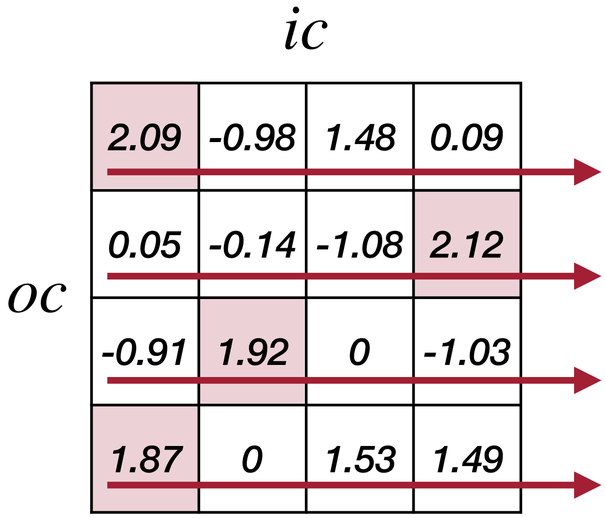
2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	0
2.12	0	2.12	2.12

Reconstructed

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

Per-Channel Weight Quantization

Example: 2-bit linear quantization



Per-Channel Quantization

$$|r|_{\max} = 2.09 \qquad S_0 = 2.09$$

$$|r|_{\max} = 2.12 \qquad S_1 = 2.12$$

$$|r|_{\max} = 1.92 \qquad S_2 = 1.92$$

$$|r|_{\max} = 1.87 \qquad S_3 = 1.87$$

Per-Tensor Quantization

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

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

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

Quantized

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

Reconstructed

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

Per-Channel Weight Quantization

Example: 2-bit linear quantization

oc \ ic				
	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

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

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

Quantized

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

Reconstructed

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

Per-Tensor Quantization

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

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

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

Quantized

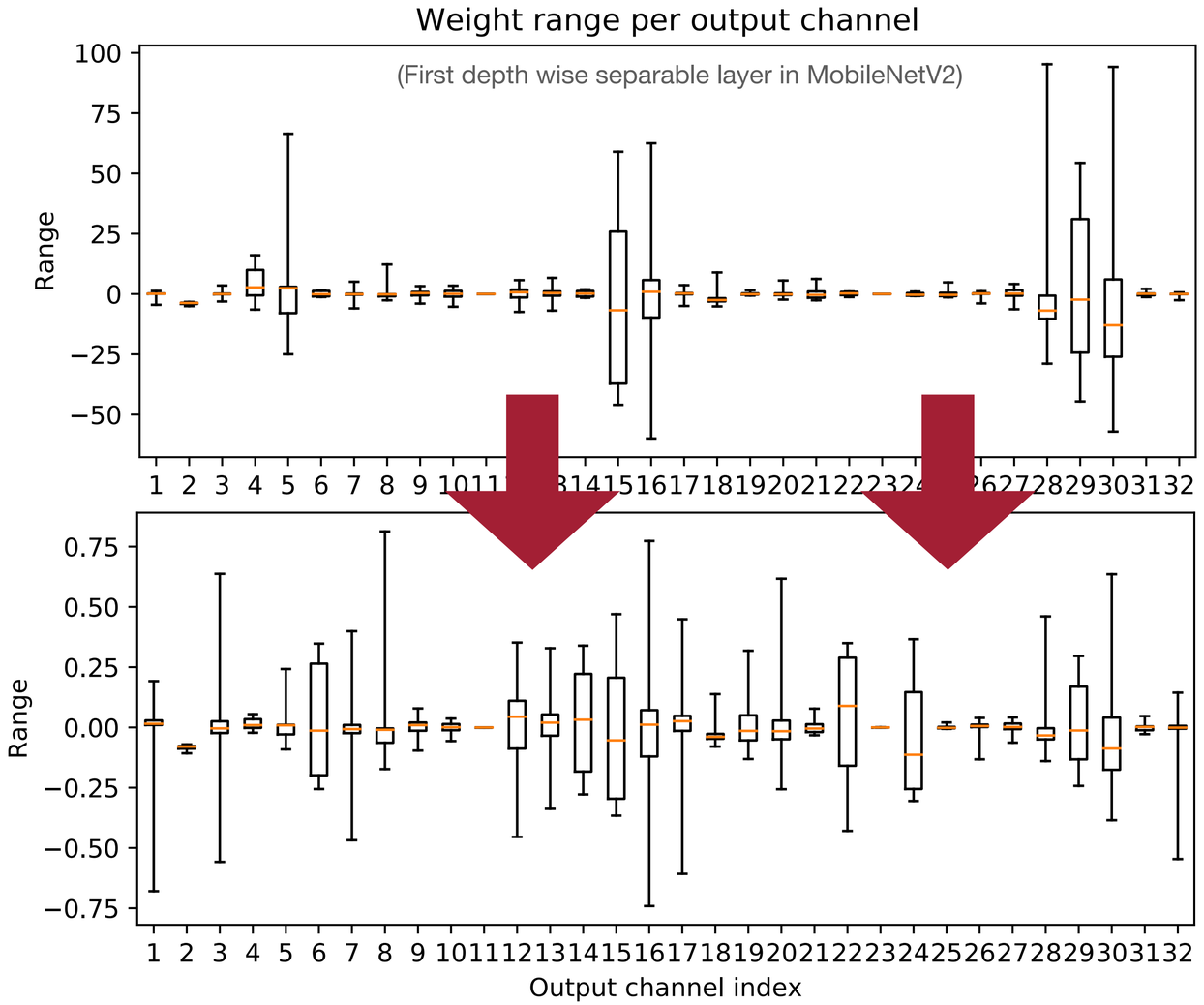
2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	0
2.12	0	2.12	2.12

Reconstructed

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

Symmetric Linear Quantization on Weights

Per-Tensor *vs.* Per-Channel Quantization



- Large differences in weight ranges for different output channels
  - → Per-Channel Weight Quantization
  - → much better performance than Per-Tensor Weight Quantization.

- However, it is not supported on all hardware, and it creates unnecessary overhead in the computation due to the necessity of scale values for each channel individually.

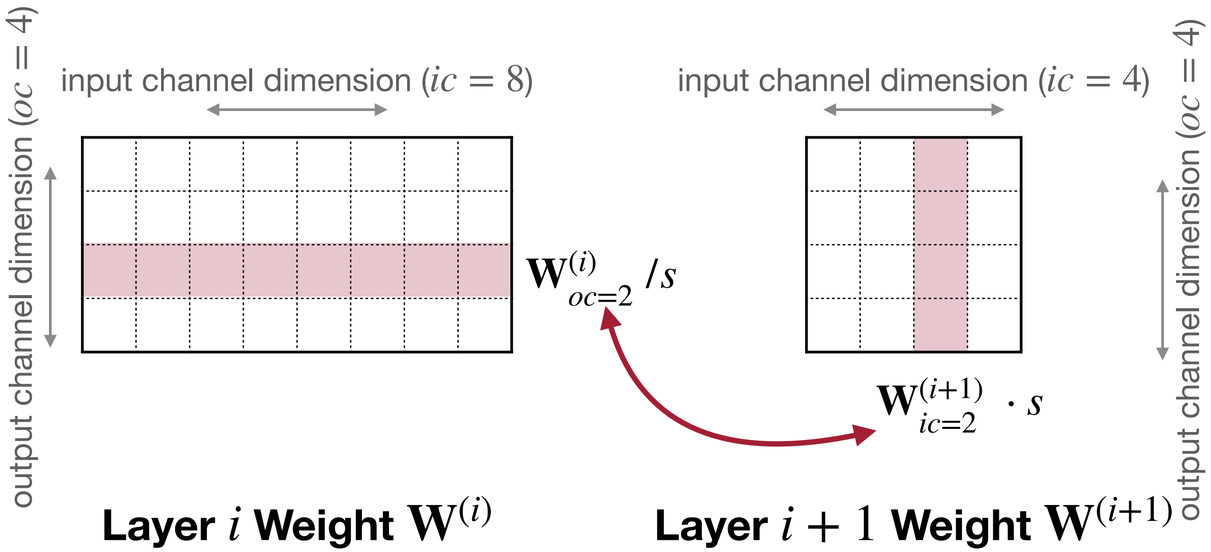
Can we make weight ranges similar to each other so that per-tensor weight quantization will work?

Data-Free Quantization Through Weight Equalization and Bias Correction [Markus et al., ICCV 2019]

Weight Equalization

Rescaling to equalize the weight ranges of different output channels

- Key idea: positive scaling equivariance
  - $=$ scaling **down** layer i weights $\mathbf{W}^{(i)}_{oc=a}$ of output channel $oc = a$ by s
  - $+$ scaling **up** layer $i+1$ weights $\mathbf{W}^{(i+1)}_{ic=a}$ of input channel $ic = a$ by s



- To make weight ranges of different output channels matched as closely as possible,

$$s_j = \frac{1}{r^{(i+1)}_{ic=j}} \sqrt{r^{(i)}_{oc=j} \cdot r^{(i+1)}_{ic=j}}$$

where $r^{(i)}_{oc=j}$ is the weight range of *output* channel j in Layer i, and $r^{(2)}_{ic=j}$ is the weight range of input channel j in Layer $i+1$.

$$r'^{(i)}_{oc=j} = r^{(i)}_{oc=j}/s = \frac{r^{(i)}_{oc=j} \cdot r^{(i+1)}_{ic=j}}{\sqrt{r^{(i)}_{oc=j} \cdot r^{(i+1)}_{ic=j}}} = \sqrt{r^{(i)}_{oc=j} \cdot r^{(i+1)}_{ic=j}}$$

$$r'^{(i+1)}_{ic=j} = r^{(i+1)}_{ic=j} \cdot s = \sqrt{r^{(i)}_{oc=j} \cdot r^{(i+1)}_{ic=j}}$$

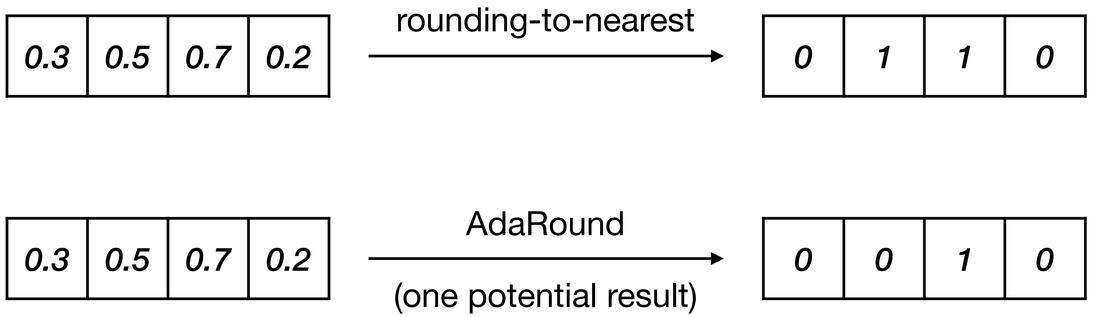
f has to be linear
e.g., ReLU() is piece-wise linear

Data-Free Quantization Through Weight Equalization and Bias Correction [Markus et al., ICCV 2019]

Adaptive Rounding for Weight Quantization

Rounding-to-nearest is not optimal

- **Philosophy**
  - Rounding-to-nearest is not optimal
  - Weights are correlated with each other. The best rounding for each weight (to nearest) is not the best rounding for the whole tensor



- What is optimal? Rounding that reconstructs the original activation the best, which may be very different
  - For weight quantization only
  - With short-term tuning, (almost) post-training quantization

Up or Down? Adaptive Rounding for Post-Training Quantization [Nagel *et al.*, PMLR 2020]

Adaptive Rounding for Weight Quantization

Rounding-to-nearest is not optimal

- **Method**:
  - Instead of $\lfloor w \rceil$, we want to choose from $\{\lfloor w \rfloor, \lceil w \rceil\}$ to get the best reconstruction
  - We took a learning-based method to find quantized value $\tilde{w} = \lfloor \lfloor w \rfloor + \delta \rceil, \delta \in [0,1]$
  - We optimize the following equation (omit the derivation):

$$\text{argmin}_{\mathbf{V}} \|\mathbf{W}\mathbf{x} - \tilde{\mathbf{W}}\mathbf{x}\|_F^2 + \lambda f_{reg}(\mathbf{V})$$

$$\rightarrow \text{argmin}_{\mathbf{V}} \|\mathbf{W}\mathbf{x} - \lfloor \lfloor \mathbf{W} \rfloor + \mathbf{h}(\mathbf{V}) \rceil \mathbf{x}\|_F^2 + \lambda f_{reg}(\mathbf{V})$$

  - $\mathbf{x}$ is the input to the layer, $\mathbf{V}$ is a random variable of the same shape
  - $\mathbf{h}()$ is a function to map the range to $(0,1)$, such as rectified sigmoid
  - $f_{reg}(\mathbf{V})$ is a regularization that encourages $\mathbf{h}(\mathbf{V})$ to be binary

Up or Down? Adaptive Rounding for Post-Training Quantization [Nagel *et al.*, PMLR 2020]

Post-Training Quantization

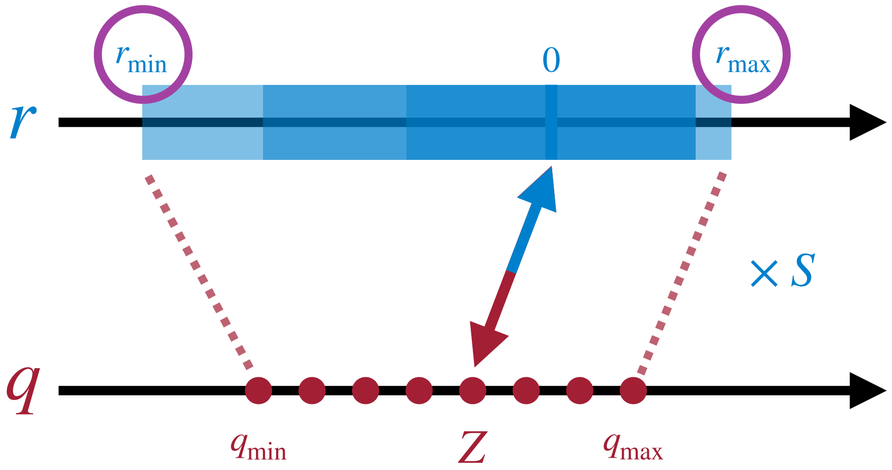
How should we get the optimal linear quantization parameters (S, Z)?

Topic I: Weight Quantization

Topic II: Activation Quantization

Topic III: Bias Quantization

Linear Quantization on Activations

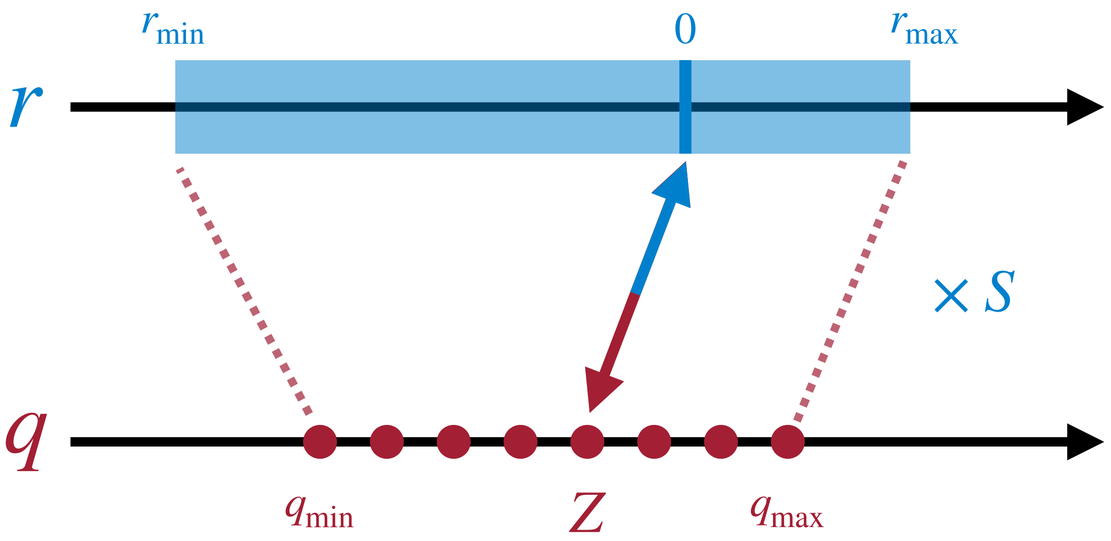


- For different inputs, the floating-point range varies.
- To determine the floating-point range, the activations statistics are gathered ***before*** deploying the model.

Dynamic Range for Activation Quantization

Collect activations statistics before deploying the model

$$\hat{r}^{(t)}_{\max,\min} = \alpha \cdot r^{(t)}_{\max,\min} + (1-\alpha) \cdot \hat{r}^{(t-1)}_{\max,\min}$$

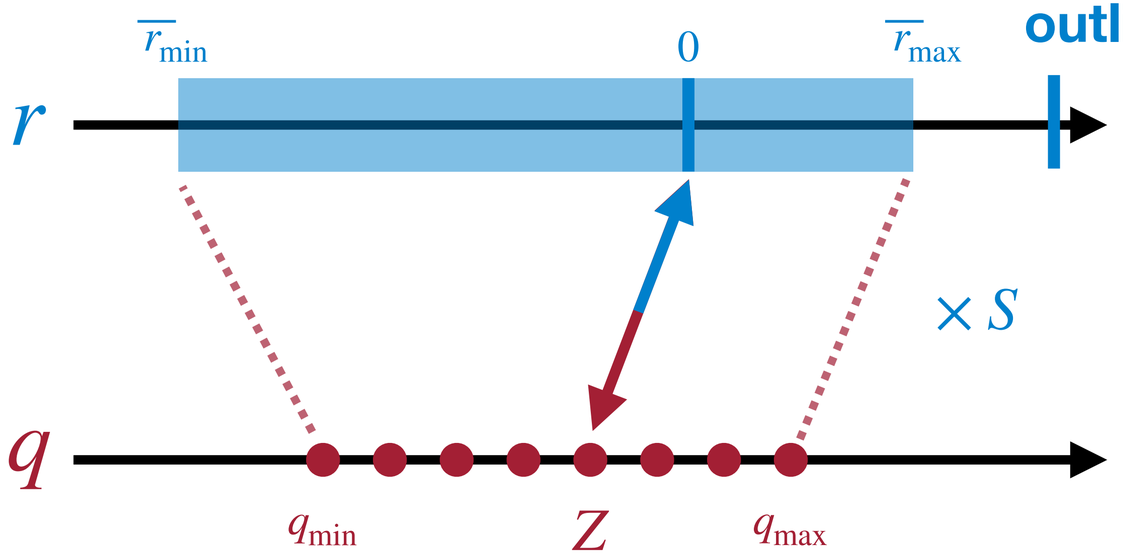


- Type 1: During training
  - Exponential moving averages (EMA)
    - observed ranges are smoothed across thousands of training steps

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

Dynamic Range for Activation Quantization

Collect activations statistics before deploying the model



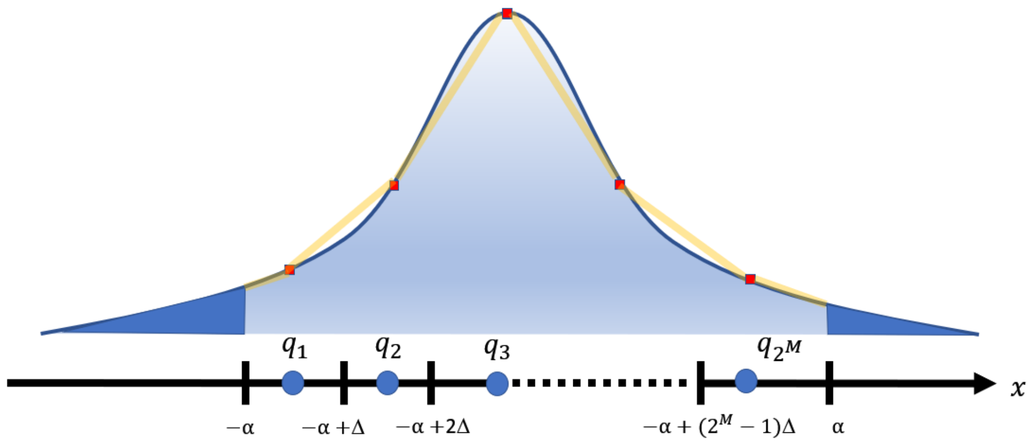
- Type 2: By running a few "calibration" batches of samples on the trained FP32 model
  - spending dynamic range on the outliers hurts the representation ability.
  - use *mean* of the min/max of each sample in the batches
  - analytical calculation (see next slide)

Neural Network Distiller

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

Dynamic Range for Activation Quantization

Collect activations statistics before deploying the model



- Type 2: By running a few "calibration" batches of samples on the trained FP32 model
  - *minimize the mean-square-error* between inputs $\mathbf{X}$ and (reconstructed) quantized inputs $Q(\mathbf{X})$,

$$\min_{|r|_{\max}} \mathbb{E}\left[(\mathbf{X} - Q(\mathbf{X}))^2\right]$$

  - assume inputs are in a Gaussian or Laplace distribution. For Laplace $(0, b)$ distribution, optimal clipping values can be solved *numerically* as

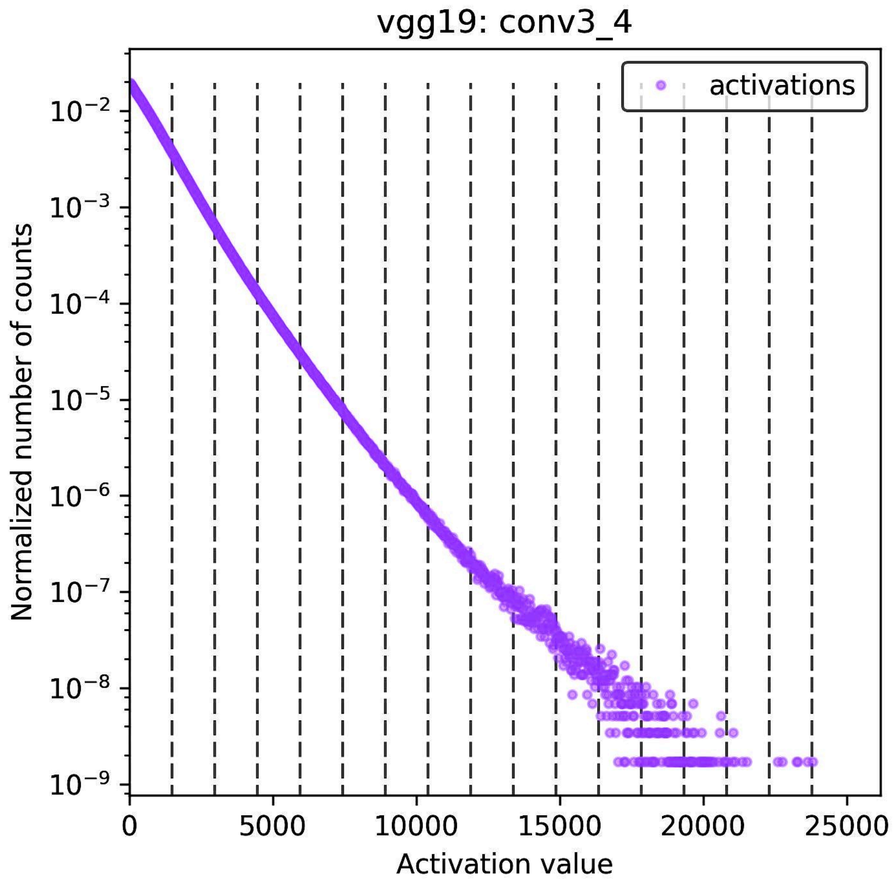
$$|r|_{\max} = 2.83b,\ 3.89b,\ 5.03b \text{ for 2, 3, 4 bits.}$$

  - the Laplace parameter b can be estimated from calibration input distribution.

Post-Training 4-Bit Quantization of Convolution Networks for Rapid-Deployment [Banner *et al.*, NeurIPS 2019]

Dynamic Range for Activation Quantization

Collect activations statistics before deploying the model



- Type 2: By running a few "calibration" batches of samples on the trained FP32 model
  - *minimize loss of information*, since integer model encodes the same information as the original floating-point model.
    - loss of information is measured by Kullback-Leibler divergence (relative entropy or information divergence):
      - for two discrete probability distributions P, Q

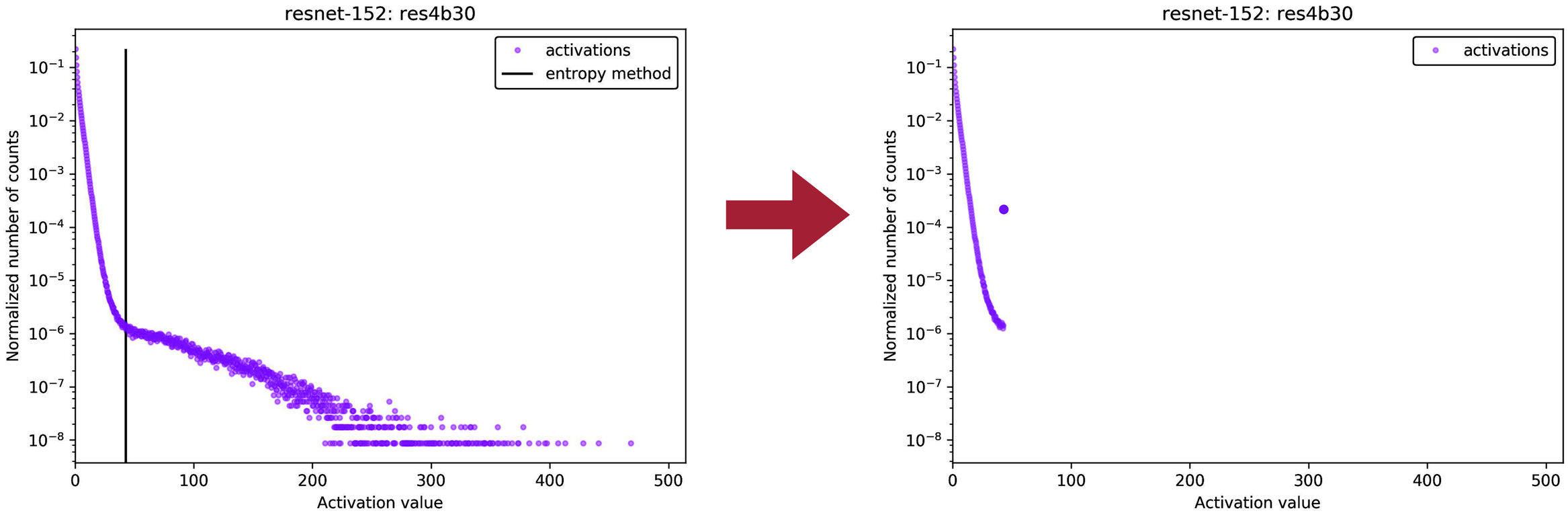
$$D_{KL}(P\|Q) = \sum_{i}^{N} P(x_i)\log\frac{P(x_i)}{Q(x_i)}$$

      - intuition: KL divergence measures the amount of information lost when approximating a given encoding.

8-bit Inference with TensorRT [Szymon Migacz, 2017]

Dynamic Range for Activation Quantization

Minimize loss of information by minimizing the KL divergence

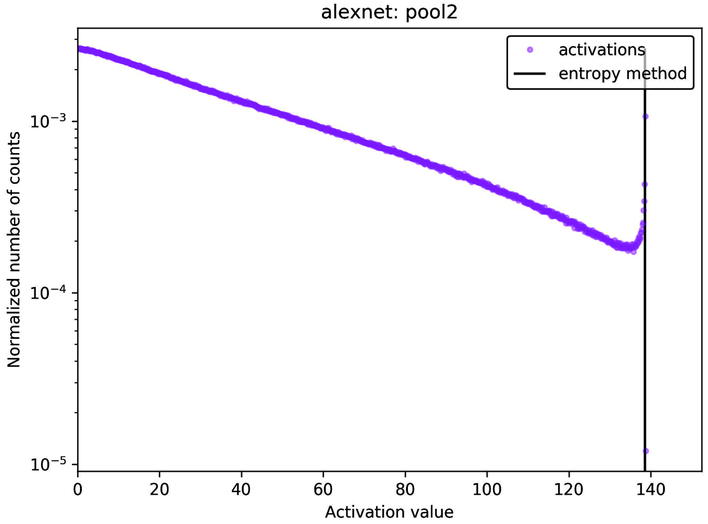


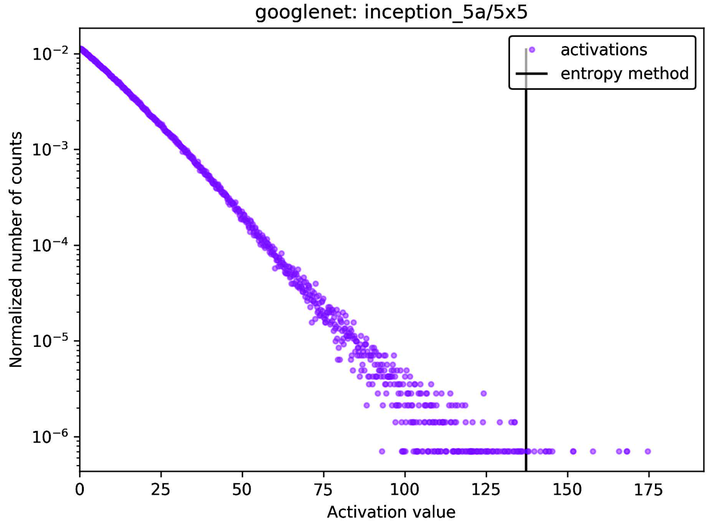
8-bit Inference with TensorRT [Szymon Migacz, 2017]

Dynamic Range for Activation Quantization

Minimize loss of information by minimizing the KL divergence

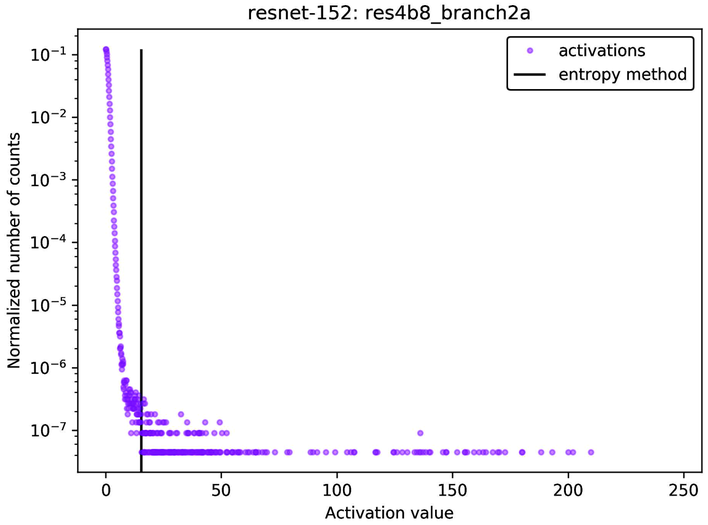
AlexNet: Pool 2

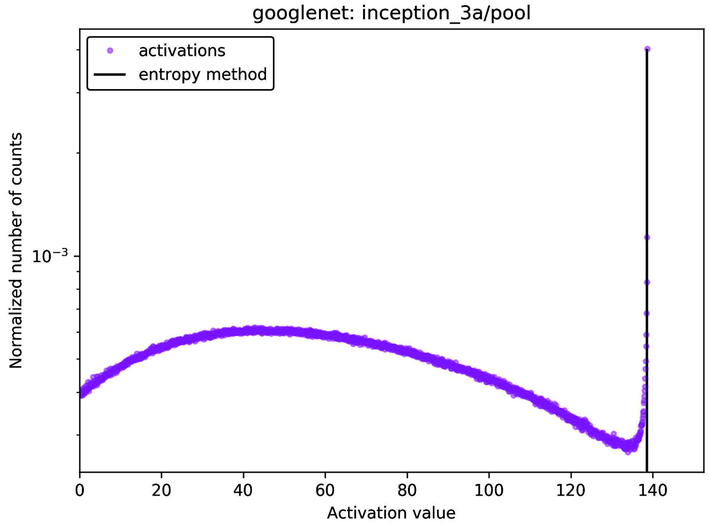




GoogleNet: incpetion_5a/5x5

ResNet-152: res4b8_branch2a





GoogleNet: incpetion_3a/pool

8-bit Inference with TensorRT [Szymon Migacz, 2017]

Post-Training Quantization

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

Topic I: Weight Quantization

Topic II: Activation Quantization

Topic III: Bias Quantization

Quantization Bias Correction

Fix the biased error introduced by quantization

- A common assumption is that quantization error is unbiased and thus cancels out in a layers' output, ensuring that the mean of a layer's output does not change as a result of quantization.
- The quantization error on weights can cause biased error on the corresponding outputs, shifting the input distribution of the next layer.

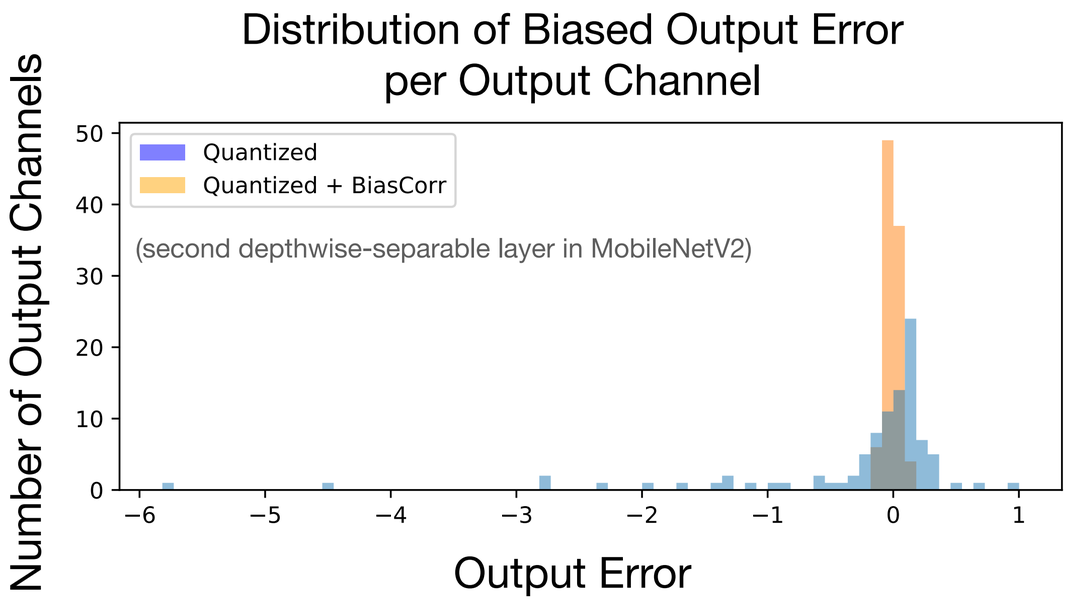
$$\mathbb{E}\left[\hat{\mathbf{y}}_j - \mathbf{y}_j\right] \approx \frac{1}{N}\sum_n \left(Q(\mathbf{W})\mathbf{x}_n\right)_j - \left(\mathbf{W}\mathbf{x}_n\right)_j$$

- Denote quantization error as $\epsilon = Q(\mathbf{W}) - \mathbf{W}$.

$$\begin{aligned}\mathbb{E}\left[\mathbf{y}\right] &= \mathbb{E}\left[\mathbf{W}\mathbf{x}\right] + \mathbb{E}\left[\epsilon\mathbf{x}\right] - \mathbb{E}\left[\epsilon\mathbf{x}\right] \\ &= \mathbb{E}\left[Q(\mathbf{W})\mathbf{x}\right] - \epsilon\mathbb{E}\left[\mathbf{x}\right]\end{aligned}$$

absorbed in the bias parameter

$\mathbb{E}\left[\mathbf{x}\right]$ can be inferred from batch normalization



Data-Free Quantization Through Weight Equalization and Bias Correction [Markus et al., ICCV 2019]

Post-Training INT8 Linear Quantization

Activation		Symmetric	Asymmertric			Asymmertric
		Per-Tensor	Per-Tensor			Per-Tensor
		Minimize KL-Divergence	Exponential Moving Average (EMA)			mean +/- 6 * std (from BatchNorm)
Weight		Symmetric	Asymmetric	Asymmetric	Symmetric	Asymmetric
		Per-Tensor	Per-Tensor	Per-Channel	Per-Channel	Per-Tensor
		-	-	-	-	Weight Equalization Bias Correction
Neural Network	GoogleNet	-0.45%	0%	0%	0%	-
	ResNet-50	-0.13%	-0.6%	-0.6%	-0.6%	-
	ResNet-152	-0.08%	-1.7%	-1.8%	-1.8%	-
	MobileNetV1	-	-70.8%	-0.6%	-11.8%	-0.3%
	MobileNetV2	-	-71.8%	-2.2%	-2.1%	-0.5%

Data-Free Quantization Through Weight Equalization and Bias Correction [Markus *et al.*, ICCV 2019]
Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper [Raghuraman Krishnamoorthi, arXiv 2018]
8-bit Inference with TensorRT [Szymon Migacz, 2017]

Post-Training INT8 Linear Quantization

Activation		Symmetric	Asymmertric			Asymmertric
		Per-Tensor	Per-Tensor			Per-Tensor
		Minimize KL-Divergence	Exponential Moving Average (EMA)			mean +/- 6 * std (from BatchNorm)
Weight		Symmetric	Asymmetric	Asymmetric	Symmetric	Asymmetric
		Per-Tensor	Per-Tensor	Per-Channel	Per-Channel	Per-Tensor
		-	-	-	-	Weight Equalization Bias Correction
Neural Network	GoogleNet	-				
	ResNet-50	-				
	ResNet-152	-				
	MobileNetV1	-	-70.8%	-0.6%	-11.8%	-0.3%
	MobileNetV2	-	-71.8%	-2.2%	-2.1%	-0.5%

Smaller models seem to not respond as well to post-training quantization, presumabley due to their smaller representational capacity.

How should we improve performance of quantized models?

Data-Free Quantization Through Weight Equalization and Bias Correction [Markus *et al.*, ICCV 2019]
Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper [Raghuraman Krishnamoorthi, arXiv 2018]
8-bit Inference with TensorRT [Szymon Migacz, 2017]

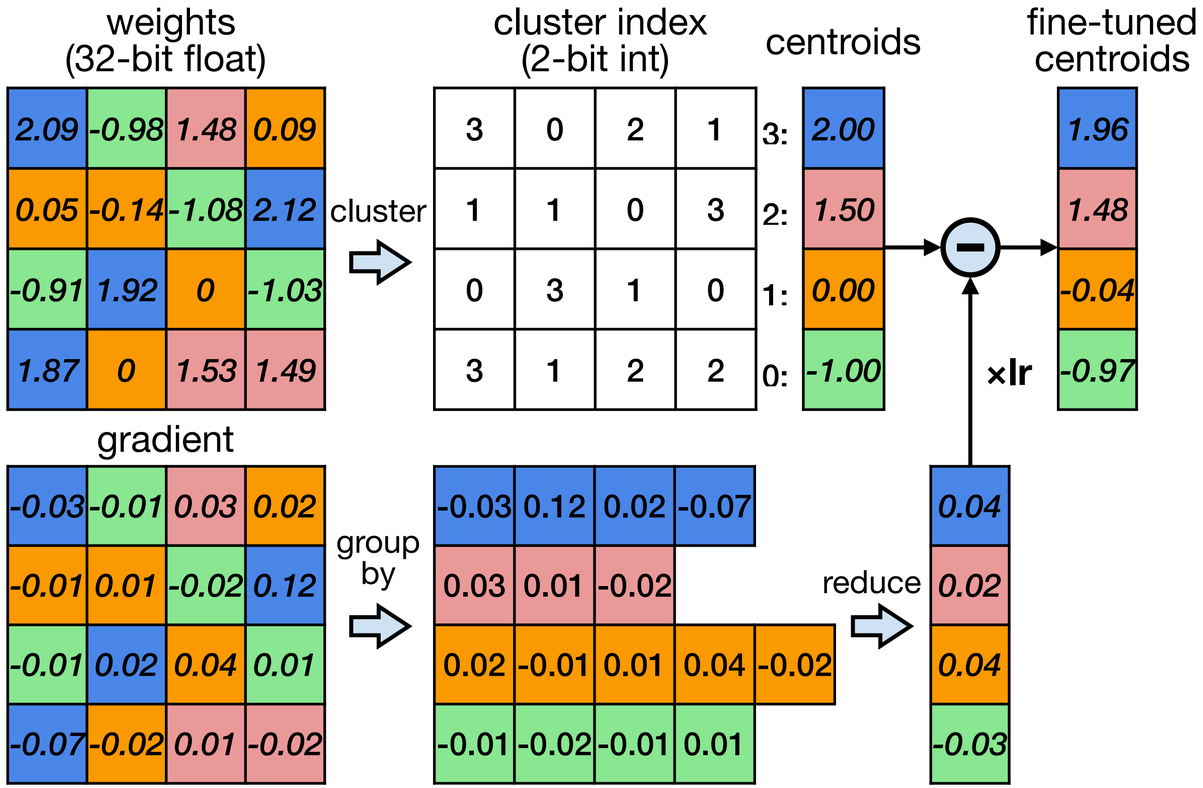
Quantization-Aware Training

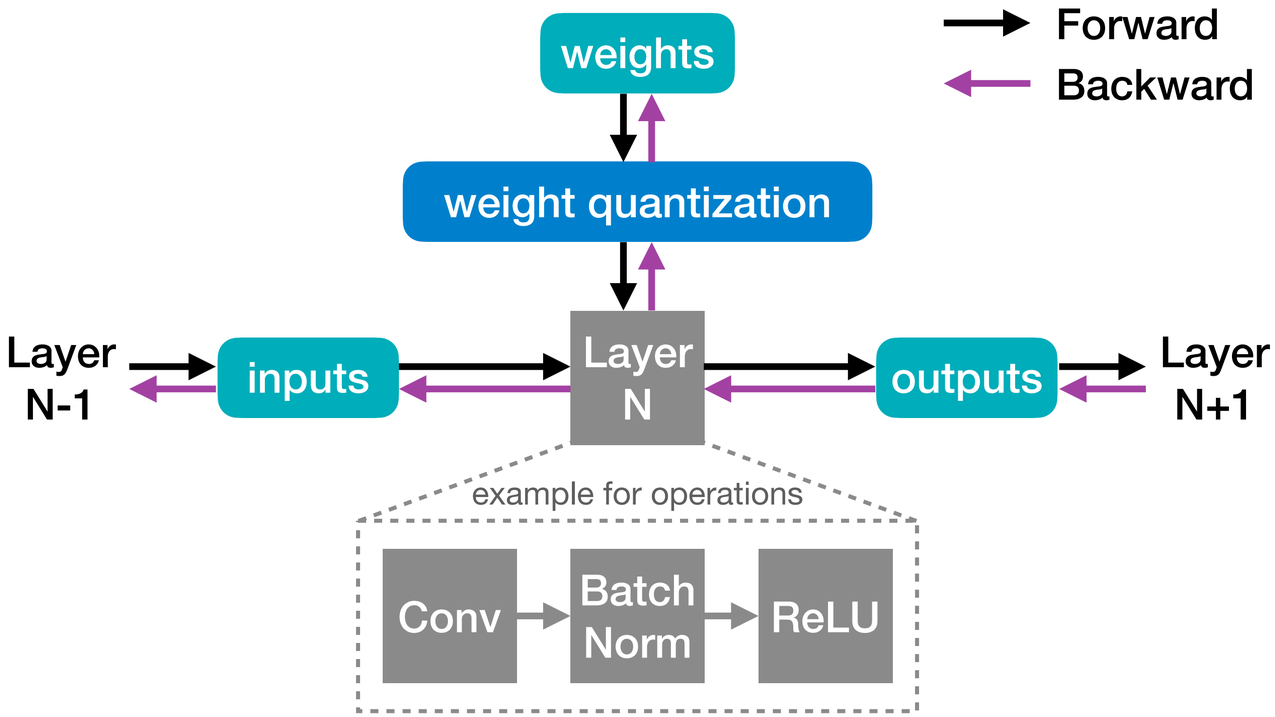
How should we improve performance of quantized models?

Quantization-Aware Training

Train the model taking quantization into consideration

- To minimize the loss of accuracy, especially aggressive quantization with 4 bits and lower bit width, neural network will be trained/fine-tuned with quantized weights and activations.
- Usually, fine-tuning a pre-trained floating point model provides better accuracy than training from scratch.



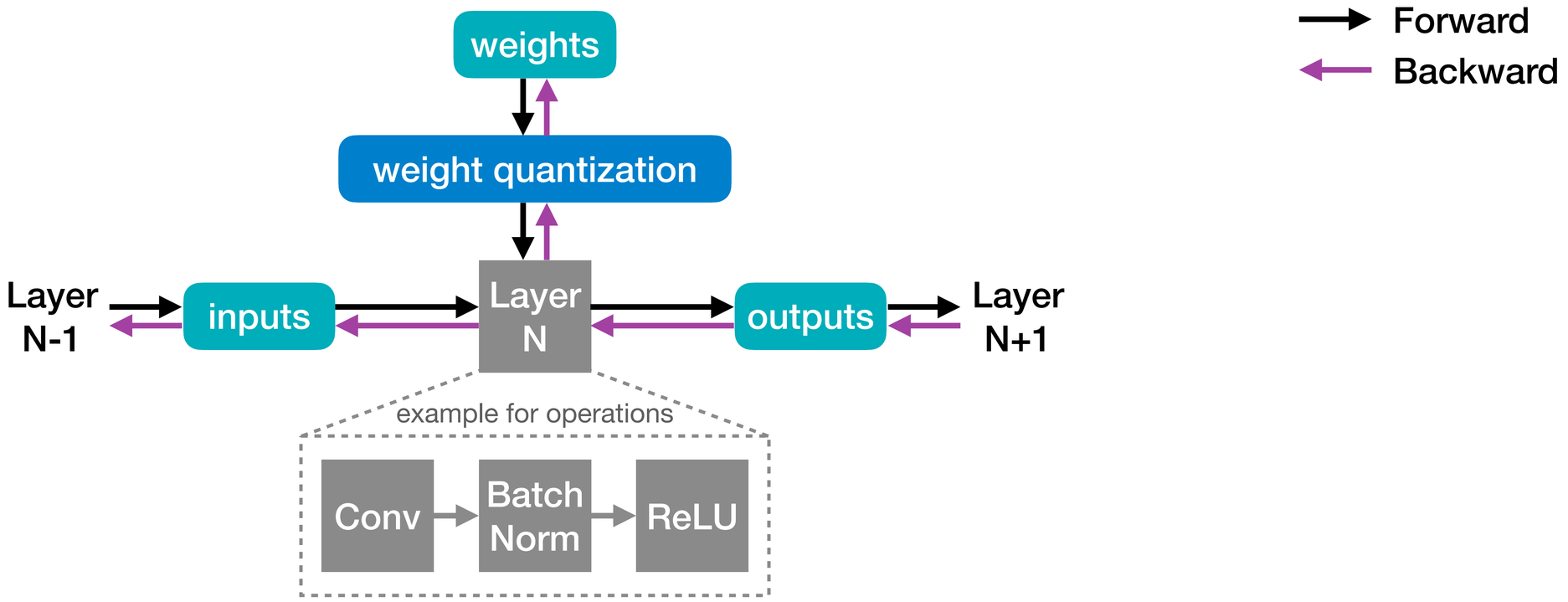


Deep Compression [Han *et al.*, ICLR 2016]

Quantization-Aware Training

Train the model taking quantization into consideration

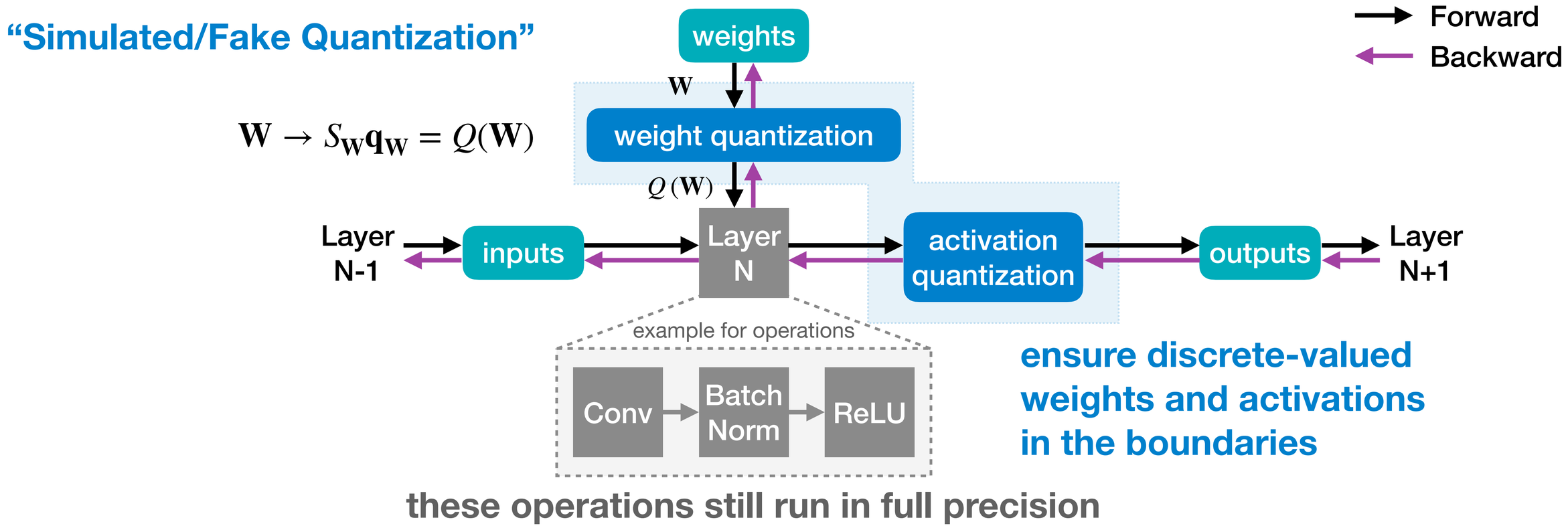
- A full precision copy of the weights W is maintained throughout the training.
- The small gradients are accumulated without loss of precision.
- Once the model is trained, only the quantized weights are used for inference.



Quantization-Aware Training

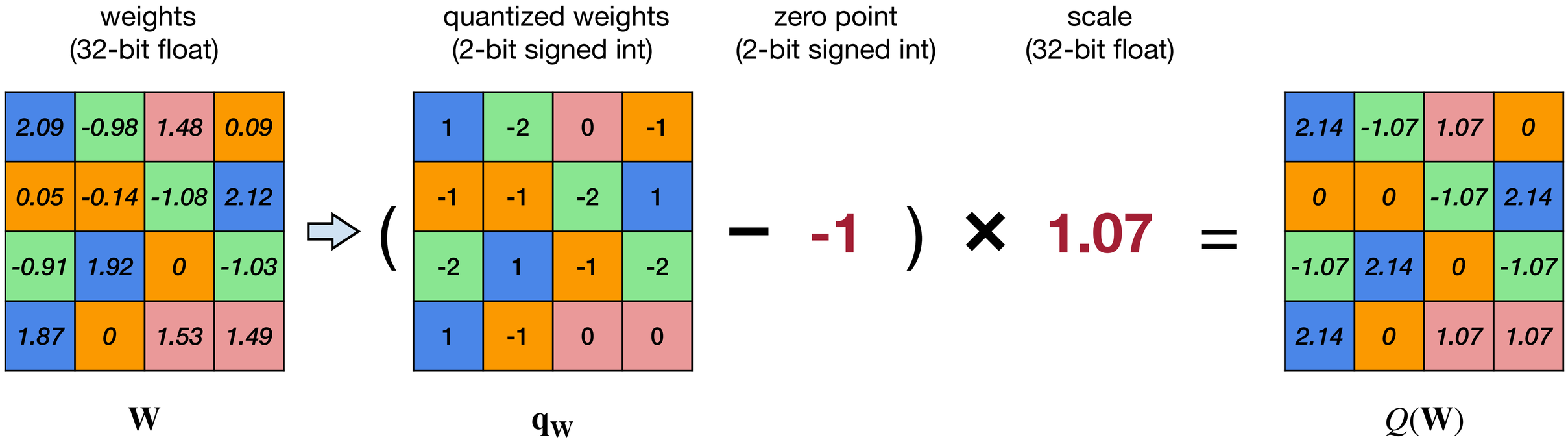
Train the model taking quantization into consideration

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



Linear Quantization

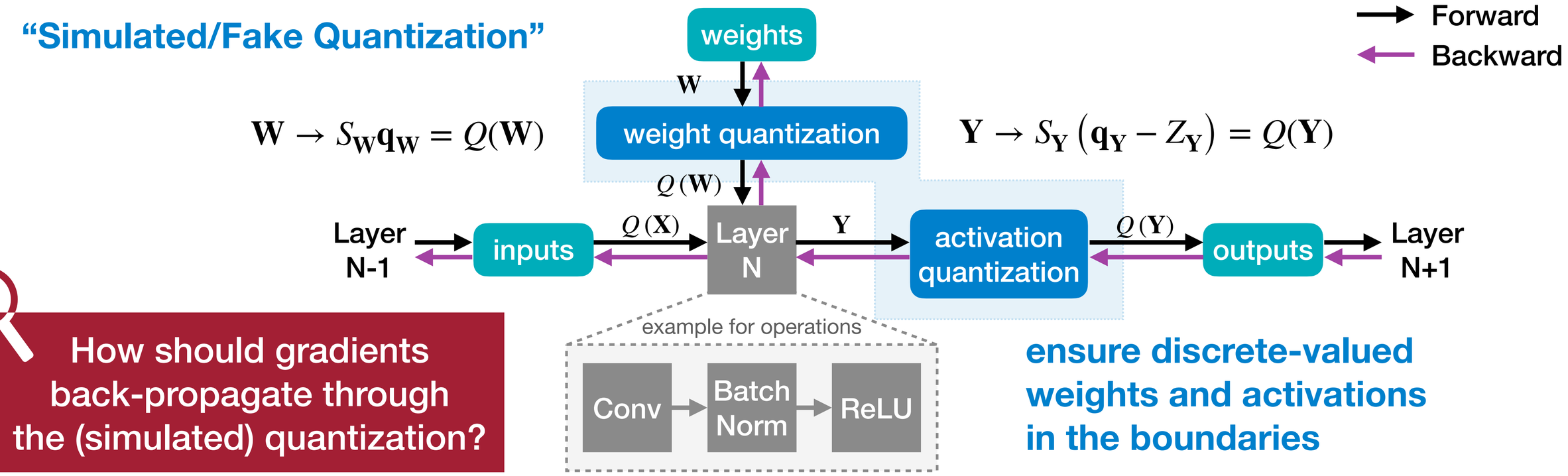
An affine mapping of integers to real numbers $r = S(q - Z)$



Quantization-Aware Training

Train the model taking quantization into consideration

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



Straight-Through Estimator (STE)

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

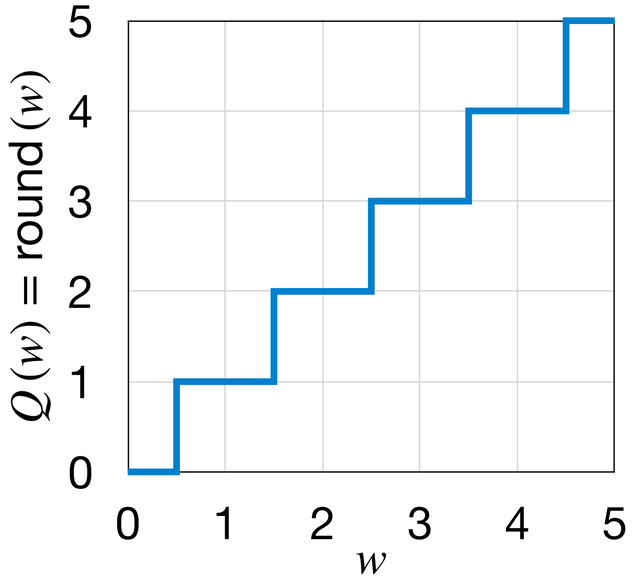
$$\frac{\partial Q(W)}{\partial W} = 0$$

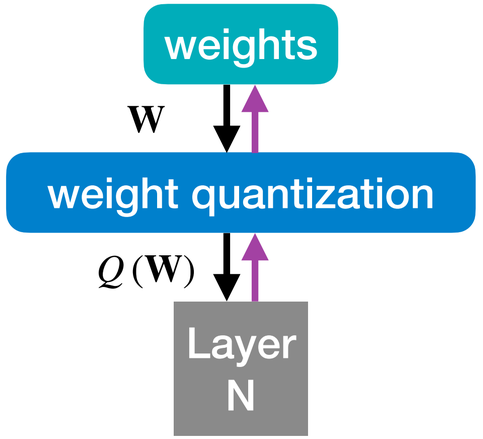
  - The neural network will learn nothing since gradients become 0 and the weights won't get updated.

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

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

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



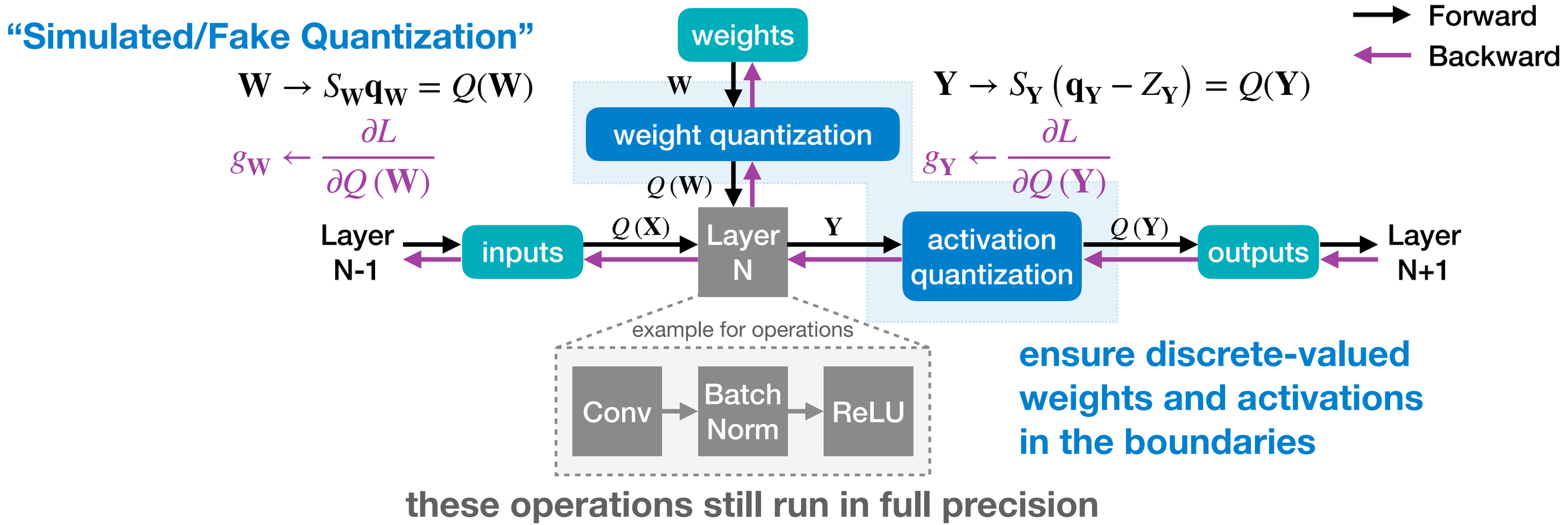


Neural Networks for Machine Learning [Hinton *et al.*, Coursera Video Lecture, 2012]
Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [Bengio, arXiv 2013]

Quantization-Aware Training

Train the model taking quantization into consideration

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



INT8 Linear Quantization-Aware Training

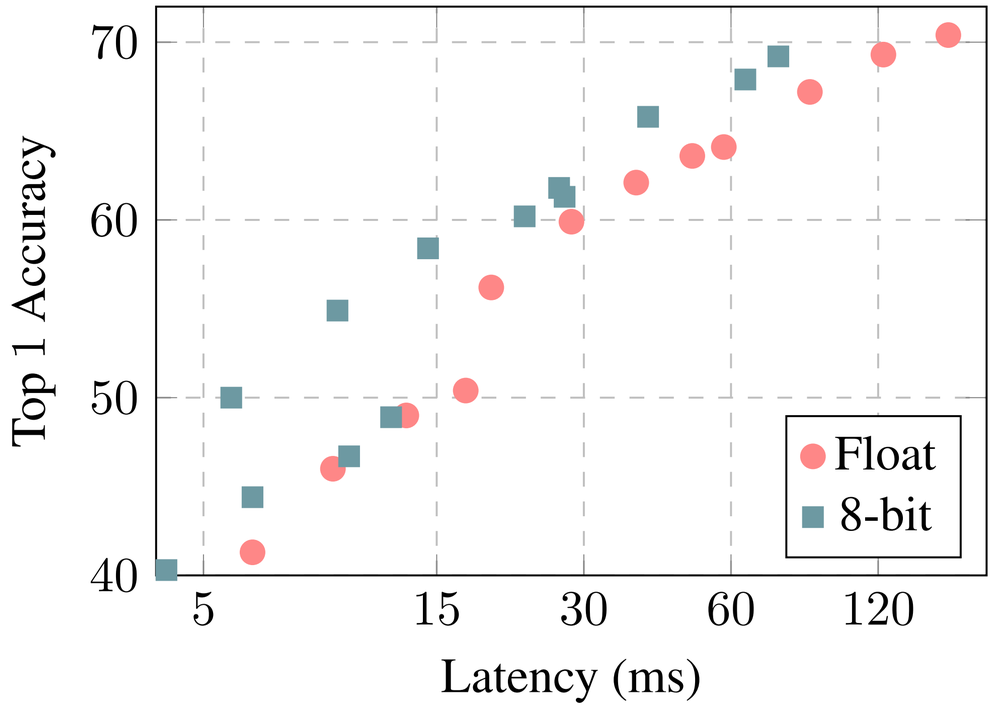
Neural Network	Floating-Point	Post-Training Quantization		Quantization-Aware Training	
		Asymmetric	Symmetric	Asymmetric	Symmetric
		Per-Tensor	Per-Channel	Per-Tensor	Per-Channel
MobileNetV1	70.9%	0.1%	59.1%	70.0%	70.7%
MobileNetV2	71.9%	0.1%	69.8%	70.9%	71.1%
NASNet-Mobile	74.9%	72.2%	72.1%	73.0%	73.0%

Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper [Raghuraman Krishnamoorthi, arXiv 2018]

INT8 Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$

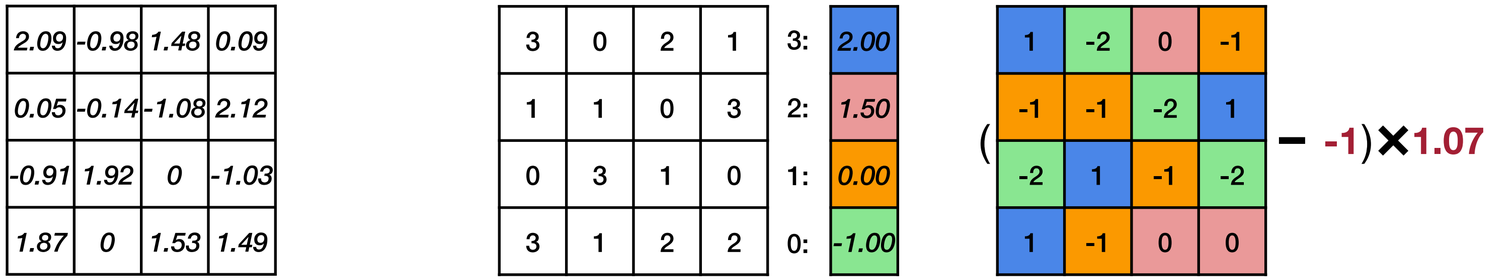
Neural Network	ResNet-50	Inception-V3
Floating-point Accuracy	76.4%	78.4%
8-bit Integer-quantized Acurracy	74.9%	75.4%



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

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

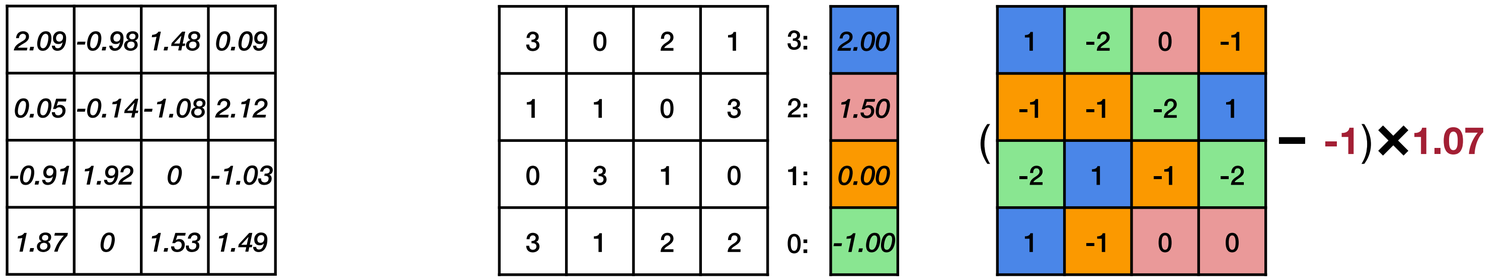
Neural Network Quantization



		K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

→ ?

Neural Network Quantization



		K-Means-based Quantization	Linear Quantization	Binary/Ternary Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights	Binary/Ternary Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic	Bit Operations

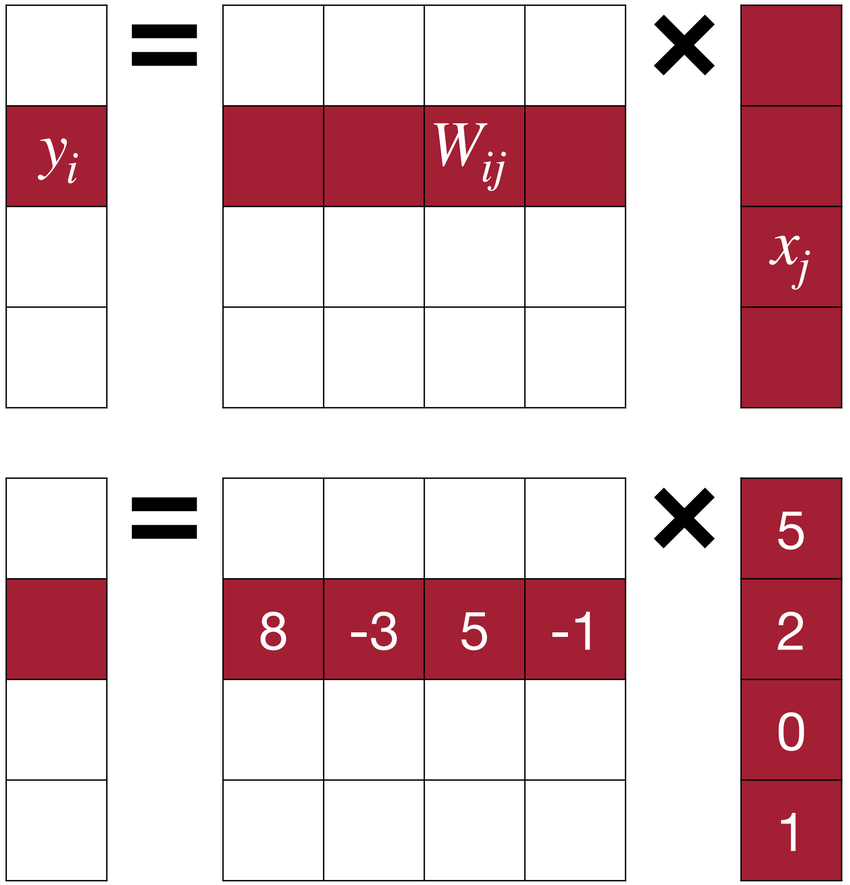
Binary/Ternary Quantization

Can we push the quantization precision to 1 bit?

Can quantization bit width go even lower?

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

$$= 8\times5 + (-3)\times2 + 5\times0 + (-1)\times1$$

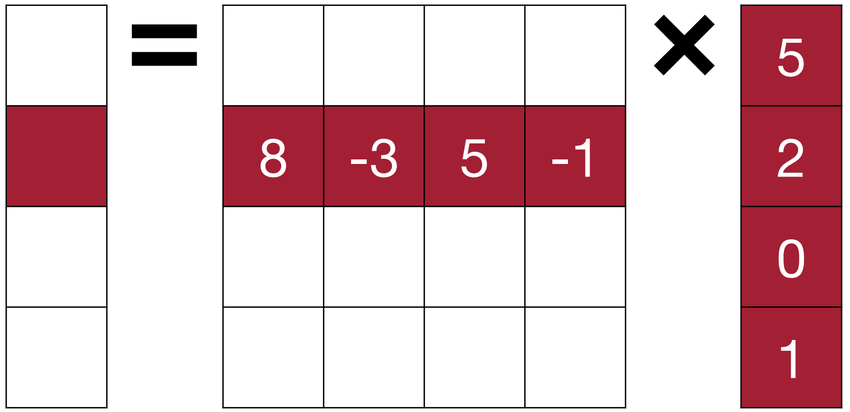


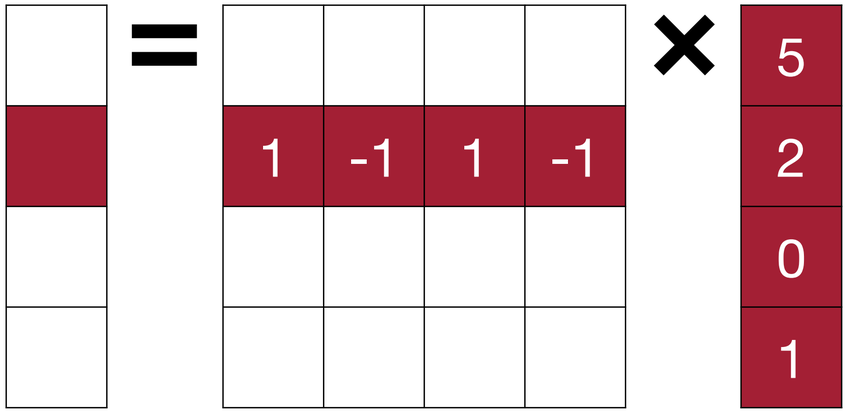
input	weight	operations	memory	computation
$\mathbb{R}$	$\mathbb{R}$	+ ×	1×	1×

If weights are quantized to +1 and -1

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

$$= 5 - 2 + 0 - 1$$





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

BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations [Courbariaux *et al.*, NeurIPS 2015]
XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

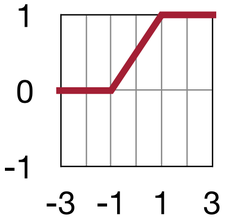
Binarization

- **Deterministic Binarization**
  - directly computes the bit value based on a threshold, usually 0, resulting in a sign function.

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

- **Stochastic Binarization**
  - use global statistics or the value of input data to determine the probability of being -1 or +1
    - *e.g.*, in Binary Connect (BC), probability is determined by hard sigmoid function $\sigma(r)$

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



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

BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations [Courbariaux *et al.*, NeurIPS 2015]
BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or –1. [Courbariaux *et al.*, Arxiv 2016]

Minimizing Quantization Error in Binarization

weights
(32-bit float)

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*

$\mathbf{W}$ → $\mathbf{W}^{\mathbb{B}}$

binary weights
(1-bit)

1	-1	1	1
1	-1	-1	1
-1	1	1	-1
1	1	1	1

$$\mathbf{W}^{\mathbb{B}} = \text{sign}(\mathbf{W})$$

$$\alpha = \frac{1}{n}\|\mathbf{W}\|_1$$

$\alpha\mathbf{W}^{\mathbb{B}}$

1	-1	1	1
1	-1	-1	1
-1	1	1	-1
1	1	1	1

$$\|\mathbf{W} - \mathbf{W}^{\mathbb{B}}\|_F^2 = 9.28$$

scale
(32-bit float)

× **1.05** $= \frac{1}{16}\|\mathbf{W}\|_1$

$$\|\mathbf{W} - \alpha\mathbf{W}^{\mathbb{B}}\|_F^2 = 9.24$$

AlexNet-based Network	ImageNet Top-1 Accuracy Delta
BinaryConnect	-21.2%
Binary Weight Network (BWN)	0.2%

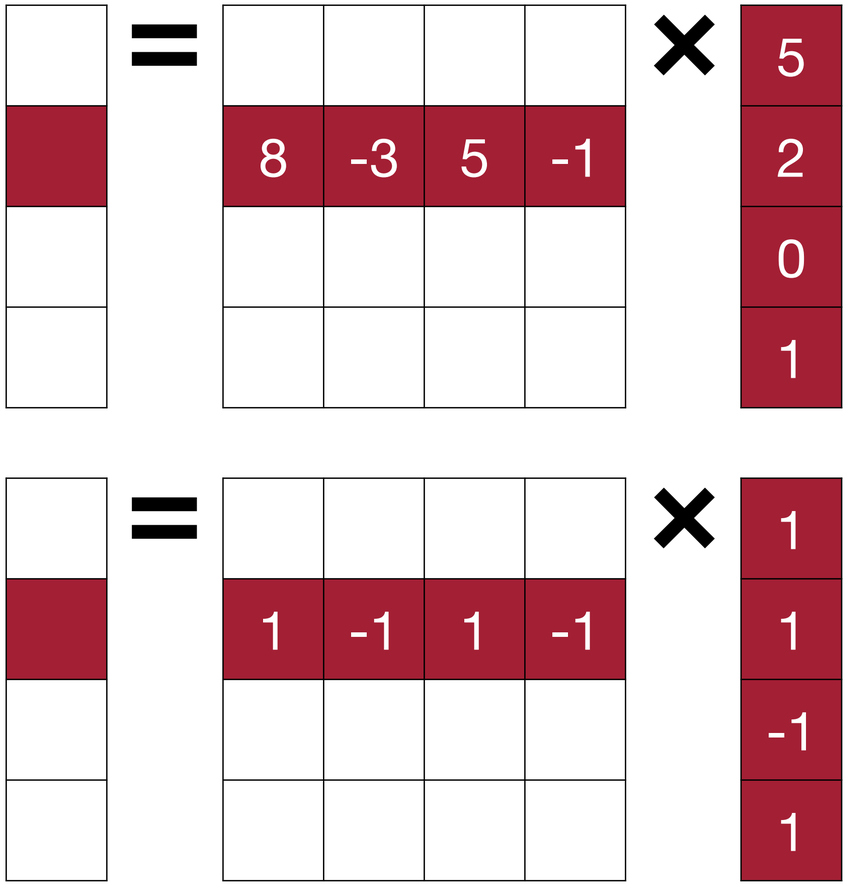
XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

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

$$= 1\times1 + (-1)\times1 + 1\times(-1) + (-1)\times1$$

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



XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

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

$$= 1\times1 + (-1)\times1 + 1\times(-1) + (-1)\times1$$

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

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

b_W	b_X	XNOR(b_W, b_X)
1	1	1
1	0	0
0	0	1
0	1	0

XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

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

= 1×1 + (-1)×1 + 1×(-1) + (-1)×1 = -2

= 1 xnor 1 + 0 xnor 1 + 1 xnor 0 + 0 xnor 1

= 1 + 0 + 0 + 0 = 1

?

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

b_W	b_X	XNOR(b_W, b_X)
1	1	1
1	0	0
0	0	1
0	1	0

XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

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

$$= 1\times1 + (-1)\times1 + 1\times(-1) + (-1)\times1$$

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

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

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

$$= 1 + 0 + 0 + 0 = 1 \times 2 + (+2) = -4 = -2$$

Assuming -1 -1 -1 -1 →

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

b_W	b_X	XNOR(b_W, b_X)
1	1	1
1	0	0
0	0	1
0	1	0

XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

$$y_i = -n + 2 \cdot \sum_j W_{ij} \text{ xnor } x_j \quad \rightarrow \quad y_i = -n + \text{popcount}\left(W_i \text{ xnor } x\right) \ll 1$$

= -4 + 2 × (1 xnor 1 + 0 xnor 1 + 1 xnor 0 + 0 xnor 1)

= -4 + 2 × (1 + 0 + 0 + 0) = -2

→ popcount: return the number of 1

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

b_W	b_X	XNOR(b_W, b_X)
1	1	1
1	0	0
0	0	1
0	1	0

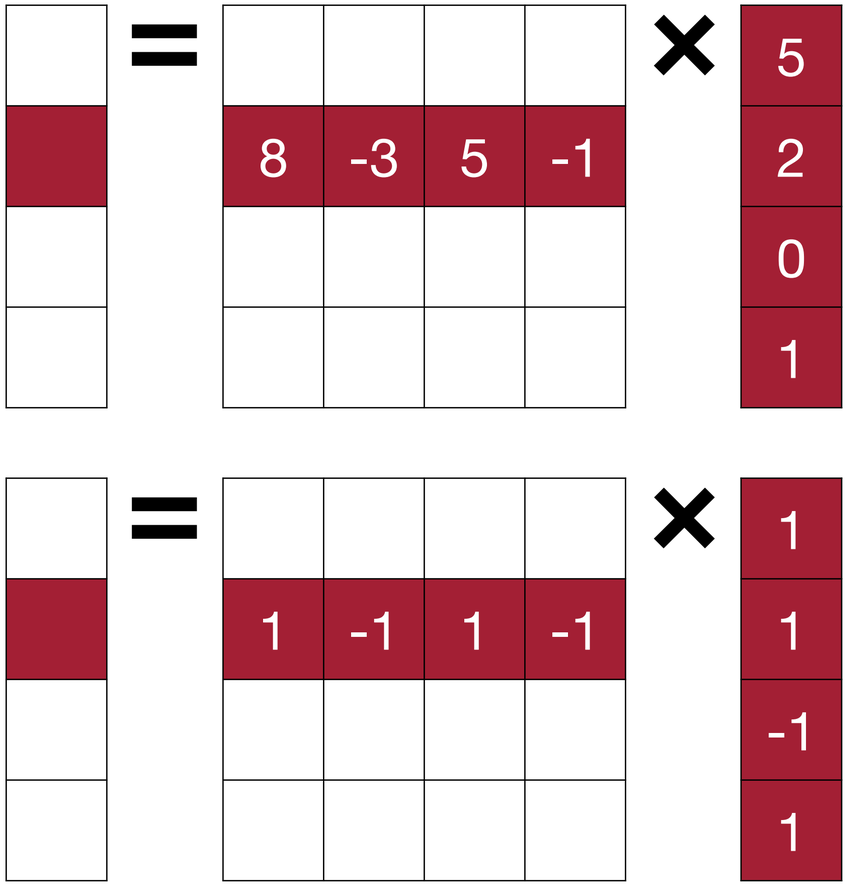
XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

$$y_i = -n + \text{popcount}\left(W_i \text{ xnor } x\right) \ll 1$$

= -4 + popcount(1010 xnor 1101) ≪ 1

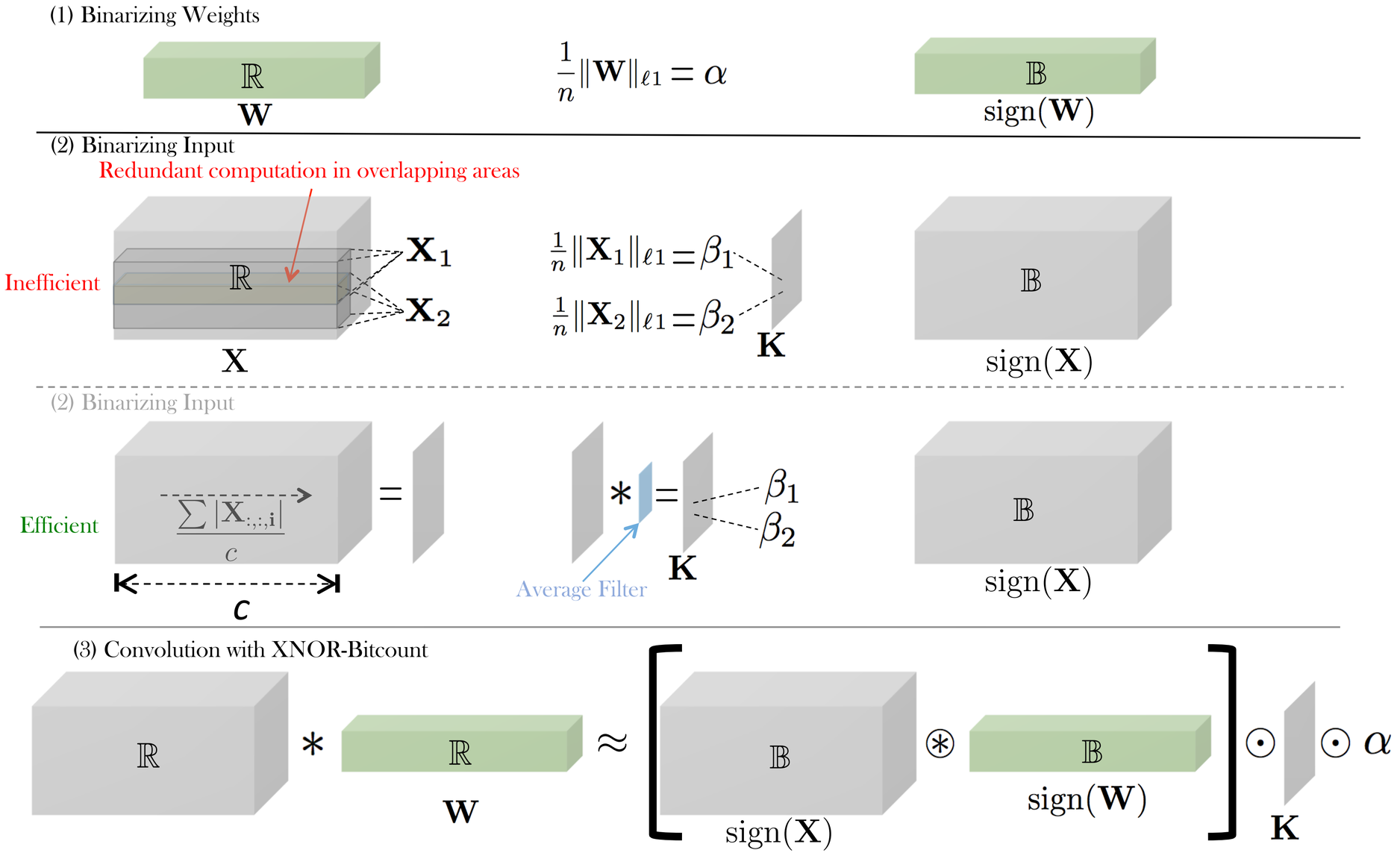
= -4 + popcount(1000) ≪ 1 = -4 + 2 = -2



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

XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

Minimizing Quantization Error in Binarization



This slide is from XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

Accuracy Degradation of Binarization

Neural Network	Quantization	Bit-Width		ImageNet Top-1 Accuracy Delta
		W	A	
AlexNet	BWN	1	32	0.2%
	BNN	1	1	-28.7%
	XNOR-Net	1	1	-12.4%
GoogleNet	BWN	1	32	-5.80%
	BNN	1	1	-24.20%
ResNet-18	BWN	1	32	-8.5%
	XNOR-Net	1	1	-18.1%

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

Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or –1. [Courbariaux *et al.*, Arxiv 2016]
XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

Ternary Weight Networks (TWN)

Weights are quantized to +1, -1 and 0

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

weights $\mathbf{W}$ (32-bit float)

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*

→

ternary weights $\mathbf{W}^{\mathbb{T}}$ (2-bit)

1	*-1*	*1*	*0*
0	*0*	*-1*	*1*
-1	*1*	*0*	*-1*
1	*0*	*1*	*1*

$$\Delta = 0.7 \times \frac{1}{16}\|\mathbf{W}\|_1 = 0.73$$

$$\times \quad \mathbf{1.5} = \frac{1}{11}\|\mathbf{W}_{\mathbf{W}^{\mathbb{T}} \neq 0}\|_1$$

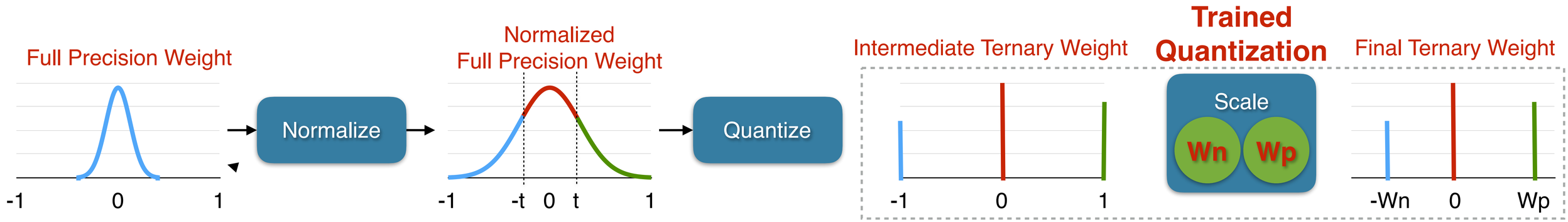
ImageNet Top-1 Accuracy	Full Precision	1 bit (BWN)	2 bit (TWN)
ResNet-18	69.6	60.8	65.3

Ternary Weight Networks [Li *et al.*, Arxiv 2016]

Trained Ternary Quantization (TTQ)

- Instead of using fixed scale r_t, TTQ introduces two *trainable* parameters w_p and w_n to represent the positive and negative scales in the quantization.

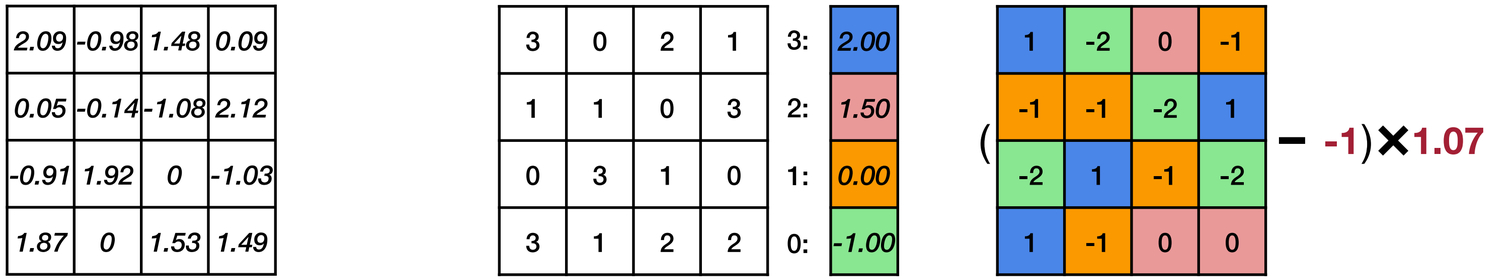
$$q = \begin{cases} w_p, & r > \Delta \\ 0, & |r| \leq \Delta \\ -w_n, & r < -\Delta \end{cases}$$



ImageNet Top-1 Accuracy	Full Precision	1 bit (BWN)	2 bit (TWN)	TTQ
ResNet-18	69.6	60.8	65.3	66.6

Trained Ternary Quantization [Zhu *et al.*, ICLR 2017]

Neural Network Quantization



		K-Means-based Quantization	Linear Quantization	Binary/Ternary Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights	Binary/Ternary Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic	Bit Operations

Low Bit-Width Quantization

How should we push quantization to 4 bits and even lower bit width?

Quantization to 4 bits and lower bit widths

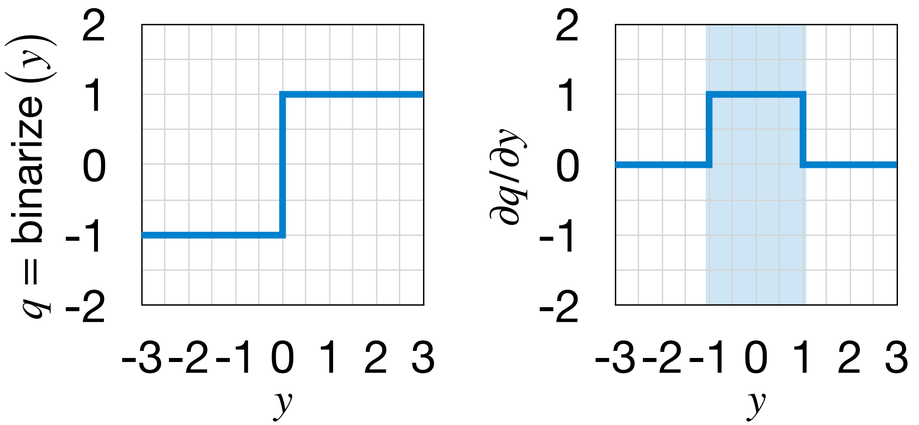
Naive low-bit-width quantization incurs significant accuracy accuracy drop

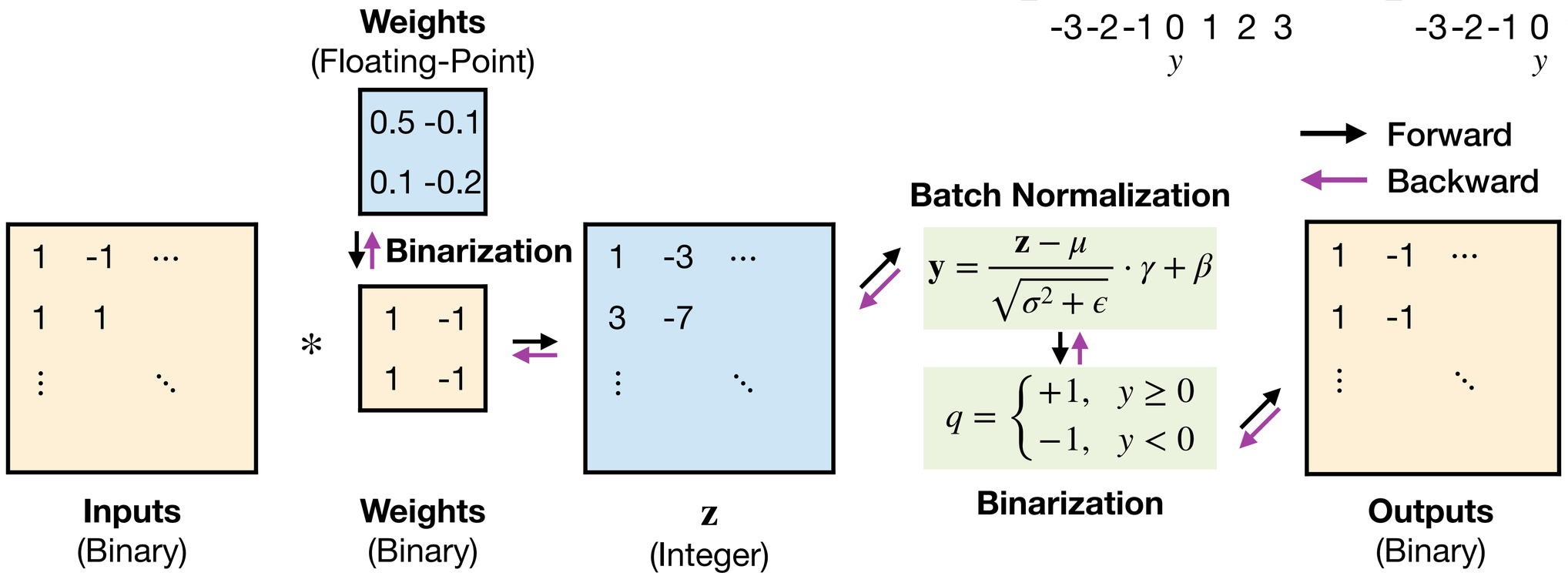
- **Quantization-Aware Training**

Quantization [Neural Network Distiller]

Train Binarized Neural Networks From Scratch

- **STE with saturation effect**
  - preserves the gradients' information and cancels the gradients when y is too large
- **Clip** the floating-point weights copy within the [-1, 1]
  - bound weights for regularization





Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or –1. [Courbariaux et al., Arxiv 2016]

DoReFa-Net with Low Bit-Width Gradients

- ***k*-bit Quantization Function in DoReFa-Net**
  - $quantize_k(r) = \frac{1}{2^k - 1} round\left((2^k - 1)r\right)$
- **Activation Quantization**
  - $Q(x) = quantize_k\left(x_f\right)$, x_f is clipped to $[0,1]$
- **Weight Quantization**
  - $$Q(w) = 2 \cdot \left[quantize_k \left(\frac{\tanh\left(w_f\right)}{2 \cdot \max\left(\left|\tanh\left(\mathbf{W}_f\right)\right|\right)} + \frac{1}{2} \right) - \frac{1}{2} \right], \quad k > 1$$

limit to [-1, 1]

limit to [-1, 1]

limit to [0, 1]

$quantize_k(r)$

$quantize_k(r) = \frac{1}{3}q$, where q is 2-bit unsigned integer

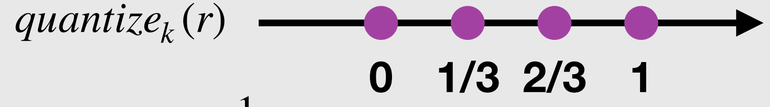
$Q(w)$

$Q(w) = \frac{2}{3}q_w - 1$, where q_w is 2-bit unsigned integer

DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients [Zhou *et al.*, arXiv 2016]

DoReFa-Net with Low Bit-Width Gradients

- ***k*-bit Quantization Function in DoReFa-Net**
  - $quantize_k(r) = \frac{1}{2^k - 1} round\left((2^k - 1)r\right)$
- **Activation Quantization**
  - $Q(x) = quantize_k\left(x_f\right)$, x_f is clipped to $[0,1]$
- **Weight Quantization**
  - $$Q(w) = 2 \cdot \left[quantize_k\left(\frac{\tanh\left(w_f\right)}{2 \cdot \max\left(\left|\tanh\left(\mathbf{W}_f\right)\right|\right)} + \frac{1}{2} \right) - \frac{1}{2} \right], \quad k > 1$$



$quantize_k(r) = \frac{1}{3}q$, where q is 2-bit unsigned integer

$Q(w) = \frac{2}{3}q_w - 1$, where q_w is 2-bit unsigned integer

- **Gradient Quantization**
  - $Q(g) = 2 \cdot \max(|\mathbf{G}|) \cdot \left[quantize_k\left(\frac{g}{2 \cdot \max\left(|\mathbf{G}|\right)} + \frac{1}{2} + N(k) \right) - \frac{1}{2} \right]$, where $N(k) = \frac{\sigma}{2^k - 1}$ and $\sigma \sim Uniform(-0.5,0.5)$
  - Noise function $N(k)$ is added to compensate the potential bias introduced by gradient quantization.

DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients [Zhou *et al.*, arXiv 2016]

DoReFa-Net with Low Bit-Width Gradients

Weights Bit Width	Activation Bit Width	Gradient Bit Width	AlexNet Accuracy on ImageNet
32	32	32	55.9%
8	8	8	53.0%
1	2	32	49.8% (initialized)
1	2	32	47.7%
1	2	6	46.1%
1	1	32	43.6% (Initialized)
1	1	32	40.1%
1	1	6	39.5%

* initialized means training from a pre-trained floating point AlexNet

DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients [Zhou *et al.*, arXiv 2016]

Quantization to 4 bits and lower bit widths

Naive low-bit-width quantization incurs significant accuracy accuracy drop

- Quantization-Aware Training
- **Replace the Activation Function**
  - The most common activation function ReLU is unbounded. The dynamic range of inputs becomes problematic for low bit-width quantization due to very limited range and resolution.
  - ReLU is replaced with hard-coded bounded activation functions: ReLU6, ReLU1, *etc*
  - The clipping value per layer can be learned as well: PACT

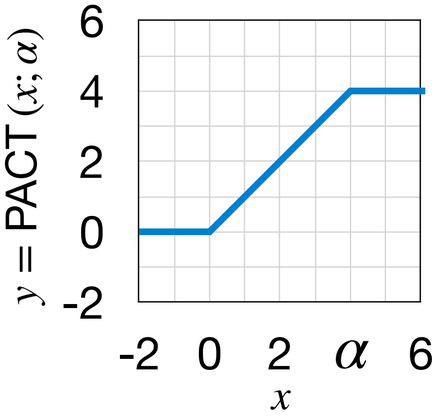
PACT: Parameterized Clipping Activation for Quantized Neural Networks [Choi *et al.*, arXiv 2018]
Quantization [Neural Network Distiller]

Parameterized Clipping Activation Function

Learn clipping value per layer

- In PACT, the ReLU activation function is replaced with

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



- The upper clipping value of the activation function is a trainable.
  - With STE, the gradient is computed as

$$\frac{\partial Q(\mathbf{y})}{\partial \alpha} = \frac{\partial Q(\mathbf{y})}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \alpha} = \begin{cases} 0 & x \in (-\infty, \alpha) \\ 1 & x \in [\alpha, +\infty) \end{cases} \quad \rightarrow \quad \frac{\partial L}{\partial \alpha} = \frac{\partial L}{\partial Q(\mathbf{y})} \cdot \frac{\partial Q(\mathbf{y})}{\partial \alpha} = \begin{cases} 0 & x \in (-\infty, \alpha) \\ \frac{\partial L}{\partial Q(\mathbf{y})} & x \in [\alpha, +\infty) \end{cases}$$

- The larger the α, the more the parameterized clipping function resembles a ReLU function.
  - To avoid large quantization errors due to a wide dynamic range $[0,\alpha]$, L2-regularizer for α is included in the training loss function.

PACT: Parameterized Clipping Activation for Quantized Neural Networks [Choi *et al.*, arXiv 2018]

Parameterized Clipping Activation Function

Learn clipping value per layer

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%

PACT: Parameterized Clipping Activation for Quantized Neural Networks [Choi *et al.*, arXiv 2018]

Quantization to 4 bits and lower bit widths

Naive low-bit-width quantization incurs significant accuracy accuracy drop

- **Quantization-Aware Training**
- **Replace the Activation Function**
  - The most common activation function ReLU is unbounded. The dynamic range of inputs becomes problematic for low bit-width quantization due to very limited range and resolution.
  - ReLU is replaced with hard-coded bounded activation functions: ReLU6, ReLU1, *etc*
  - The clipping value per layer can be learned as well: PACT
- **Modify the Neural Network Architecture**
  - Widen the neural network to compensate for the loss of information due to quantization

WRPN: Wide Reduced-Precision Networks [Mishra *et al.*, ICLR 2018]
Quantization [Neural Network Distiller]

Wide Reduced-Precision Networks

Double the channels, reduce the quantization precision

ResNet-32 Width	Precision		ImageNet Top-1 Accuracy	Compute Cost
	Activation	Weight		
1× wide	32 bits	32 bits	73.59	1×
	1 bit	1 bit	60.54	0.03×
2× wide	4 bits	2 bits	73.58	0.39×
	1 bit	1 bit	69.85	0.15×
3× wide	1 bit	1 bit	72.38	0.30×

WRPN: Wide Reduced-Precision Networks [Mishra *et al.*, ICLR 2018]

Quantization to 4 bits and lower bit widths

Naive low-bit-width quantization incurs significant accuracy accuracy drop

- **Quantization-Aware Training**
- **Replace the Activation Function**
  - The most common activation function ReLU is unbounded. The dynamic range of inputs becomes problematic for low bit-width quantization due to very limited range and resolution.
  - ReLU is replaced with hard-coded bounded activation functions: ReLU6, ReLU1, *etc*
  - The clipping value per layer can be learned as well: PACT
- **Modify the Neural Network Architecture**
  - Widen the neural network to compensate for the loss of information due to quantization
  - Replace a single floating-point convolution with multiple binary convolutions

Towards Accurate Binary Convolutional Neural Network [Lin *et al.*, NeurIPS 2017]
Quantization [Neural Network Distiller]

Quantization to 4 bits and lower bit widths

Naive low-bit-width quantization incurs significant accuracy accuracy drop

- **Quantization-Aware Training**
- **Replace the Activation Function**
  - The most common activation function ReLU is unbounded. The dynamic range of inputs becomes problematic for low bit-width quantization due to very limited range and resolution.
  - ReLU is replaced with hard-coded bounded activation functions: ReLU6, ReLU1, *etc*
  - The clipping value per layer can be learned as well: PACT
- **Modify the Neural Network Architecture**
  - Widen the neural network to compensate for the loss of information due to quantization
  - Replace a single floating-point convolution with multiple binary convolutions
- **No Quantization on First and Last Layer**
  - 1) more sensitive to quantization; 2) small portion of the overall computation
  - Quantizing these layers to 8-bit integer does not reduce accuracy

Quantization [Neural Network Distiller]

Quantization to 4 bits and lower bit widths

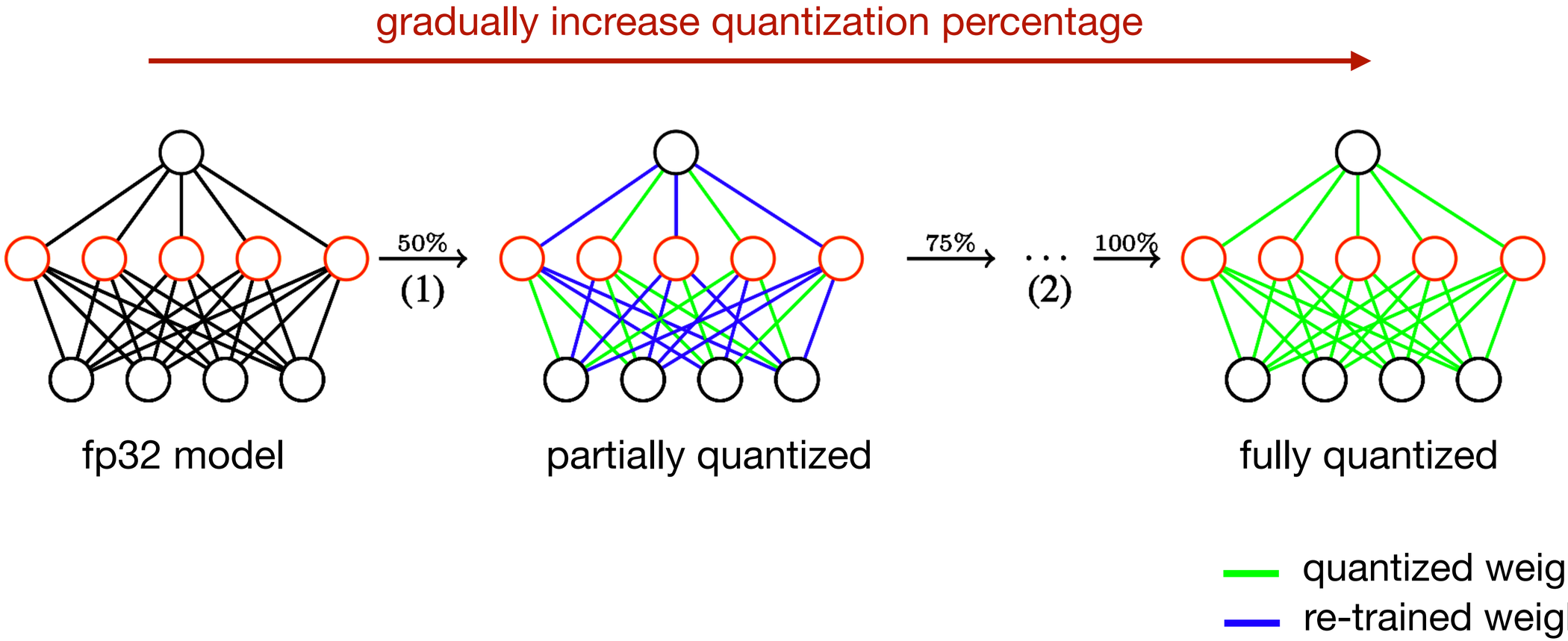
Naive low-bit-width quantization incurs significant accuracy accuracy drop

- **Quantization-Aware Training**
- **Replace the Activation Function**
  - The most common activation function ReLU is unbounded. The dynamic range of inputs becomes problematic for low bit-width quantization due to very limited range and resolution.
  - ReLU is replaced with hard-coded bounded activation functions: ReLU6, ReLU1, *etc*
  - The clipping value per layer can be learned as well: PACT
- **Modify the Neural Network Architecture**
  - Widen the neural network to compensate for the loss of information due to quantization
  - Replace a single floating-point convolution with multiple binary convolutions
- **No Quantization on First and Last Layer**
  - 1) more sensitive to quantization; 2) small portion of the overall computation
  - Quantizing these layers to 8-bit integer does not reduce accuracy
- **Iterative Quantization**

Quantization [Neural Network Distiller]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy



Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

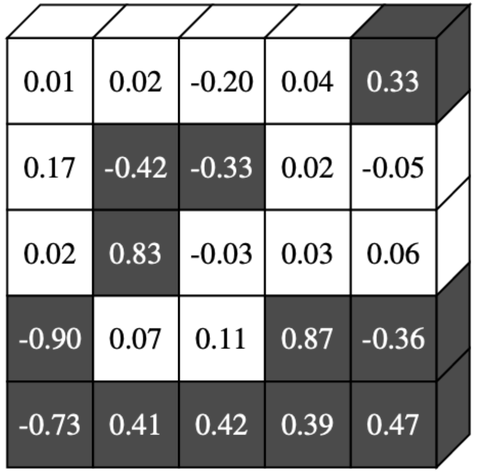
- **Setting:**
  - Weight quantization only
  - Quantize weights to 2^n for faster computation (bit shift instead of multiply)
- **Algorithm**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group to recover accuracy
  - Repeat until all the weights are quantized (a popular stride is $\{50\%, 75\%, 87.5\%, 100\%\}$)

Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

- **Algorithm (visual)**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group
  - Repeat until all the weights are quantized

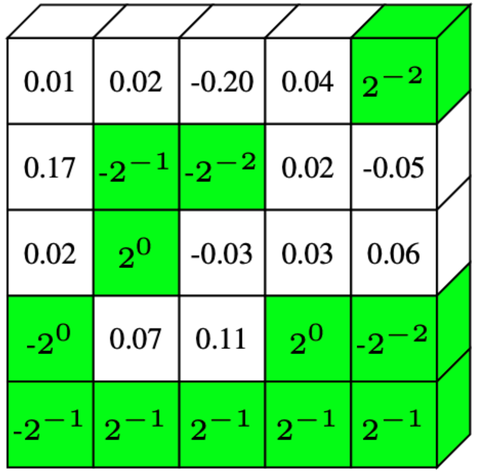


Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

- **Algorithm (visual)**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group
  - Repeat until all the weights are quantized



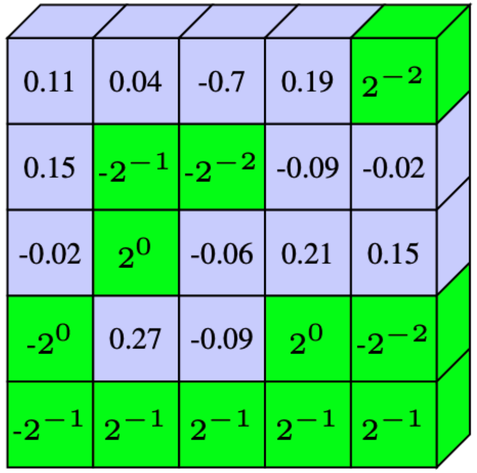
Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

- **Algorithm (visual)**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group
  - Repeat until all the weights are quantized

50% quantized



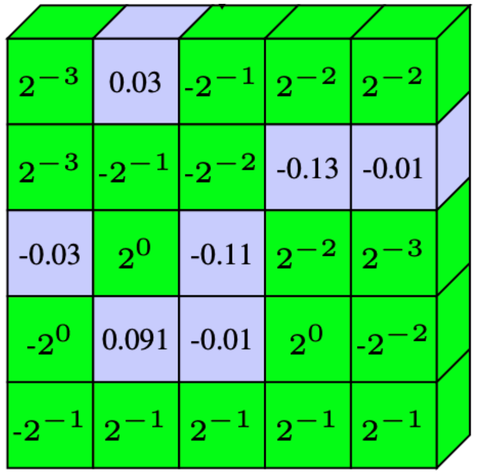
Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

- **Algorithm (visual)**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group
  - Repeat until all the weights are quantized

75% quantized



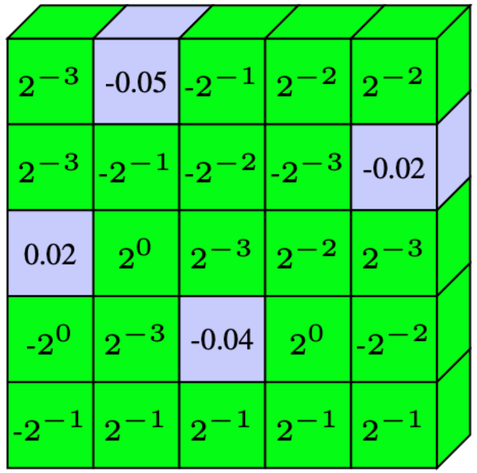
Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

- **Algorithm (visual)**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group
  - Repeat until all the weights are quantized

87.5% quantized



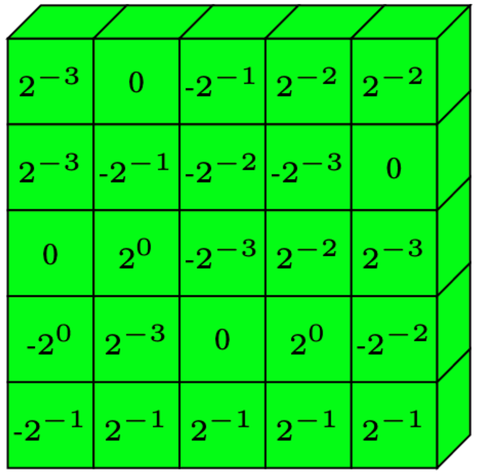
Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Incremental Network Quantization

Incrementally quantize all the weights lead to a better accuracy

- **Algorithm (visual)**
  - Start from a pre-trained fp32 model
  - For the remaining fp32 weights
    - Partition into two disjoint groups (e.g., according to magnitude)
    - Quantize the first groups (higher magnitude), and re-train the other group
  - Repeat until all the weights are quantized

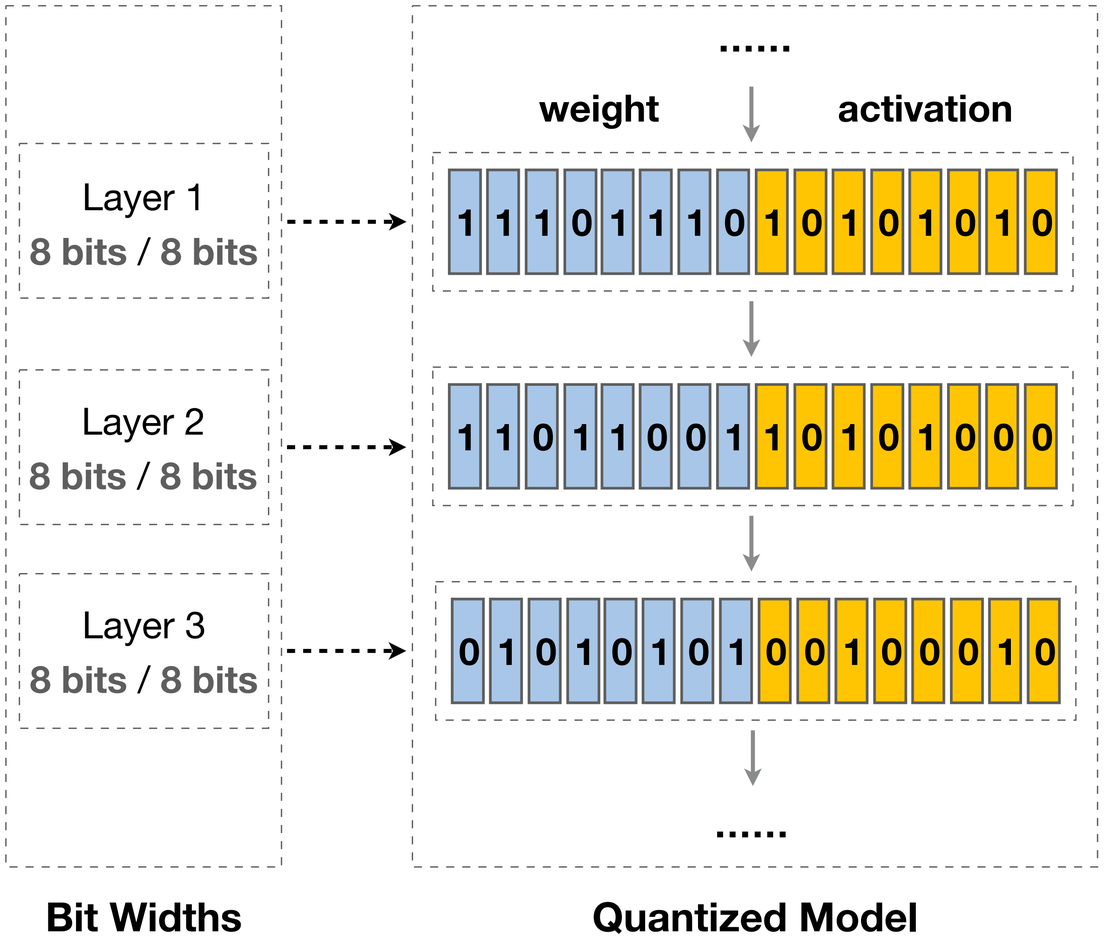
100% quantized



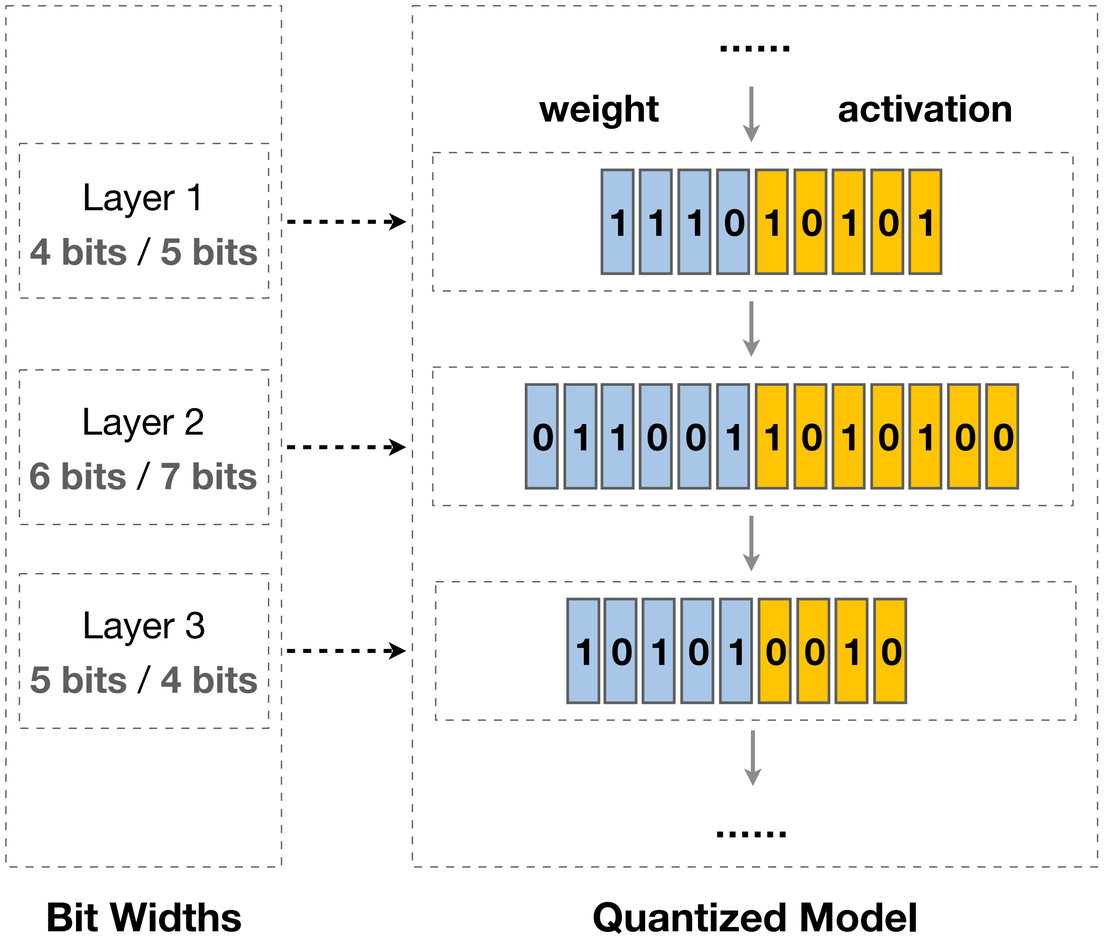
Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou *et al.*, ICLR 2017]

Mixed-Precision Quantization

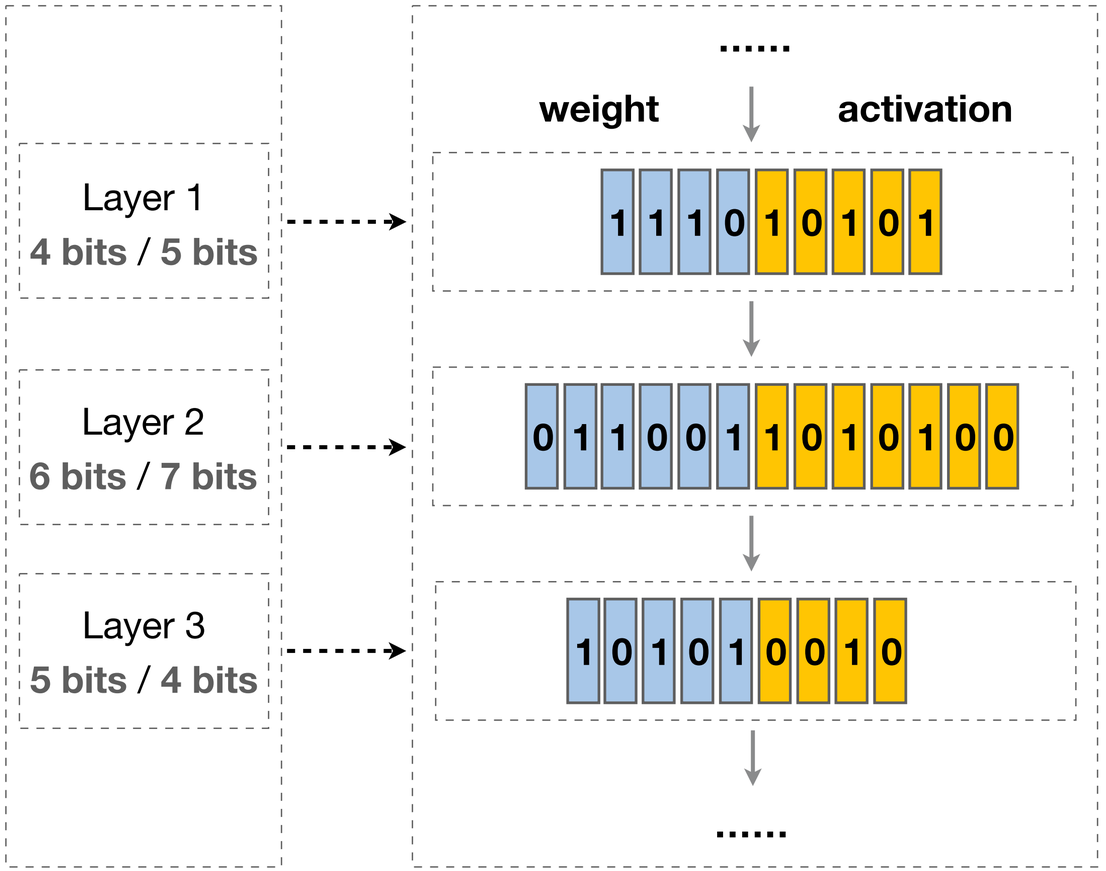
Uniform Quantization



Mixed-Precision Quantization



Challenge: Huge Design Space



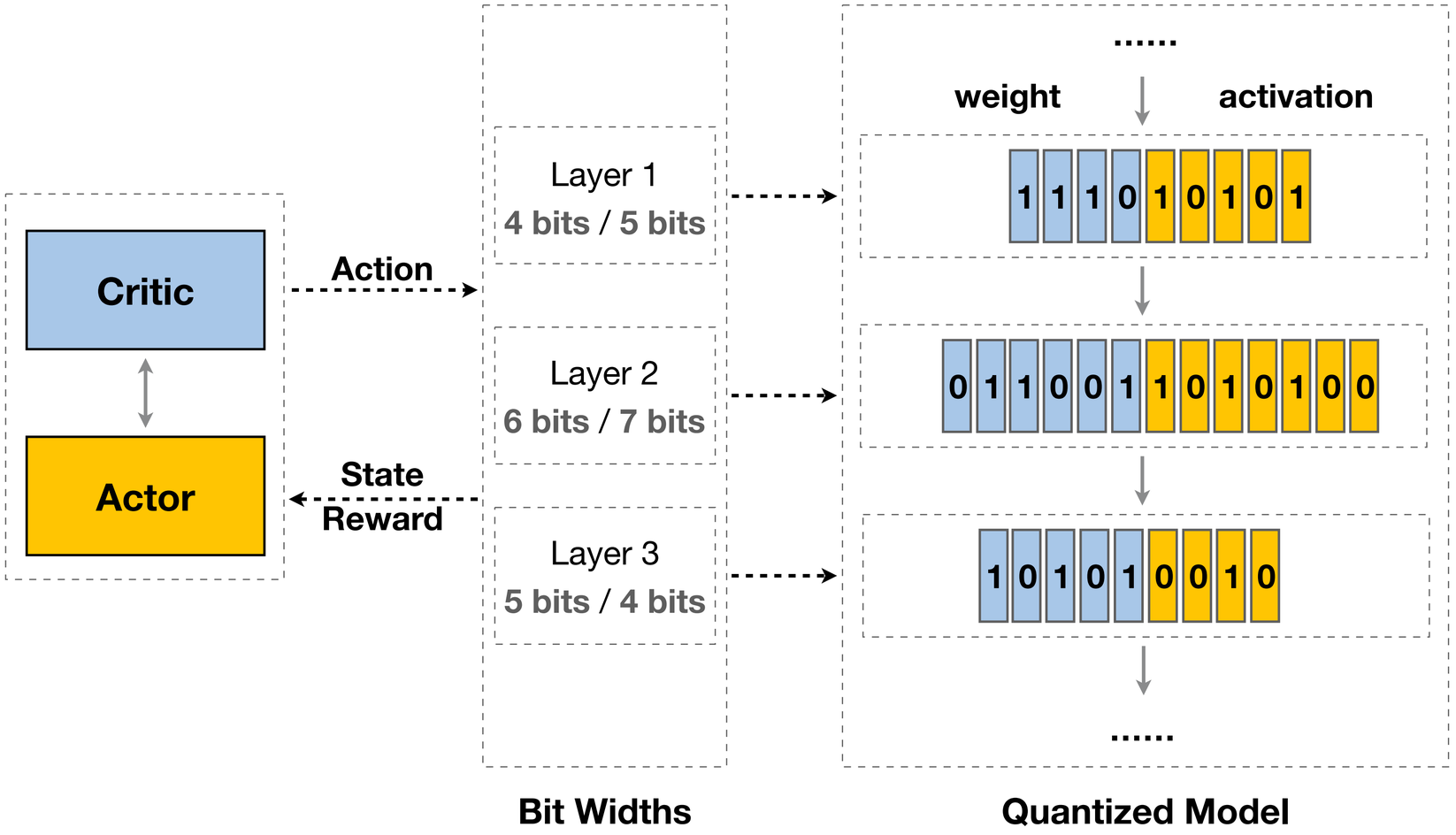
Choices: **8 x 8 = 64**

Choices: **8 x 8 = 64**

Choices: **8 x 8 = 64**

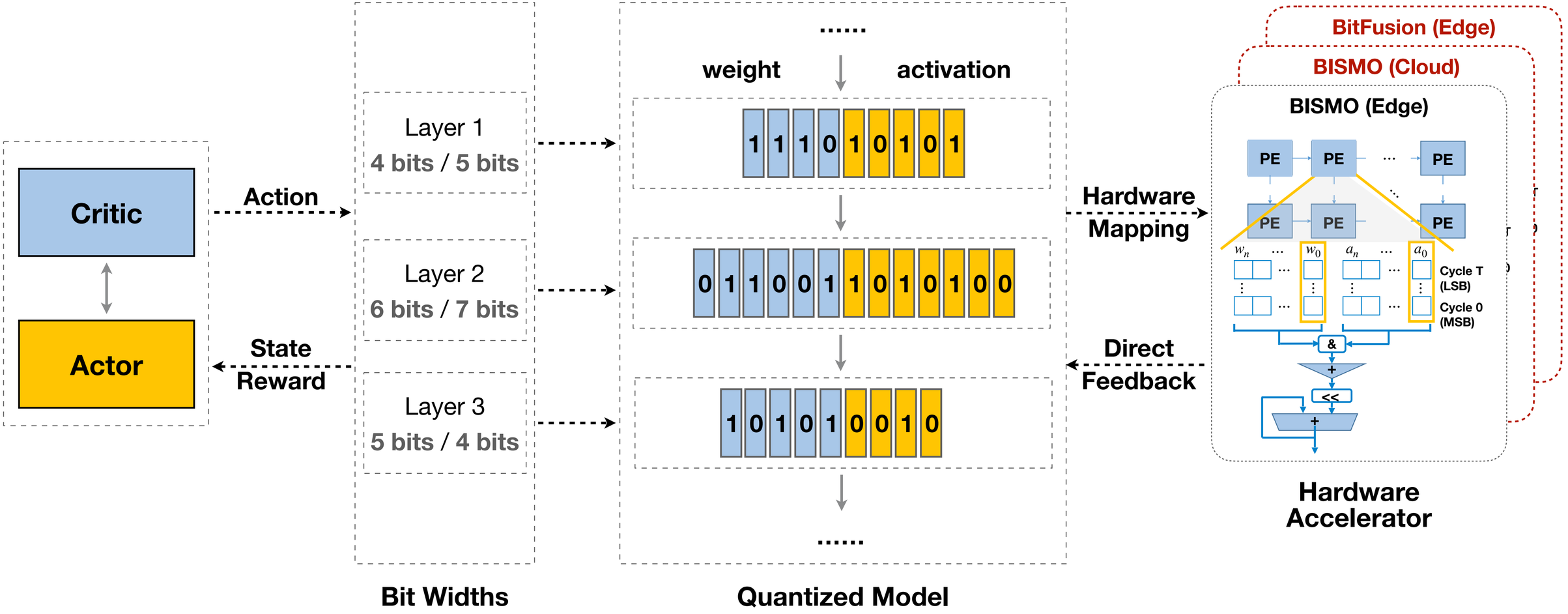
Design Space: 64^n

Solution: Design Automation



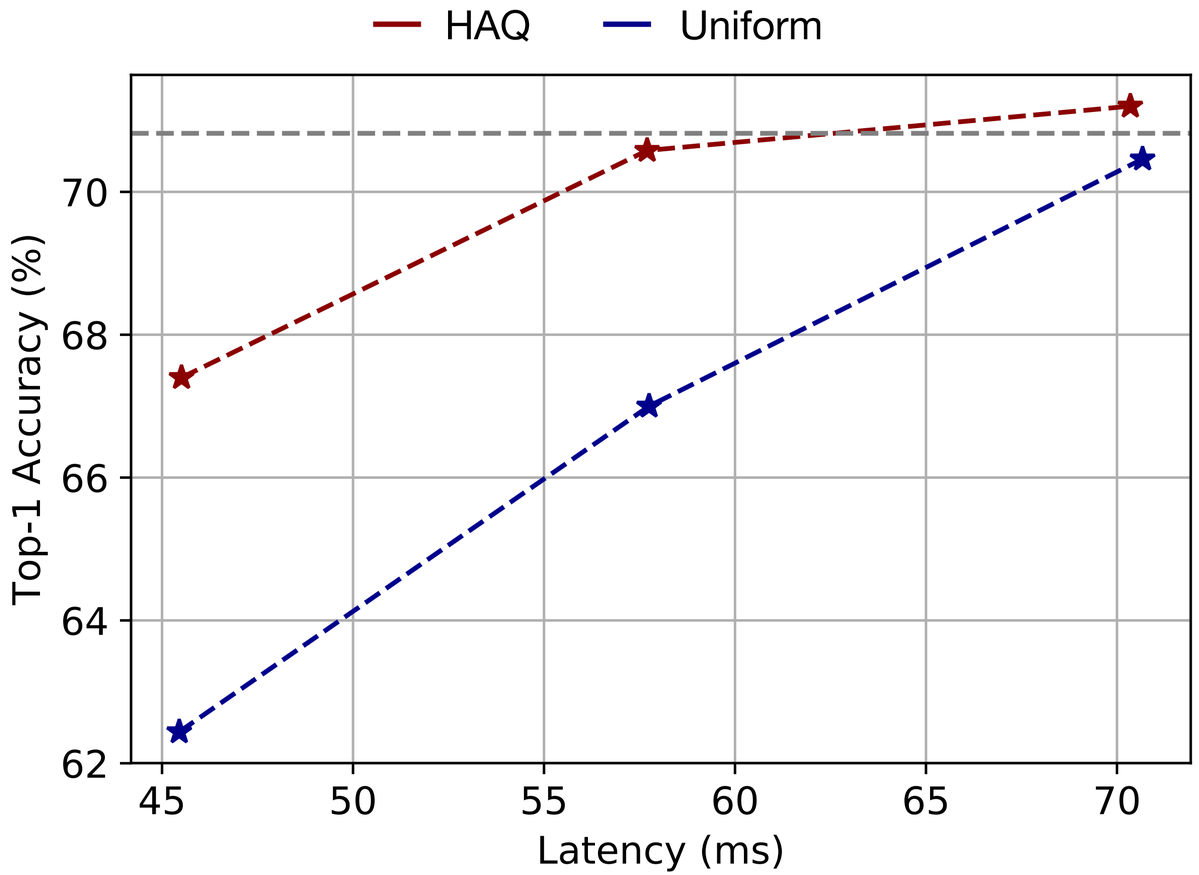
HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

Solution: Design Automation



HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

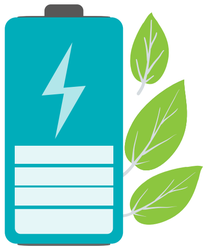
HAQ Outperforms Uniform Quantization



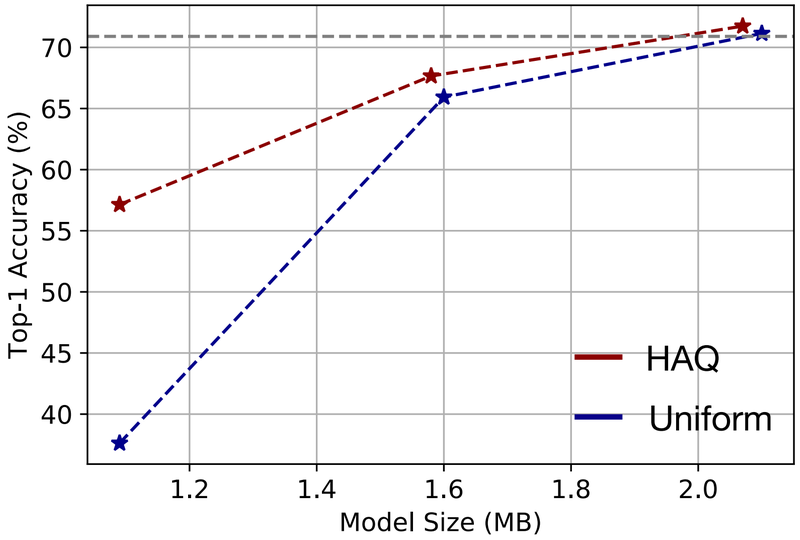
Mixed-Precision Quantized MobileNetV1

HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

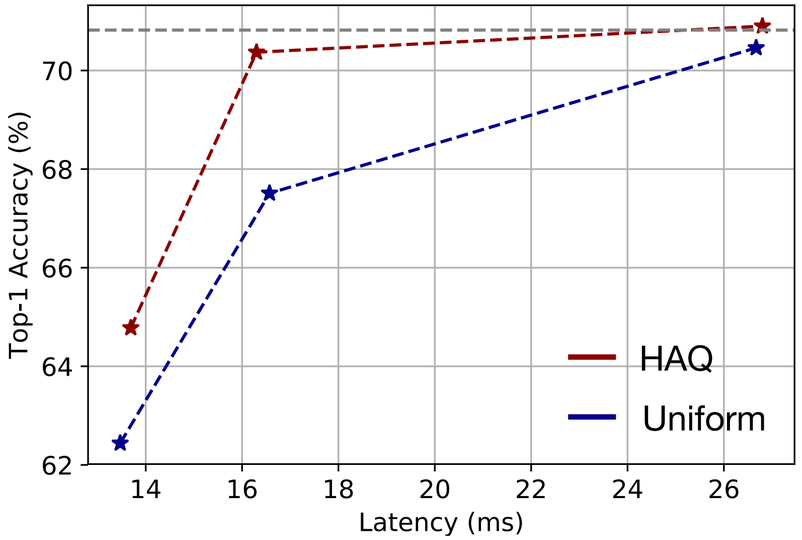
HAQ Supports Multiple Objectives



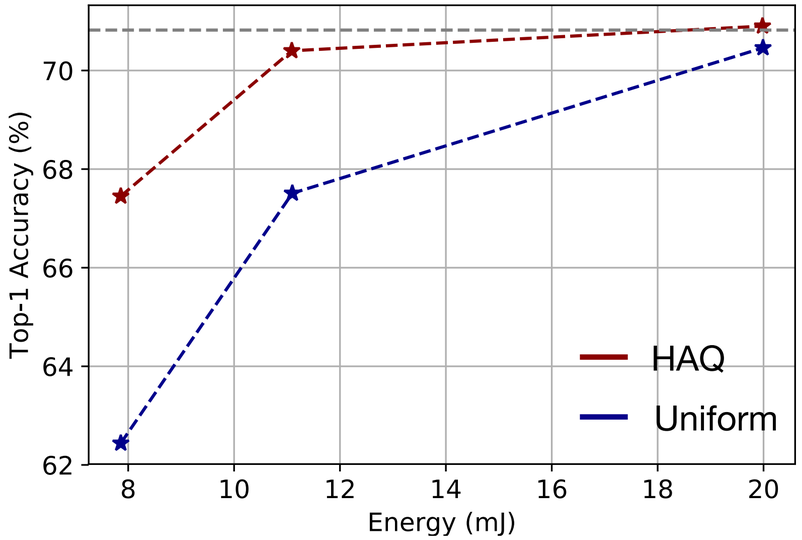
Model Size Constrained



Latency Constrained



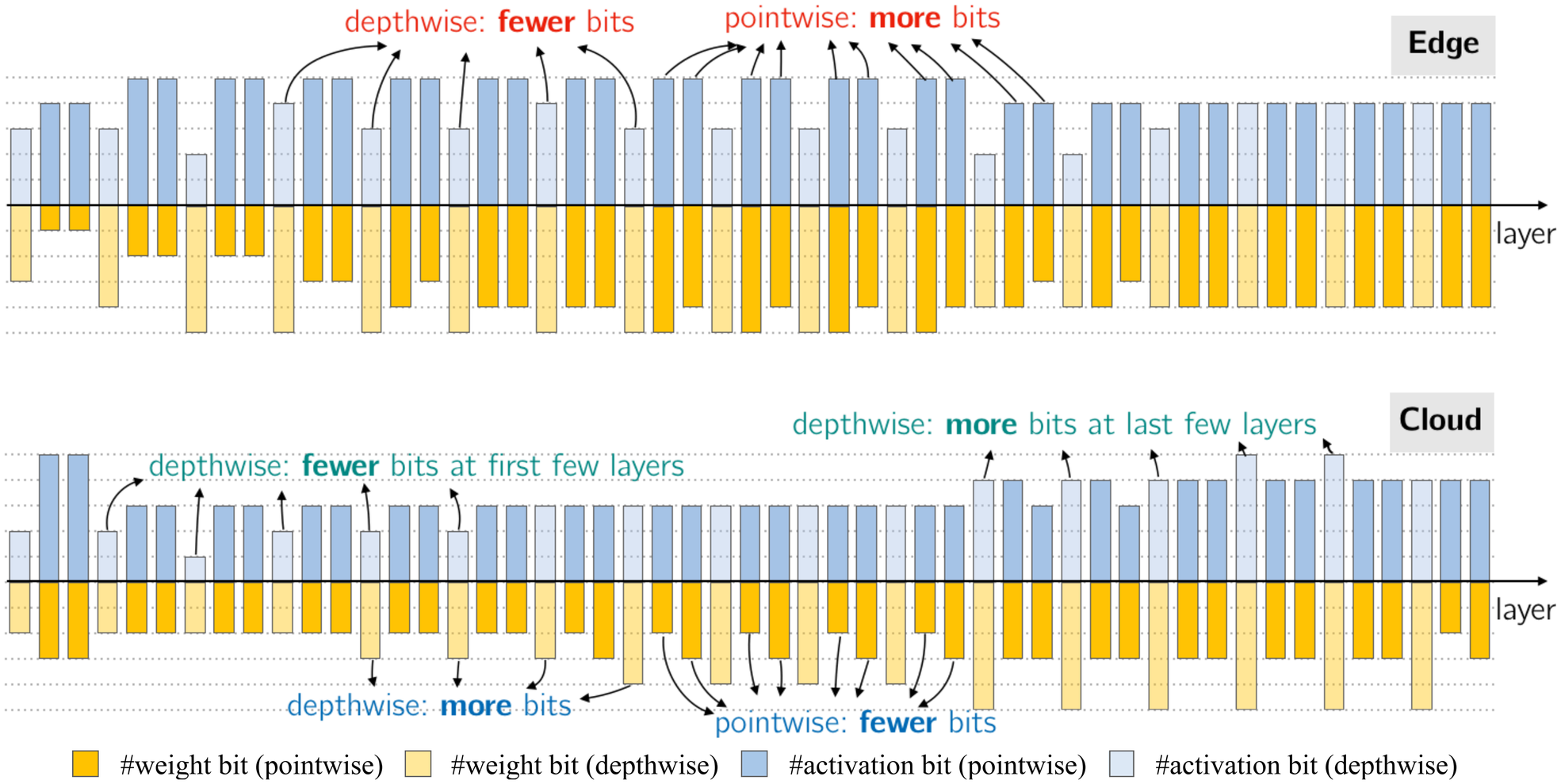
Energy Constrained



Mixed-Precision Quantized MobileNetV1

HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

Quantization Policy for Edge and Cloud



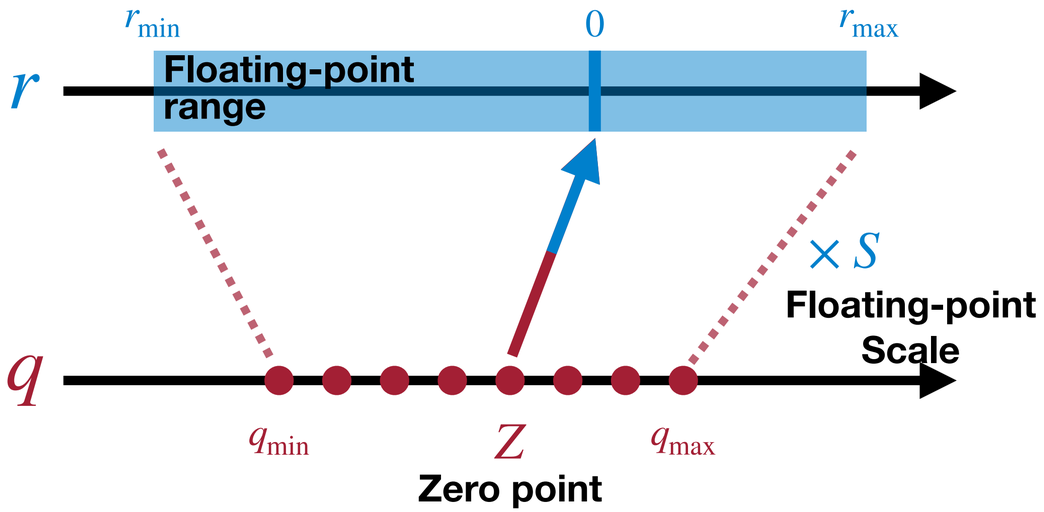
Mixed-Precision Quantized MobileNetV2

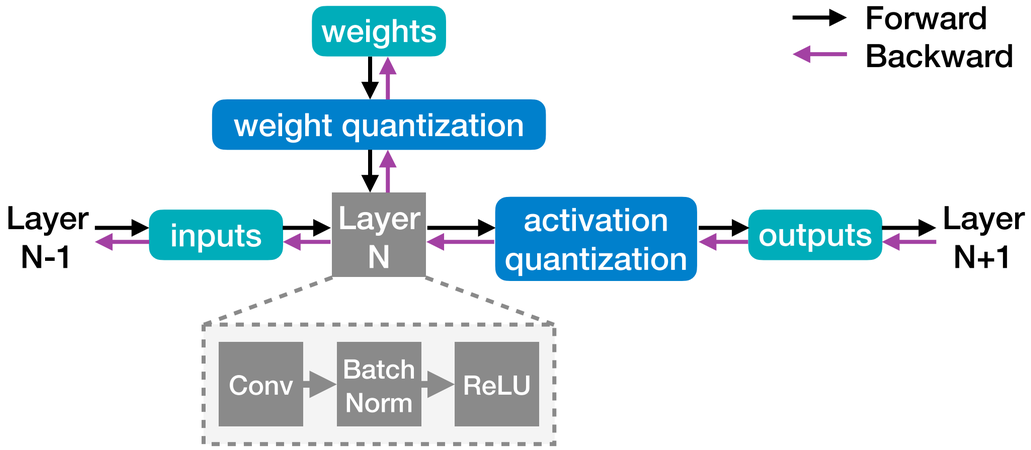
HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

Summary of Today's Lecture

In this lecture, we

1. Reviewed Linear Quantization.
2. Introduced **Post-Training Quantization (PTQ)** that quantizes an already-trained floating-point neural network model.
   - Per-tensor *vs.* per-channel weight quantization
   - How to determine dynamic range of activations
3. Introduced **Quantization-Aware Training (QAT)** that emulates inference-time quantization during the training/fine-tuning.
   - Straight-Through Estimator (STE)
4. Introduced **binary and ternary** quantization.
5. Introduced automatic **mixed-precision** quantization.





References

1. Deep Compression [Han et al., ICLR 2016]
2. Neural Network Distiller: https://intellabs.github.io/distiller/algo_quantization.html
3. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]
4. Data-Free Quantization Through Weight Equalization and Bias Correction [Markus et al., ICCV 2019]
5. Post-Training 4-Bit Quantization of Convolution Networks for Rapid-Deployment [Banner et al., NeurIPS 2019]
6. 8-bit Inference with TensorRT [Szymon Migacz, 2017]
7. Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper [Raghuraman Krishnamoorthi, arXiv 2018]
8. Neural Networks for Machine Learning [Hinton et al., Coursera Video Lecture, 2012]
9. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [Bengio, arXiv 2013]
10. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or –1. [Courbariaux et al., Arxiv 2016]
11. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients [Zhou et al., arXiv 2016]
12. PACT: Parameterized Clipping Activation for Quantized Neural Networks [Choi et al., arXiv 2018]
13. WRPN: Wide Reduced-Precision Networks [Mishra et al., ICLR 2018]
14. Towards Accurate Binary Convolutional Neural Network [Lin et al., NeurIPS 2017]
15. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou et al., ICLR 2017]
16. HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang et al., CVPR 2019]