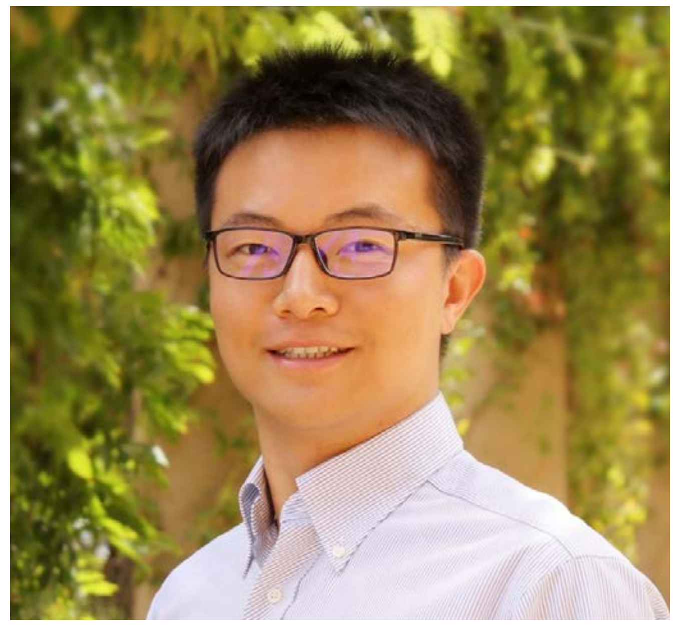
MIT HAN LAB

EfficientML.ai Lecture 05 Quantization

Part I



Song Han

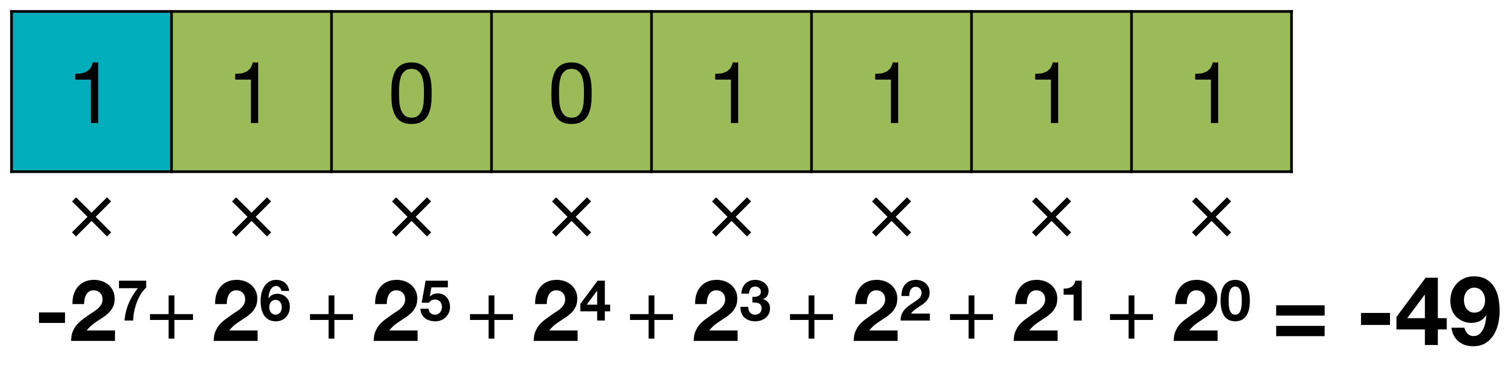
Associate Professor, MIT
Distinguished Scientist, NVIDIA

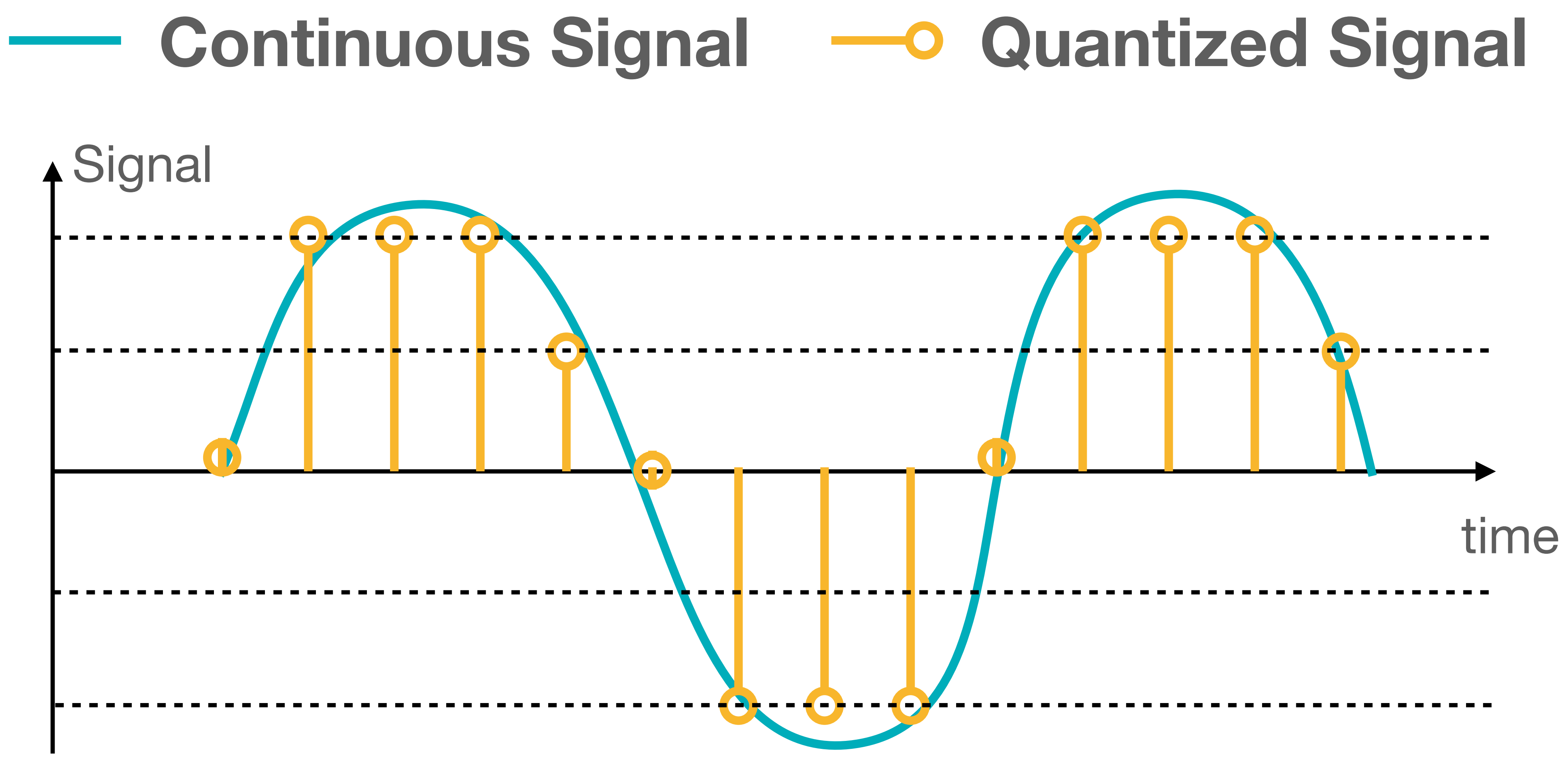
@SongHan_MIT

Lecture Plan

Today we will:

1. Review the numeric ***data types*** used in the modern computing systems, including integers and floating-point numbers.
2. Learn the basic concept of ***neural network quantization***
3. Learn three types of common neural network quantization:
    1. K-Means-based Quantization
    2. Linear Quantization
    3. Binary and Ternary Quantization

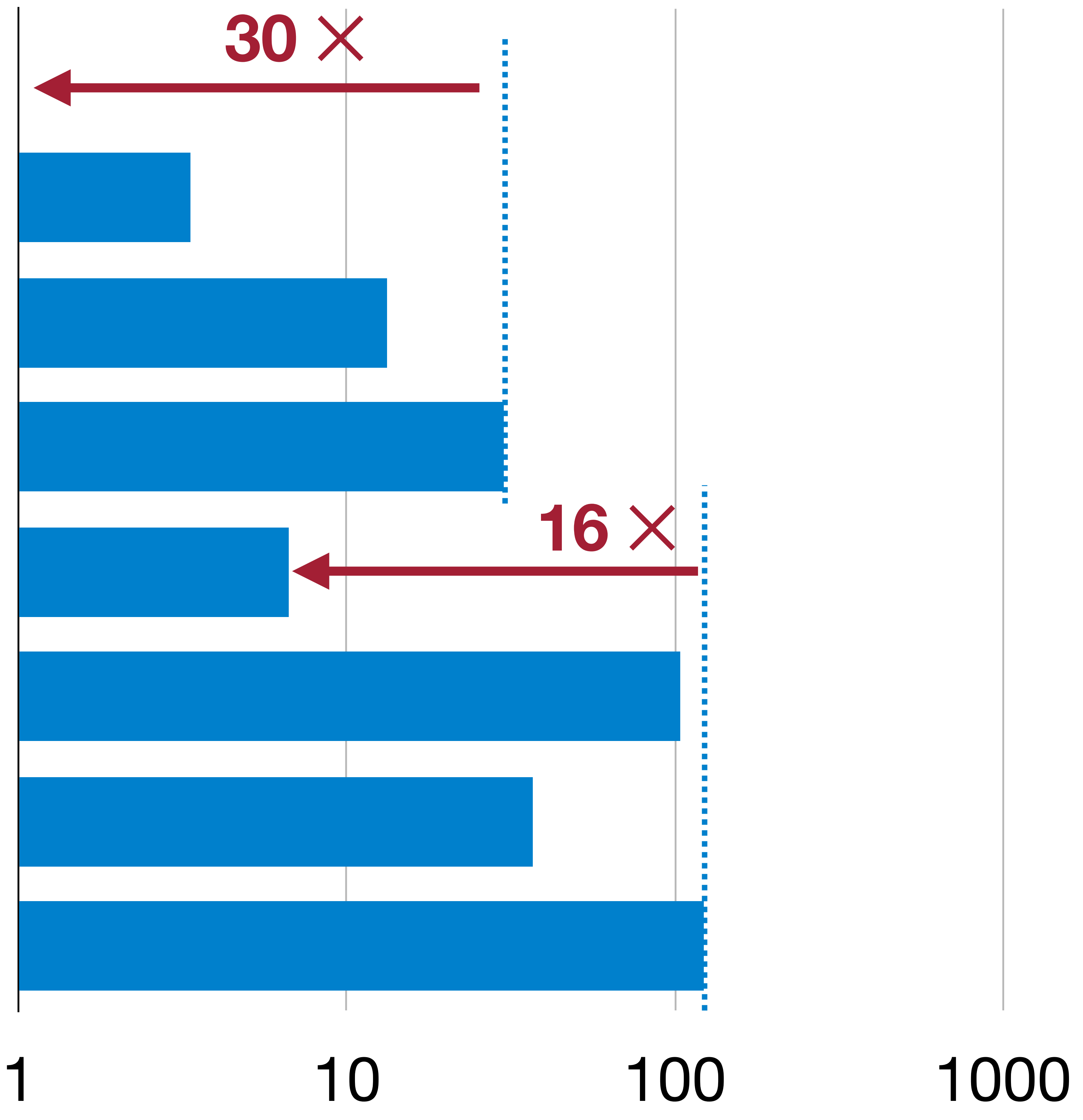




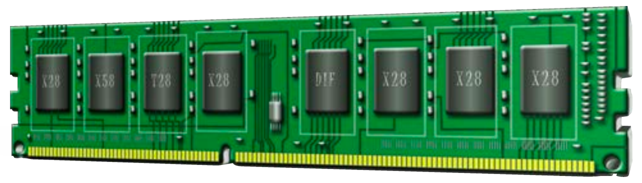
Low Bit-Width Operations are Cheap

Less Bit-Width → Less Energy

Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7



Rough Energy Cost For Various Operations in 45nm 0.9V

1  = 200 ×+

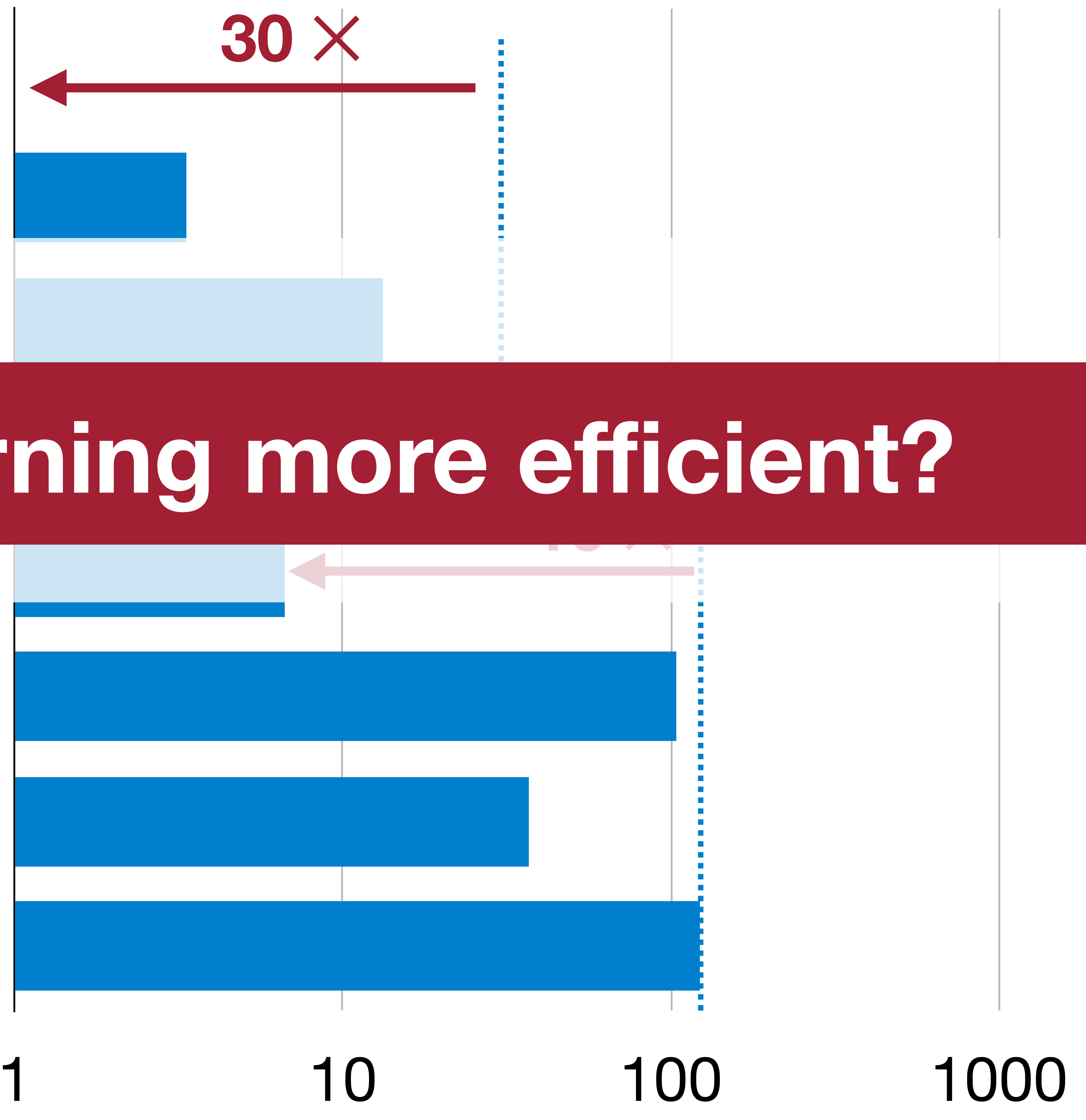
Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]

Low Bit-Width Operations are Cheap

Less Bit-Width → Less Energy

Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7

Rough Energy Cost For Various Operations in 45nm 0.9V



How should we make deep learning more efficient?

Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]

Numeric Data Types

How is numeric data represented in modern computing systems?

Integer

- Unsigned Integer
  - n-bit Range: $[0,\ 2^n - 1]$
- Signed Integer
  - Sign-Magnitude Representation
    - n-bit Range: $[-2^{n-1} - 1,\ 2^{n-1} - 1]$
    - Both 000…00 and 100…00 represent 0
  - Two's Complement Representation
    - n-bit Range: $[-2^{n-1},\ 2^{n-1} - 1]$
    - 000…00 represents 0
    - 100…00 represents -2^{n-1}

0	0	1	1	0	0	0	1
×	×	×	×	×	×	×	×

$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 49$$

Sign Bit

1	0	1	1	0	0	0	1
	×	×	×	×	×	×	×

$$- \quad 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

1	1	0	0	1	1	1	1
×	×	×	×	×	×	×	×

$$-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

Fixed-Point Number

Integer . Fraction

"Decimal" Point

0	0	1	1	0	0	0	1
×	×	×	×	×	×	×	×

$$-2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 3.0625$$

0	0	1	1	0	0	0	1
×	×	×	×	×	×	×	×

$$(-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \times 2^{-4} = 49 \times 0.0625 = 3.0625$$

(using 2's complement representation)

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754

$2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

Sign 8 bit Exponent 23 bit Fraction

$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$

Exponent Bias = 127 = 2^{8-1}-1

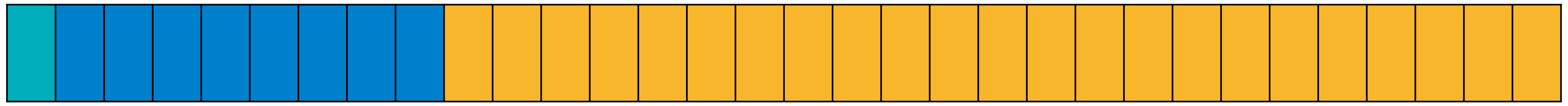
(significant / mantissa)

$$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$$

$$0\ \underbrace{0\ 1\ 1\ 1\ 1\ 1\ 0\ 1}_{125}\ \underbrace{0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{0.0625}$$

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$

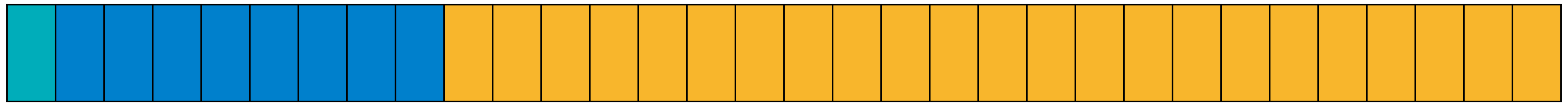
Exponent Bias = 127 = $2^{8-1}-1$

(significant / mantissa)

How should we represent 0?

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



Sign 8 bit Exponent **23 bit Fraction**

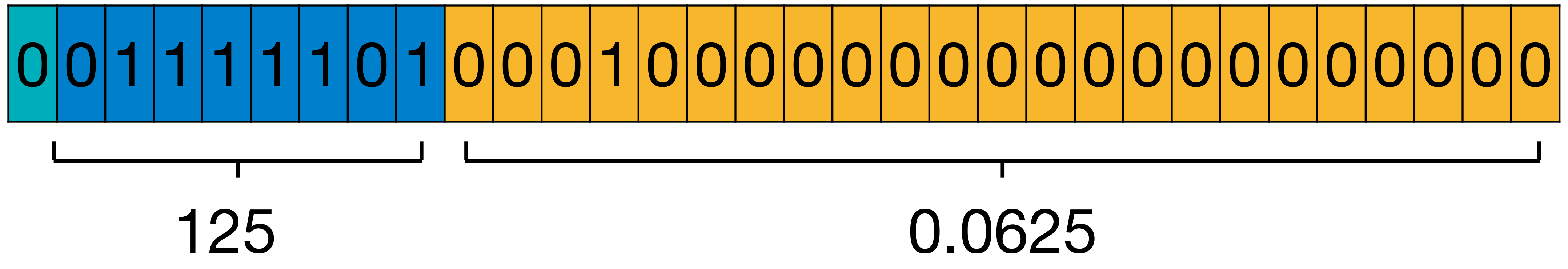
$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

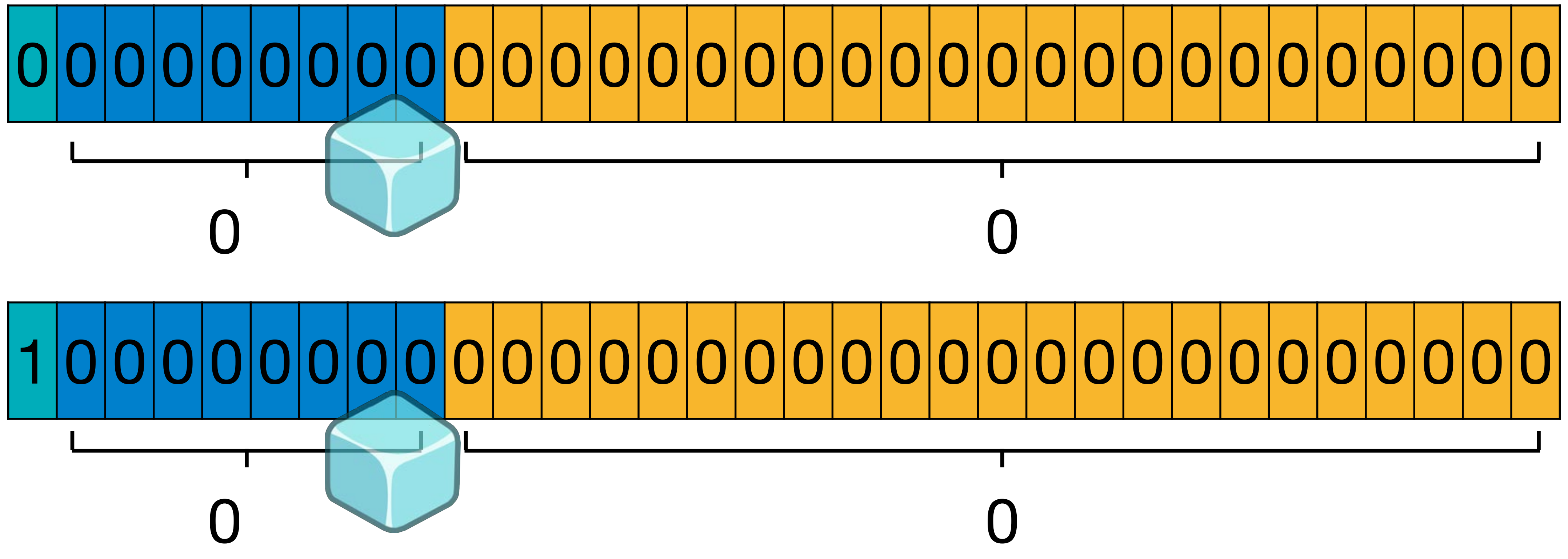
(Normal Numbers, Exponent≠0)

Should have been $(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\mathbf{0}-127}$

But we force to be $(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{\mathbf{1}-127}$

(Subnormal Numbers, Exponent=0)



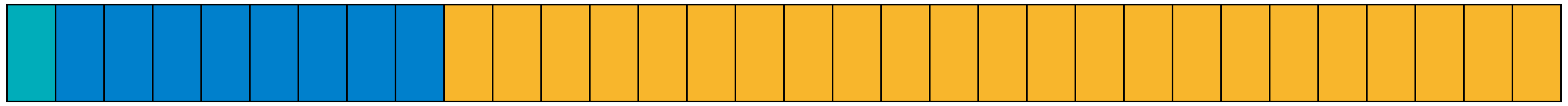


$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$

$0 = 0 \times 2^{-126}$

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



Sign 8 bit Exponent **23 bit Fraction**

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

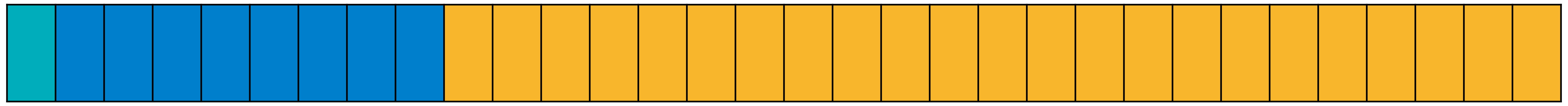
$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

What is the minimum positive value?

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$

(Normal Numbers, Exponent≠0)

0 | 00000001 | 00000000000000000000000

$\underbrace{00000001}_{1}$ $\underbrace{00000000000000000000000}_{0}$

$$2^{-126} = (1 + \underline{0}) \times 2^{\underline{1}-127}$$

$$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$$

(Subnormal Numbers, Exponent=0)

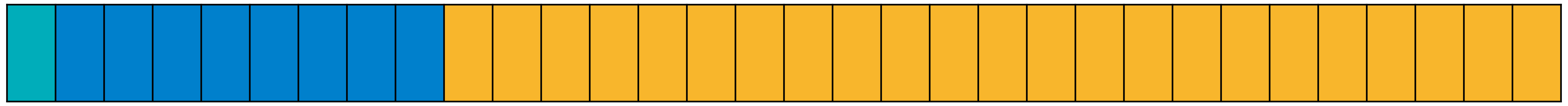
0 | 00000000 | 00000000000000000000001

$\underbrace{00000000}_{0}$ $\underbrace{00000000000000000000001}_{2^{-23}}$

$$2^{-149} = 2^{-23} \times 2^{-126}$$

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



Sign 8 bit Exponent 23 bit Fraction

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

What is the maximum positive subnormal value?

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754

Sign **8 bit Exponent** **23 bit Fraction**

$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$

(Normal Numbers, Exponent≠0)

0 | 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$$\underbrace{00000001}_{1} \quad \underbrace{00000000000000000000000}_{0}$$

$$2^{-126} = (1 + \underline{0}) \times 2^{\underline{1}-127}$$

$$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$$

(Subnormal Numbers, Exponent=0)

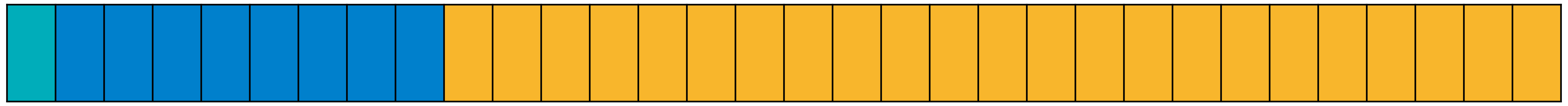
0 | 0 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$$\underbrace{00000000}_{0} \quad \underbrace{11111111111111111111111}_{2^{-23} + 2^{-22} + \ldots + 2^{-1} = 1 - 2^{-23}}$$

$$2^{-126} - 2^{-149} = (1 - 2^{-23}) \times 2^{-126}$$

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



Sign **8 bit Exponent** **23 bit Fraction**

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

+∞ (positive infinity)

1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

-∞ (negative infinity)

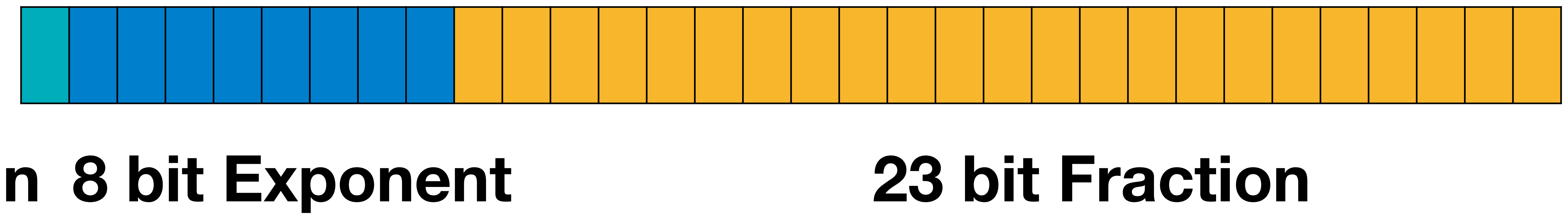
\- 1 1 1 1 1 1 1 1 -

NaN (Not a Number)

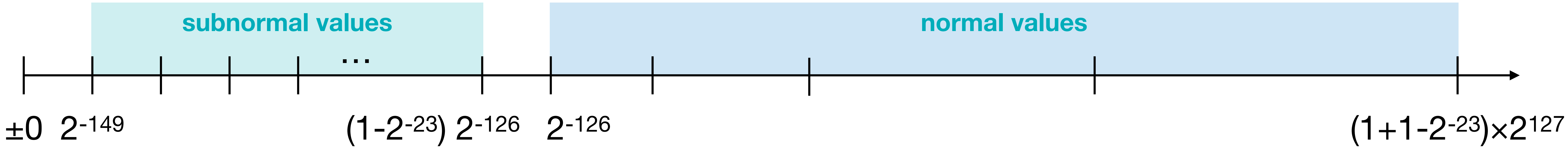
much waste. Revisit in fp8.

Floating-Point Number

Example: 32-bit floating-point number in IEEE 754



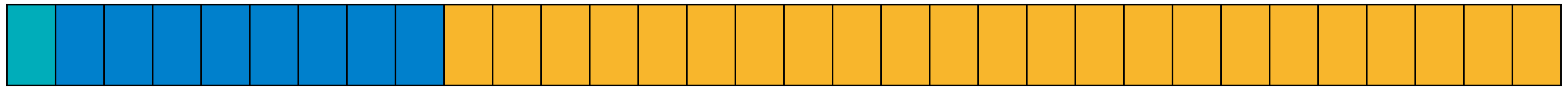
Exponent	Fraction=0	Fraction≠0	Equation
$00_H = 0$	±0	subnormal	$(-1)^{sign} \times \textbf{Fraction} \times 2^{\mathbf{1}-127}$
$01_H \ldots FE_H = 1 \ldots 254$	normal		$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$
$FF_H = 255$	±INF	NaN	



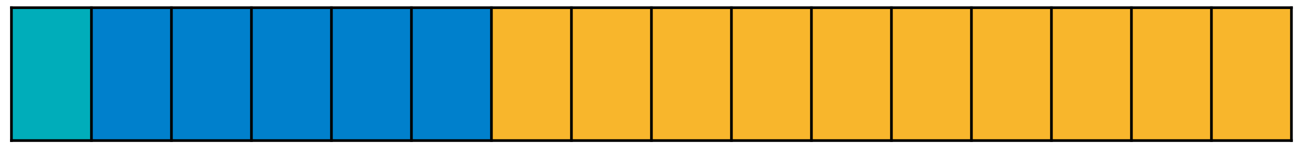
Floating-Point Number

Exponent Width → Range; Fraction Width → Precision

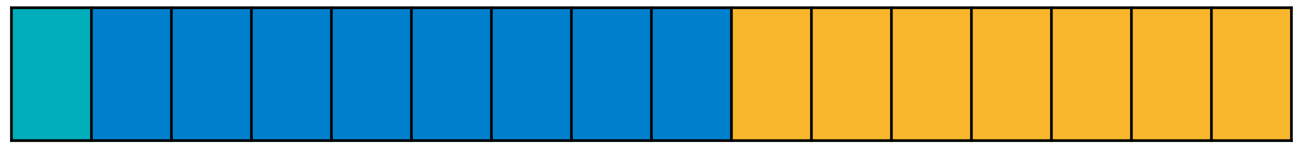
IEEE 754 Single Precision 32-bit Float (IEEE FP32)



IEEE 754 Half Precision 16-bit Float (IEEE FP16)



Google Brain Float (BF16)



Exponent (bits)	Fraction (bits)	Total (bits)
8	23	32
5	10	16
8	7	16

Numeric Data Types

- **Question**: What is the following IEEE half precision (IEEE FP16) number in decimal?

1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0

Sign 5 bit Exponent 10 bit Fraction

Exponent Bias = 15_{10}

  - Sign: -
  - Exponent: $10001_2 - 15_{10} = 17_{10} - 15_{10} = 2_{10}$
  - Fraction: $1100000000_2 = 0.75_{10}$
  - Decimal Answer = $-(1 + 0.75) \times 2^2 = -1.75 \times 2^2 = -7.0_{10}$

Numeric Data Types

- **Question**: What is the decimal 2.5 in Brain Float (BF16)?

$$2.5_{10} = 1.\underline{25}_{10} \times 2^{\underline{1}}$$

Exponent Bias = 127_{10}

  - Sign: +
  - Exponent Binary: $1_{10} + 127_{10} = 128_{10} = 10000000_2$
  - Fraction Binary: $0.25_{10} = 0100000_2$
  - Binary Answer

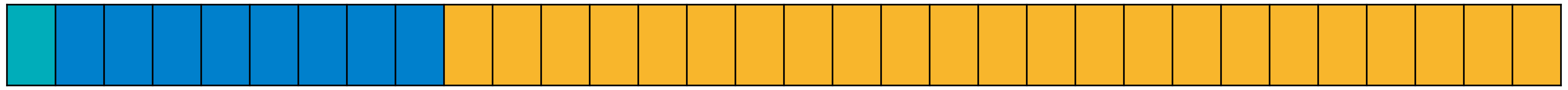
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Sign 8 bit Exponent 7 bit Fraction

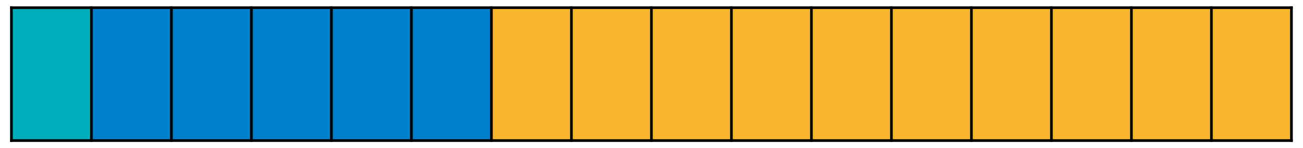
Floating-Point Number

Exponent Width → Range; Fraction Width → Precision

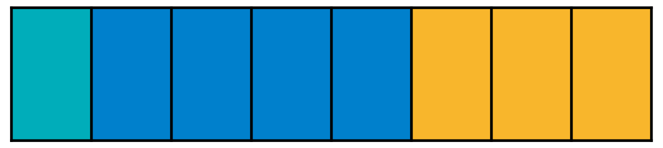
IEEE 754 Single Precision 32-bit Float (IEEE FP32)



IEEE 754 Half Precision 16-bit Float (IEEE FP16)

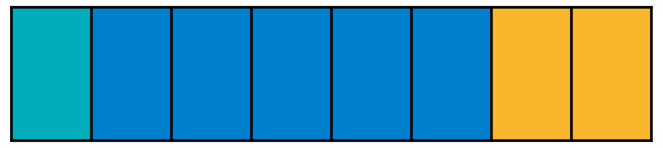


Nvidia FP8 (E4M3)



* FP8 E4M3 does not have INF, and $S.1111.111_2$ is used for NaN.
* Largest FP8 E4M3 normal value is $S.1111.110_2 = 448$.

Nvidia FP8 (E5M2) for gradient in the backward



* FP8 E5M2 have INF ($S.11111.00_2$) and NaN ($S.11111.XX_2$).
* Largest FP8 E5M2 normal value is $S.11110.11_2 = 57344$.

	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE FP32	8	23	32
IEEE FP16	5	10	16
FP8 (E4M3)	4	3	8
FP8 (E5M2)	5	2	8

INT4 and FP4

Exponent Width → Range; Fraction Width → Precision

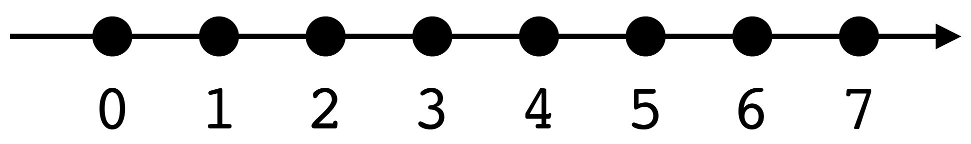
INT4

S			
0	0	0	1
0	1	1	1

=1

=7

-1,-2,-3,-4,-5,-6,-7,-8
0, 1, 2, 3, 4, 5, 6, 7



-1,-2,-3,-4,-5,-6,-7,-8
0, 1, 2, 3, 4, 5, 6, 7

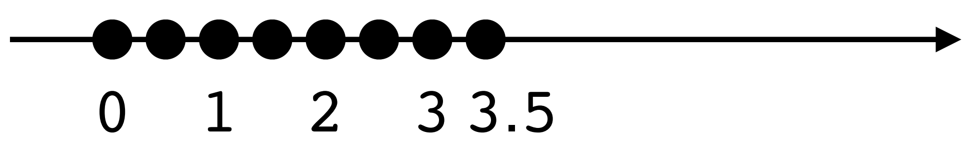
FP4 (E1M2)

S	E	M	M
0	0	0	1
0	1	1	1

$=0.25\times2^{1-0}=0.5$

$=(1+0.75)\times2^{1-0}=3.5$

-0,-0.5,-1,-1.5,-2,-2.5,-3,-3.5
0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5



-0,-1,-2,-3,-4,-5,-6,-7
0, 1, 2, 3, 4, 5, 6, 7 ×0.5

FP4 (E2M1)

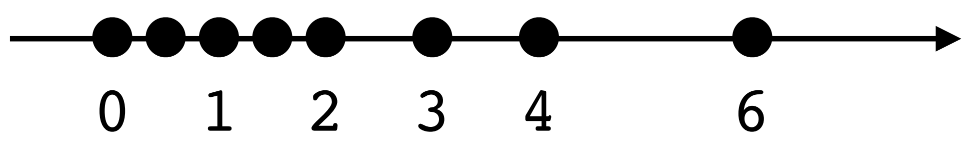
S	E	E	M
0	0	0	1
0	1	1	1

$=0.5\times2^{1-1}=0.5$

$=(1+0.5)\times2^{3-1}=1$

no inf, no NaN

-0,-0.5,-1,-1.5,-2,-3,-4,-6
0, 0.5, 1, 1.5, 2, 3, 4, 6



-0,-1,-2,-3,-4,-6,-8,-12
0, 1, 2, 3, 4, 6, 8, 12 ×0.5

FP4 (E3M0)

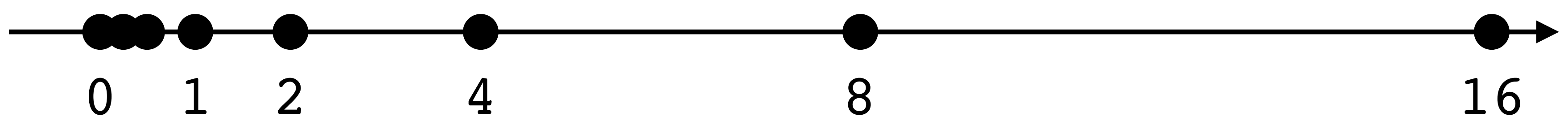
S	E	E	E
0	0	0	1
0	1	1	1

$=(1+0)\times2^{1-3}=0.25$

$=(1+0)\times2^{7-3}=16$

no inf, no NaN

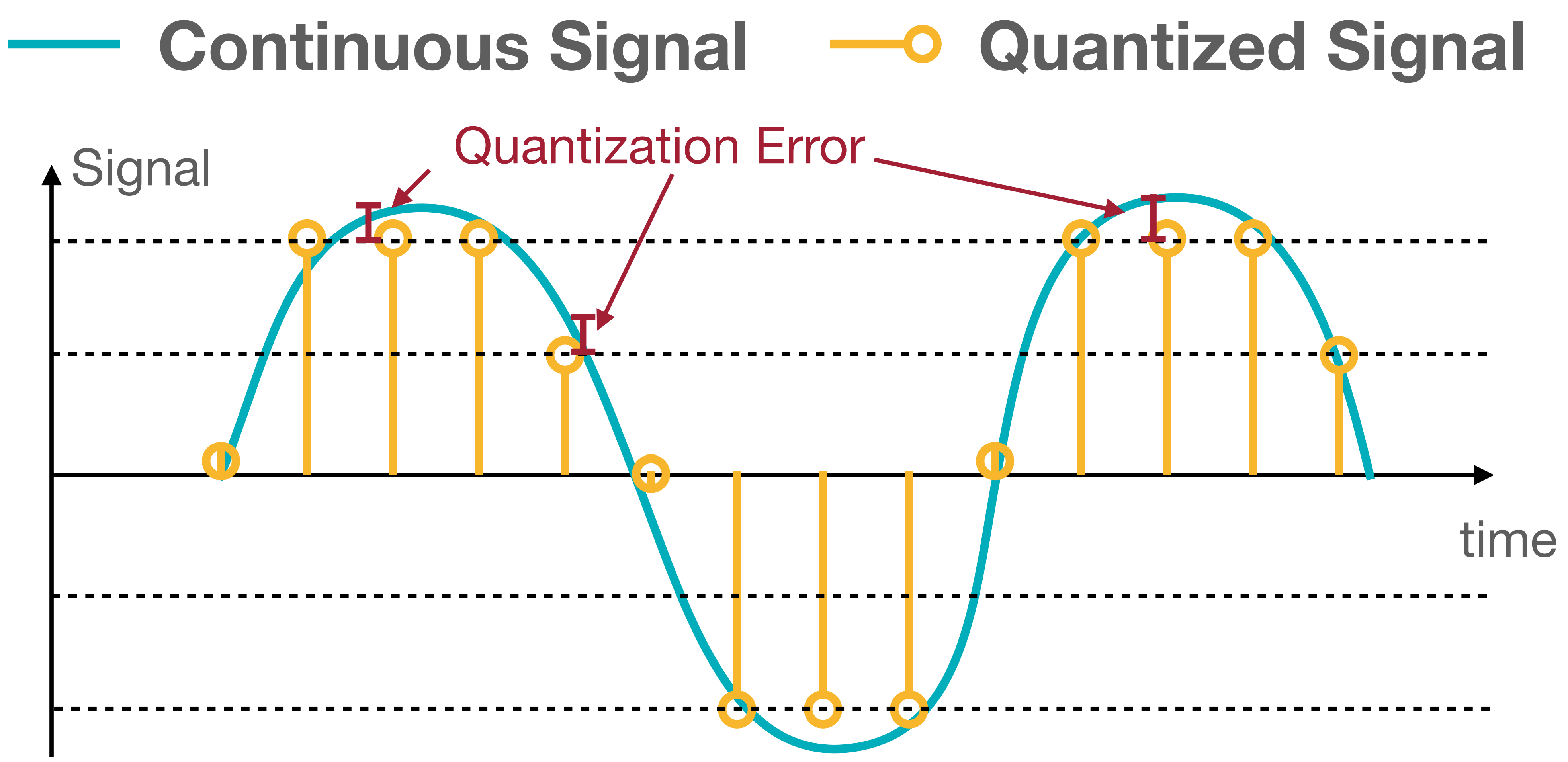
-0,-0.25,-0.5,-1,-2,-4,-8,-16
0, 0.25, 0.5, 1, 2, 4, 8, 16



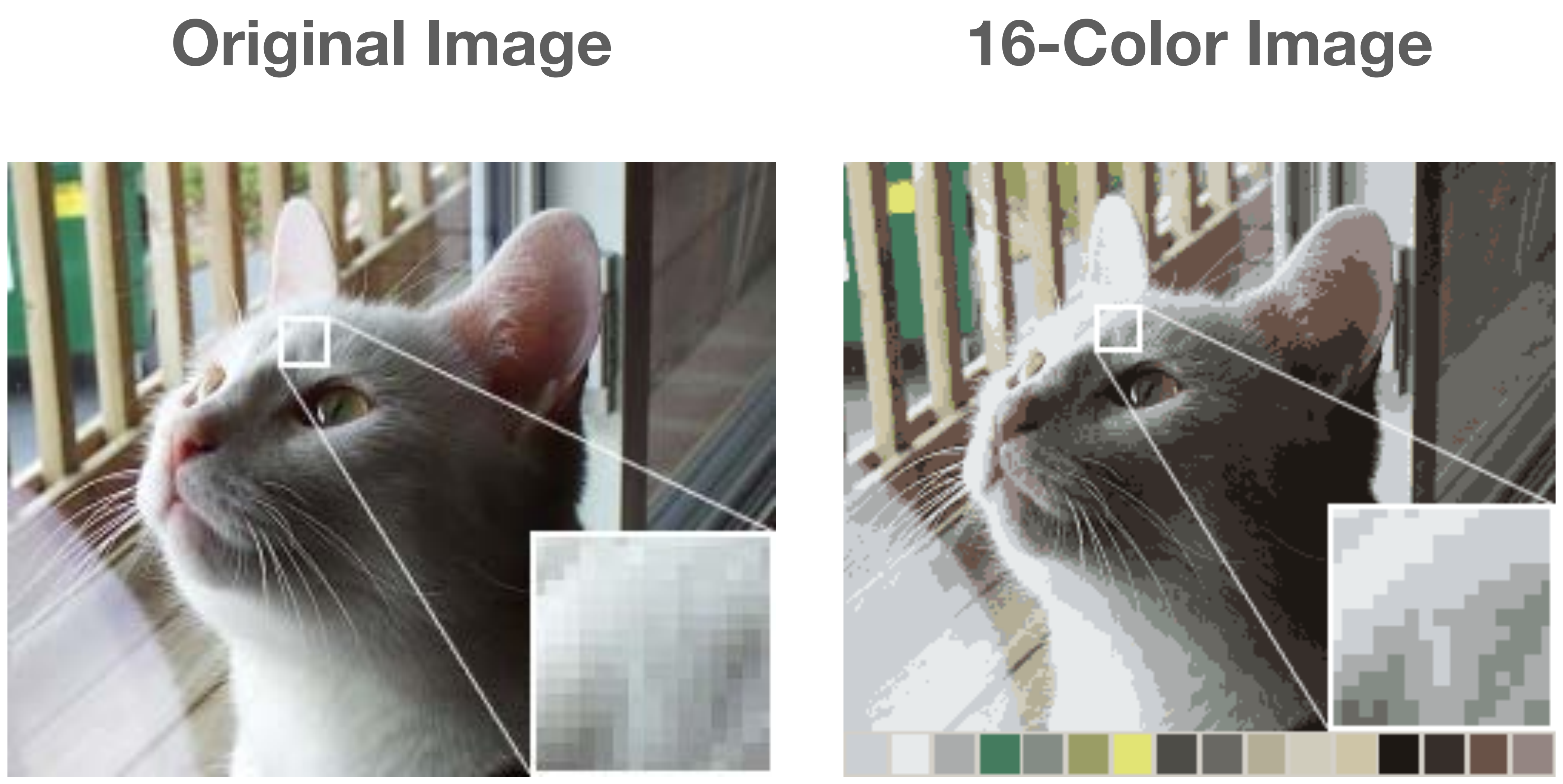
-0,-1,-2,-4,-8,-16,-32,-64
0, 1, 2, 4, 8, 16, 32, 64 ×0.25

What is Quantization?

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



The difference between an input value and its quantized value is referred to as quantization error.



Images are in the public domain.

"Palettization"

Quantization [Wikipedia]

Neural Network Quantization: Agenda

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

3	0	2	1	3:	2.00
1	1	0	3	2:	1.50
0	3	1	0	1:	0.00
3	1	2	2	0:	-1.00

K-Means-based Quantization

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1\right) \times 1.07$$

Linear Quantization

1	0	1	1
1	0	0	1
0	1	1	0
1	1	1	1

Binary/Ternary Quantization

Storage	Floating-Point Weights			
Computation	Floating-Point Arithmetic			

Neural Network Quantization: Agenda

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

3	0	2	1	3:	2.00
1	1	0	3	2:	1.50
0	3	1	0	1:	0.00
3	1	2	2	0:	-1.00

K-Means-based Quantization

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

(– -1)×1.07

Linear Quantization

1	0	1	1
1	0	0	1
0	1	1	0
1	1	1	1

Binary/Ternary Quantization

Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic

Neural Network Quantization

Weight Quantization

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*

Neural Network Quantization

Weight Quantization

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*

~~2.09, 2.12, 1.92, 1.87~~

2.0

K-Means-based Weight Quantization

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*

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

K-Means-based Weight Quantization

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*

cluster

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

indexes

centroids

3:	*2.00*
2:	*1.50*
1:	*0.00*
0:	*-1.00*

codebook

reconstructed weights
(32-bit float)

2.00	*-1.00*	*1.50*	*0.00*
0.00	*0.00*	*-1.00*	*2.00*
-1.00	*2.00*	*0.00*	*-1.00*
2.00	*0.00*	*1.50*	*1.50*

quantization error

0.09	*0.02*	*-0.02*	*0.09*
0.05	*-0.14*	*-0.08*	*0.12*
0.09	*-0.08*	*0*	*-0.03*
-0.13	*0*	*0.03*	*-0.01*

storage

32 bit × 16
= 512 bit = 64 B

2 bit × 16
= 32 bit = 4 B

+

32 bit × 4
= 128 bit = 16 B

= 20 B

3.2 × smaller

Assume *N*-bit quantization, and #parameters = *M* >> 2^N.

32 bit × *M*
= 32*M* bit

N bit × *M*
= *NM* bit

$\cancel{32 \text{ bit} \times 2^N = 2^{N+5} \text{ bit}}$

32/*N* × smaller

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

K-Means-based Weight Quantization

Fine-tuning Quantized Weights

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*

cluster

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

centroids

3:	*2.00*
2:	*1.50*
1:	*0.00*
0:	*-1.00*

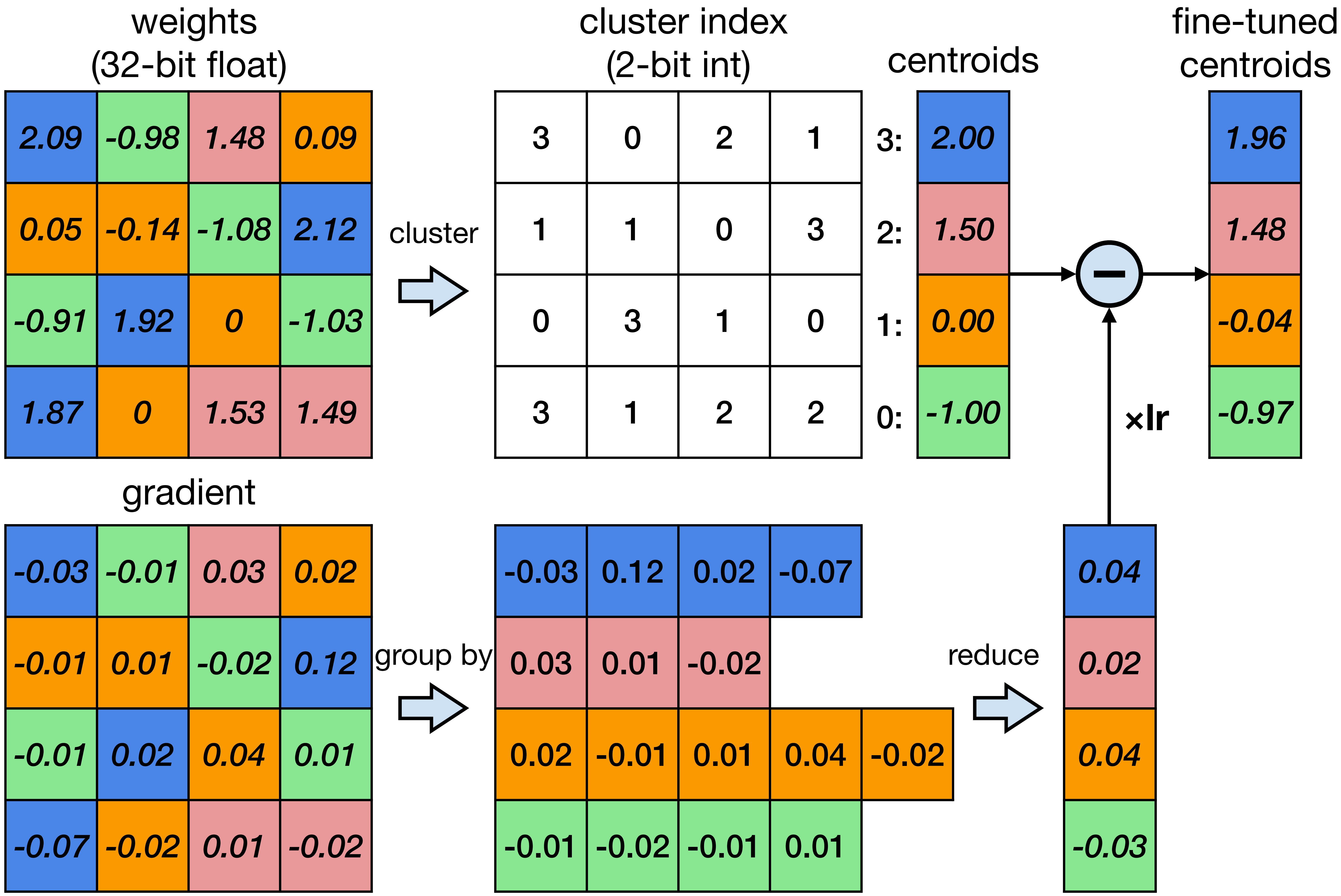
gradient

-0.03	*-0.01*	*0.03*	*0.02*
-0.01	*0.01*	*-0.02*	*0.12*
-0.01	*0.02*	*0.04*	*0.01*
-0.07	*-0.02*	*0.01*	*-0.02*

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

K-Means-based Weight Quantization

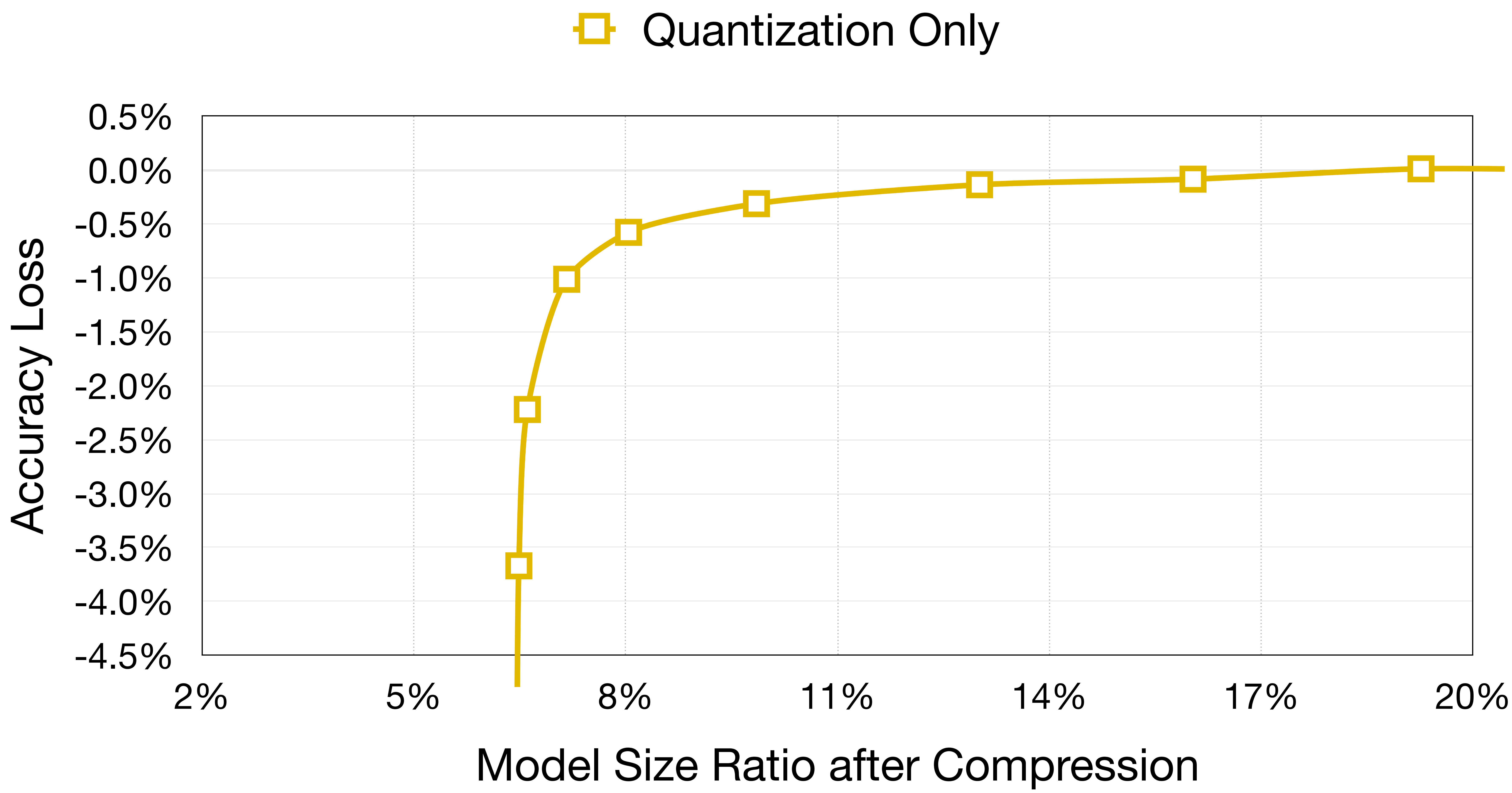
Fine-tuning Quantized Weights



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

K-Means-based Weight Quantization

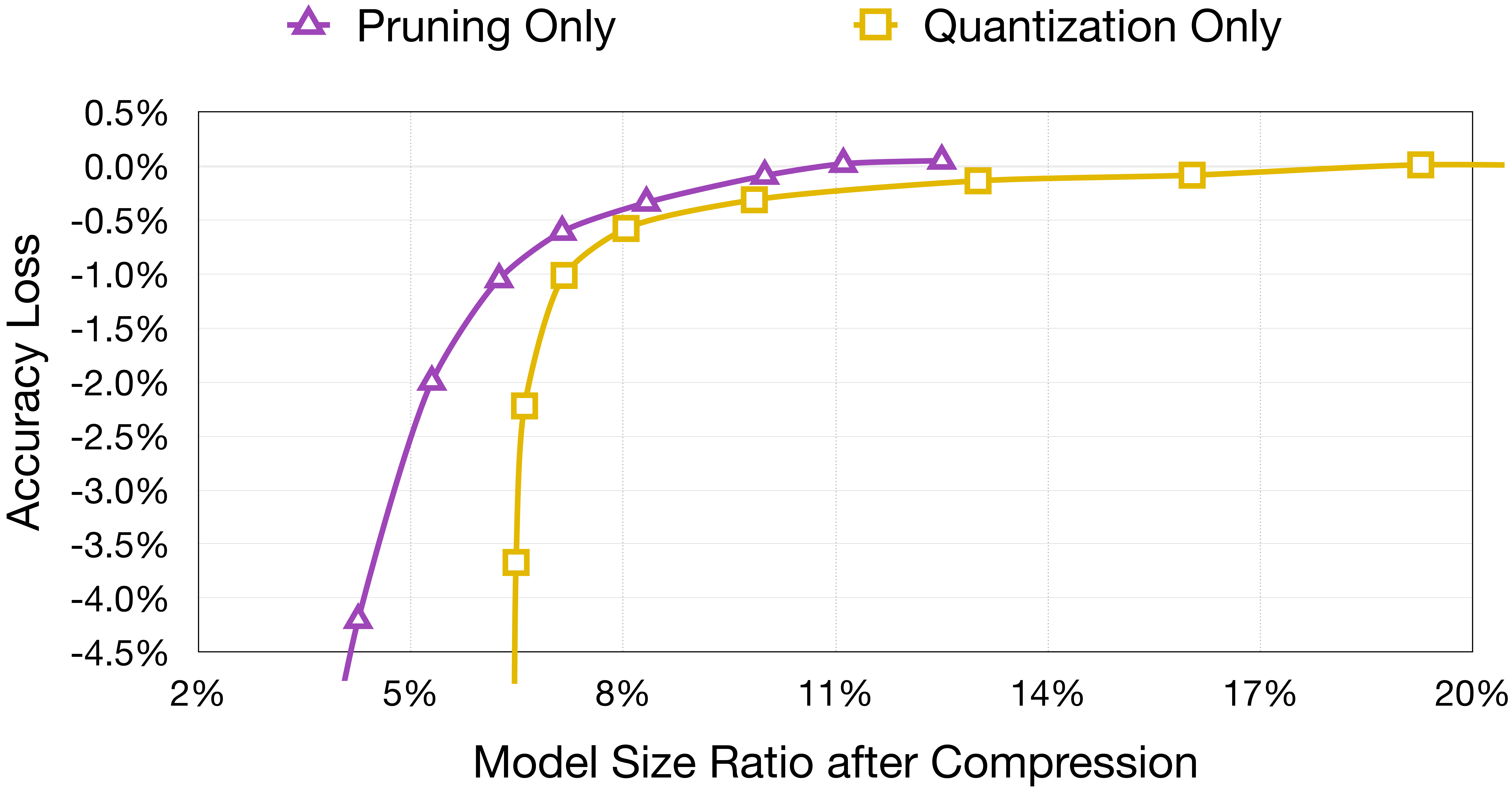
Accuracy vs. compression rate for AlexNet on ImageNet dataset



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

K-Means-based Weight Quantization

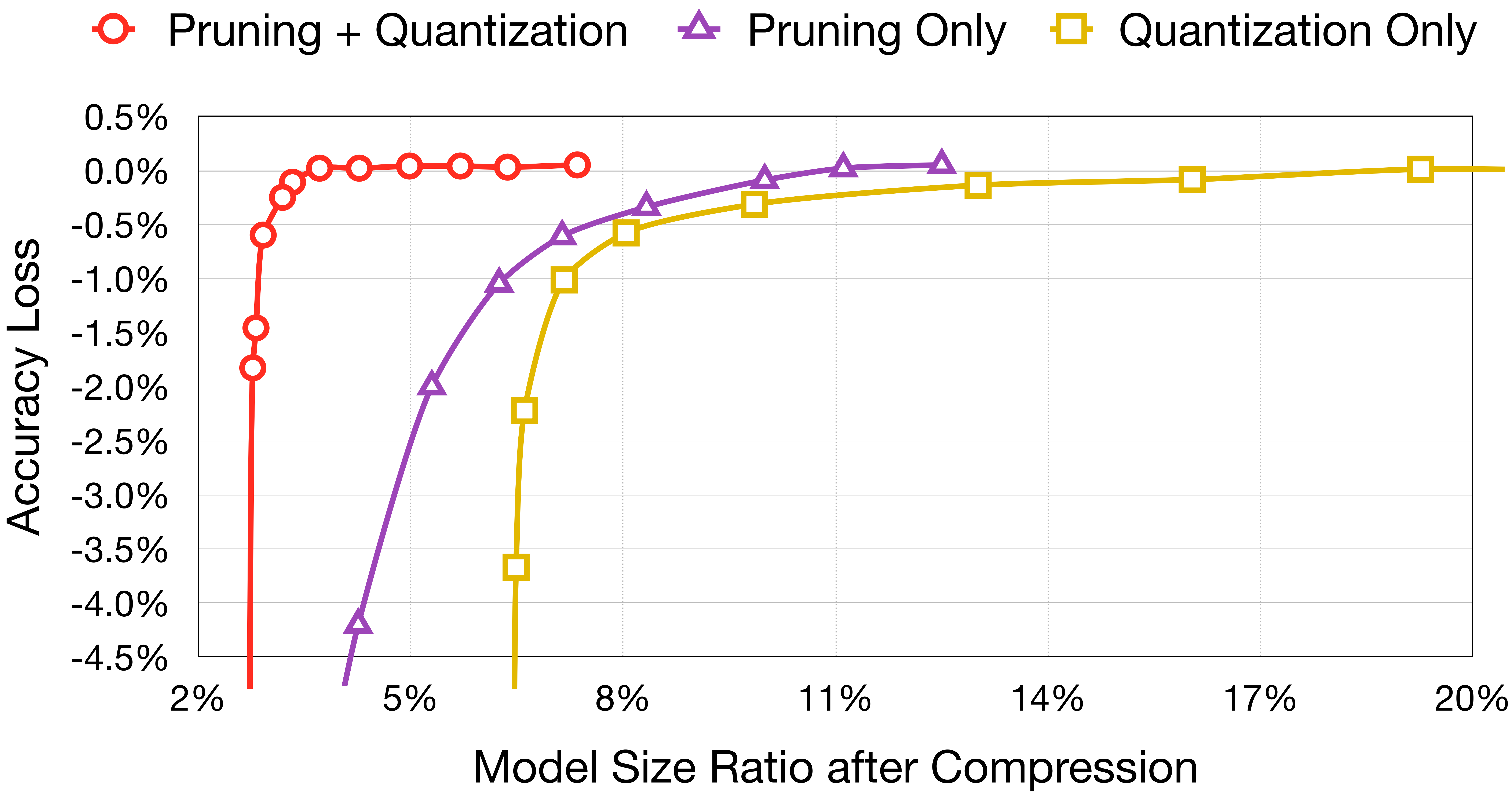
Accuracy vs. compression rate for AlexNet on ImageNet dataset



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

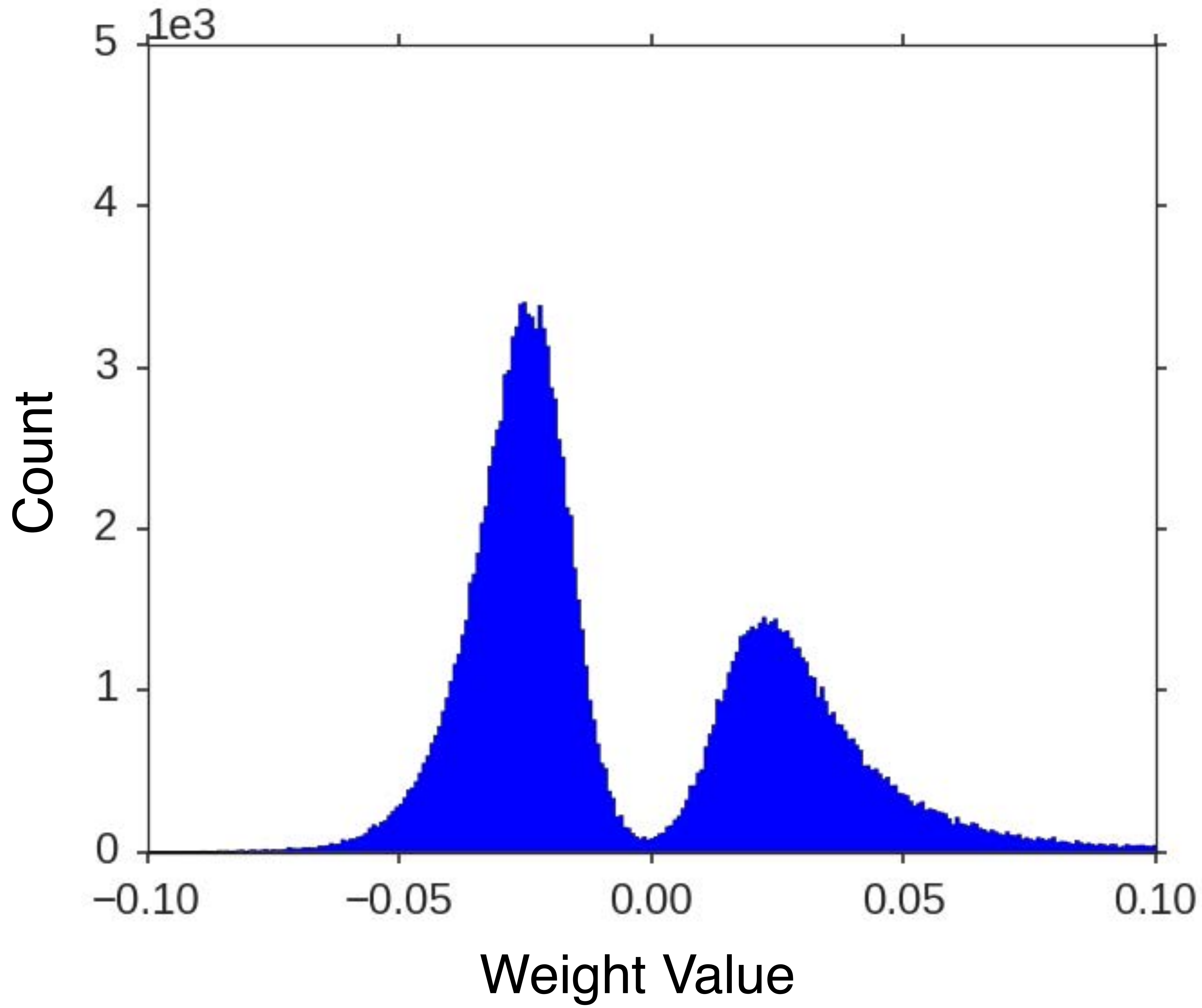
K-Means-based Weight Quantization

Accuracy vs. compression rate for AlexNet on ImageNet dataset



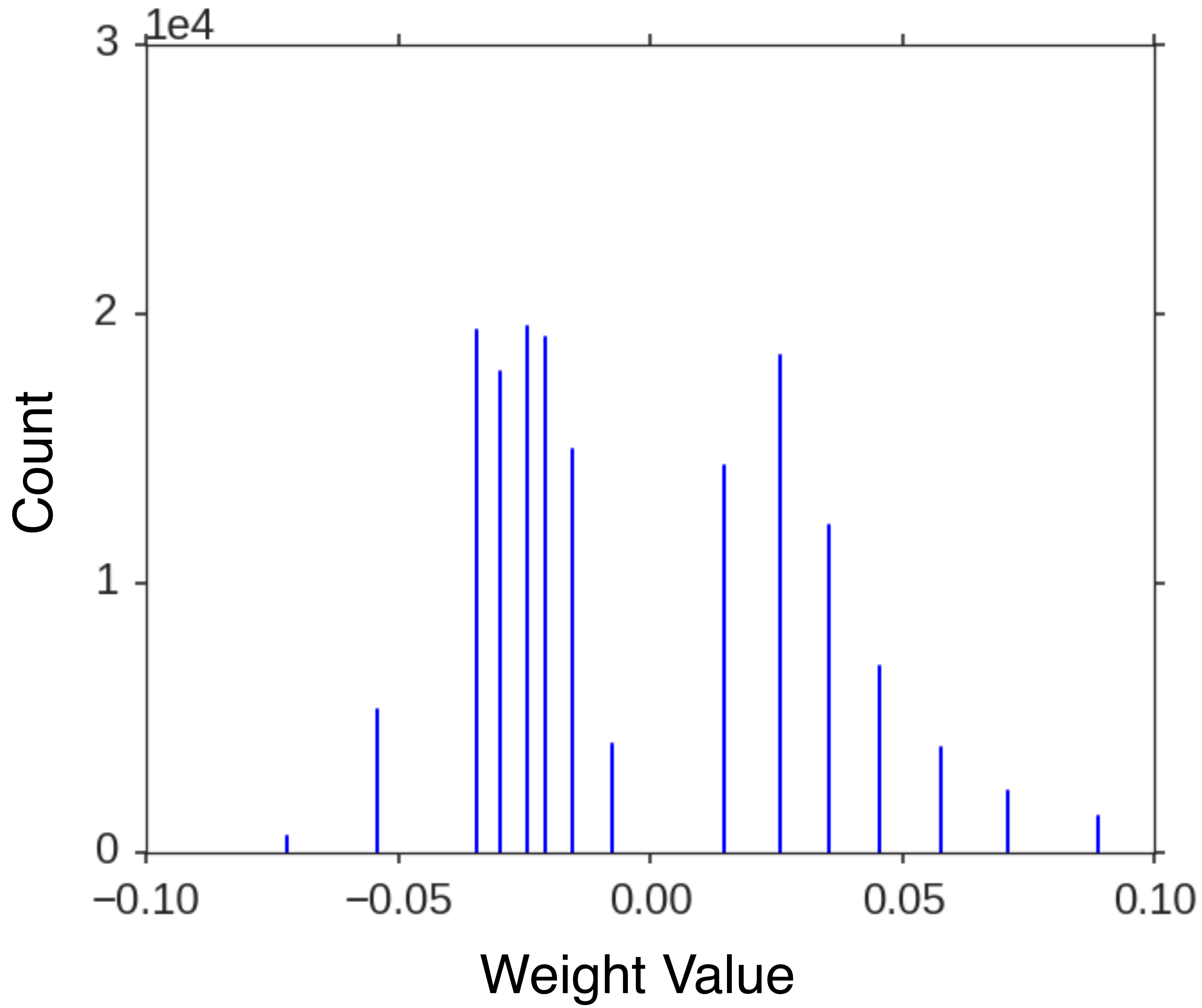
Deep Compression [Han *et al.*, ICLR 2016]

Before Quantization: Continuous Weight



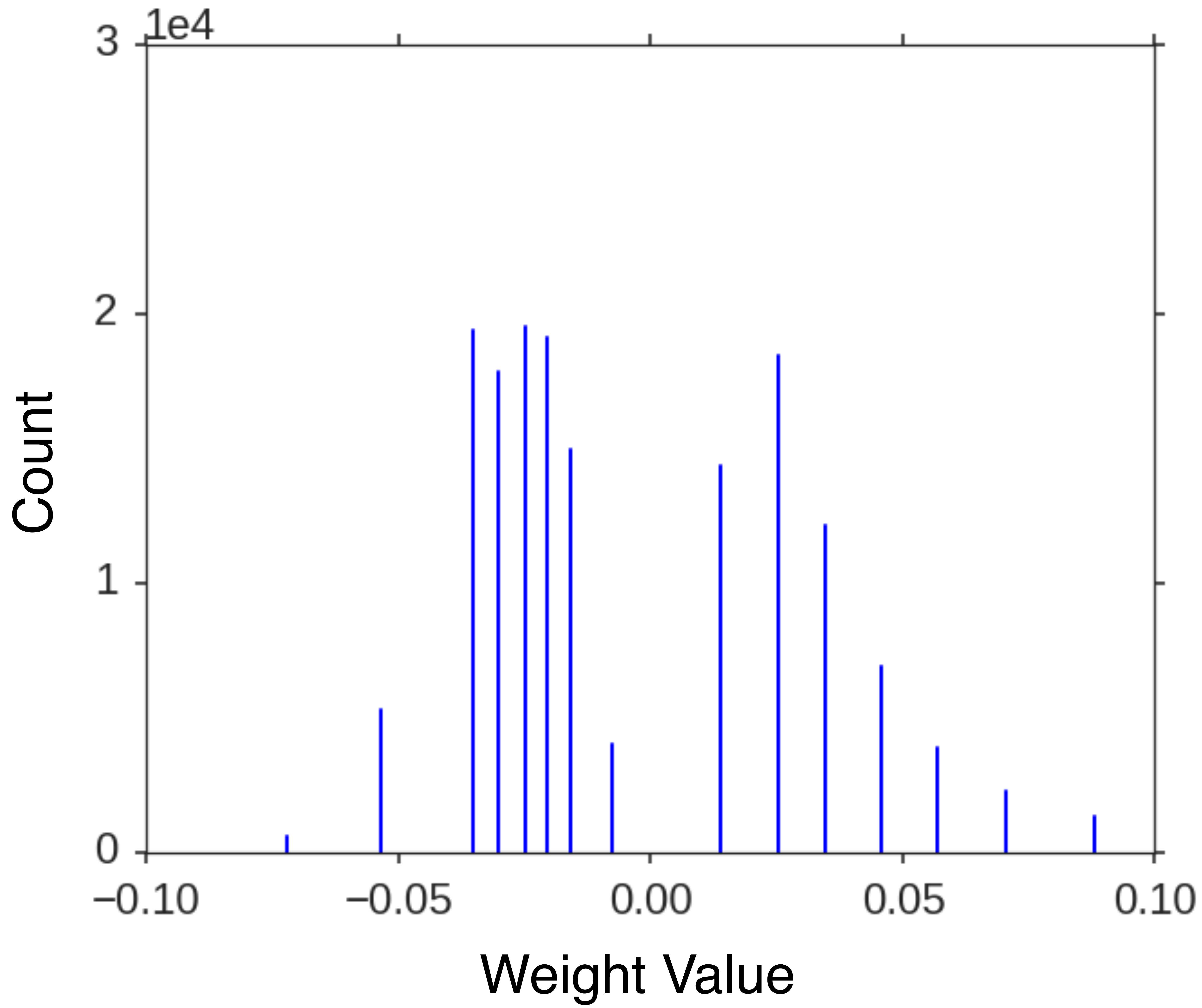
Deep Compression [Han *et al.*, ICLR 2016]

After Quantization: Discrete Weight



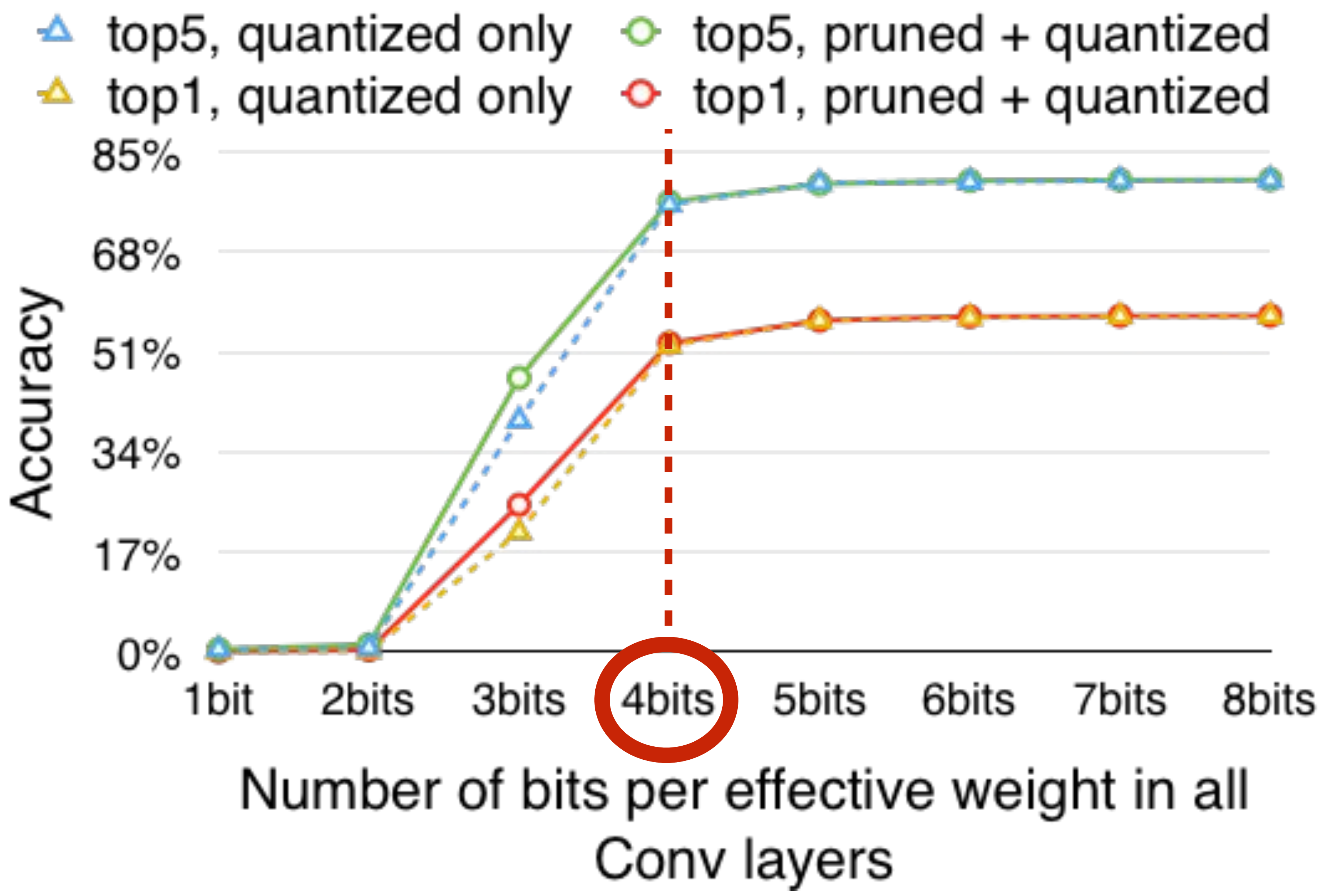
Deep Compression [Han *et al.*, ICLR 2016]

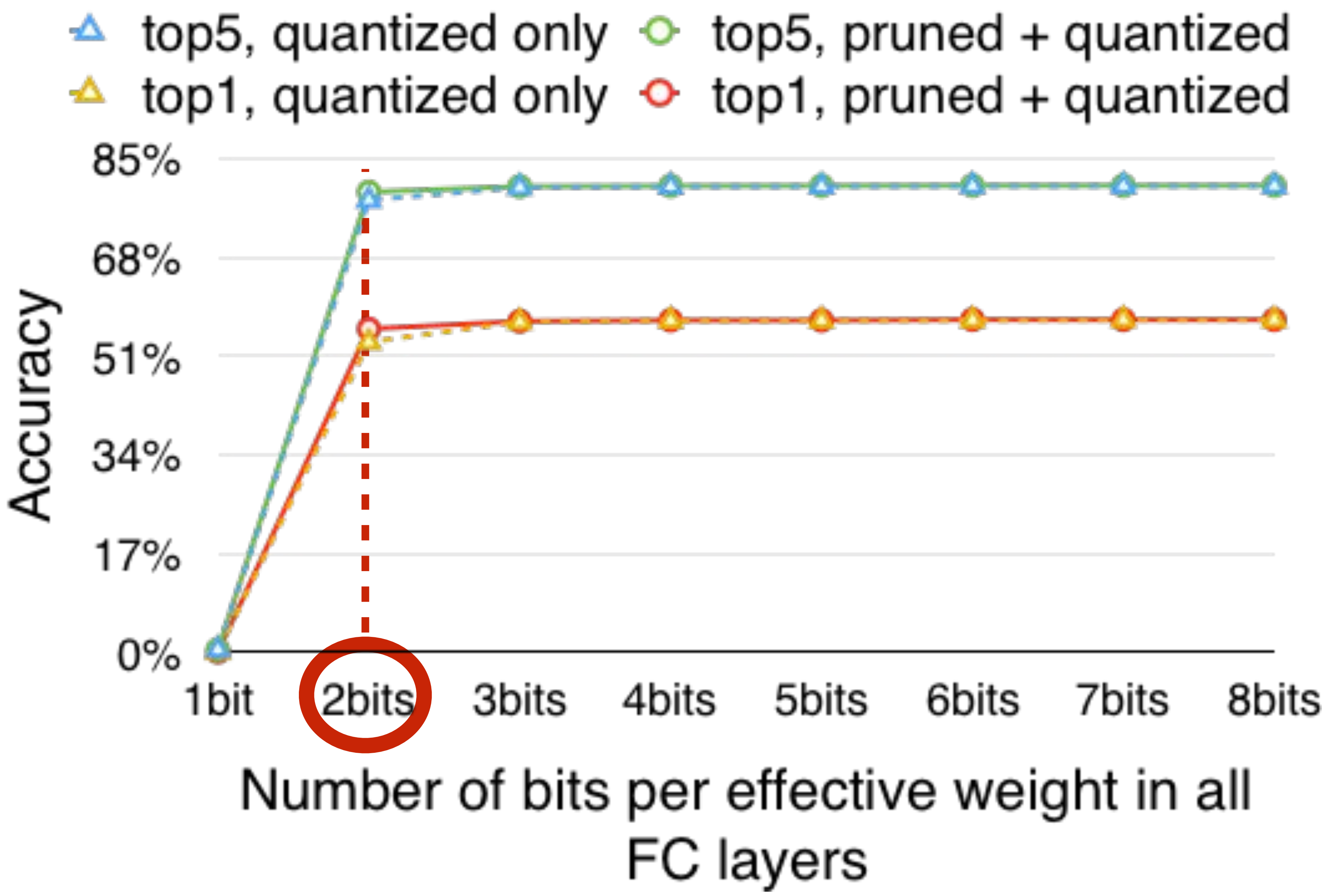
After Quantization: Discrete Weight after Training



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

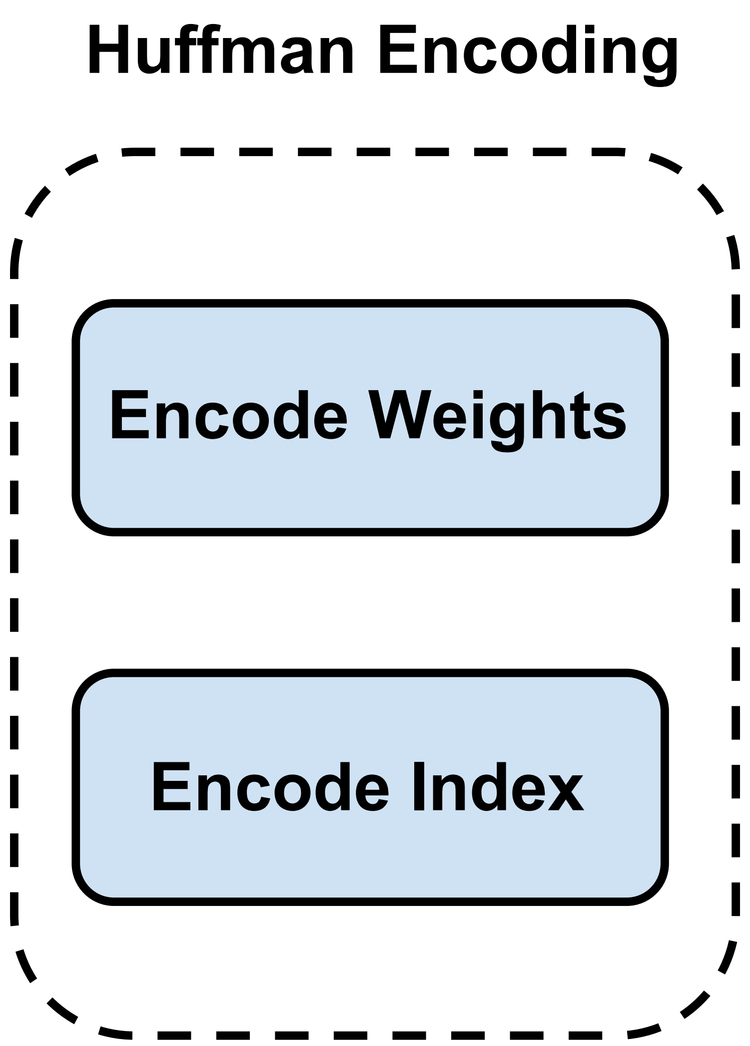
How Many Bits do We Need?

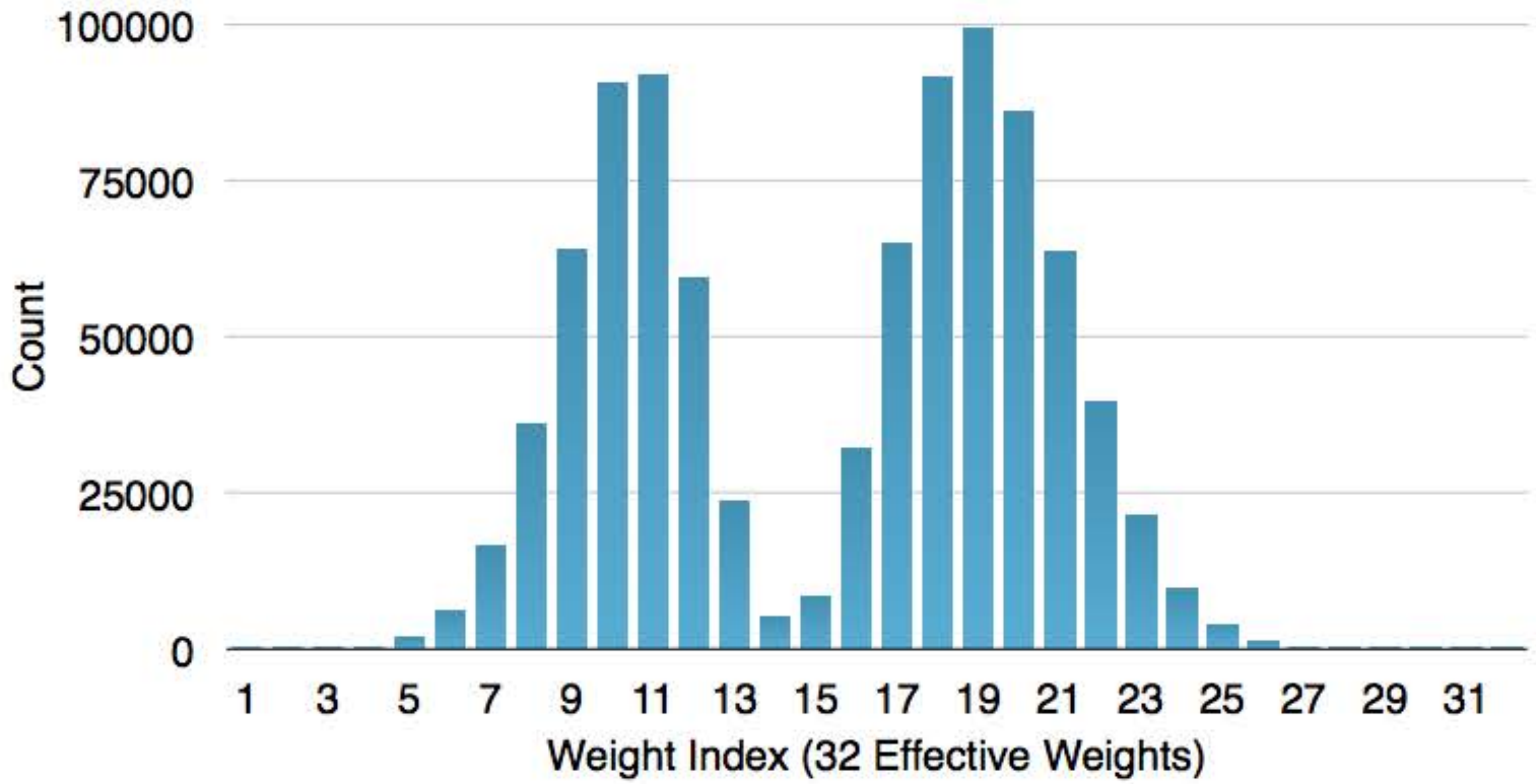




Deep Compression [Han *et al.*, ICLR 2016]

Huffman Coding

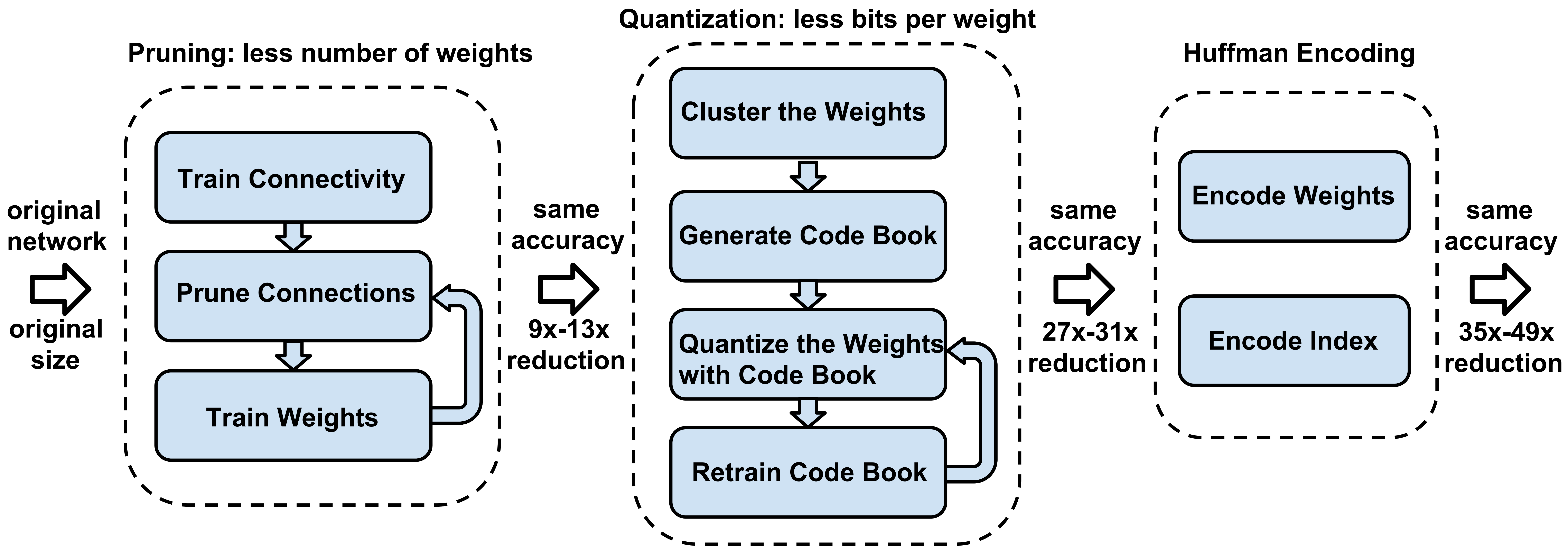




- In-frequent weights: use more bits to represent
- Frequent weights: use less bits to represent

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

Summary of Deep Compression



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

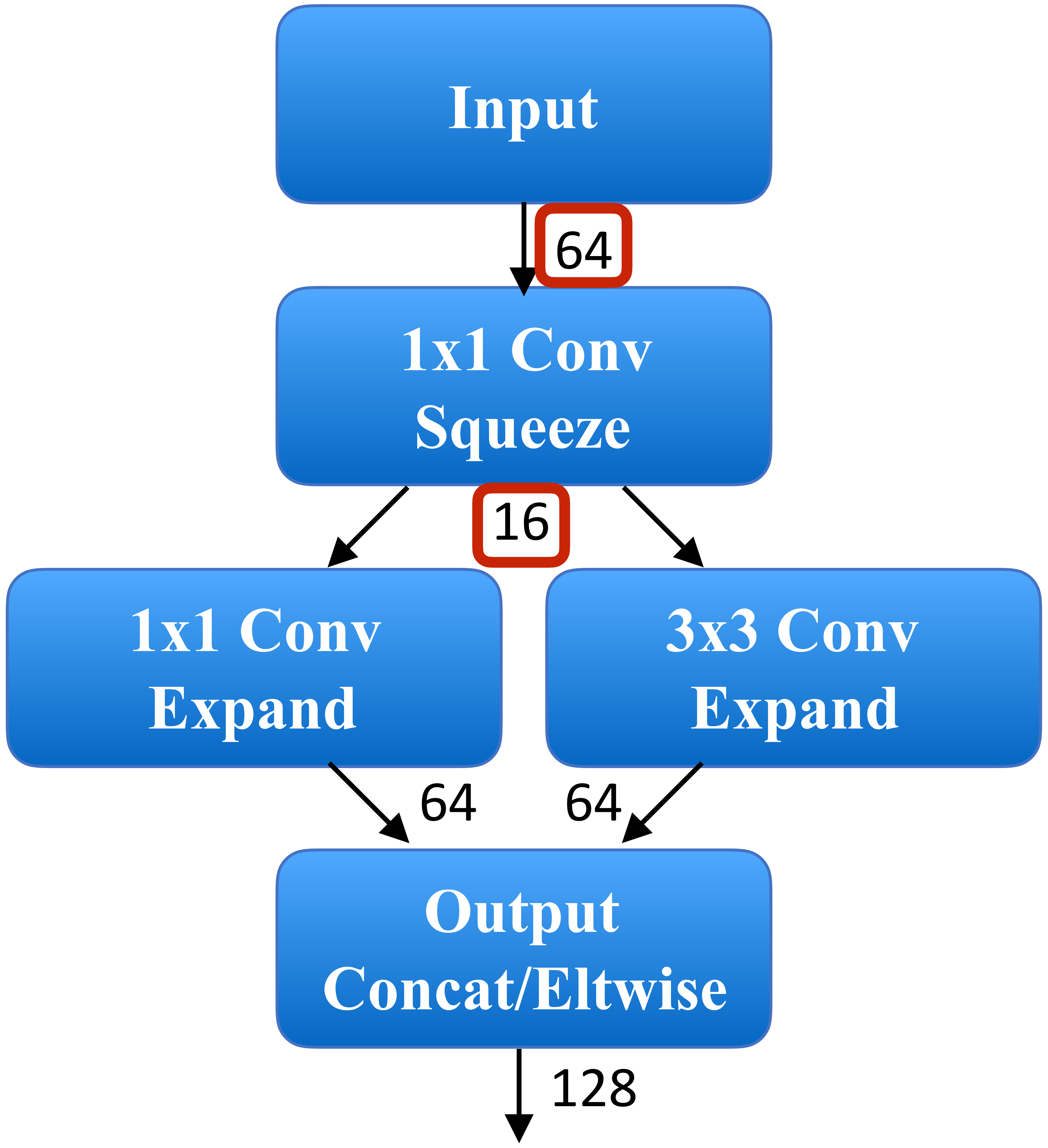
Deep Compression Results

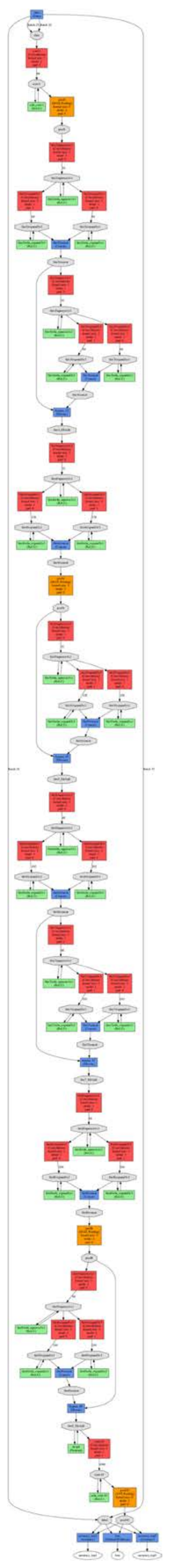
Network	Original Size	Compressed Size	Compression Ratio	Original Accuracy	Compressed Accuracy
LeNet-300	1070KB	27KB	**40x**	98.36%	98.42%
LeNet-5	1720KB	44KB	**39x**	99.20%	99.26%
AlexNet	240MB	6.9MB	**35x**	80.27%	80.30%
VGGNet	550MB	11.3MB	**49x**	88.68%	89.09%
GoogleNet	28MB	2.8MB	**10x**	88.90%	88.92%
ResNet-18	44.6MB	4.0MB	**11x**	89.24%	89.28%

Can we make compact models to begin with?

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

SqueezeNet





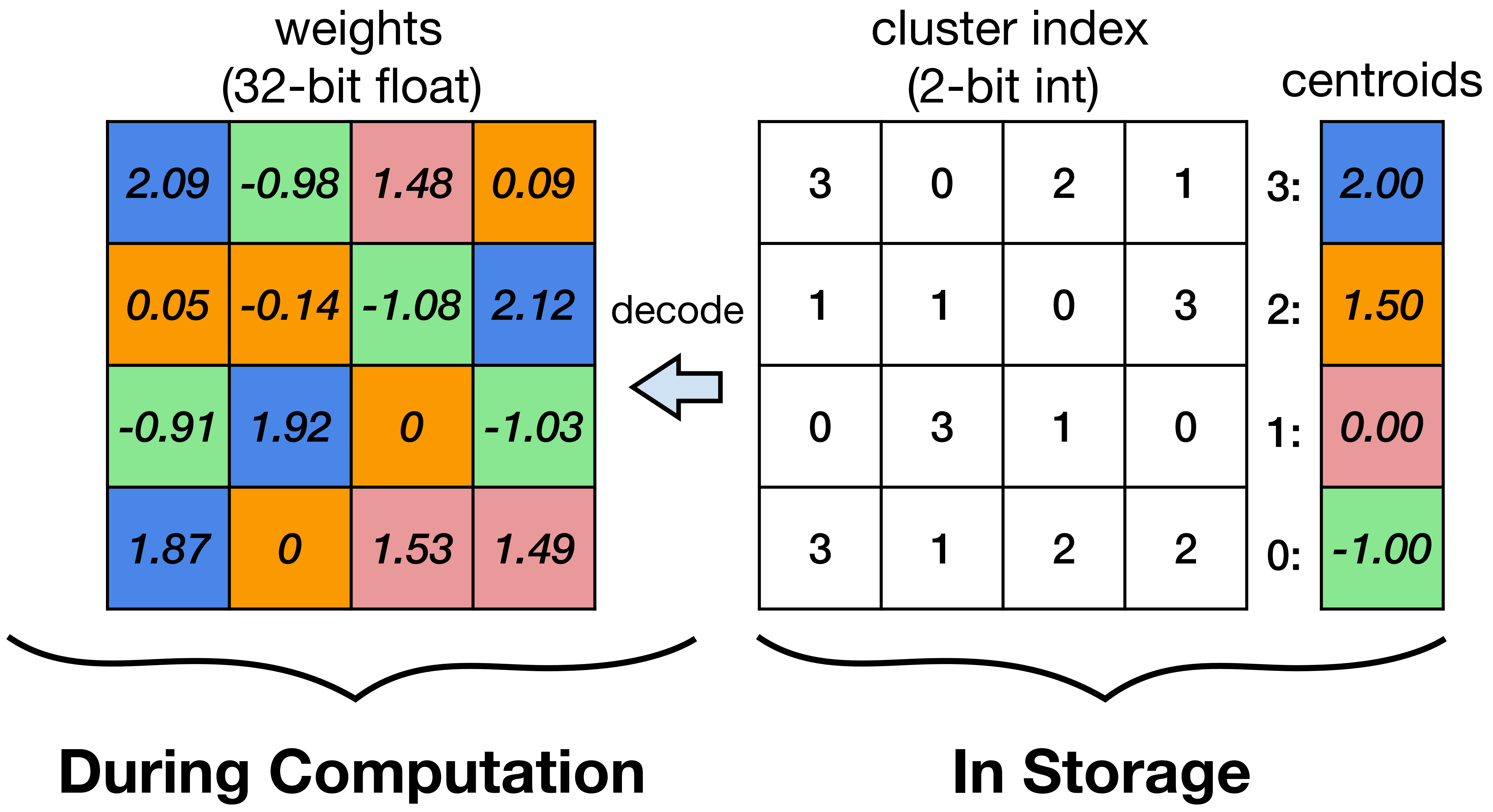
SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size [Iandola *et al.*, arXiv 2016]

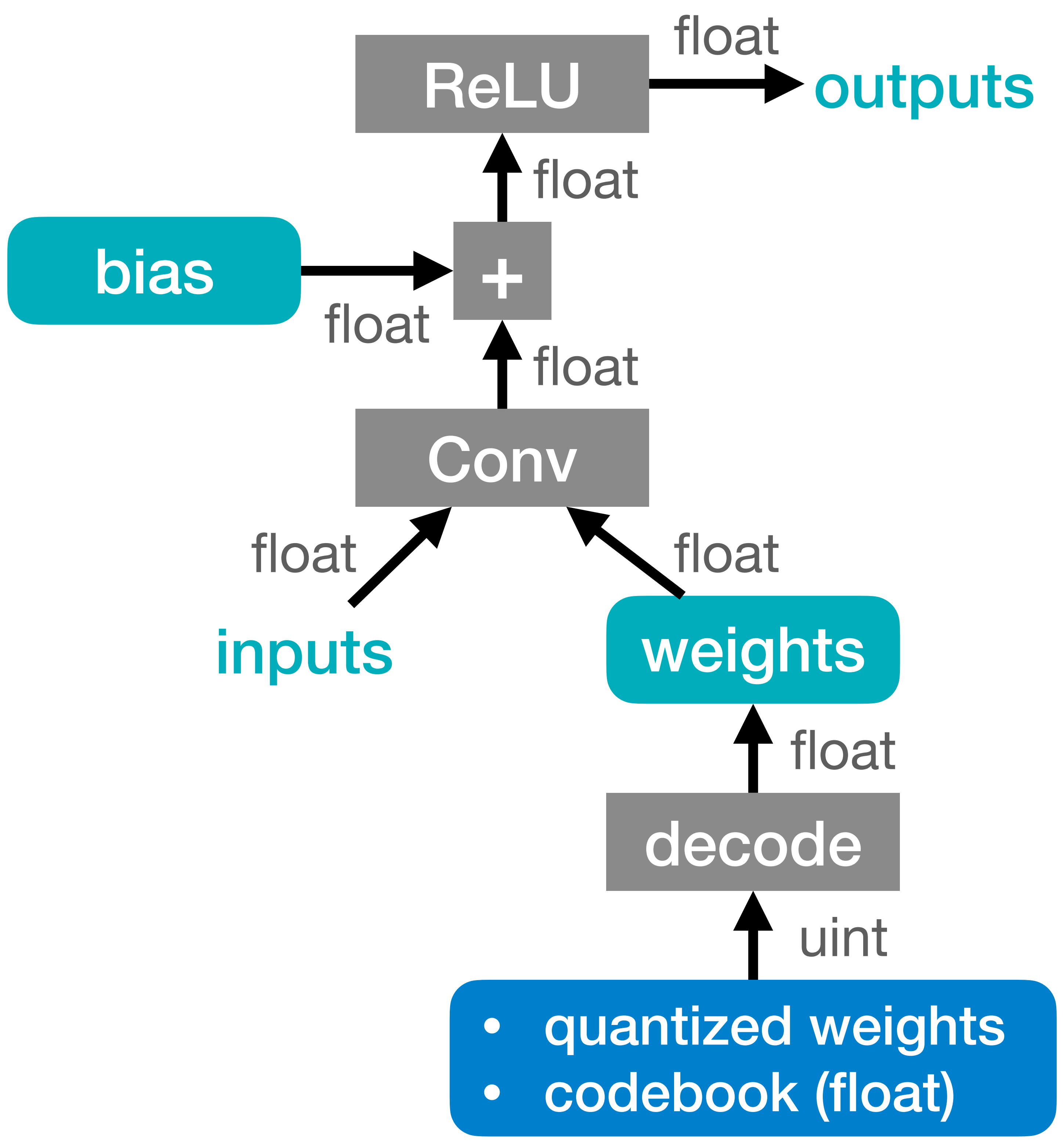
Deep Compression on SqueezeNet

Network	Approach	Size	Ratio	Top-1 Accuracy	Top-5 Accuracy
AlexNet	-	240MB	**1x**	57.2%	80.3%
AlexNet	SVD	48MB	**5x**	56.0%	79.4%
AlexNet	Deep Compression	6.9MB	**35x**	57.2%	80.3%
SqueezeNet	-	4.8MB	**50x**	57.5%	80.3%
SqueezeNet	Deep Compression	0.47MB	**510x**	57.5%	80.3%

SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size [Iandola *et al.*, arXiv 2016]

K-Means-based Weight Quantization





- The weights are decompressed using a lookup table (*i.e.*, codebook) during runtime inference.
- K-Means-based Weight Quantization only saves storage cost of a neural network model.
  - All the computation and memory access are still floating-point.

Neural Network Quantization

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

3	0	2	1	3:	2.00
1	1	0	3	2:	1.50
0	3	1	0	1:	0.00
3	1	2	2	0:	-1.00

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1\right) \times 1.07$$

1	0	1	1
1	0	0	1
0	1	1	0
1	1	1	1

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

Linear Quantization

What is Linear Quantization?

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*

What is Linear Quantization?

An affine mapping of integers to real numbers

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

$\Rightarrow$ (quantized weights (2-bit signed int) − zero point (2-bit signed int)) × scale (32-bit float) = reconstructed weights (32-bit float)

quantized weights (2-bit signed int)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

zero point (2-bit signed int): **-1**

scale (32-bit float): **1.07**

we will learn how to determine these parameters later

reconstructed weights (32-bit float)

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

quantization error

-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

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

What is Linear Quantization?

An affine mapping of integers to real numbers

$$
\underbrace{\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}}_{\text{weights (32-bit float)}}
\Rightarrow
\left(\underbrace{\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix}}_{\text{quantized weights (2-bit signed int)}} - \underbrace{-1}_{\underline{\text{zero point}}\text{ (2-bit signed int)}} \right) \times \underbrace{1.07}_{\underline{\text{scale}}\text{ (32-bit float)}} = \underbrace{\begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}}_{\text{reconstructed weights (32-bit float)}}
$$

we will learn how to determine these parameters later

quantization error

-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

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

Linear Quantization

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

weights (32-bit float)	quantized weights (2-bit signed int)	zero point (2-bit signed int)	scale (32-bit float)

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \Rightarrow \left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - \; -1 \right) \times 1.07$$

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

Floating-point **Integer**

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

Linear Quantization

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

weights (32-bit float)		quantized weights (2-bit signed int)		zero point (2-bit signed int)		scale (32-bit float)

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \Rightarrow \left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1 \right) \times 1.07$$

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

Floating-point ← **Integer**

Floating-point
- quantization parameter

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

Linear Quantization

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

weights (32-bit float)	quantized weights (2-bit signed int)	zero point (2-bit signed int)	scale (32-bit float)

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \Rightarrow \left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - \mathbf{-1} \right) \times \mathbf{1.07}$$

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

Floating-point (r)

Integer (q)

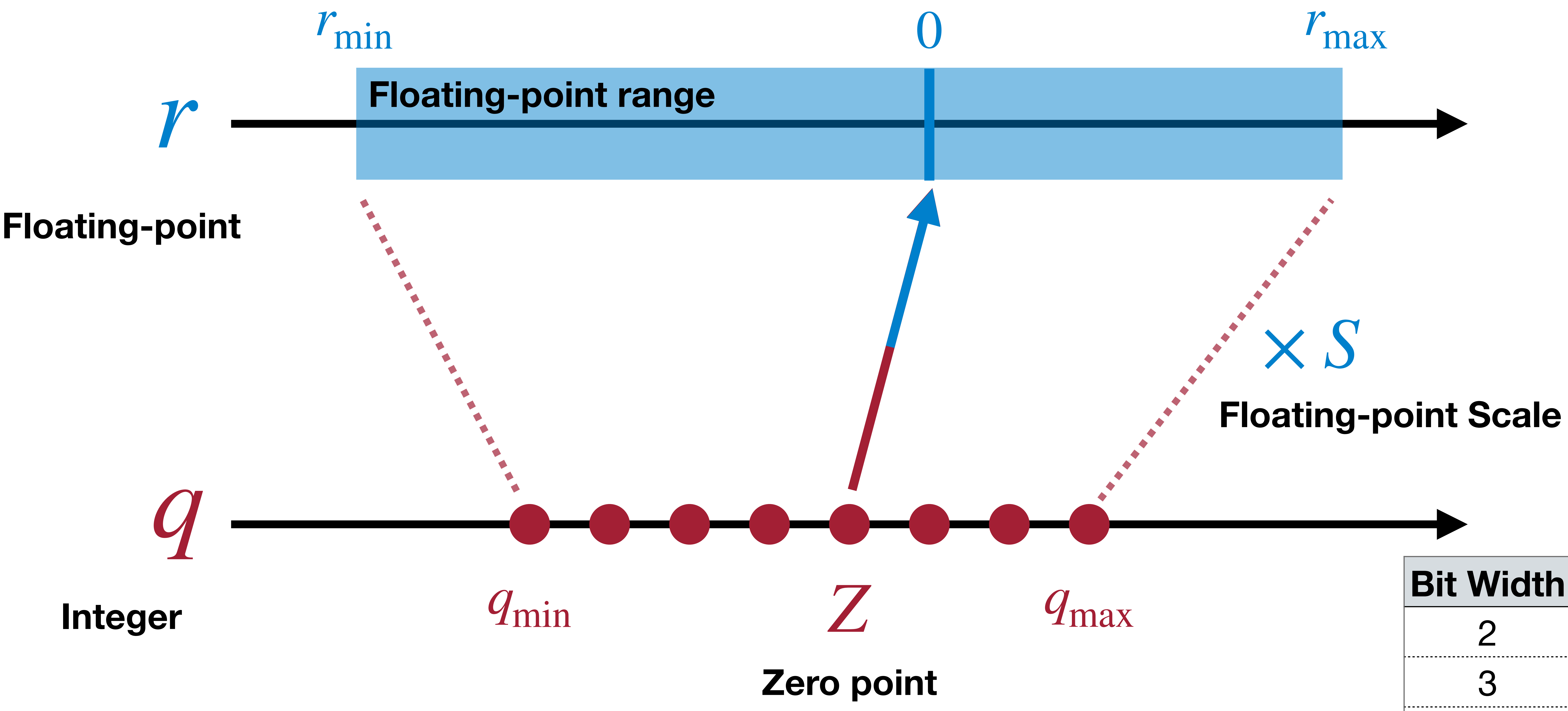
Integer (Z)
- quantization parameter
- allow real number *r=0* be exactly representable by a quantized integer Z

Floating-point (S)
- quantization parameter

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

Linear Quantization

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

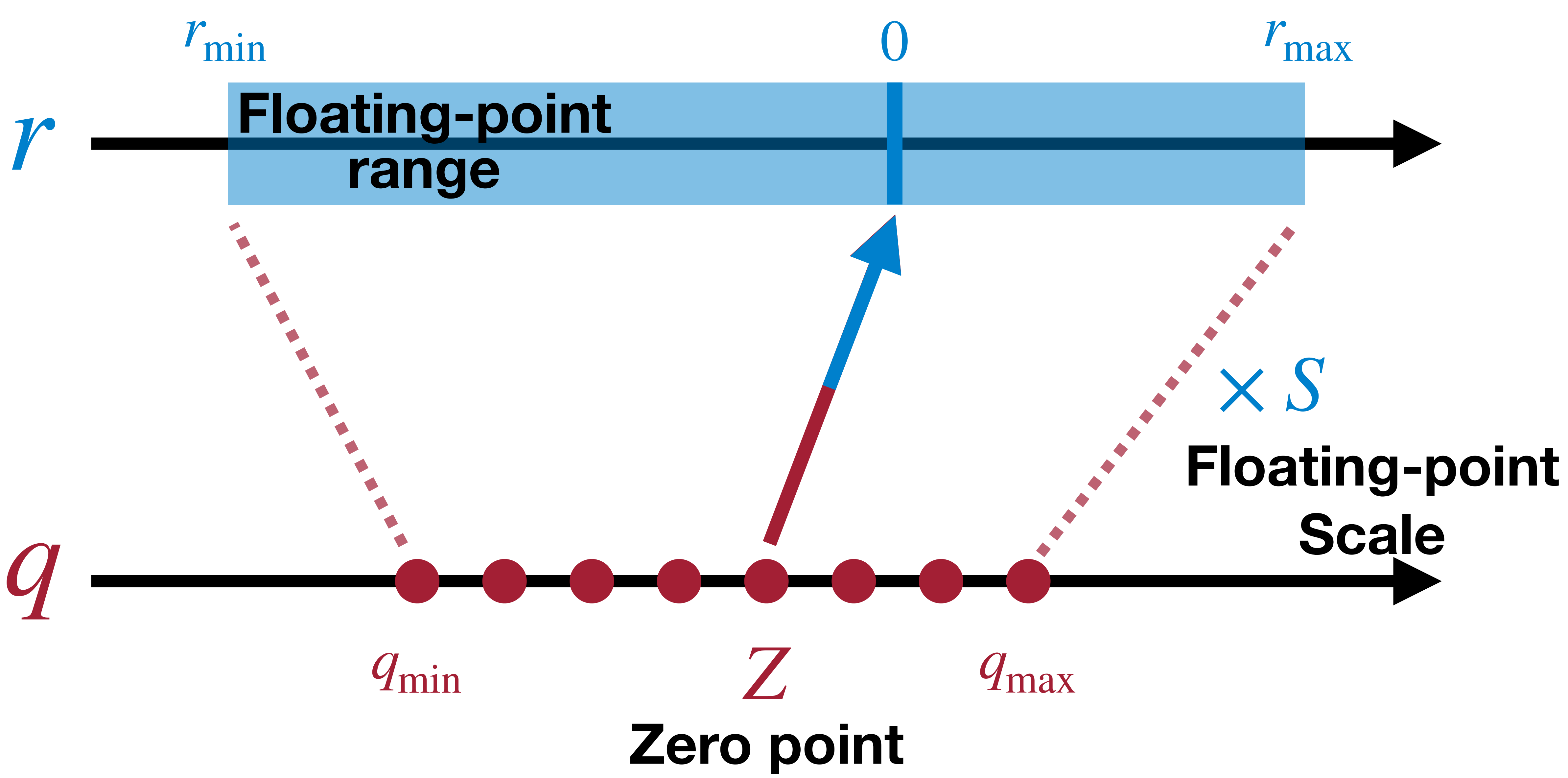


Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

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

Scale of Linear Quantization

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



$$r_{\max} = S\left(q_{\max} - Z\right)$$

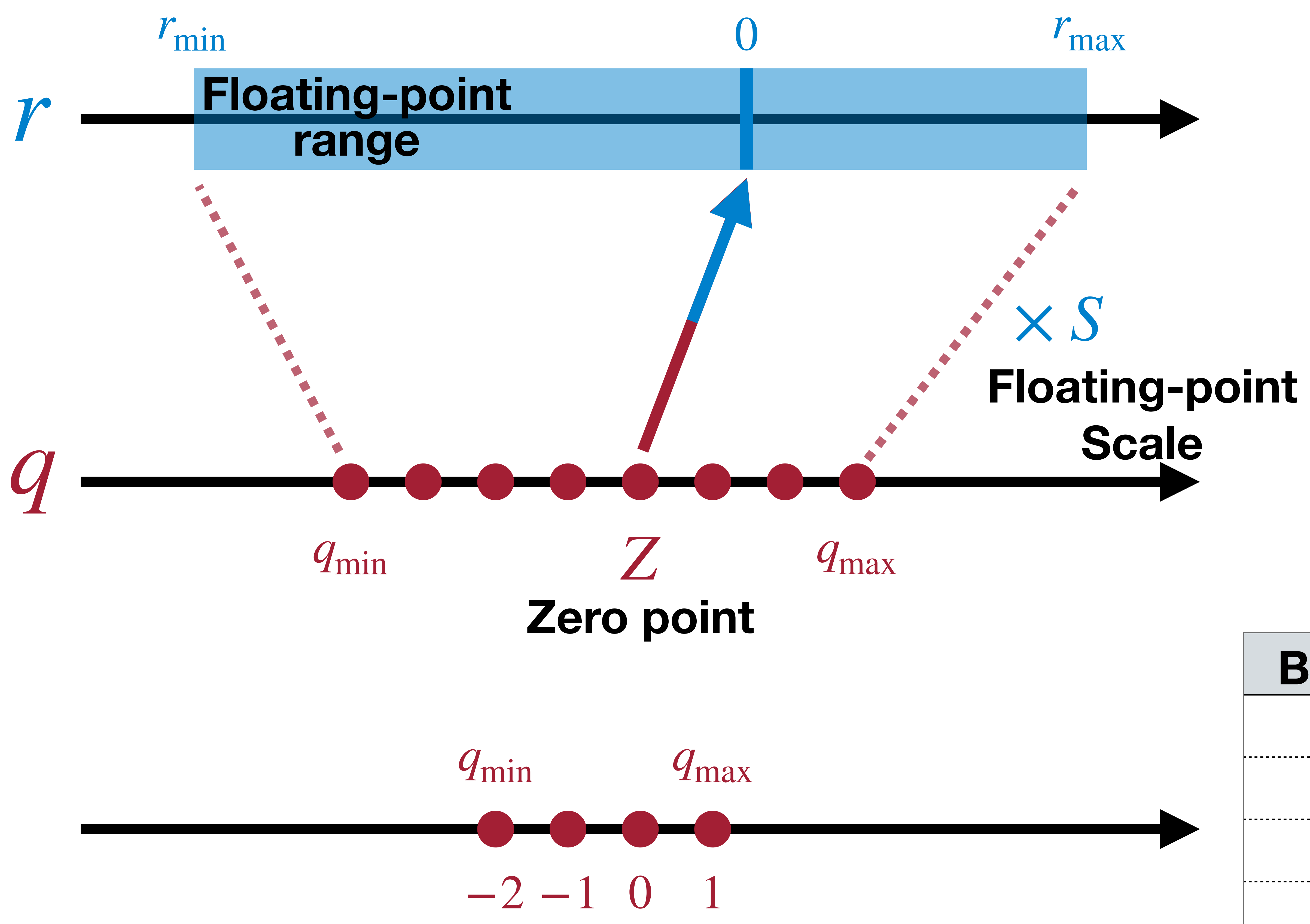
$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$r_{\max} - r_{\min} = S\left(q_{\max} - q_{\min}\right)$$

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

Scale 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*

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

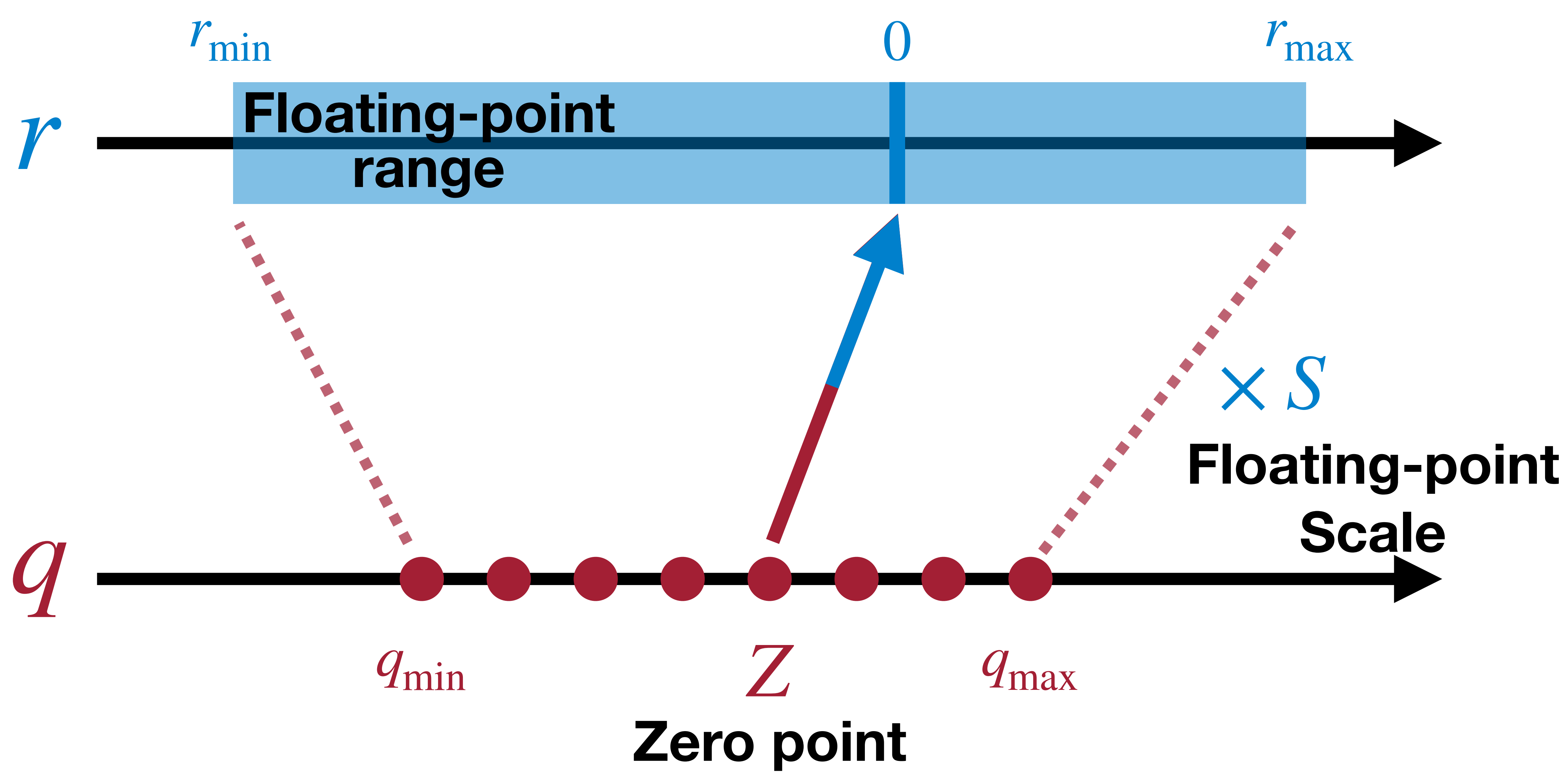
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

$$= \frac{2.12 - (-1.08)}{1 - (-2)}$$

$$= 1.07$$

Zero Point of Linear Quantization

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



$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$\downarrow$$

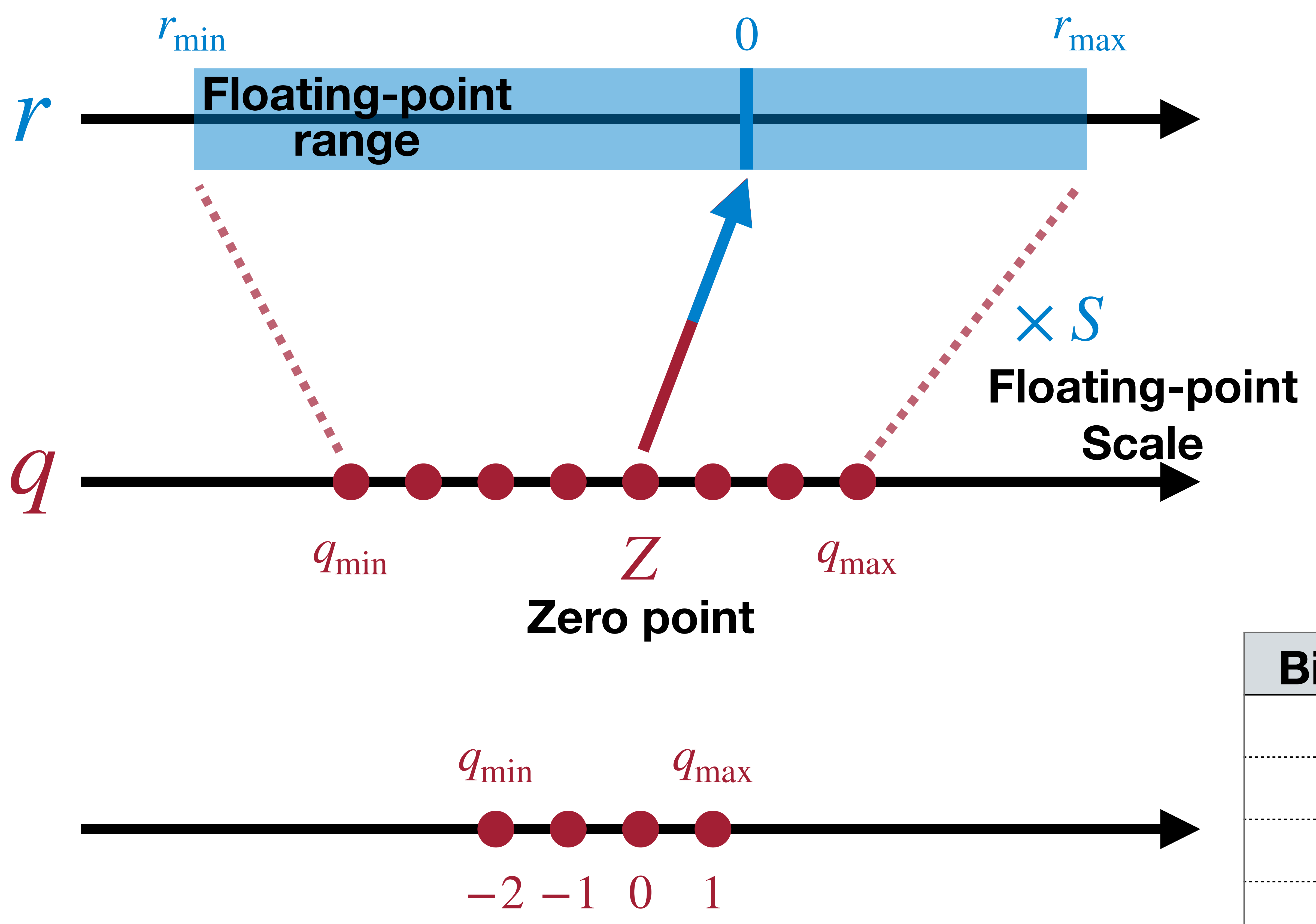
$$Z = q_{\min} - \frac{r_{\min}}{S}$$

$$\downarrow$$

$$Z = \mathrm{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

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

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

$$= \mathrm{round}\left(-2 - \frac{-1.08}{1.07}\right)$$

$$= -1$$

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

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication.

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

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right) \cdot S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right)$$

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

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

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

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication.

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

$$\mathbf{q_Y} = \boxed{\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}} \left(\mathbf{q_W}\mathbf{q_X} - Z_\mathbf{W}\mathbf{q_X} \underbrace{- Z_\mathbf{X}\mathbf{q_W} + Z_\mathbf{W}Z_\mathbf{X}}_{\text{Precompute}} \right) + \boxed{Z_\mathbf{Y}}$$

N-bit Integer Multiplication
32-bit Integer Addition/Subtraction

N-bit Integer
Addition

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

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication.

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

$$\mathbf{q_Y} = \boxed{\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}} \boxed{\left(\mathbf{q_W}\mathbf{q_X} - Z_\mathbf{W}\mathbf{q_X} - Z_\mathbf{X}\mathbf{q_W} + Z_\mathbf{W} Z_\mathbf{X}\right)} + \boxed{Z_\mathbf{Y}}$$

- Empirically, the scale $\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}$ is always in the interval (0, 1).

Fixed-point Multiplication

$$\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}} = 2^{-n} M_0, \quad \text{where} \quad M_0 \in [0.5, 1)$$

Bit Shift

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

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication.

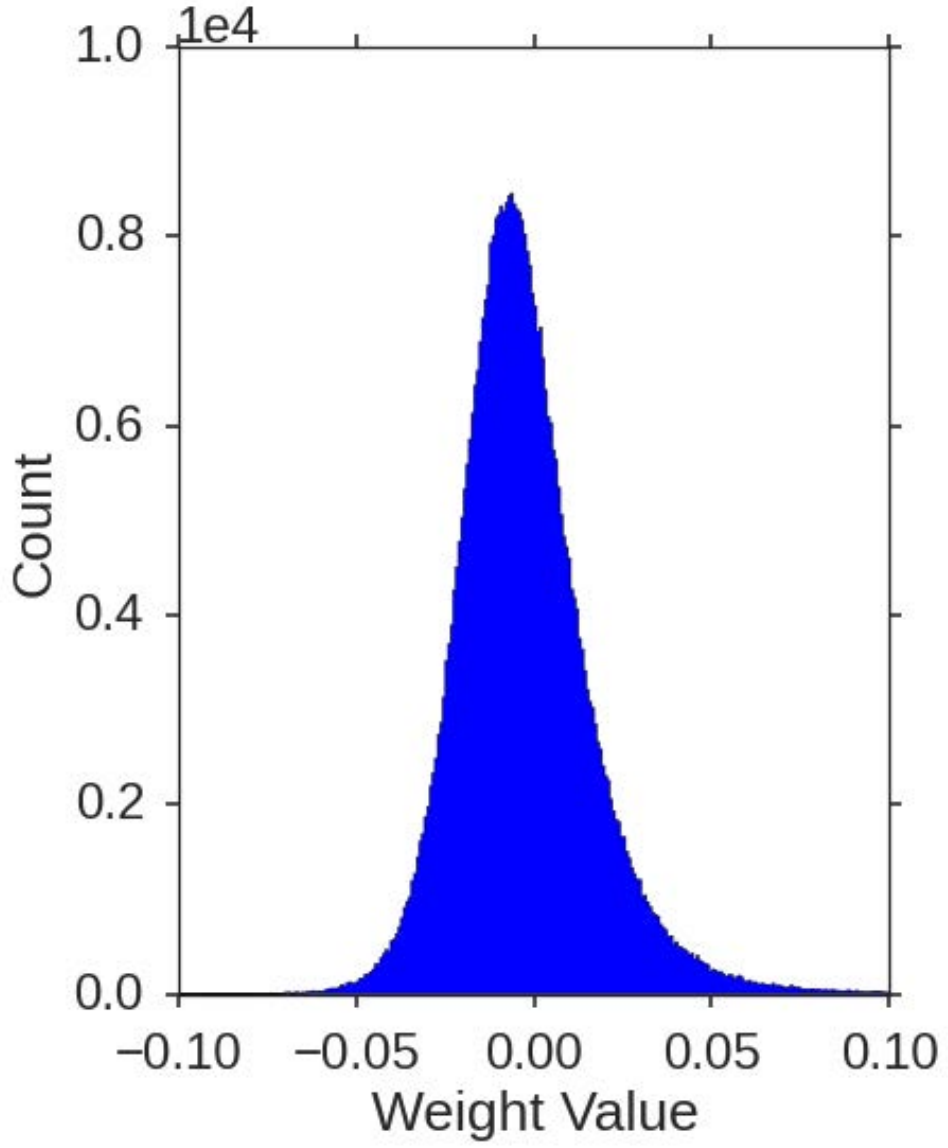
$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$$\mathbf{q_Y} = \boxed{\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}} \boxed{\left(\mathbf{q_W}\mathbf{q_X} - Z_\mathbf{W}\mathbf{q_X} \underbrace{- Z_\mathbf{X}\mathbf{q_W} + Z_\mathbf{W} Z_\mathbf{X}}_{\text{Precompute}}\right)} + \boxed{Z_\mathbf{Y}}$$

Rescale to *N*-bit Integer

N-bit Integer Multiplication
32-bit Integer Addition/Subtraction

N-bit Integer Addition

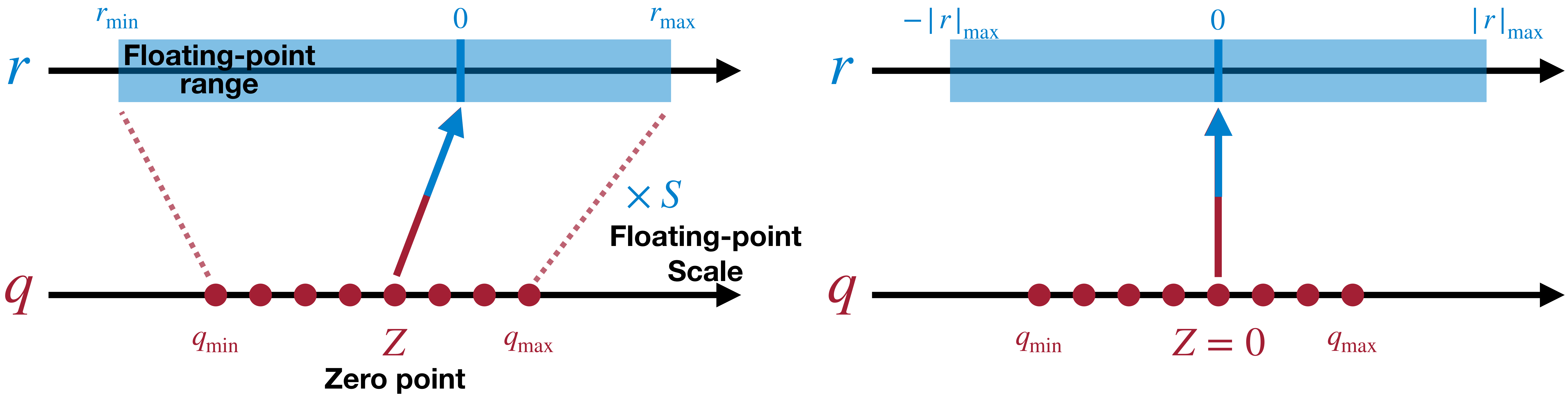


$$Z_\mathbf{W} = 0?$$

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

Symmetric Linear Quantization

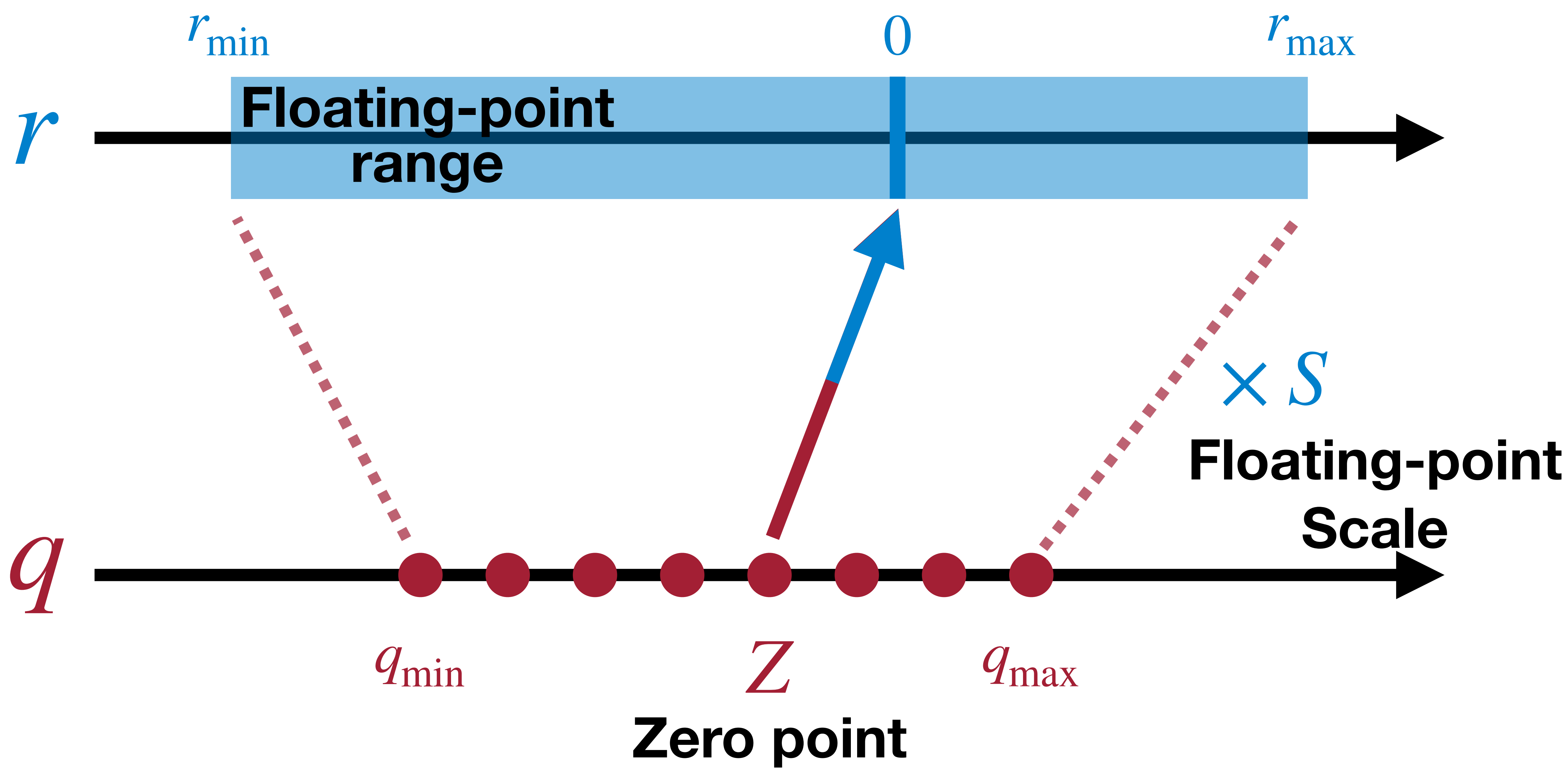
Zero point $Z = 0$ and Symmetric floating-point range

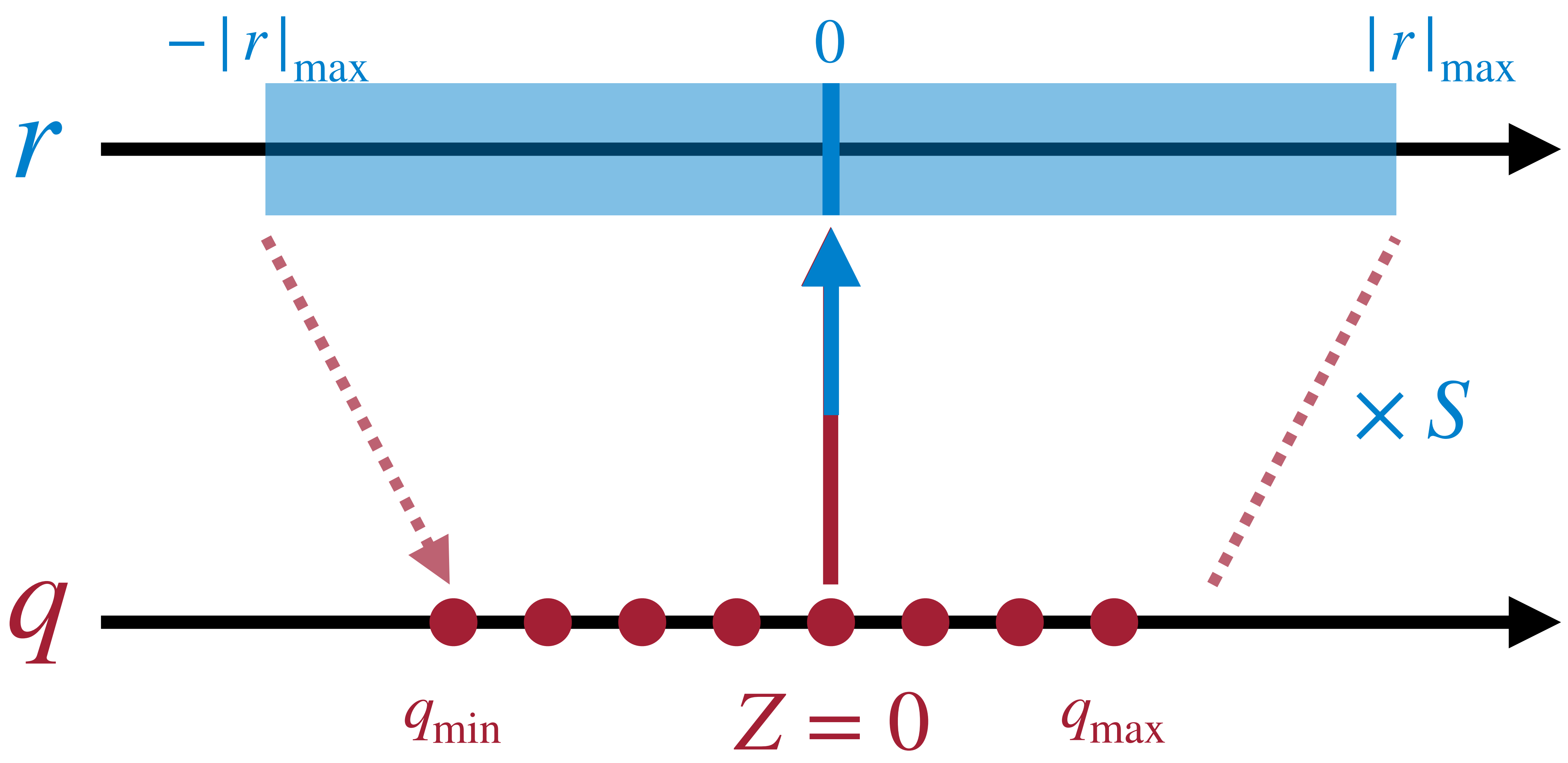


Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Symmetric Linear Quantization

Full range mode





$r_{\min} = S\left(q_{\min} - Z\right)$

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

$$S = \frac{r_{\min}}{q_{\min} - Z} = \frac{-|r|_{\max}}{q_{\min}} = \frac{|r|_{\max}}{2^{N-1}}$$

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

- use full range of quantized integers
- example: PyTorch's native quantization, ONNX

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication, when Zw=0.

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

$$\mathbf{q_Y} = \boxed{\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}} \boxed{\left(\mathbf{q_W}\mathbf{q_X} - Z_\mathbf{W}\mathbf{q_X} \underbrace{- Z_\mathbf{X}\mathbf{q_W} + Z_\mathbf{W} Z_\mathbf{X}}_{\text{Precompute}}\right)} + \boxed{Z_\mathbf{Y}}$$

Rescale to *N*-bit Integer

N-bit Integer Multiplication
32-bit Integer Addition/Subtraction

N-bit Integer Addition

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

$$\mathbf{q_Y} = \boxed{\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}} \boxed{\left(\mathbf{q_W}\mathbf{q_X} - Z_\mathbf{X}\mathbf{q_W}\right)} + \boxed{Z_\mathbf{Y}}$$

Linear Quantized Fully-Connected Layer

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

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

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

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right) \cdot S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right) + S_{\mathbf{b}}\left(\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{b}}\right)$$

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

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = \underline{\boxed{S_{\mathbf{W}}S_{\mathbf{X}}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}}\right)} + \underline{\boxed{S_{\mathbf{b}}}\left(\mathbf{q}_{\mathbf{b}} - \boxed{Z_{\mathbf{b}}}\right)}$$

Linear Quantized Fully-Connected Layer

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

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

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

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right) \cdot S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right) + S_{\mathbf{b}}\left(\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{b}}\right)$$

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

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

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

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

Linear Quantized Fully-Connected Layer

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

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

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

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

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

$$\Big\downarrow$$

Precompute

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

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

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

We will discuss how to compute activation zero point in the next lecture.

Linear Quantized Fully-Connected Layer

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

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

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

$$\downarrow \quad \begin{aligned} & Z_{\mathbf{W}} = 0 \\ & Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}} \\ & \mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}} \end{aligned}$$

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

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.

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

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

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

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

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

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.

$$\mathbf{Y} = \mathrm{Conv}\left(\mathbf{W}, \mathbf{X}\right) + \mathbf{b}$$

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

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

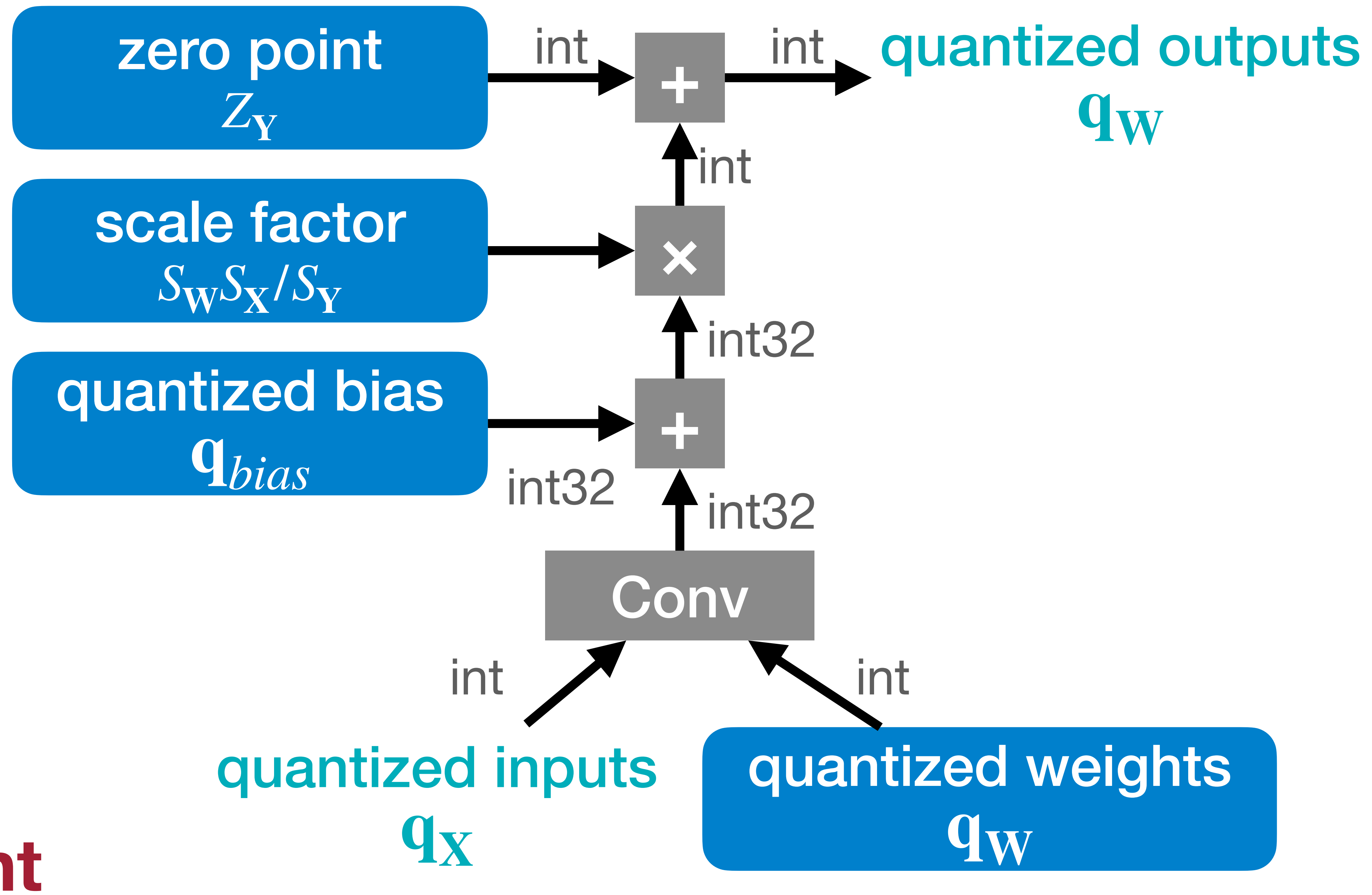
$$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - \mathrm{Conv}\left(\mathbf{q}_{\mathbf{W}}, Z_{\mathbf{X}}\right)$$

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

Rescale to *N*-bit Int

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

***N*-bit Int Add**

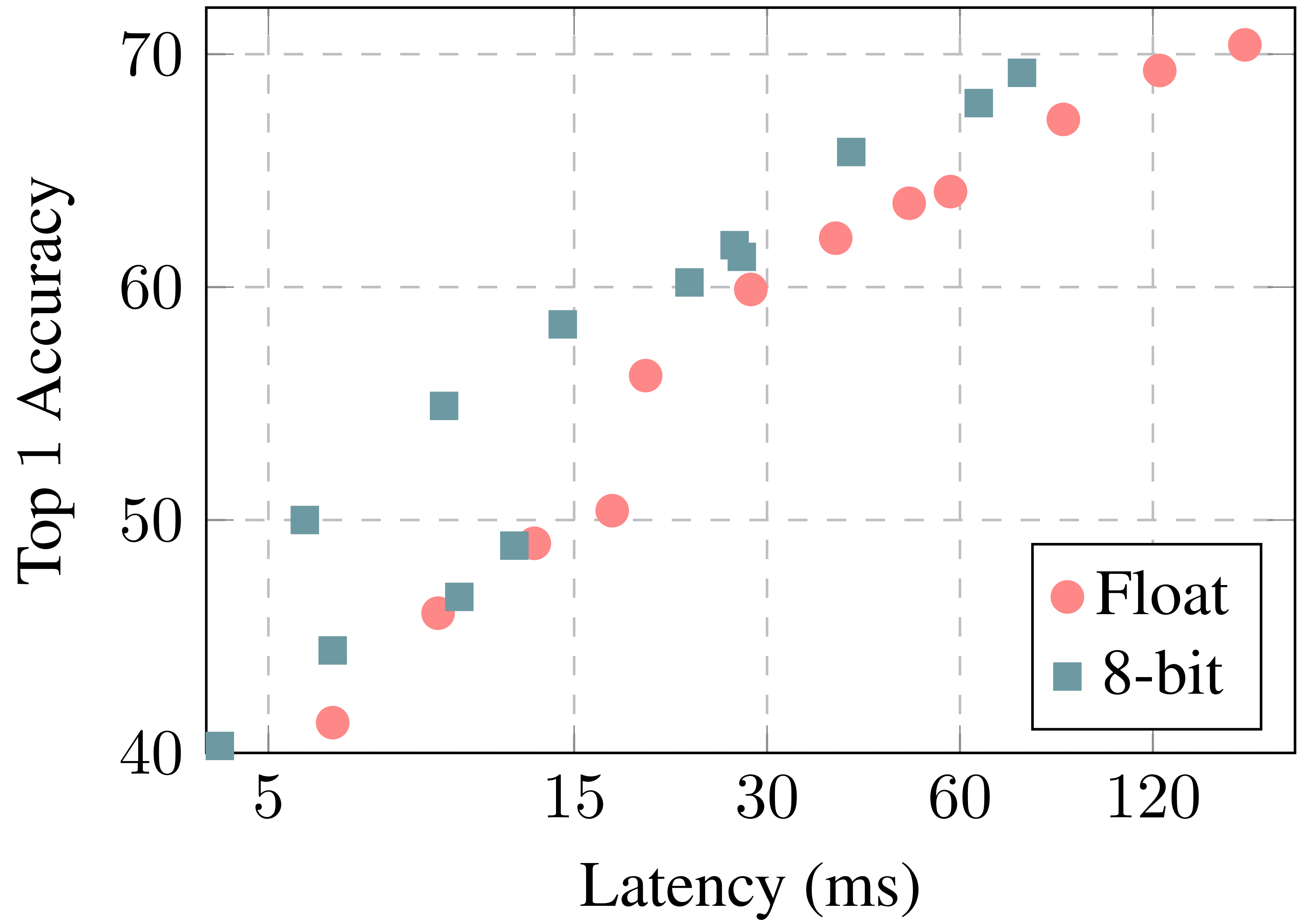


Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.

INT8 Linear Quantization

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

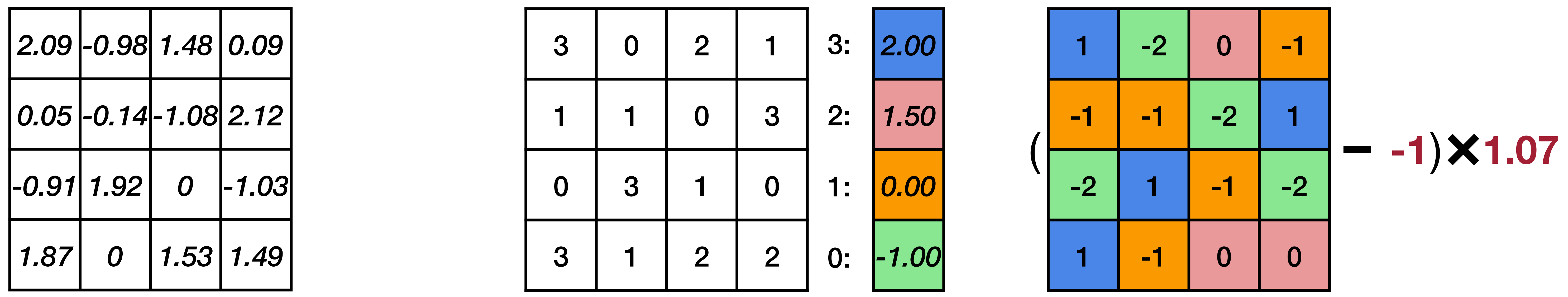
Neural Network	ResNet-50	Inception-V3
Floating-point Accuracy	76.4%	78.4%
8-bit Integer-quantized Acurracy	74.9%	75.4%



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



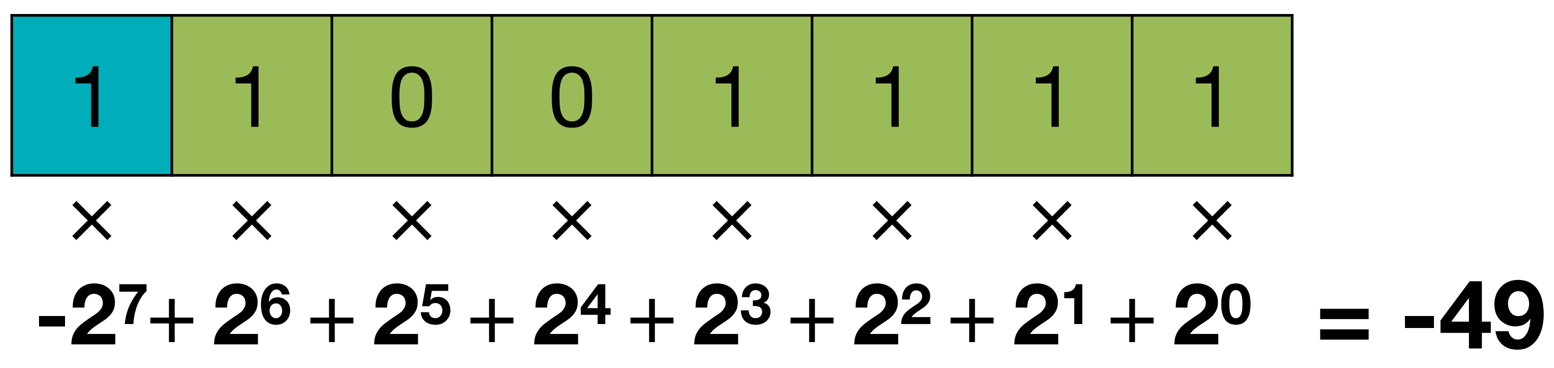
		K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

?

Summary of Today's Lecture

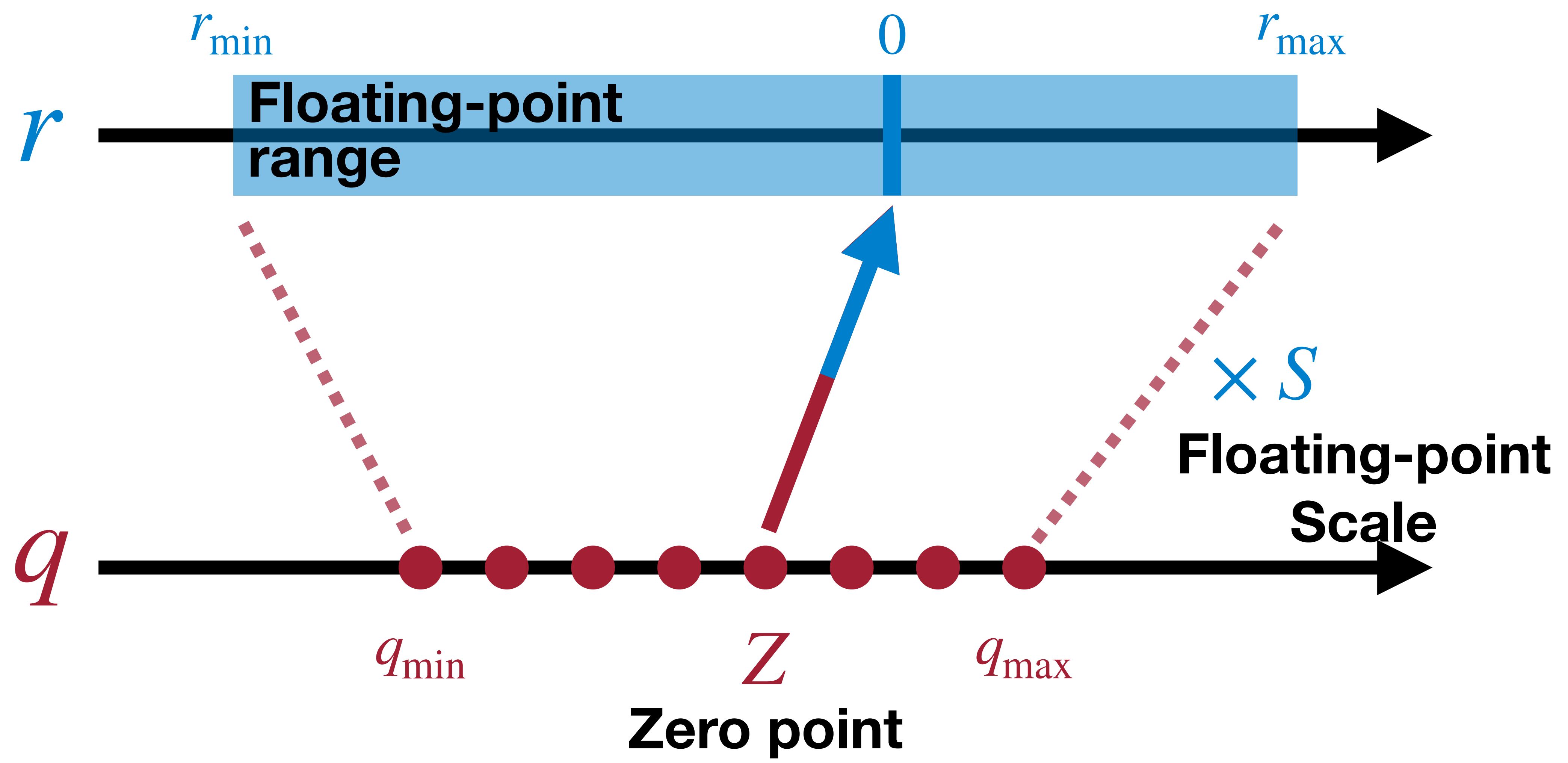
Today, we reviewed and learned

- the numeric data types used in the modern computing systems, including integers and floating-point numbers.



- the basic concept of **neural network quantization**: *converting the weights and activations of neural networks into a limited discrete set of numbers.*

- two types of common neural network quantization:
    - K-Means-based Quantization
    - Linear Quantization

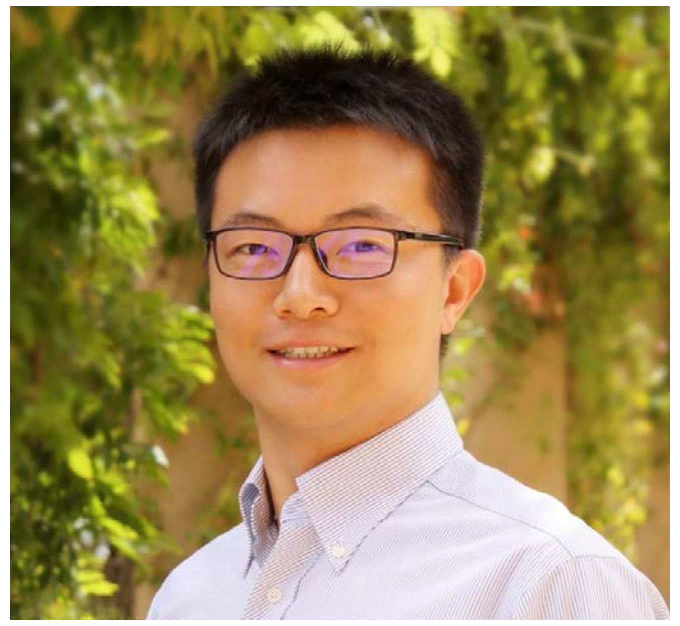


References

1. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey [Deng et al., IEEE 2020]
2. Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]
3. Deep Compression [Han et al., ICLR 2016]
4. Neural Network Distiller: https://intellabs.github.io/distiller/algo_quantization.html
5. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]
6. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations [Courbariaux et al., NeurIPS 2015]
7. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or –1. [Courbariaux et al., Arxiv 2016]
8. XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari et al., ECCV 2016]
9. Ternary Weight Networks [Li et al., Arxiv 2016]
10. Trained Ternary Quantization [Zhu et al., ICLR 2017]

EfficientML.ai Lecture 06
Quantization

Part II



Song Han

Associate Professor, MIT
Distinguished Scientist, NVIDIA

@SongHan_MIT

Lecture Plan

Today we will:

1. Review Linear Quantization.

2. Introduce **Post-Training Quantization (PTQ)** that quantizes a floating-point neural network model, including: channel quantization, group quantization, and range clipping.

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

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

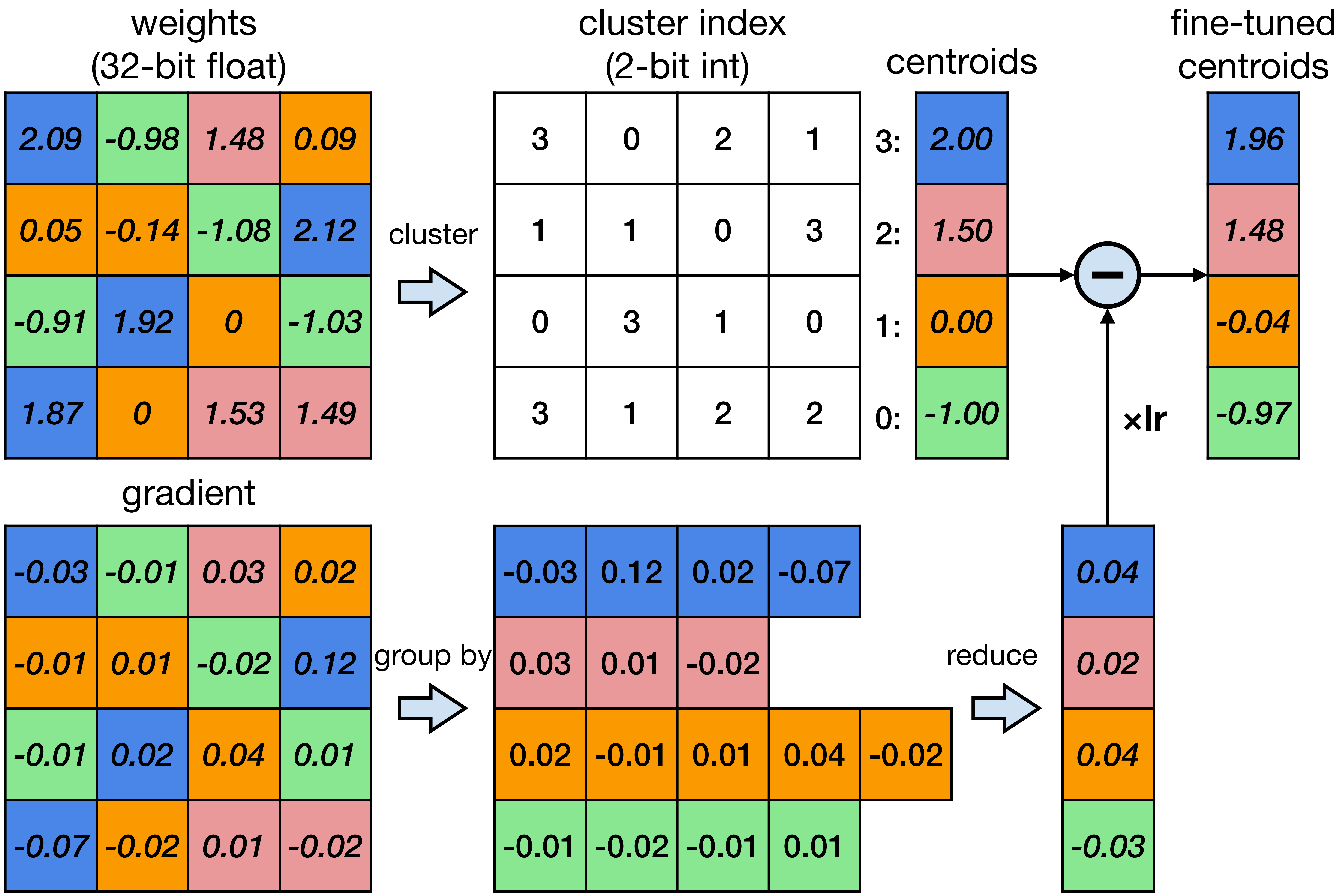
3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

$$\left(\begin{matrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{matrix} - -1\right) \times 1.07$$

		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

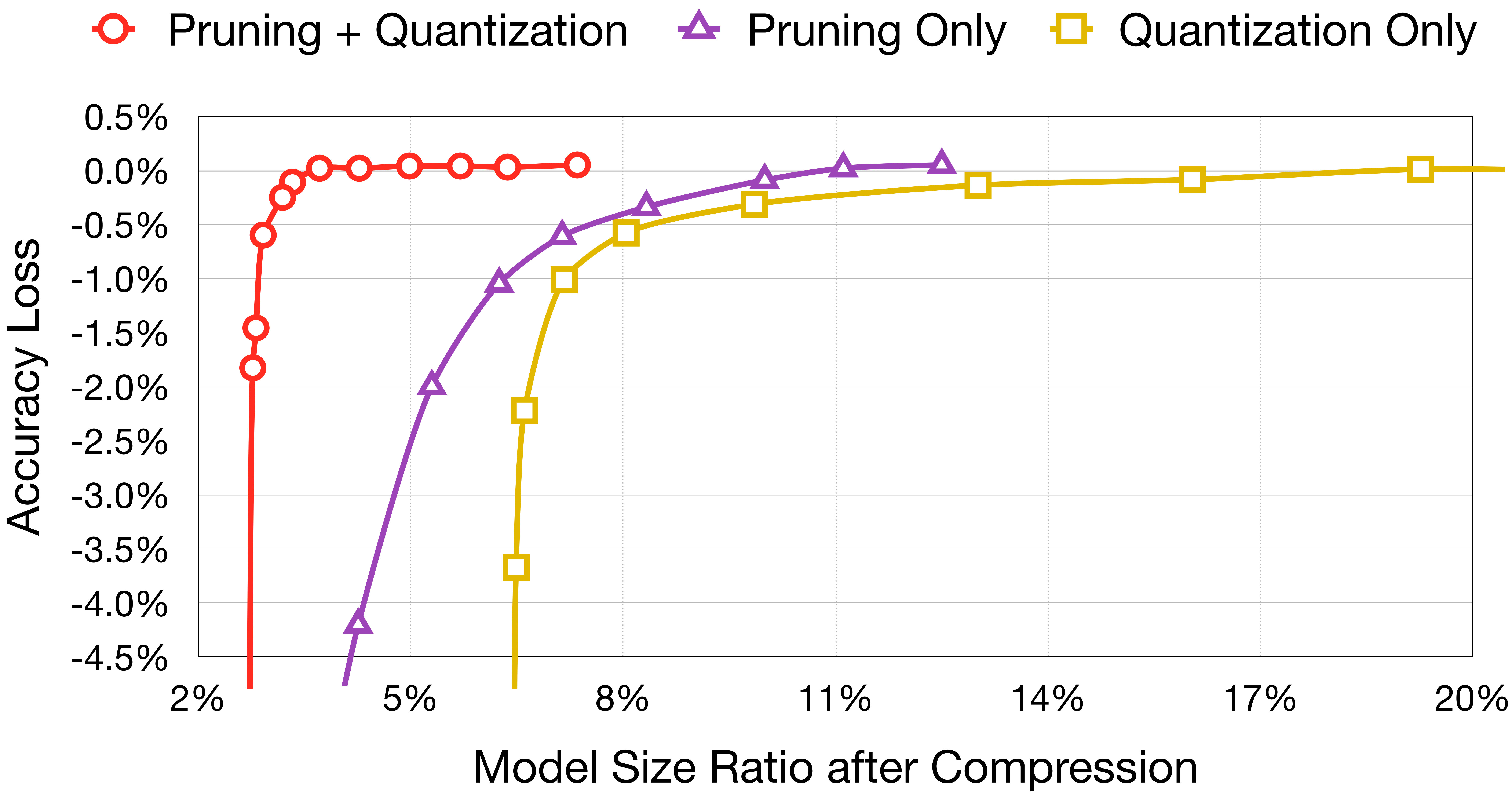
K-Means-based Weight Quantization



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 $r = S(q - Z)$

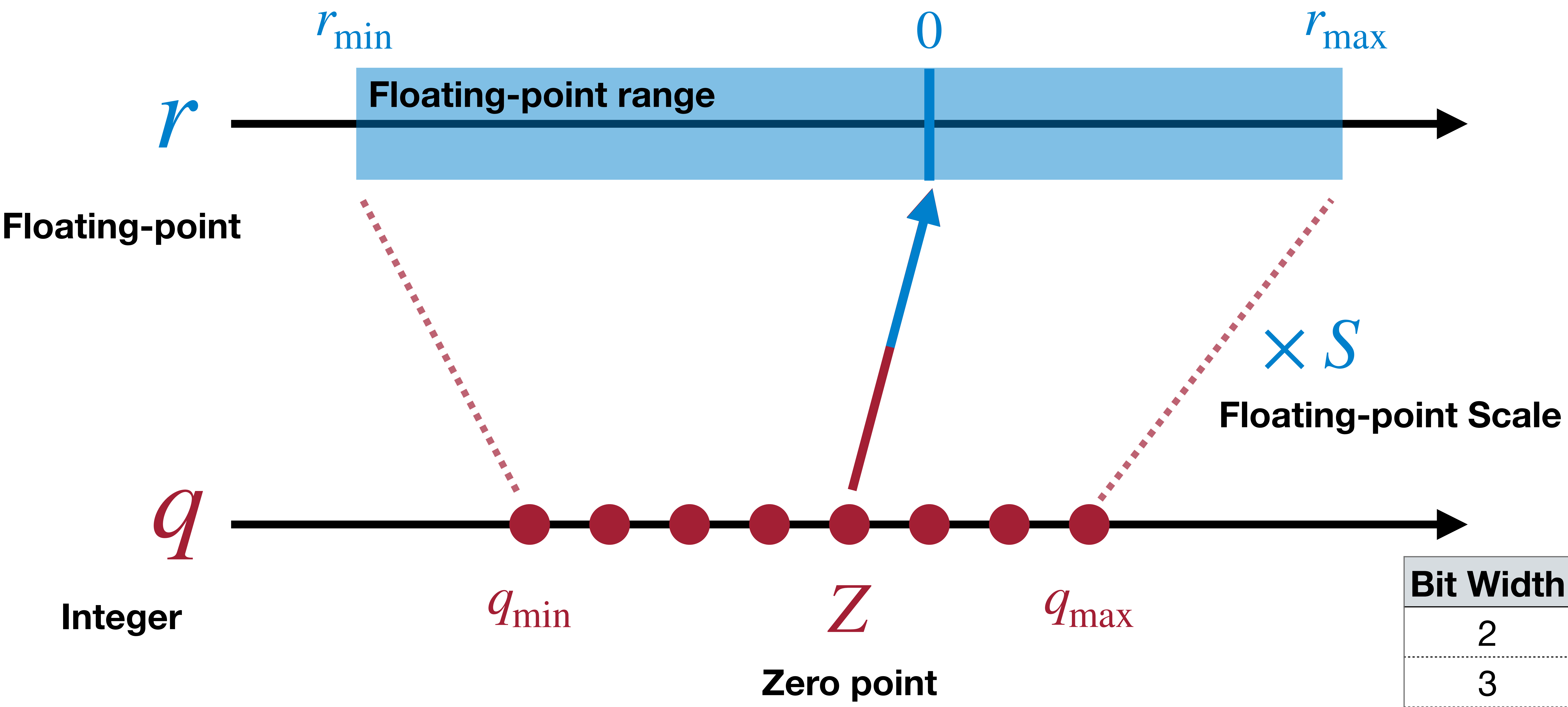
weights (32-bit float)	quantized weights (2-bit signed int)	zero point (2-bit signed int)	scale (32-bit float)

$$
\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}
\Rightarrow
\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - (-1) \right) \times 1.07 =
\begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}
$$

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.

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

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

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

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

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

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.

$$\mathbf{Y} = \mathrm{Conv}\left(\mathbf{W}, \mathbf{X}\right) + \mathbf{b}$$

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

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

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

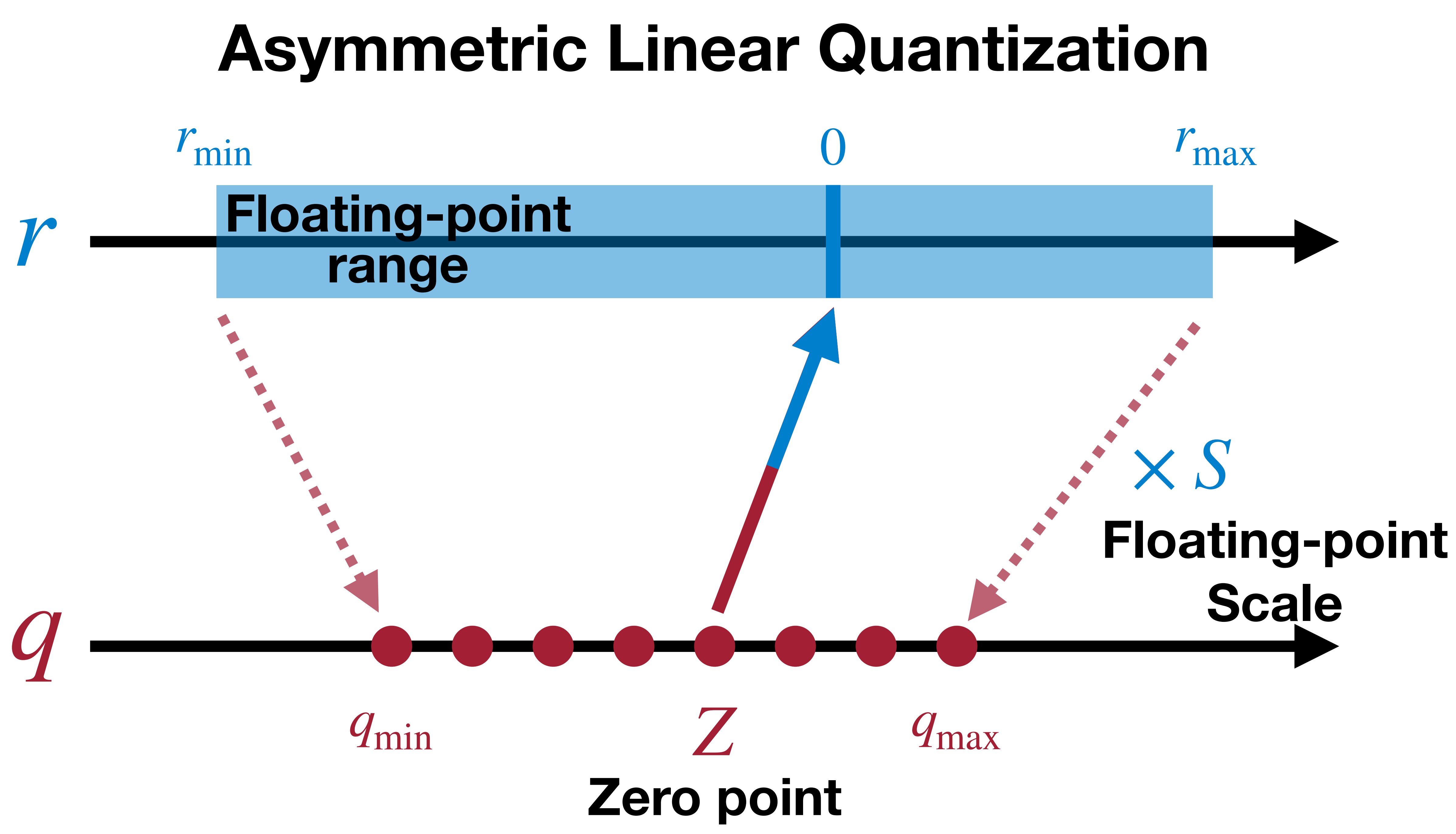
$$\mathbf{q}_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \boxed{\left(\mathrm{Conv}\left(\mathbf{q}_{\mathbf{W}}, \mathbf{q}_{\mathbf{X}}\right) + \mathbf{q}_{bias}\right)} + \boxed{Z_{\mathbf{Y}}}$$

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

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

$$= \frac{2.12 - (-1.08)}{1 - (-2)}$$

$$= 1.07$$

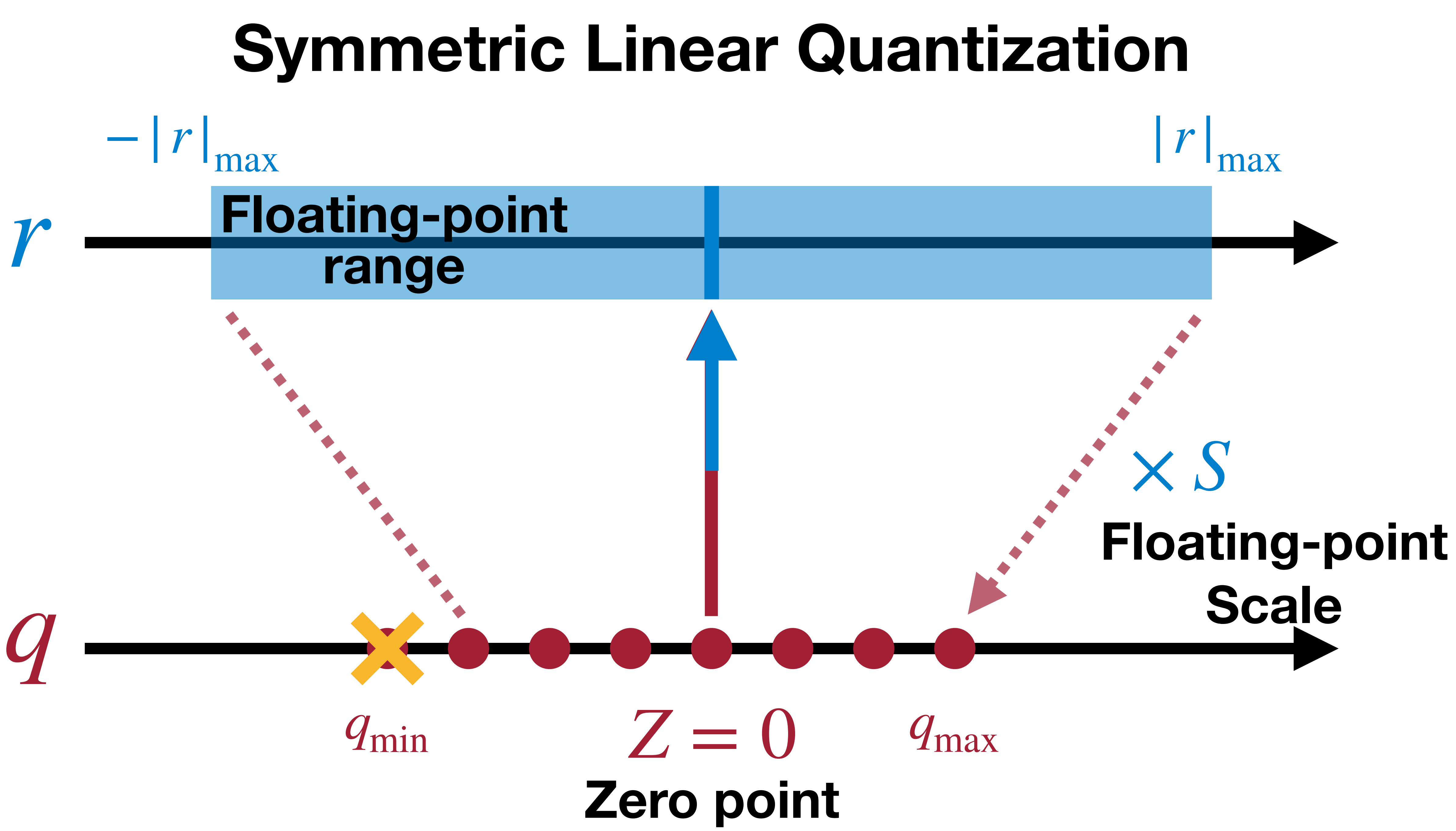
$$Z = q_{\min} - \frac{r_{\min}}{S}$$

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

$$= -1$$

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

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

$$Z = 0$$

Post-Training Quantization

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

Topic I: Quantization Granularity
Topic II: Dynamic Range Clipping
Topic III: Rounding

Post-Training Quantization

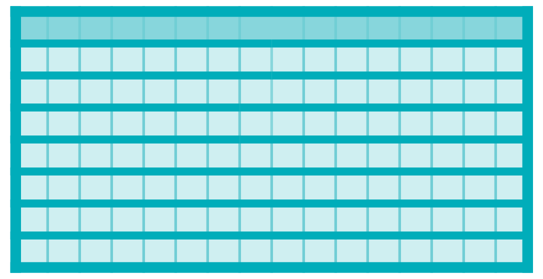
How should we get the optimal linear quantization parameters (S, Z)?

Topic I: Quantization Granularity
Topic II: Dynamic Range Clipping
Topic III: Rounding

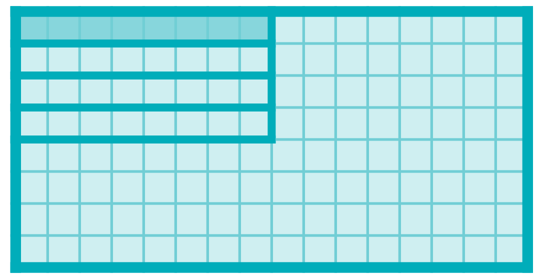
Quantization Granularity

- Per-Tensor Quantization

- Per-Channel Quantization



- Group Quantization



    - Per-Vector Quantization
    - Shared Micro-exponent (MX) data type

Quantization Granularity

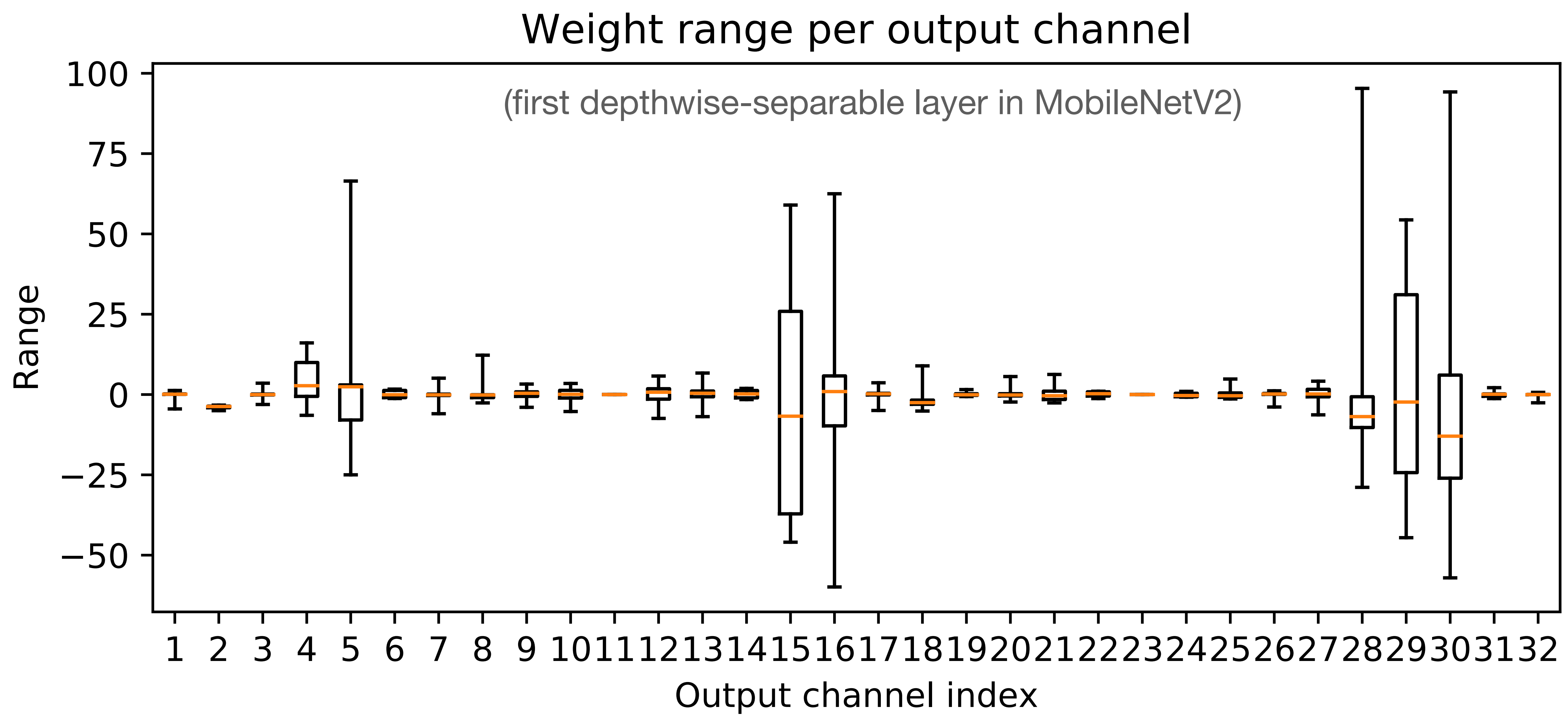
- **Per-Tensor Quantization**

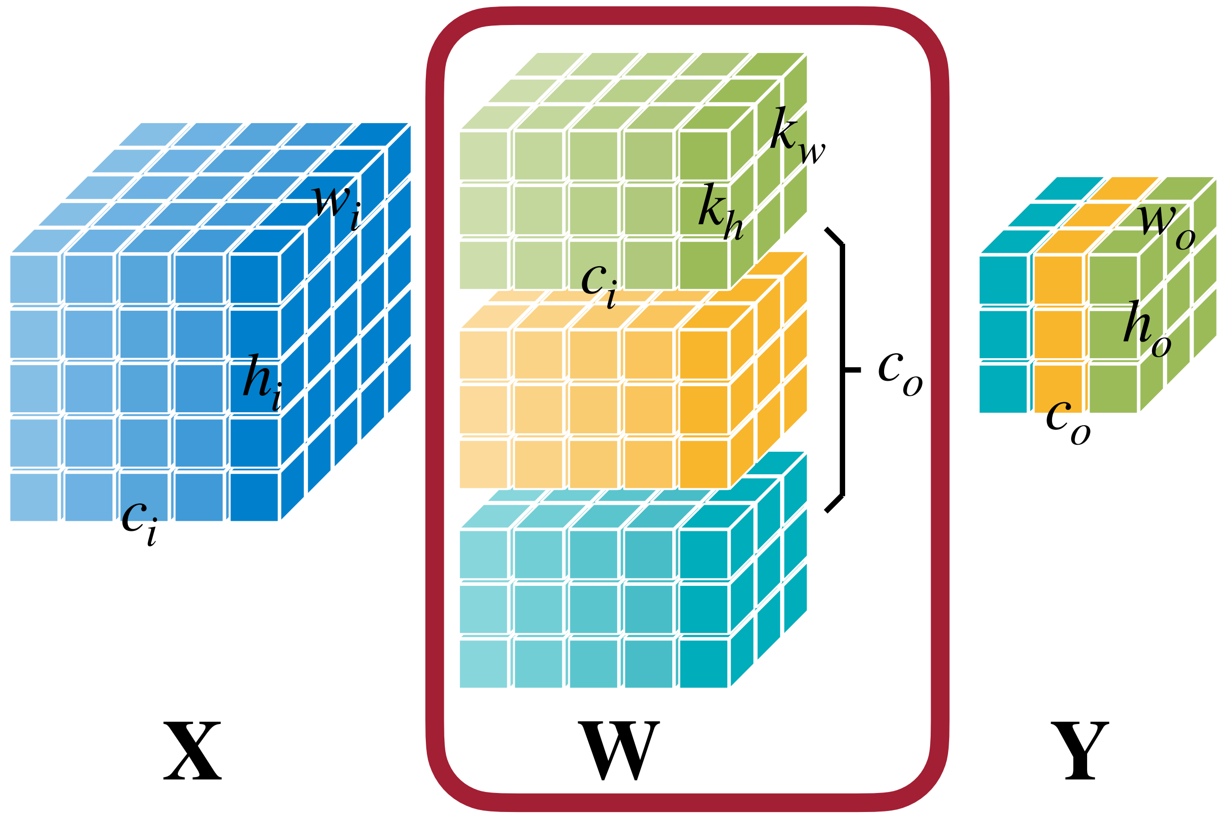
- Per-Channel Quantization

- Group Quantization

  - Per-Vector Quantization
  - Shared Micro-exponent (MX) data type

Symmetric Linear Quantization on Weights





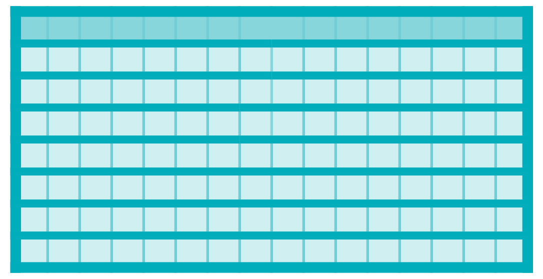
- $|r|_{\max} = |\mathbf{W}|_{\max}$
- Using *single* scale S for whole weight tensor (**Per-Tensor Quantization**)
  - works well for large models
  - accuracy drops for small models
- Common failure results from
  - large differences (more than 100×) in ranges of weights for different output channels — outlier weight
- 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]

Quantization Granularity

- Per-Tensor Quantization

- **Per-Channel Quantization**



- Group Quantization

  - Per-Vector Quantization
  - Shared Micro-exponent (MX) data type

Per-Channel Weight Quantization

Example: 2-bit linear quantization

$$
\begin{array}{cc}
 & ic \\
oc & \begin{array}{|c|c|c|c|}
\hline
2.09 & -0.98 & 1.48 & 0.09 \\
\hline
0.05 & -0.14 & -1.08 & 2.12 \\
\hline
-0.91 & 1.92 & 0 & -1.03 \\
\hline
1.87 & 0 & 1.53 & 1.49 \\
\hline
\end{array}
\end{array}
$$

Per-Channel Quantization

Per-Tensor Quantization

Per-Channel Weight Quantization

Example: 2-bit linear quantization

ic (columns), oc (rows)

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

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

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

Per-Channel Weight Quantization

Example: 2-bit linear quantization

Weight matrix (oc rows × ic columns):

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

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

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

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

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

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

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

Per-Channel Weight Quantization

Example: 2-bit linear quantization

Weight matrix (ic columns, oc rows):

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

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

Per-Channel Weight Quantization

Example: 2-bit linear quantization

ic (columns), oc (rows)

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

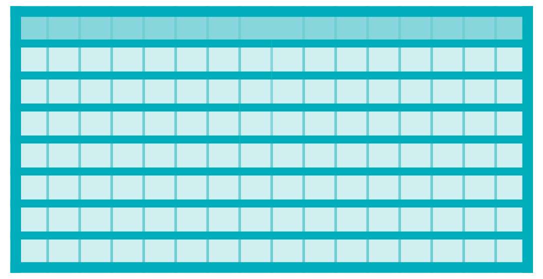
2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	0
2.12	0	2.12	2.12

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

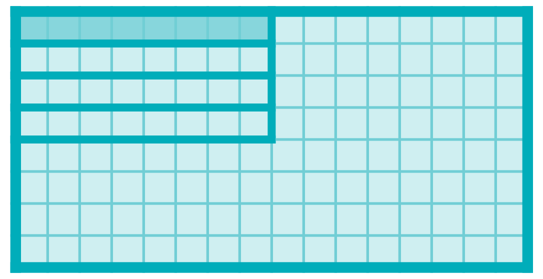
Quantization Granularity

- Per-Tensor Quantization

- Per-Channel Quantization



- **Group Quantization**

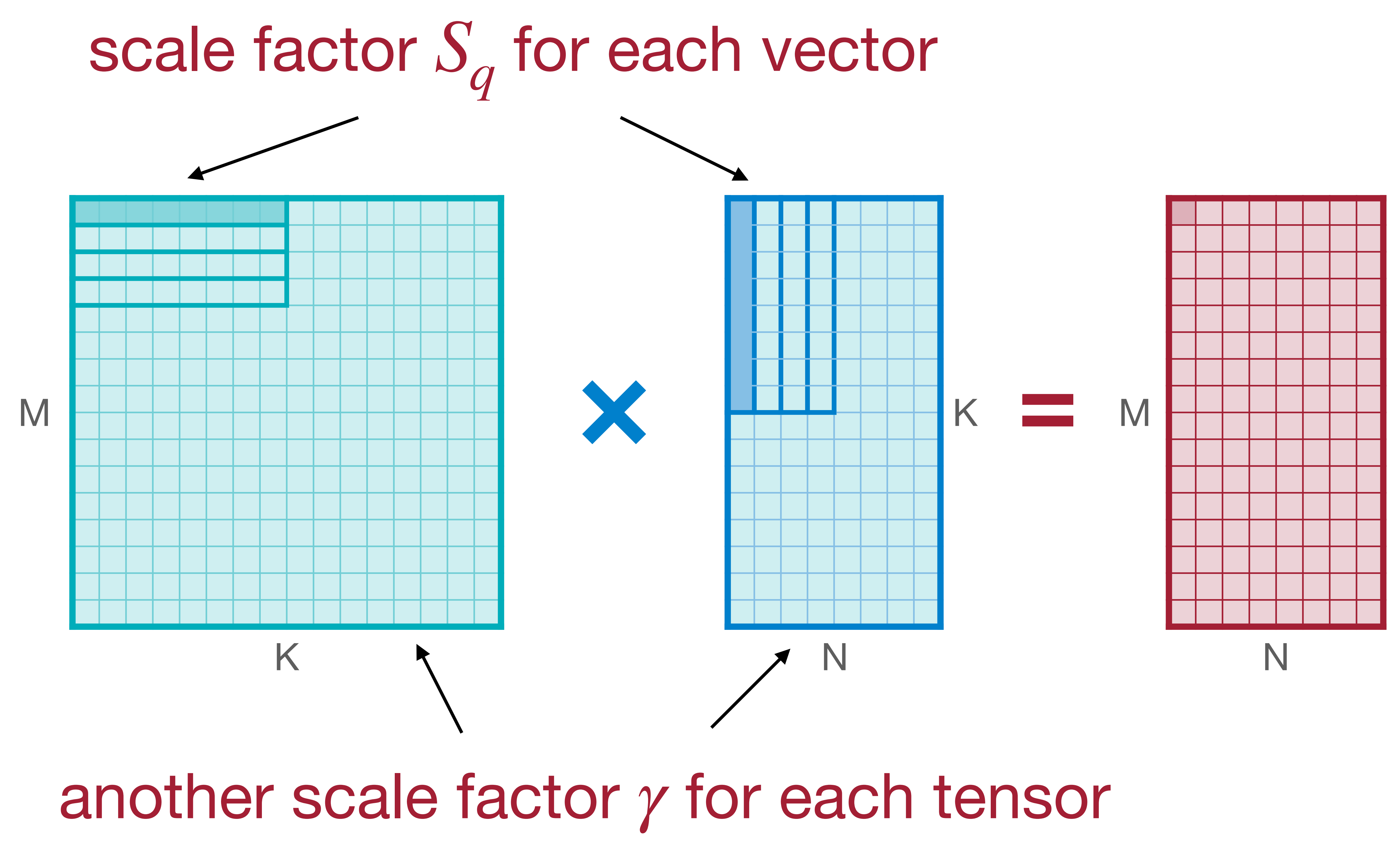


  - **Per-Vector Quantization**
  - **Shared Micro-exponent (MX) data type**

VS-Quant: Per-vector Scaled Quantization

Hierarchical scaling factor

- $r = S(q - Z) \rightarrow r = \gamma \cdot S_q(q - Z)$
  - γ is a floating-point coarse grained scale factor
  - S_q is an integer per-vector scale factor
  - achieves a balance between accuracy and hardware efficiency by
    - less expensive integer scale factors at finer granularity
    - more expensive floating-point scale factors at coarser granularity

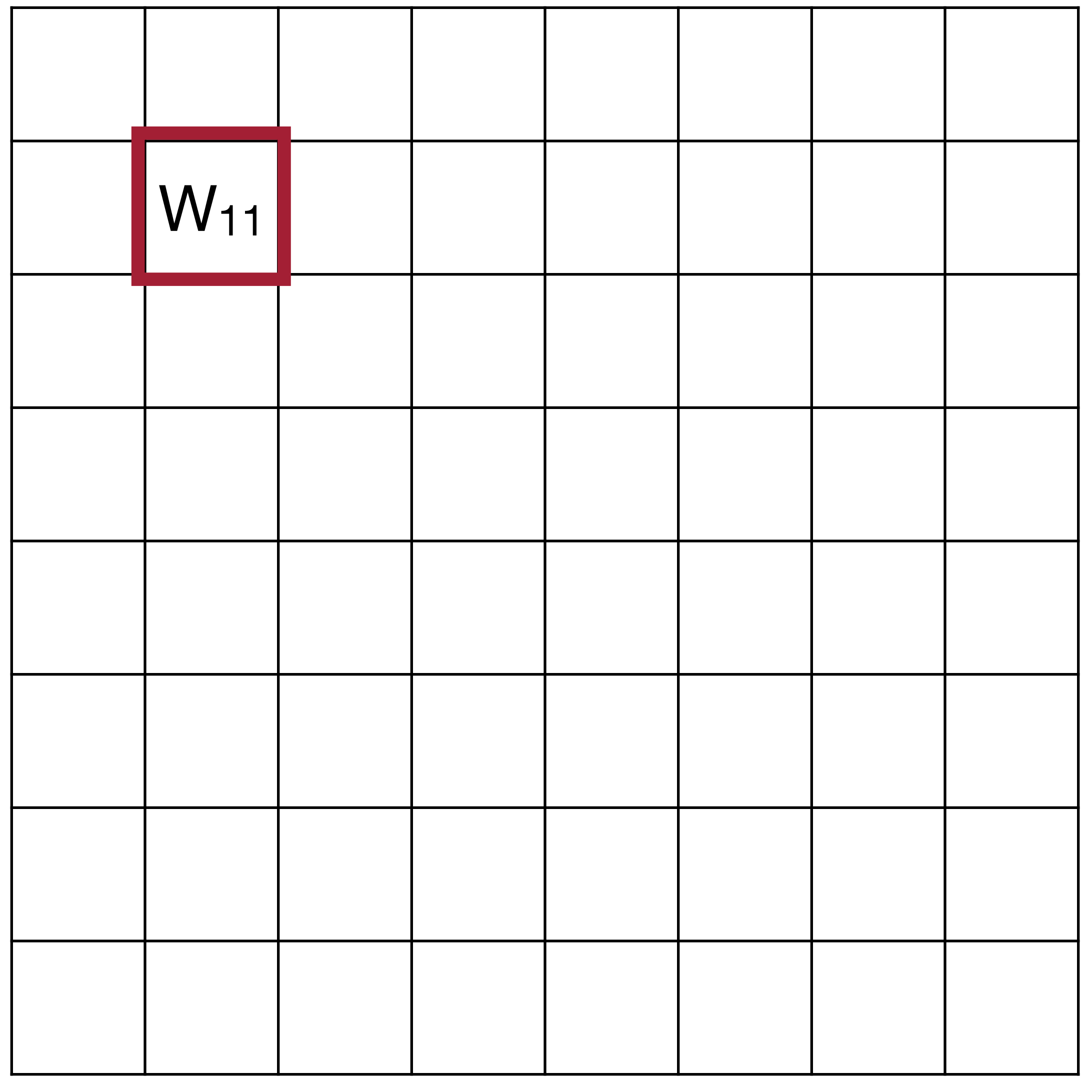


- Memory Overhead of two-level scaling:
  - Given 4-bit quantization with 4-bit per-vector scale for every 16 elements, the effective bit width is 4 + 4 / 16 = 4.25 bits.

VS-Quant: Per-Vector Scaled Quantization for Accurate Low-Precision Neural Network Inference [Steve Dai, *et al.*]

Group Quantization

Multi-level scaling scheme



$$r = (q - z) \cdot s \rightarrow$$

$$r = (q - z) \cdot s_{l_0} \cdot s_{l_1} \cdot \cdots$$

r : real number value

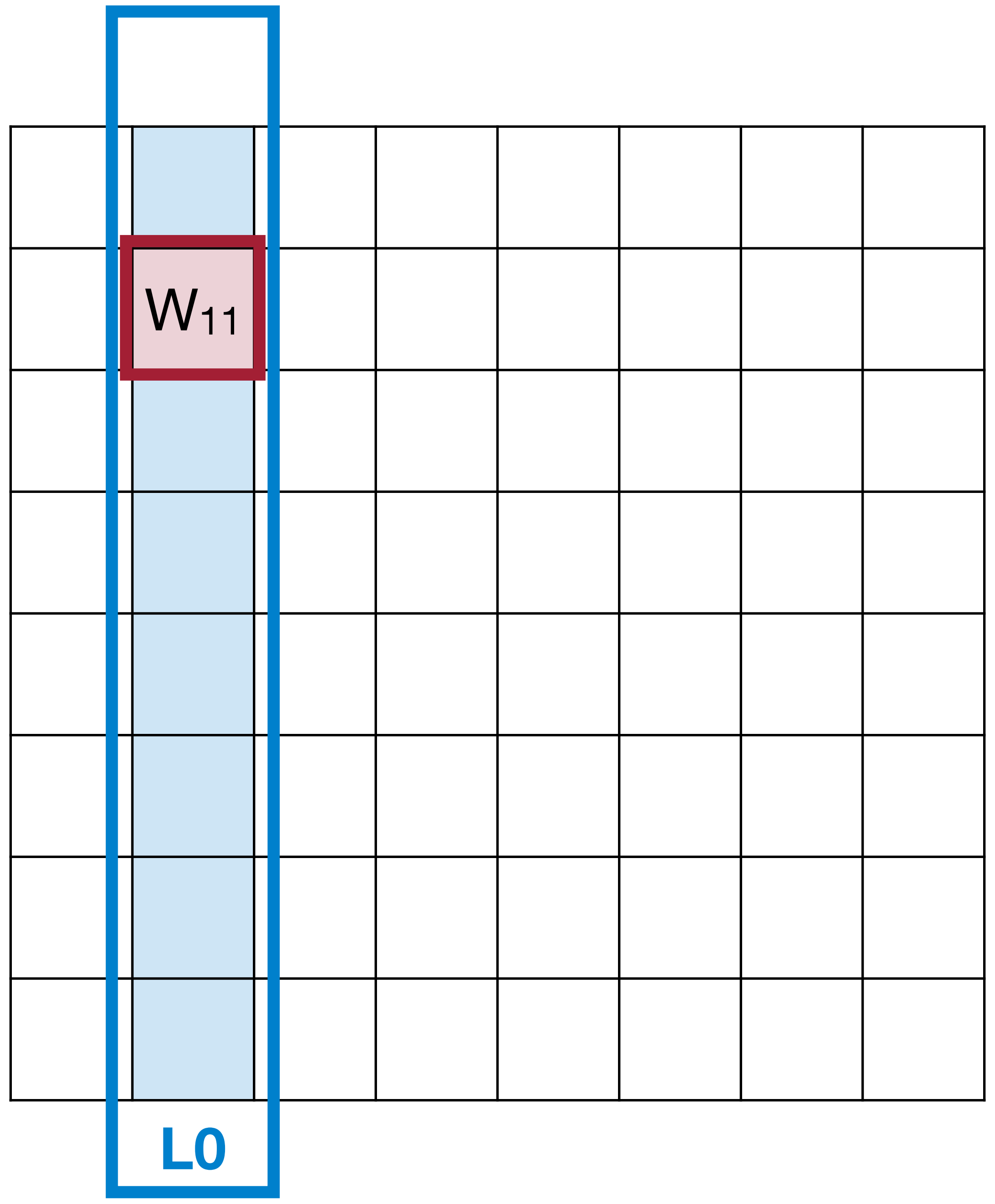
q : quantized value

z : zero point ($z = 0$ is symmetric quantization)

s : scale factors of different levels

Group Quantization

Multi-level scaling scheme



$$r = (q - z) \cdot s_{l_0} \cdot s_{l_1} \cdot \cdots$$

r : real number value

q : quantized value

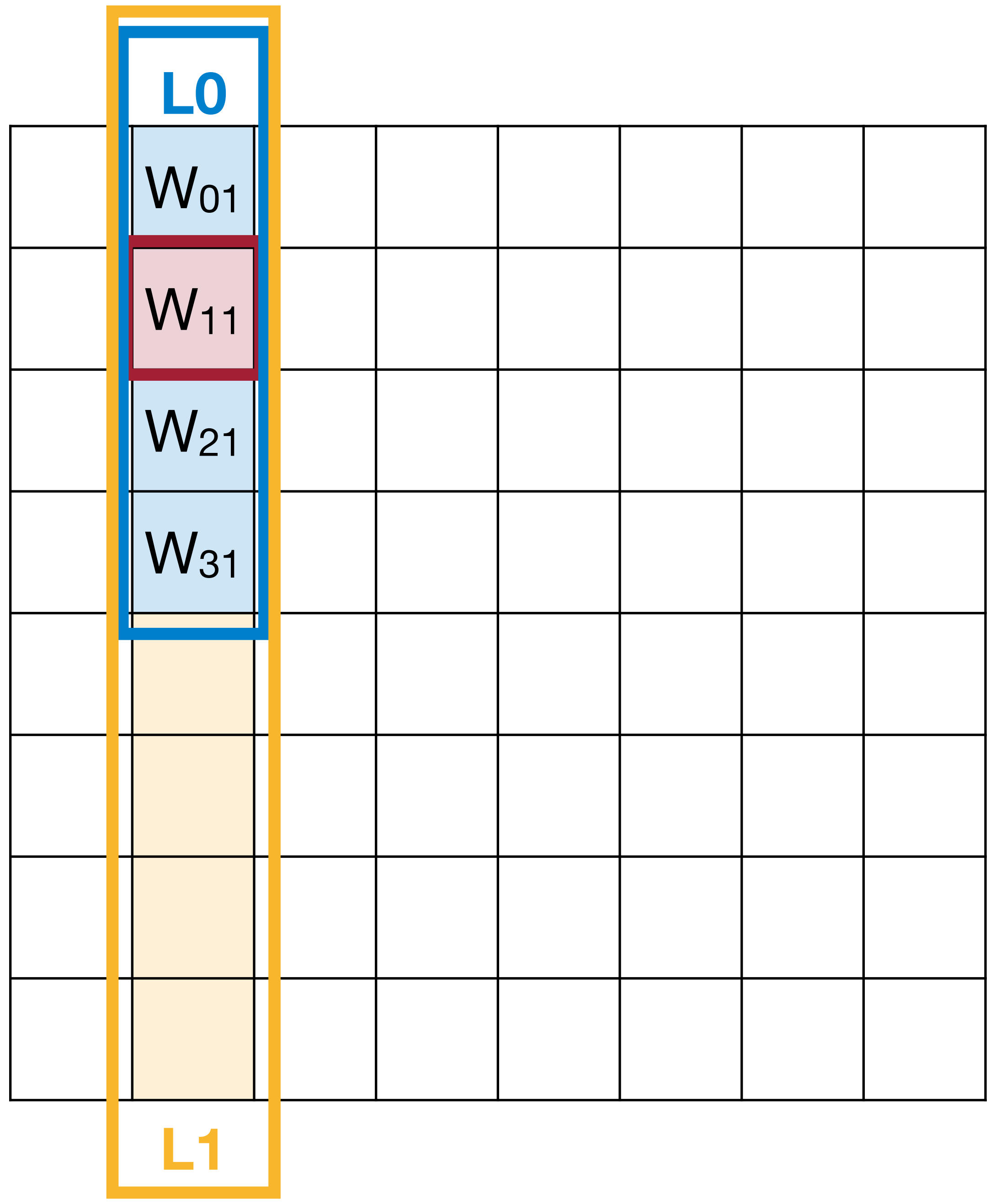
z : zero point ($z = 0$ is symmetric quantization)

s : scale factors of different levels

Quantization Approach	Data Type	L0 Group Size	L0 Scale Data Type	L1 Group Size	L1 Scale Data Type	Effective Bit Width
Per-Channel Quant	INT4	Per Channel	FP16	-	-	4

Group Quantization

Multi-level scaling scheme



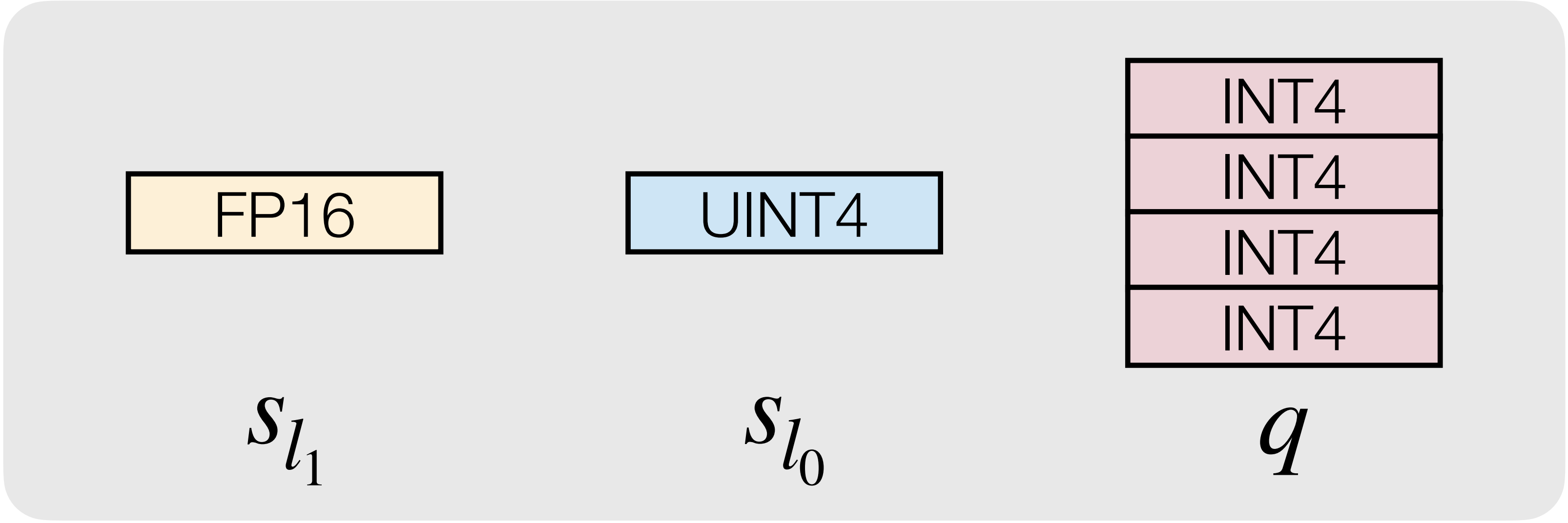
$$r = (q - z) \cdot s_{l_0} \cdot s_{l_1} \cdots$$

r : real number value

q : quantized value

z : zero point ($z = 0$ is symmetric quantization)

s : scale factors of different levels

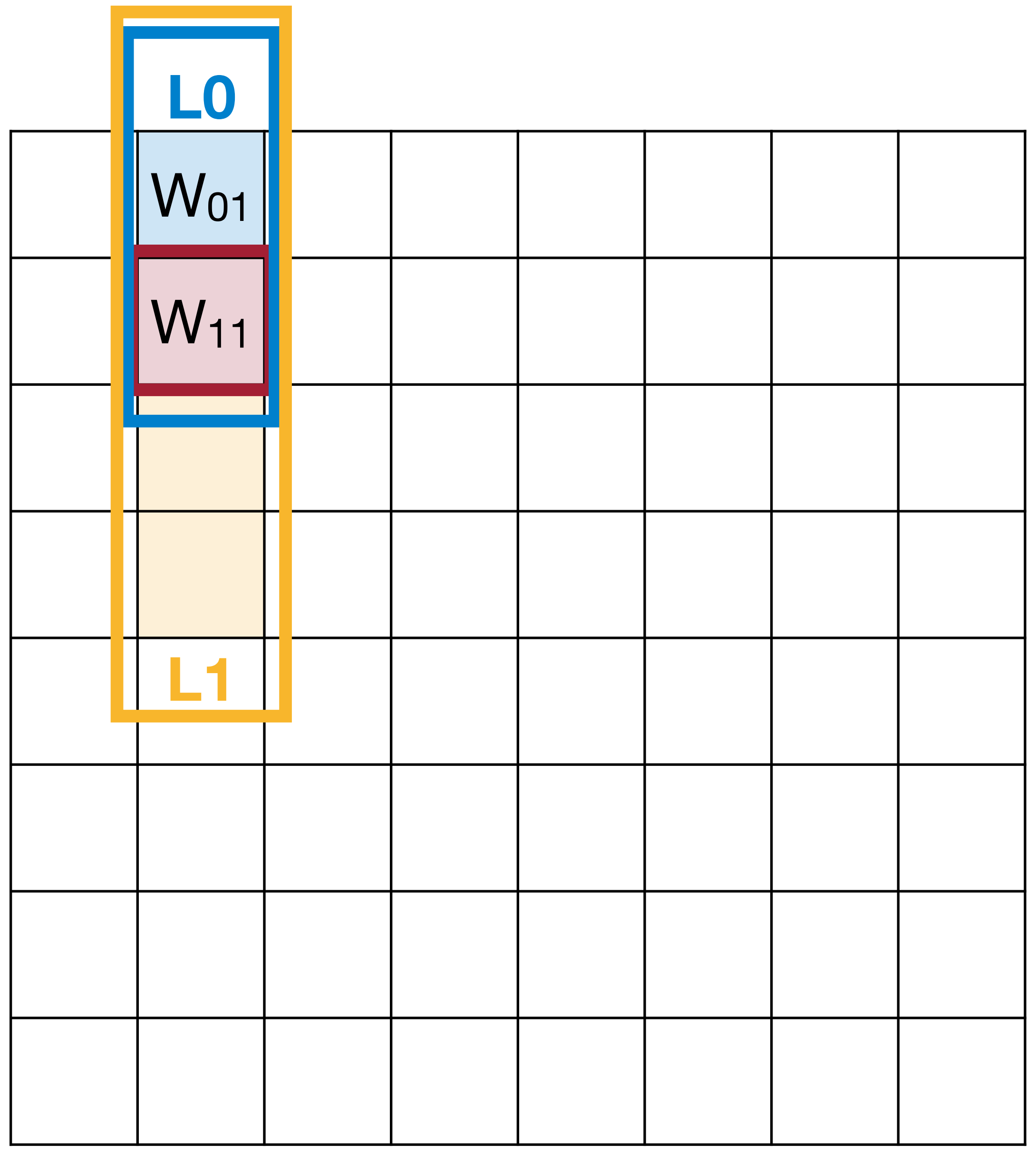


Quantization Approach	Data Type	L0 Group Size	L0 Scale Data Type	L1 Group Size	L1 Scale Data Type	Effective Bit Width
Per-Channel Quant	INT4	Per Channel	FP16	-	-	4
VSQ	INT4	16	UINT4	Per Channel	FP16	4+4/16=4.25

VS-Quant: Per-Vector Scaled Quantization for Accurate Low-Precision Neural Network Inference [Steve Dai, *et al.*]

Group Quantization

Multi-level scaling scheme



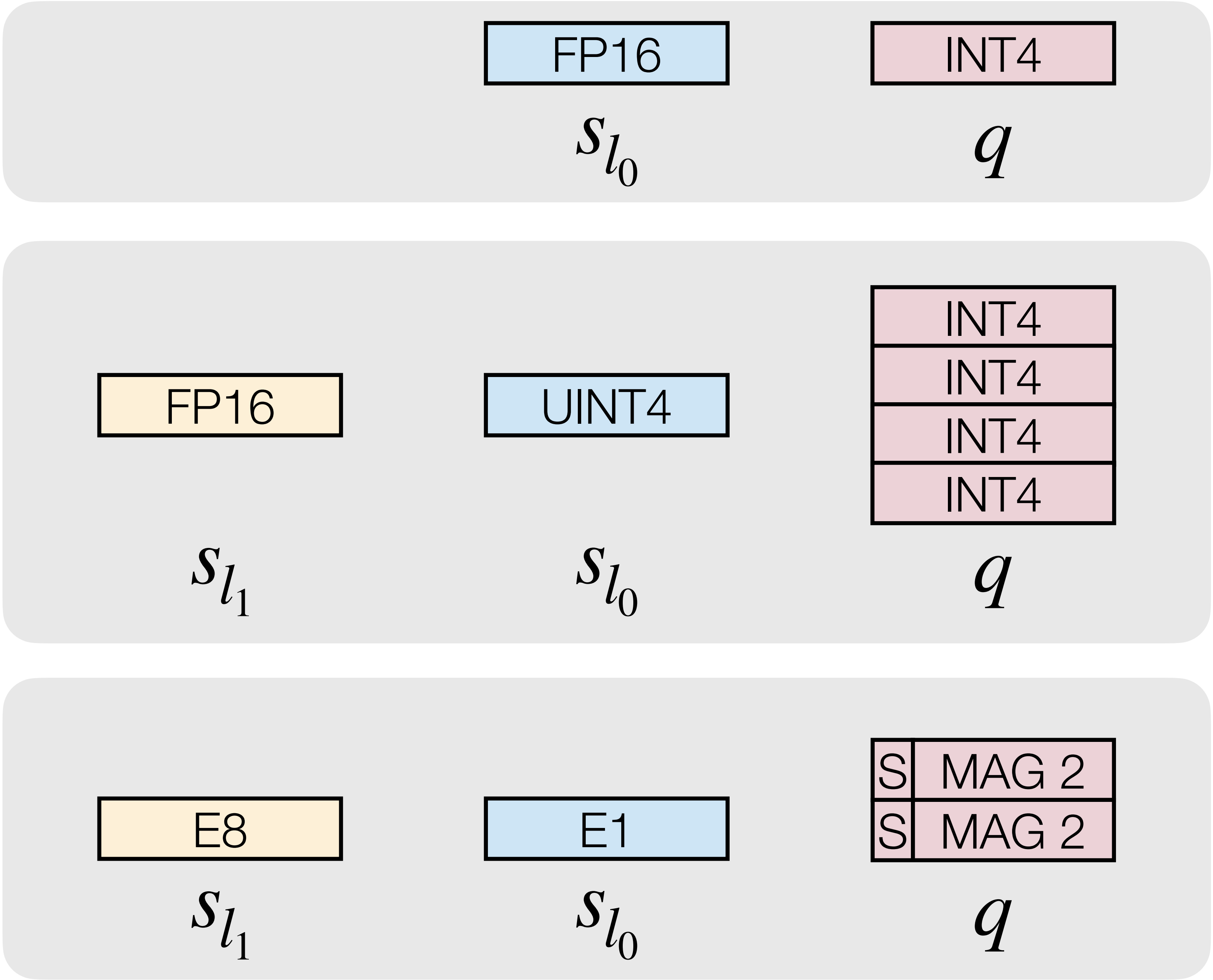
$$r = (q - z) \cdot s_{l_0} \cdot s_{l_1} \cdot \cdots$$

r : real number value

q : quantized value

z : zero point ($z = 0$ is symmetric quantization)

s : scale factors of different levels



Quantization Approach	Data Type	L0 Group Size	L0 Scale Data Type	L1 Group Size	L1 Scale Data Type	Effective Bit Width
Per-Channel Quant	INT4	Per Channel	FP16	-	-	4
VSQ	INT4	16	UINT4	Per Channel	FP16	4+4/16=4.25
MX4	S1M2	2	E1M0	16	E8M0	3+1/2+8/16=4
MX6	S1M4	2	E1M0	16	E8M0	5+1/2+8/16=6
MX9	S1M7	2	E1M0	16	E8M0	8+1/2+8/16=9

With Shared Microexponents, A Little Shifting Goes a Long Way [Bita Rouhani *et al.*]

Post-Training Quantization

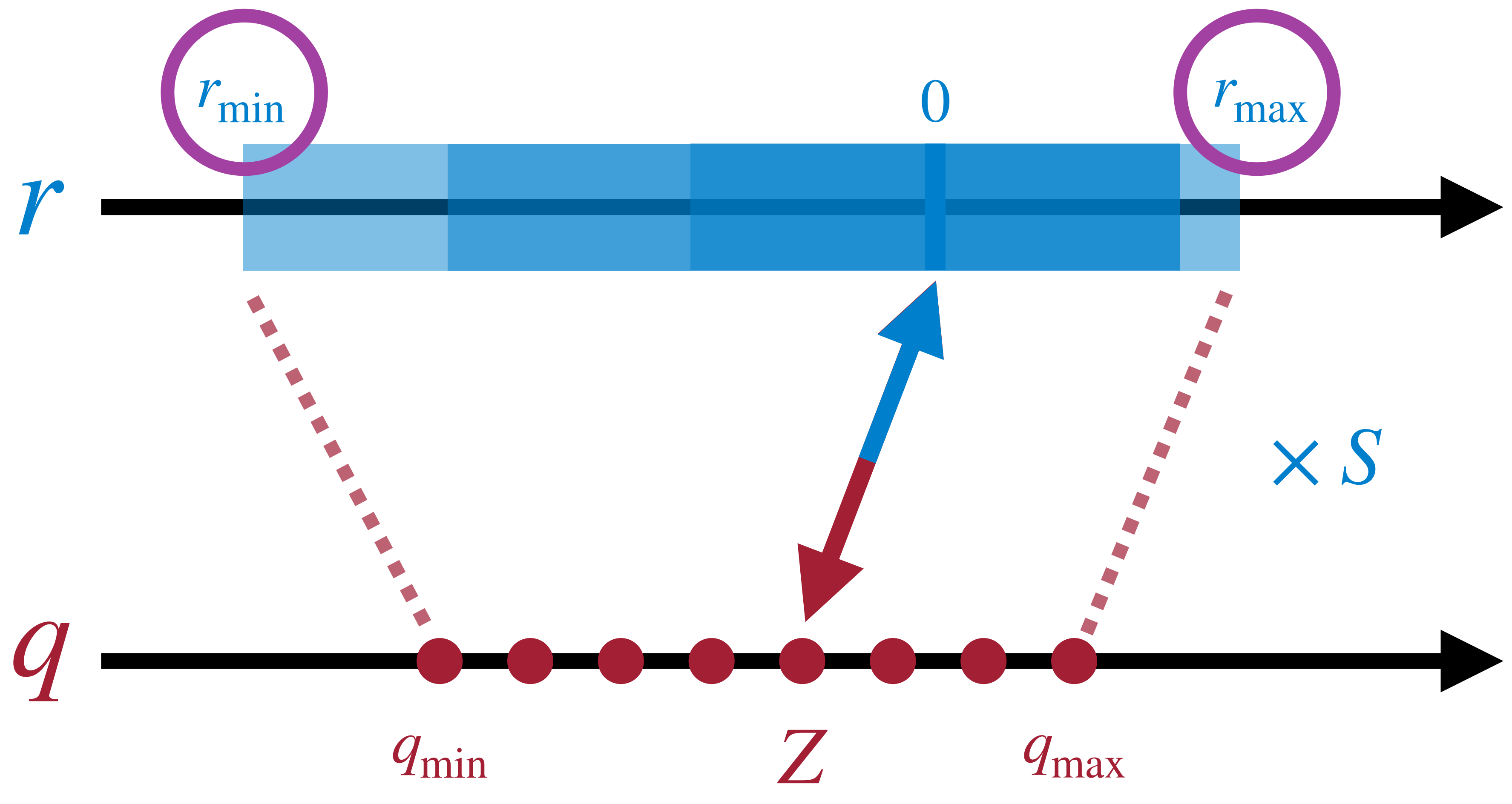
How should we get the optimal linear quantization parameters (S, Z)?

Topic I: Quantization Granularity

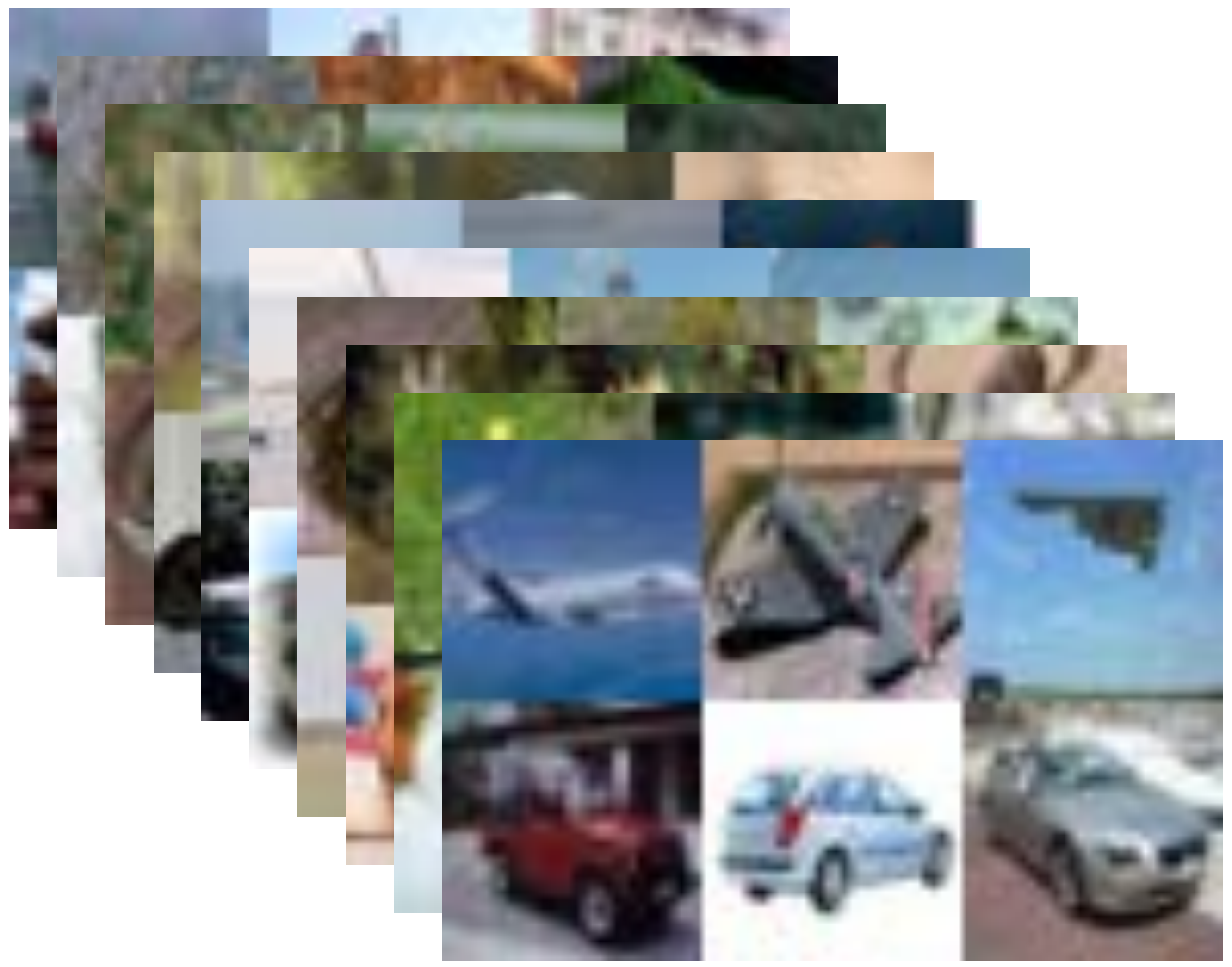
Topic II: Dynamic Range Clipping

Topic III: Rounding

Linear Quantization on Activations



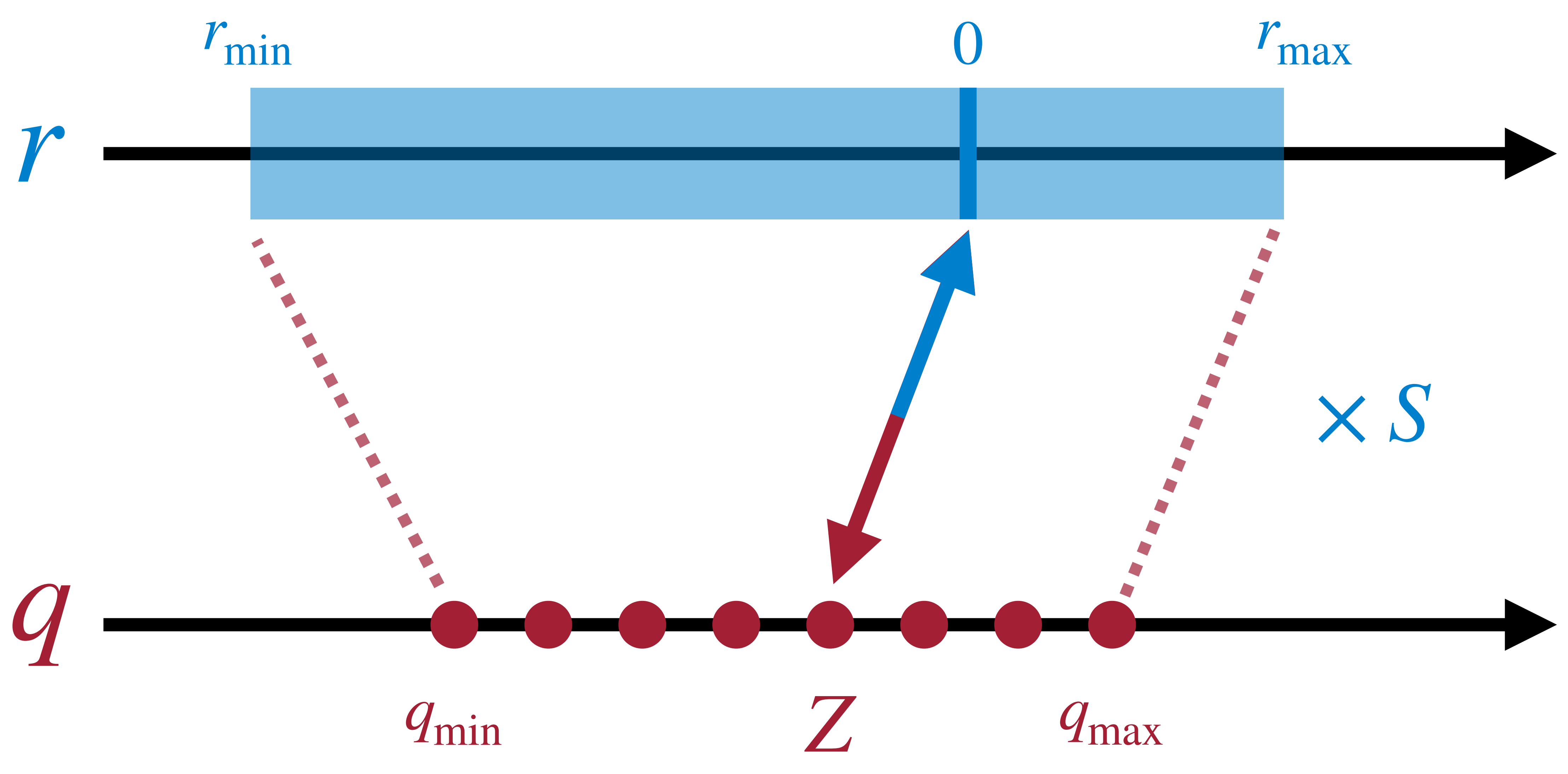
- Unlike weights, the activation range varies across inputs.
- To determine the floating-point range, the activations statistics are gathered ***before*** deploying the model.



Dynamic Range for Activation Quantization

Collect activations statistics before deploying the model

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

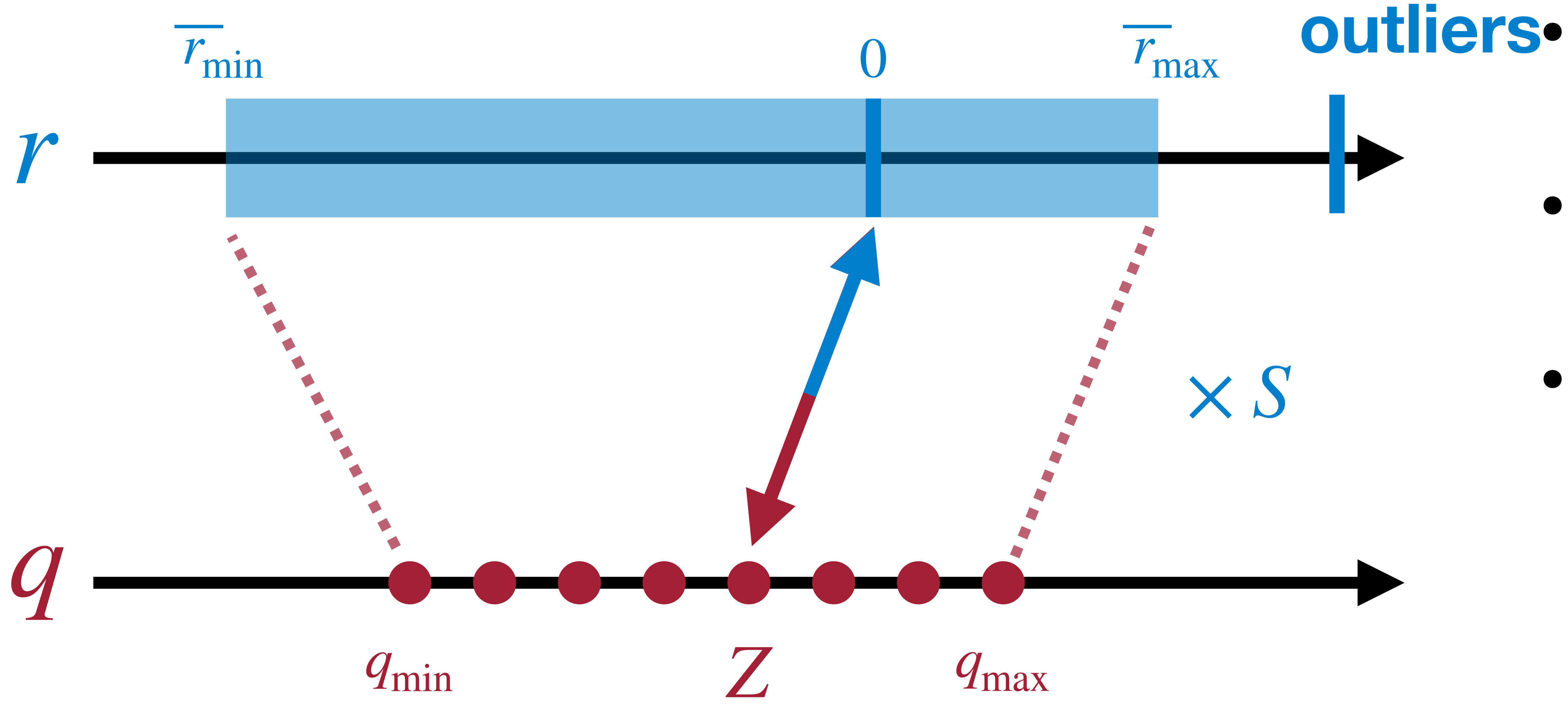


- Type 1: During training
  - Exponential moving averages (EMA)
    - observed ranges are smoothed across thousands of training steps

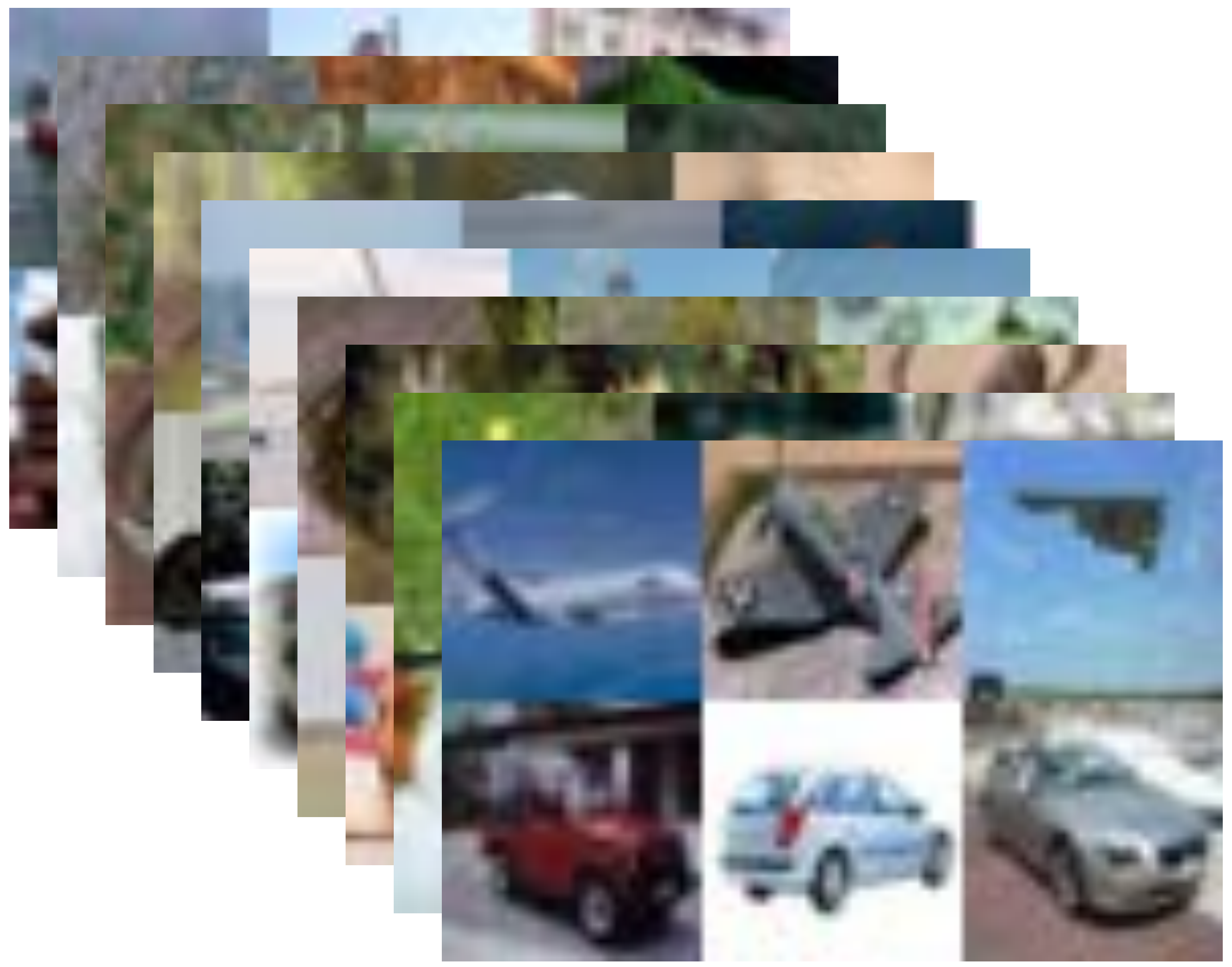
Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

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)

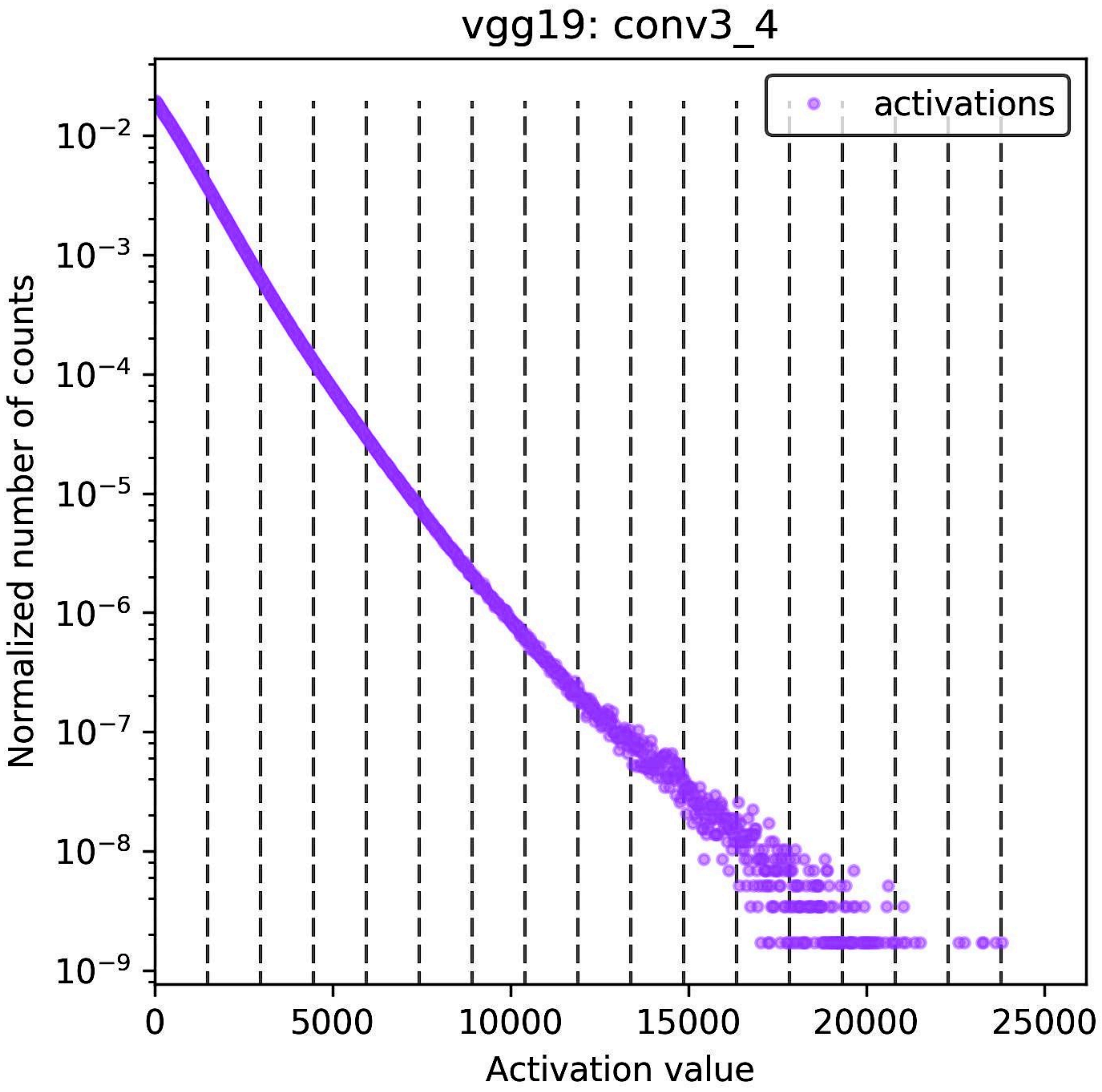


Neural Network Distiller

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 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

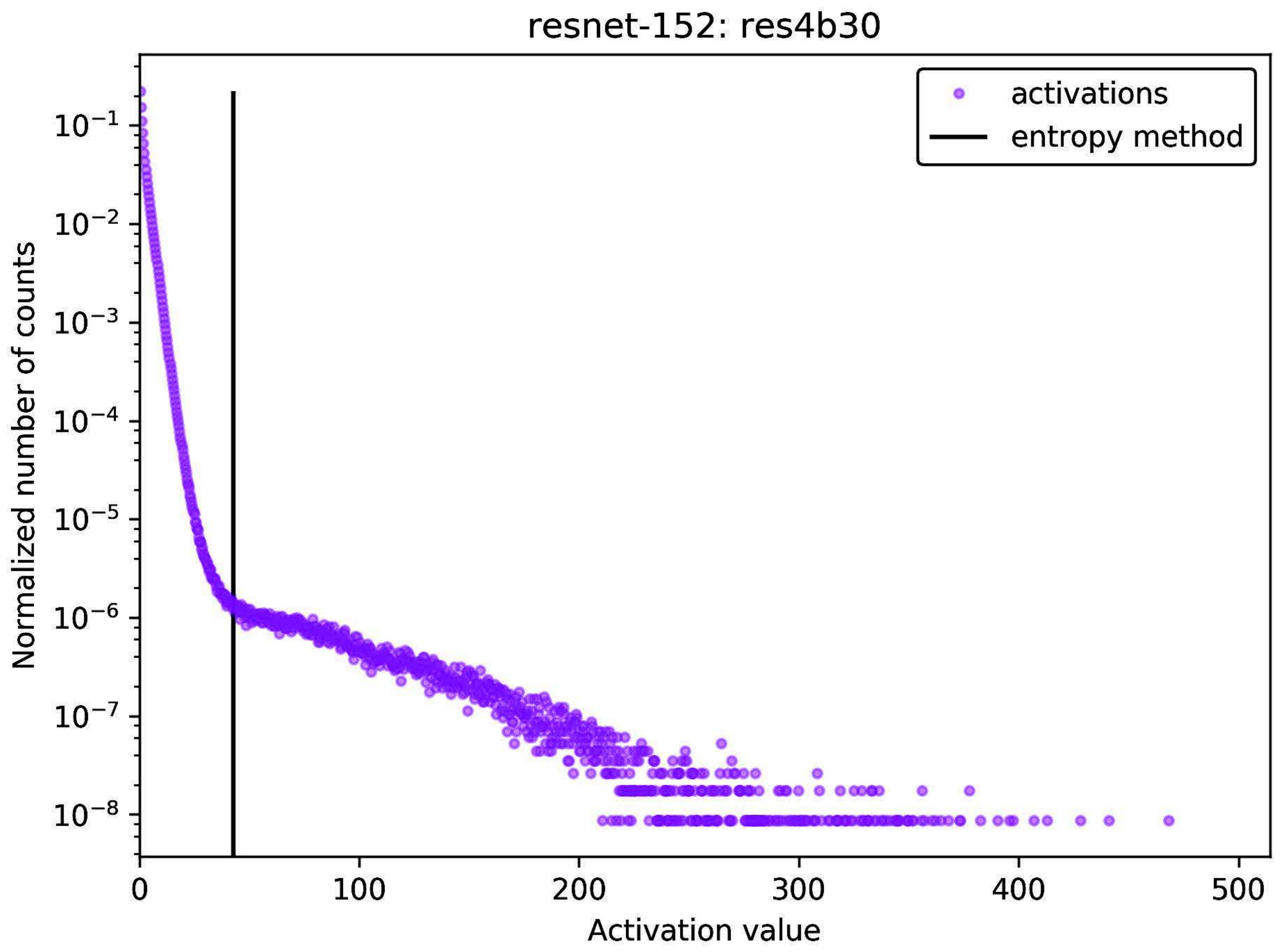
$$D_{KL}(P\|Q) = \sum_{i}^{N} P(x_i)\log\frac{P(x_i)}{Q(x_i)}$$

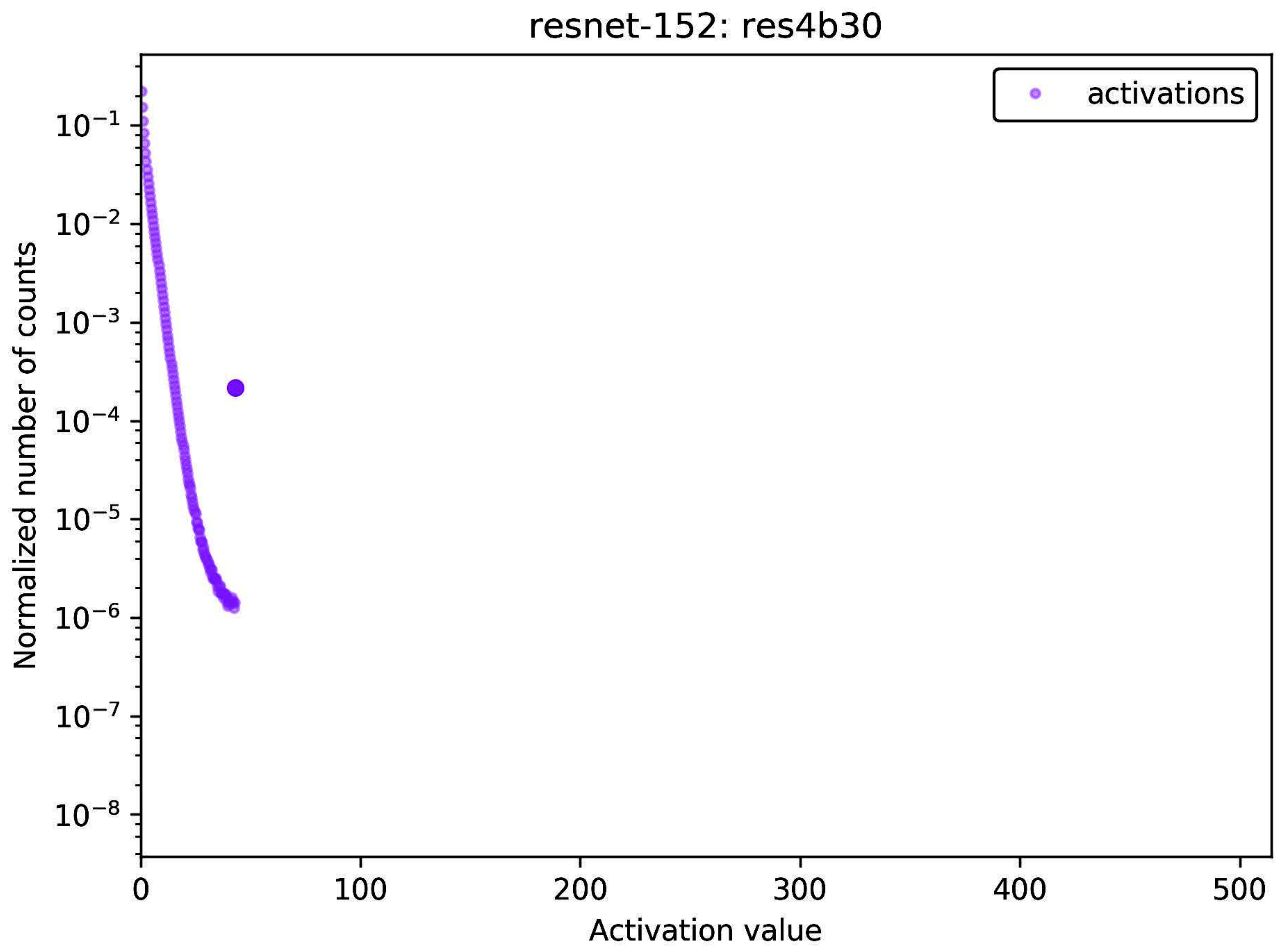
      - intuition: KL divergence measures the amount of information lost when approximating a given encoding.

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

Dynamic Range for Activation Quantization

Minimize loss of information by minimizing the KL divergence



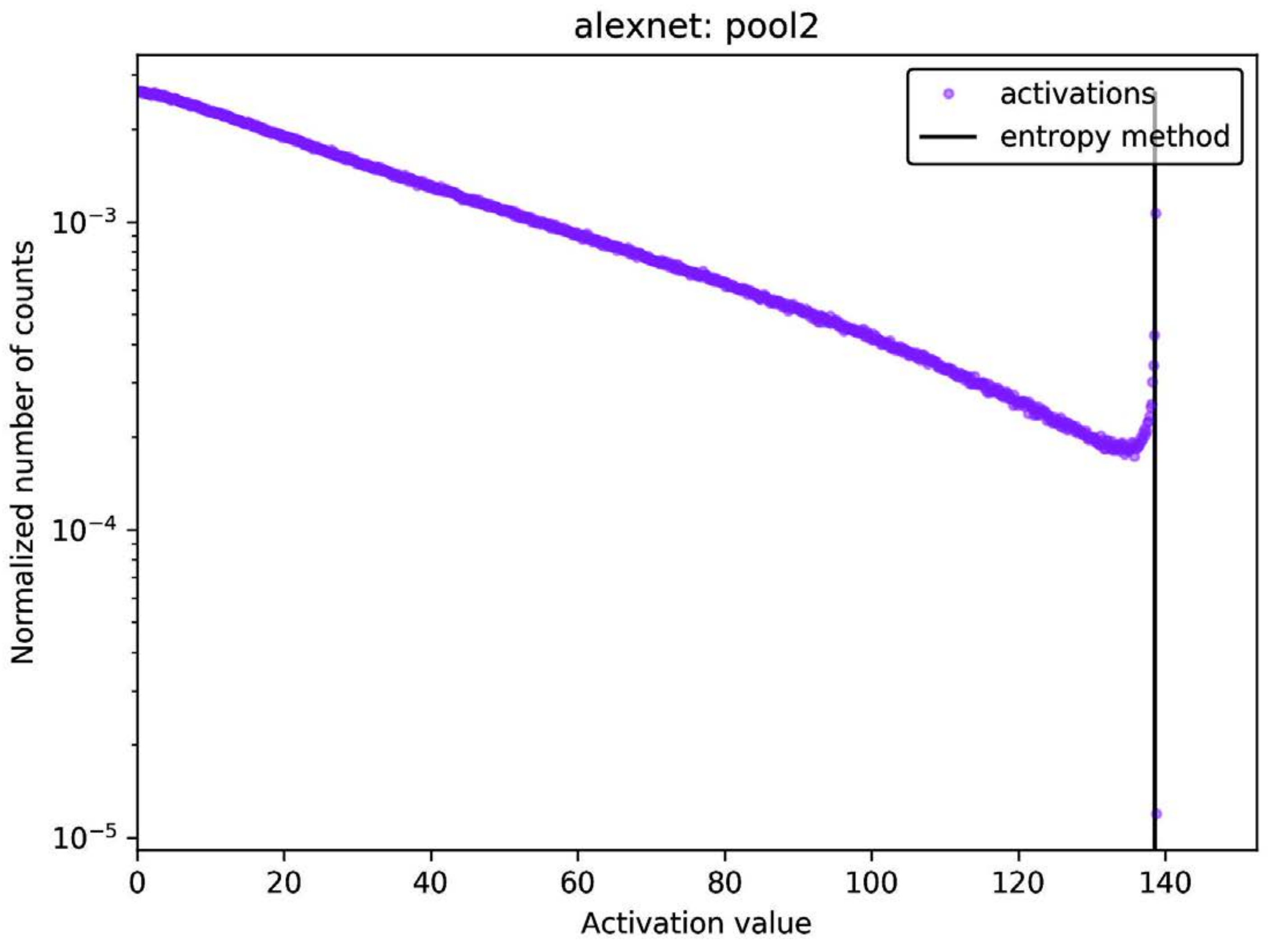


8-bit Inference with TensorRT [Szymon Migacz, 2017]

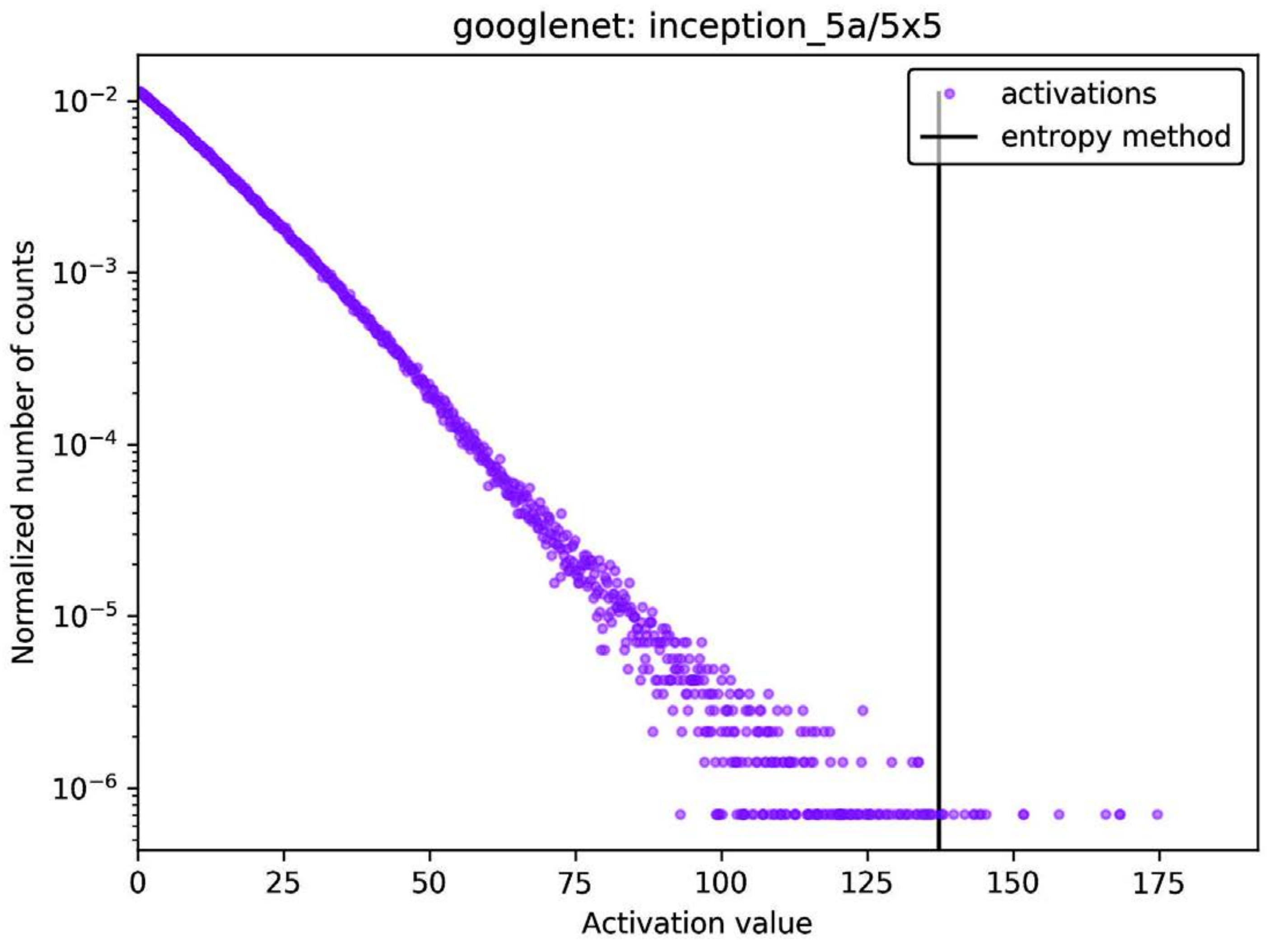
Dynamic Range for Activation Quantization

Minimize loss of information by minimizing the KL divergence

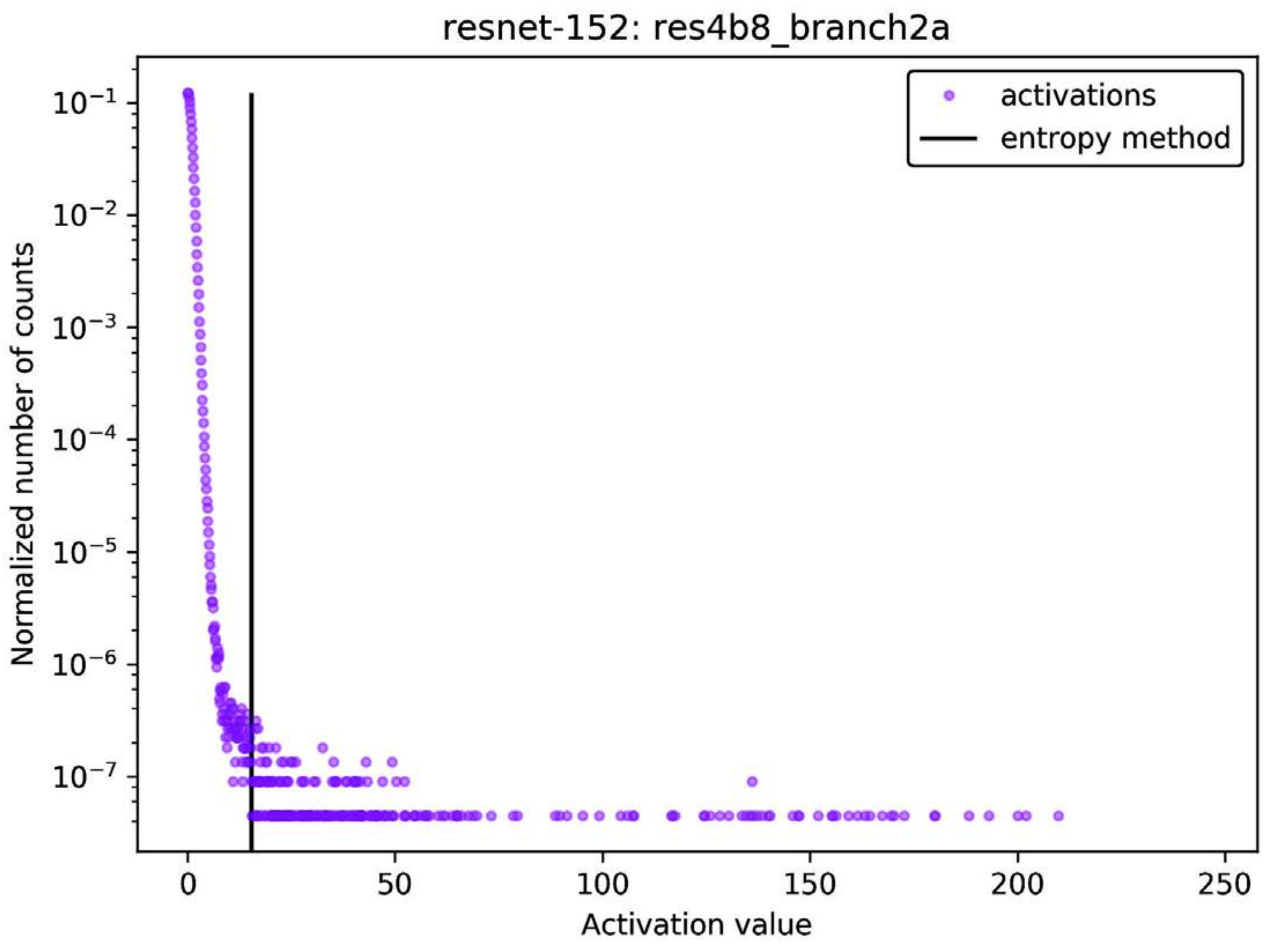
AlexNet: Pool 2



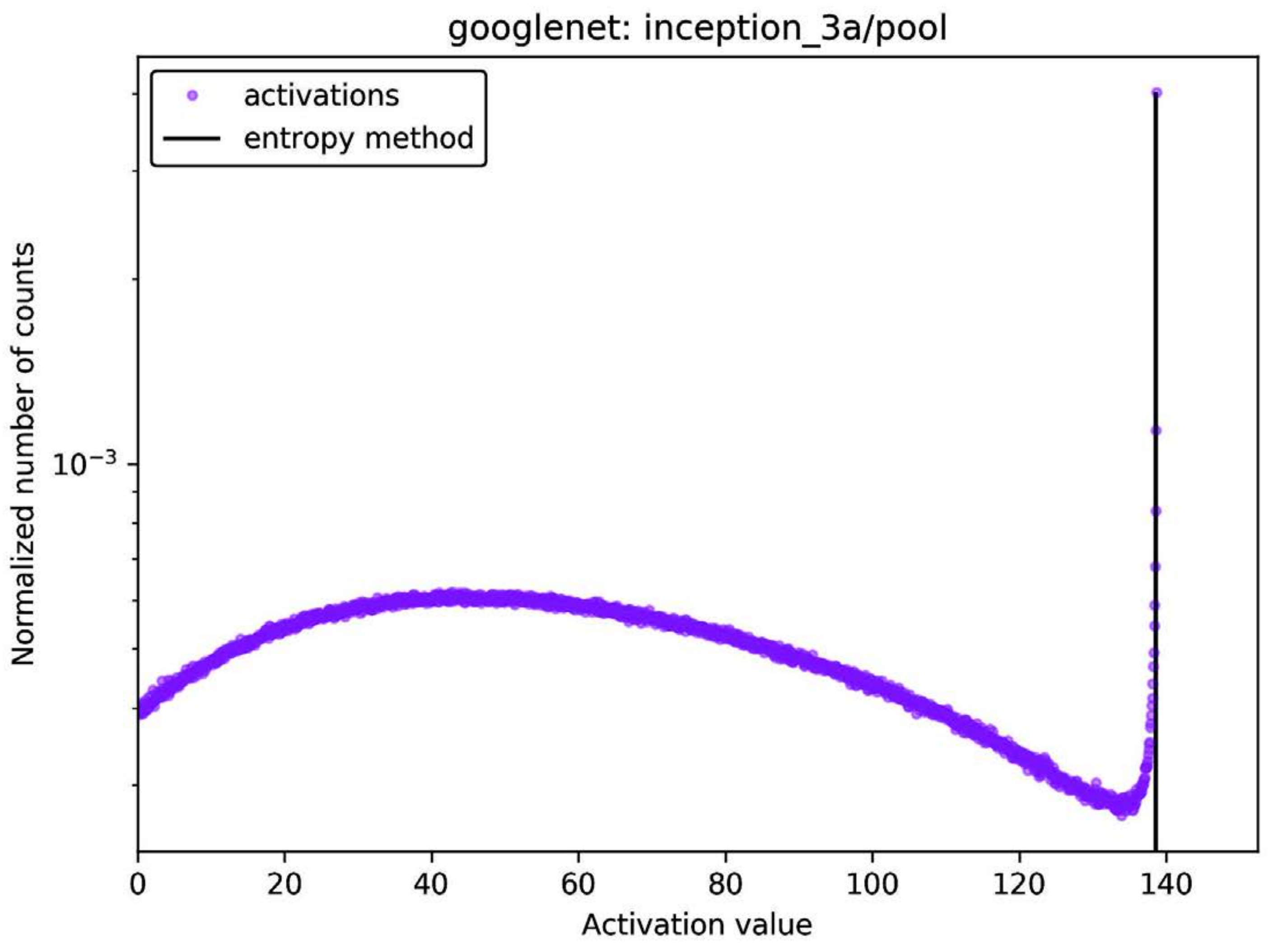
GoogleNet: incpetion_5a/5x5



ResNet-152: res4b8_branch2a



GoogleNet: incpetion_3a/pool

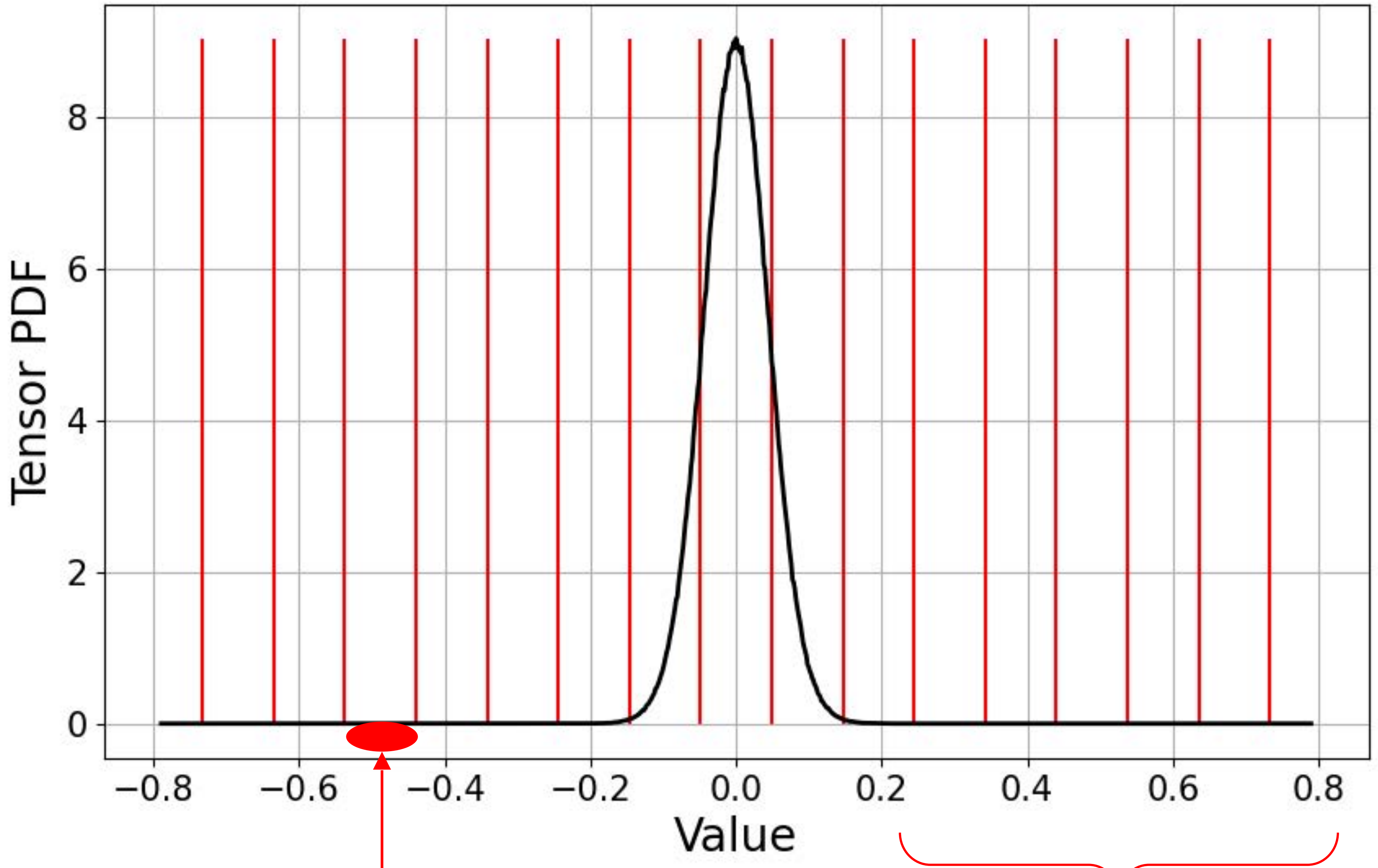


8-bit Inference with TensorRT [Szymon Migacz, 2017]

Dynamic Range for Quantization

Minimize mean-square-error (MSE) using Newton-Raphson method

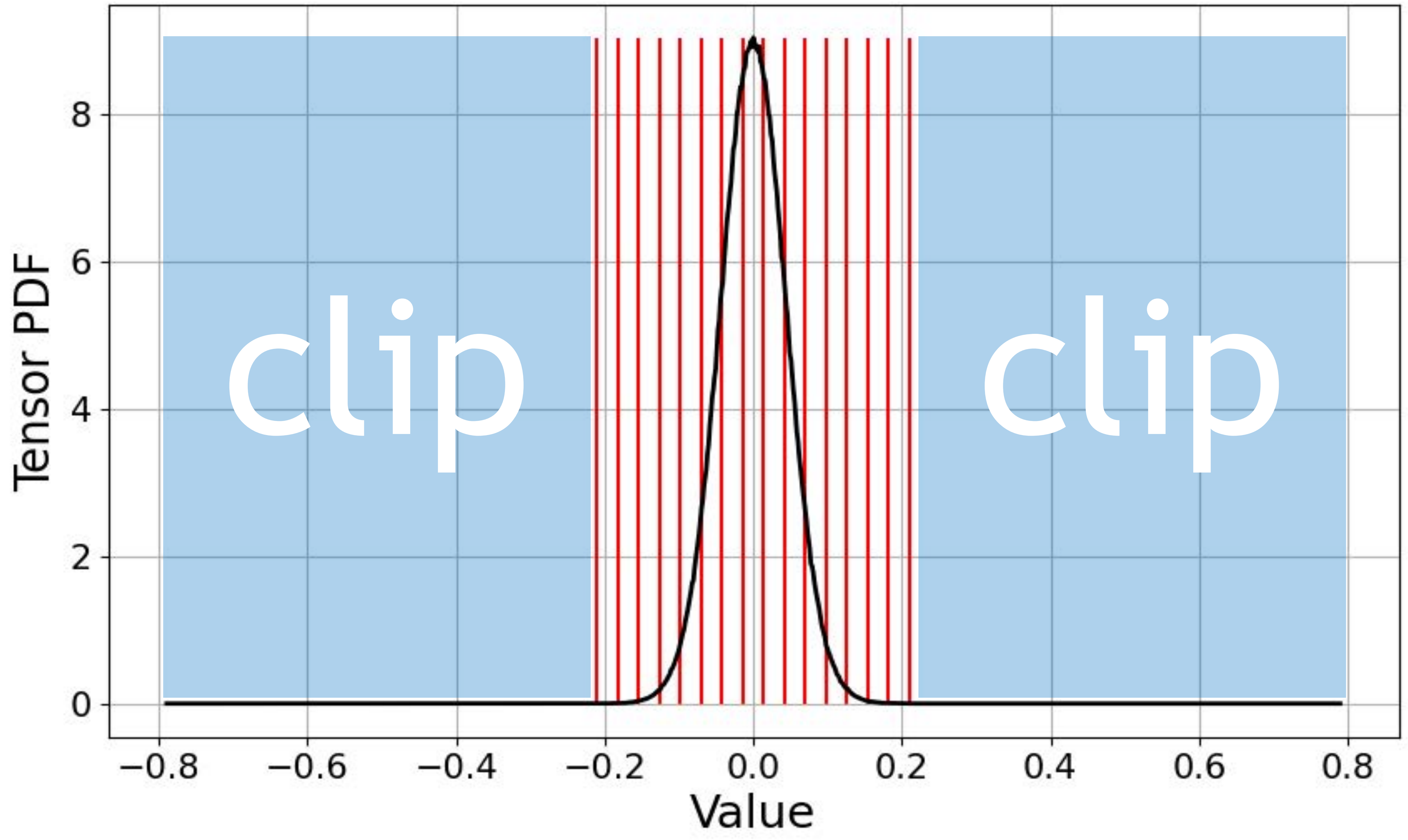
max-scaled quantization



large quantization noise

low density data

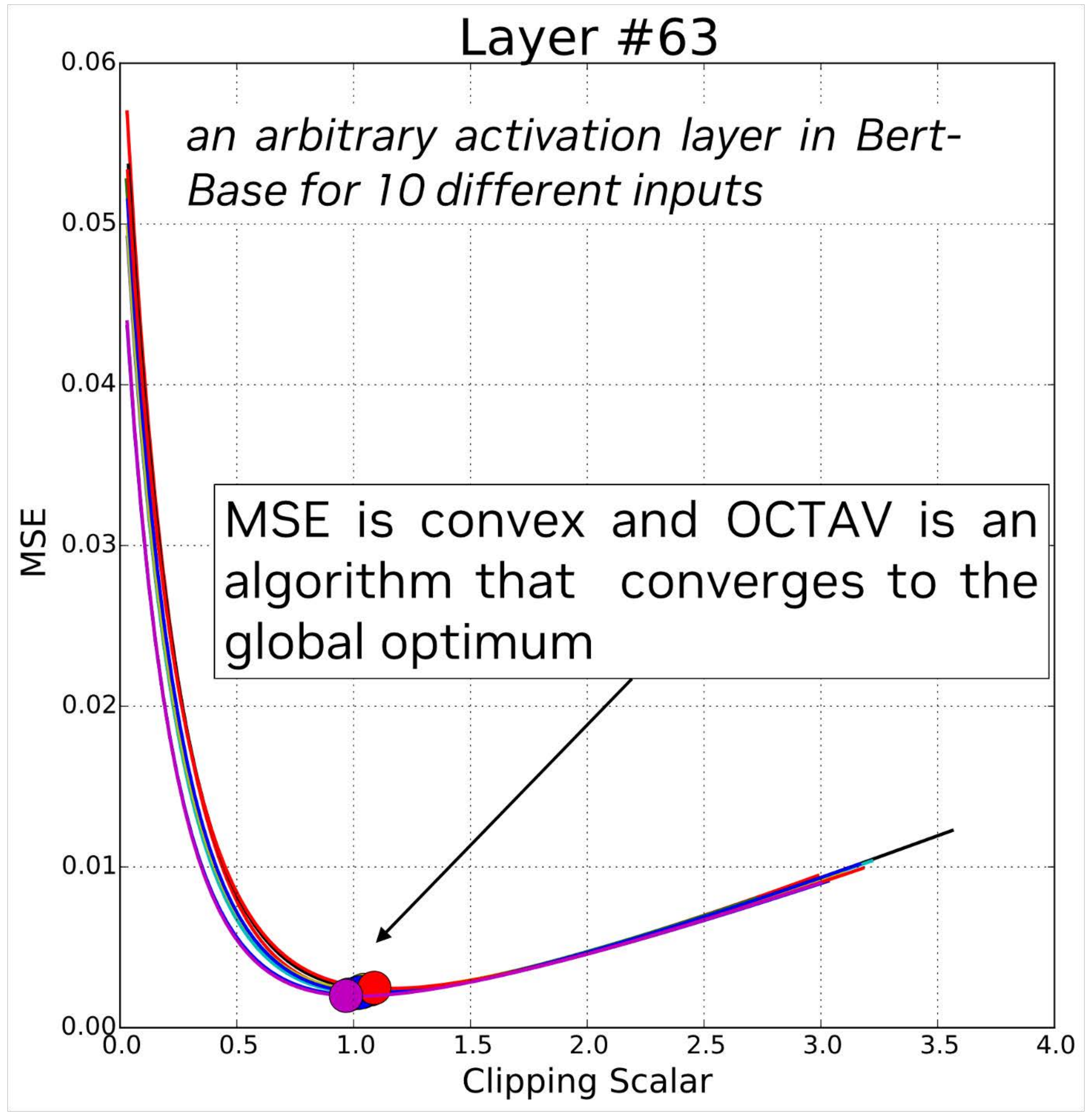
clipped quantization

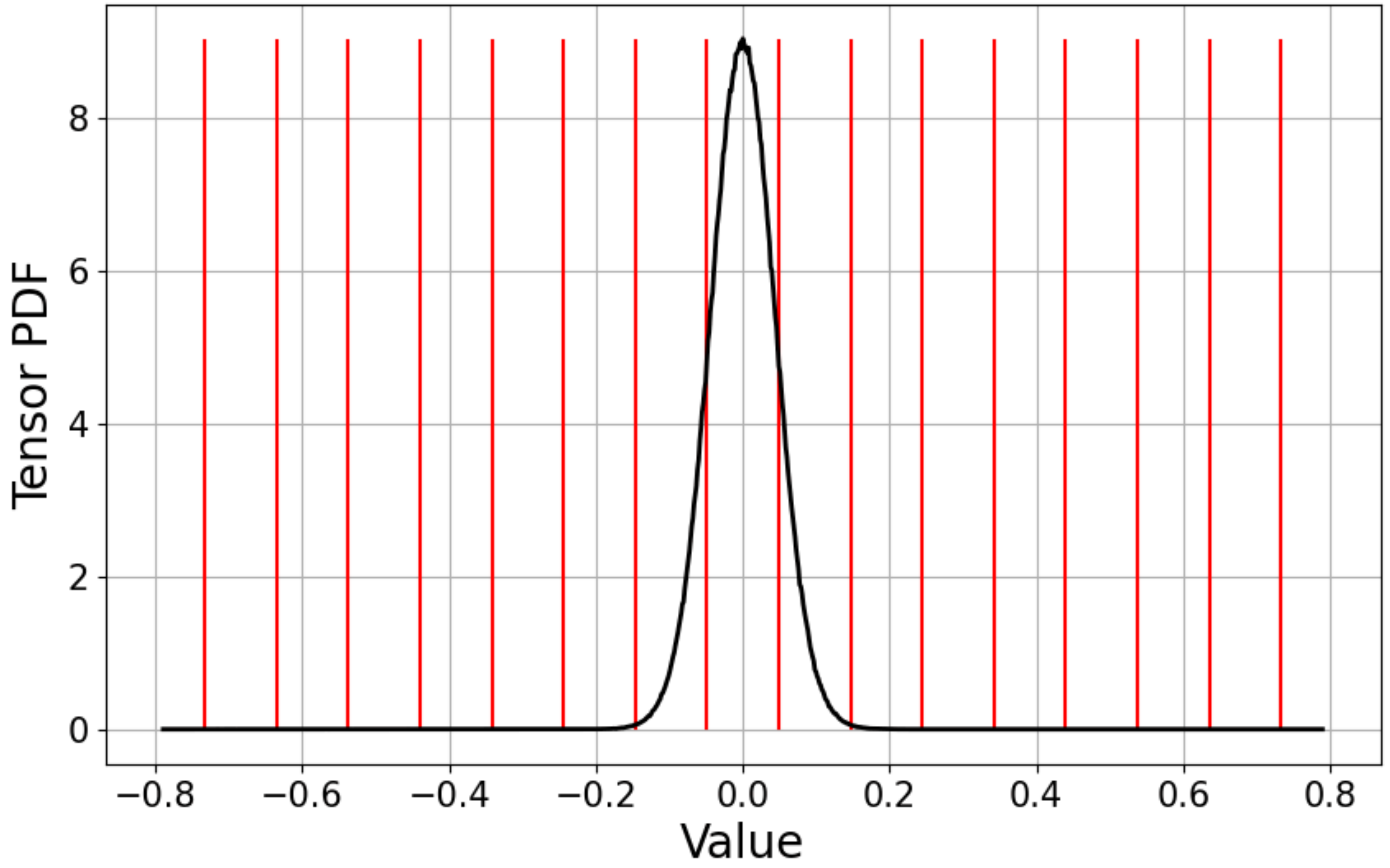


Optimal Clipping and Magnitude-aware Differentiation for Improved Quantization-aware Training [Sakr *et al.*, ICML 2022]

Dynamic Range for Quantization

Minimize mean-square-error (MSE) using Newton-Raphson method





Network	FP32 Accuracy	OCTAV int4
ResNet-50	76.07	75.84
MobileNet-V2	71.71	70.88
Bert-Large	91.00	87.09

Optimal Clipping and Magnitude-aware Differentiation for Improved Quantization-aware Training [Sakr *et al.*, ICML 2022]

Post-Training Quantization

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

Topic I: Quantization Granularity
Topic II: Dynamic Range Clipping
Topic III: Rounding

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

$$\boxed{0.3 \mid 0.5 \mid 0.7 \mid 0.2} \xrightarrow{\text{rounding-to-nearest}} \boxed{0 \mid 1 \mid 1 \mid 0}$$

$$\boxed{0.3 \mid 0.5 \mid 0.7 \mid 0.2} \xrightarrow[\text{(one potential result)}]{\text{AdaRound}} \boxed{0 \mid 0 \mid 1 \mid 0}$$

- What is optimal? Rounding that reconstructs the original <u>activation</u> 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 $\tilde{w} = \lfloor \lfloor w \rfloor + \delta \rceil, \delta \in [0,1]$

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

Adaptive Rounding for Weight Quantization

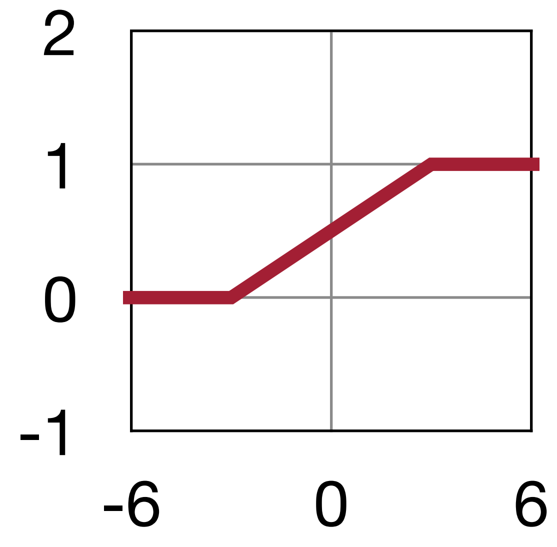
Rounding-to-nearest is not optimal

- **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):

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

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

  - $\mathbf{x}$ 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
    - $f_{reg}(\mathbf{V}) = \sum_{i,j} 1 - |2h(\mathbf{V}_{i,j}) - 1|^{\beta}$



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

Neural Network Quantization

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

3	0	2	1	3:	2.00
1	1	0	3	2:	1.50
0	3	1	0	1:	0.00
3	1	2	2	0:	-1.00

K-Means-based Quantization

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1\right) \times 1.07$$

Linear 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

- **Zero Point**
  - Asymmetric
  - Symmetric
- **Scaling Granularity**
  - Per-Tensor
  - Per-Channel
  - Group Quantization
- **Range Clipping**
  - Exponential Moving Average
  - Minimizing KL Divergence
  - Minimizing Mean-Square-Error
- **Rounding**
  - Round-to-Nearest
  - AdaRound

Post-Training INT8 Linear Quantization

Activation		Symmetric	Asymmertric
		Per-Tensor	Per-Tensor
		Minimize KL-Divergence	Exponential Moving Average (EMA)
Weight		Symmetric	Symmetric
		Per-Tensor	Per-Channel
Neural Network	GoogleNet	-0.45%	0%
	ResNet-50	-0.13%	-0.6%
	ResNet-152	-0.08%	-1.8%
	MobileNetV1	-	-11.8%
	MobileNetV2	-	-2.1%

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
		Per-Tensor	Per-Tensor
		Minimize KL-Divergence	Exponential Moving Average (EMA)
Weight		Symmetric	Symmetric
		Per-Tensor	Per-Channel
Neural Network	MobileNetV1	-	-11.8%
	MobileNetV2	-	-2.1%

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]

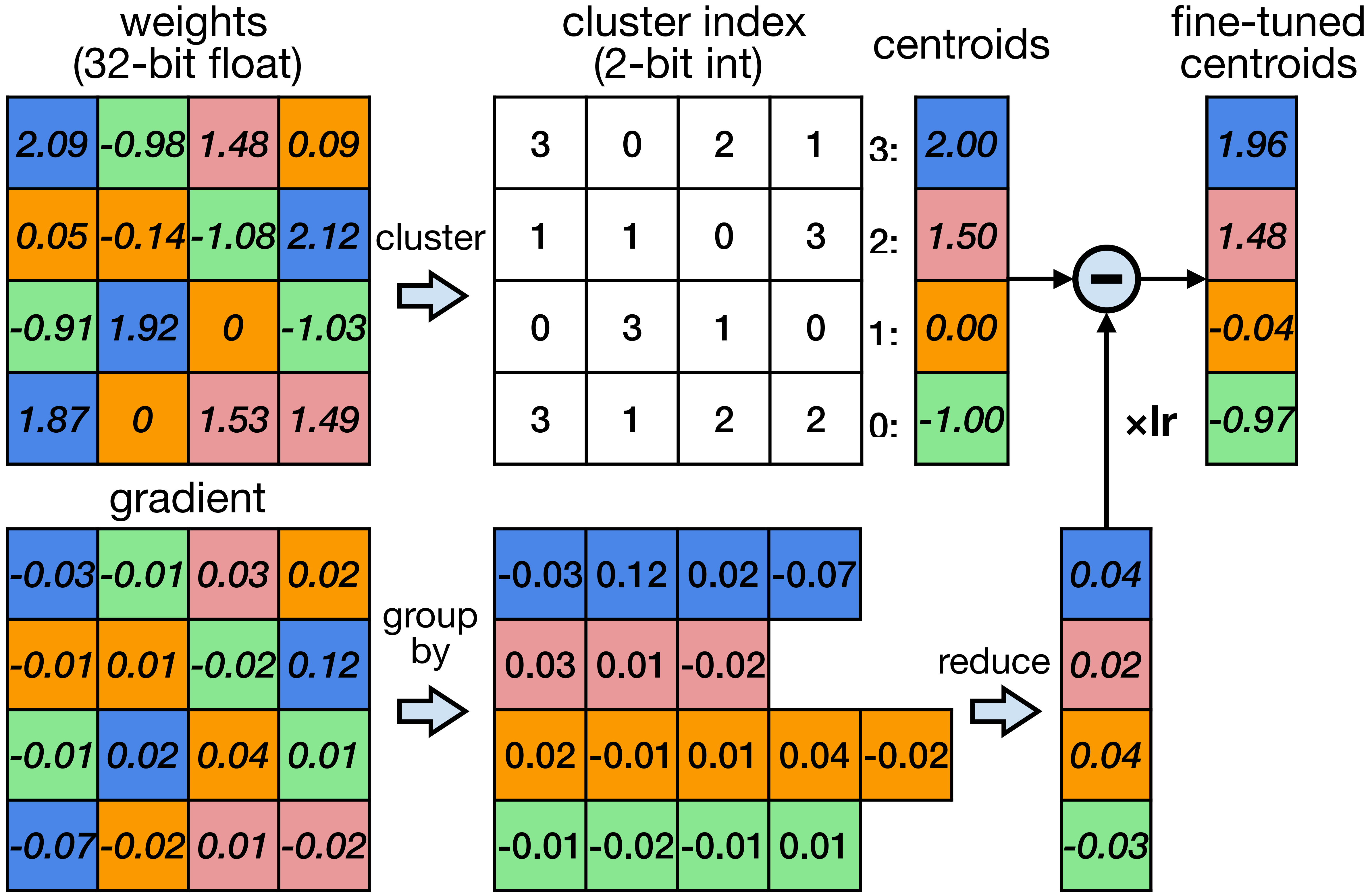
Quantization-Aware Training

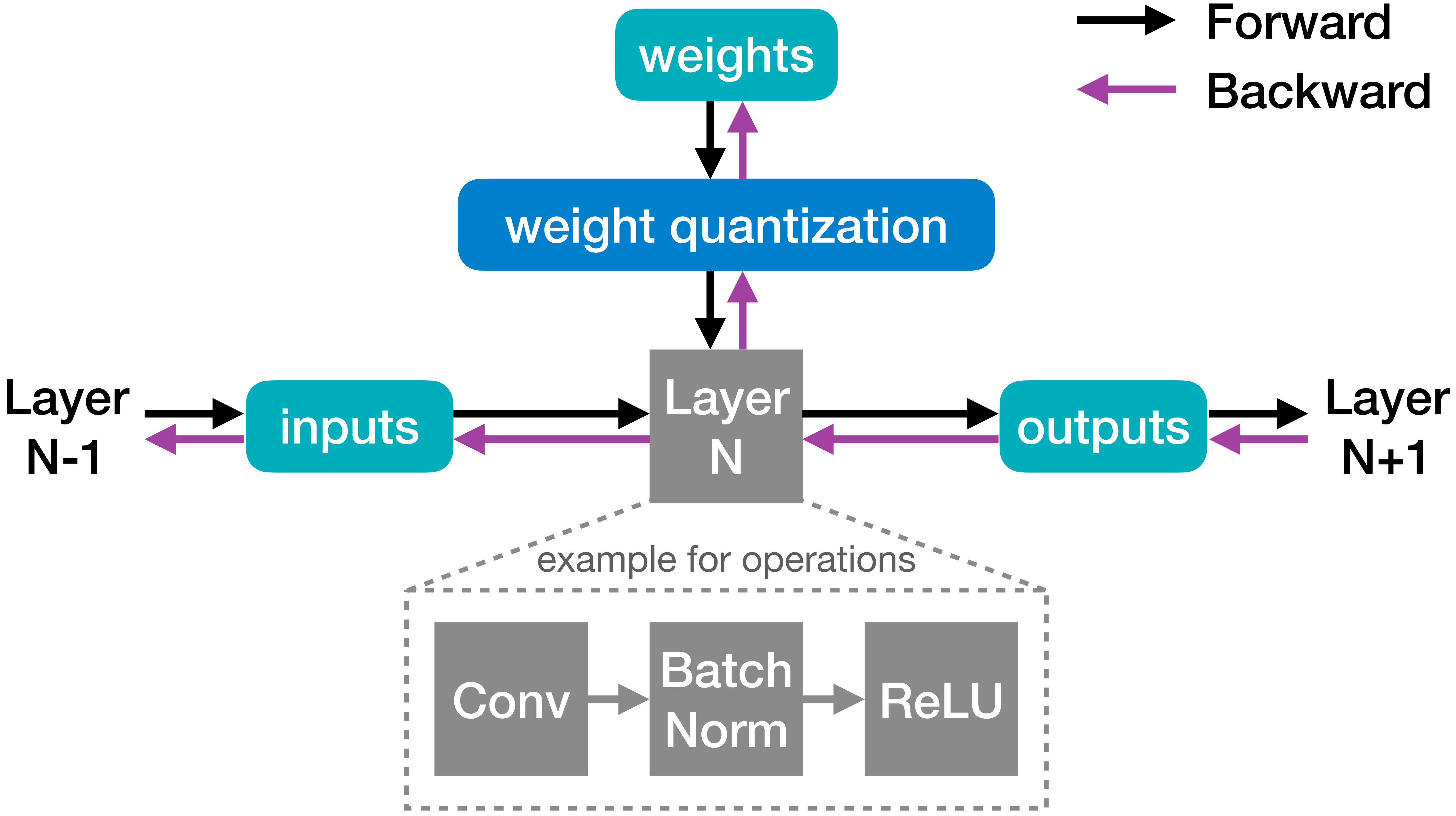
How should we improve performance of quantized models?

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.



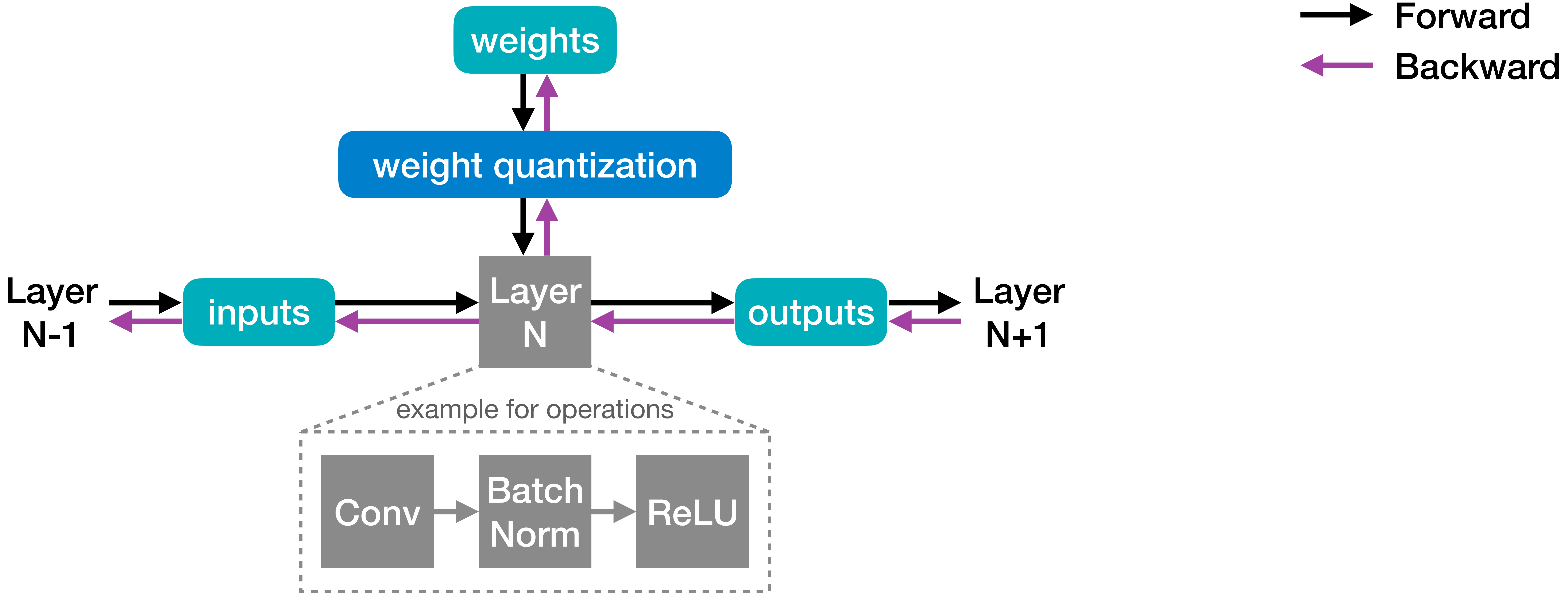


Deep Compression [Han *et al.*, ICLR 2016]

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.



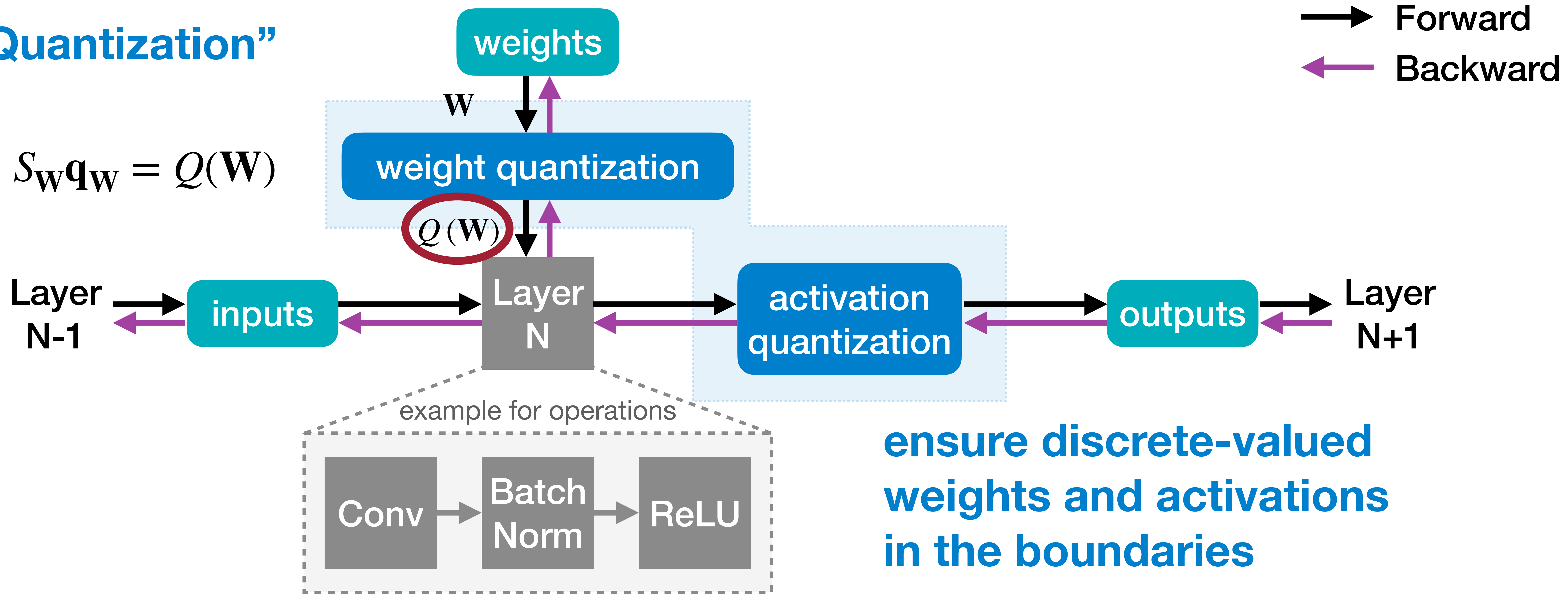
Quantization-Aware Training

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.

"Simulated/Fake Quantization"

$$\mathbf{W} \rightarrow S_{\mathbf{W}}\mathbf{q}_{\mathbf{W}} = Q(\mathbf{W})$$



ensure discrete-valued weights and activations in the boundaries

these operations still run in full precision

Linear Quantization

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

weights (32-bit float) | quantized weights (2-bit signed int) | zero point (2-bit signed int) | scale (32-bit float)

$$\underbrace{\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}}_{\mathbf{W}} \Rightarrow \left(\underbrace{\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix}}_{\mathbf{q_W}} - (-1) \right) \times 1.07 = \underbrace{\begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}}_{Q(\mathbf{W})}$$

Quantization-Aware Training

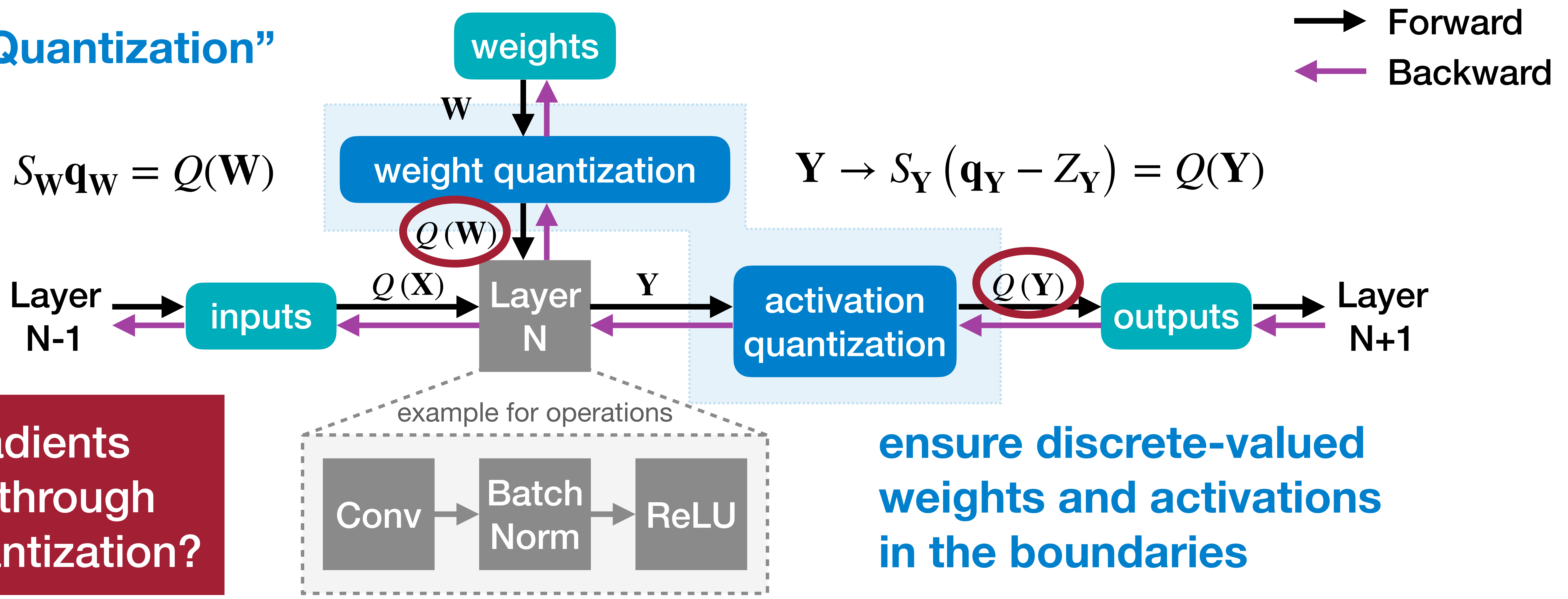
Train the model taking quantization into consideration

- 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.

"Simulated/Fake Quantization"

$$\mathbf{W} \rightarrow S_{\mathbf{W}}\mathbf{q}_{\mathbf{W}} = Q(\mathbf{W})$$

$$\mathbf{Y} \rightarrow S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = Q(\mathbf{Y})$$



How should gradients back-propagate through the (simulated) quantization?

ensure discrete-valued weights and activations in the boundaries

these operations still run in full precision

Straight-Through Estimator (STE)

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

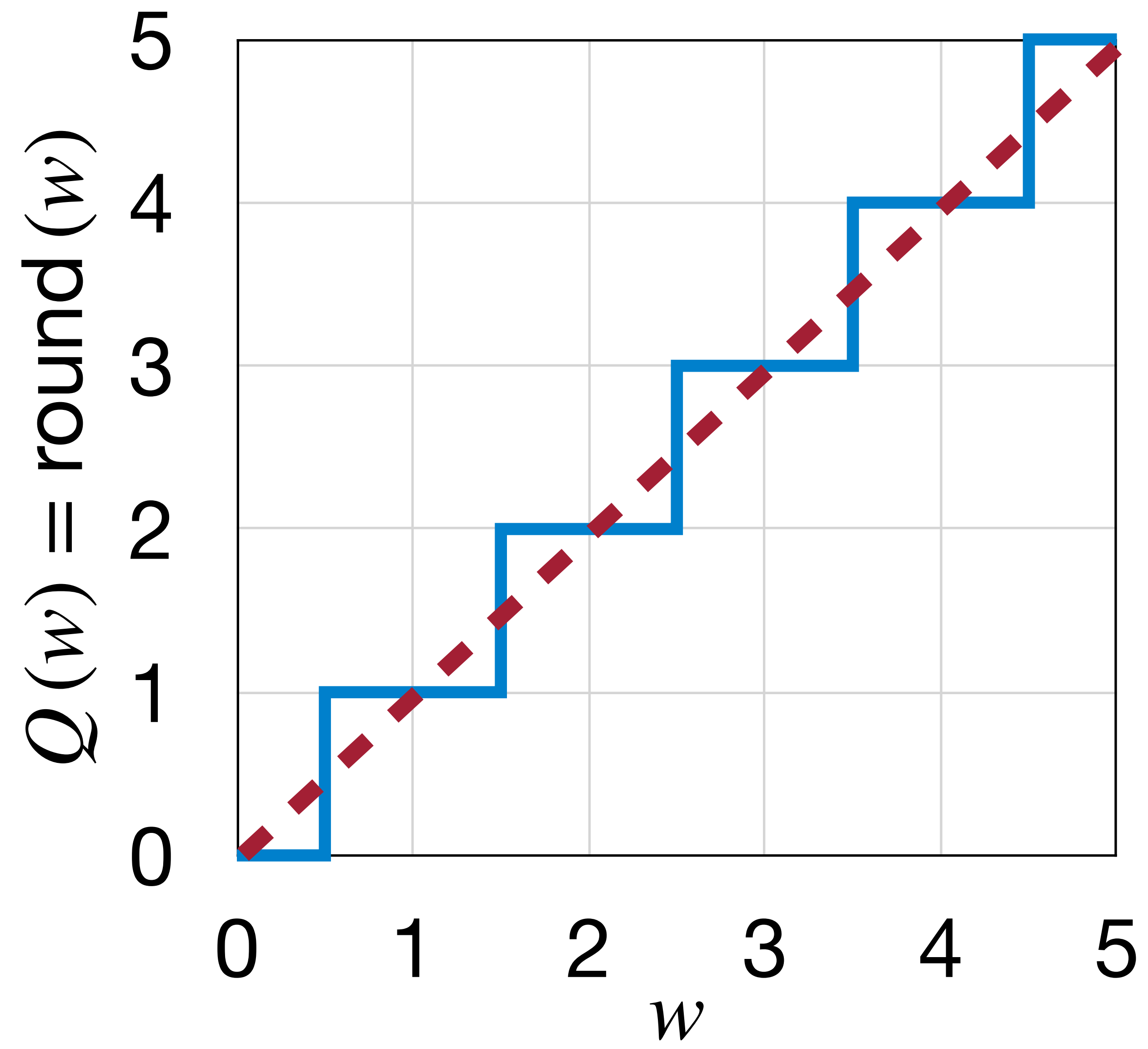
$$\frac{\partial Q(W)}{\partial W} = 0$$

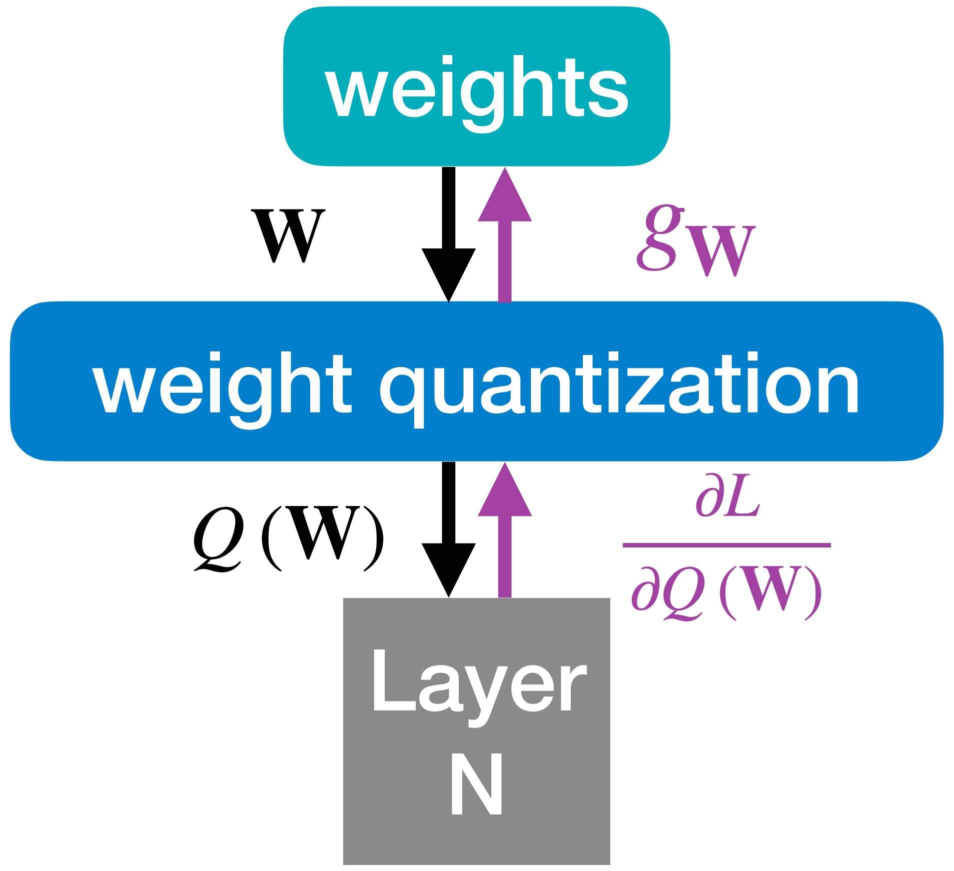
  - The neural network will learn nothing since gradients become 0 and the weights won't get updated.

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

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

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





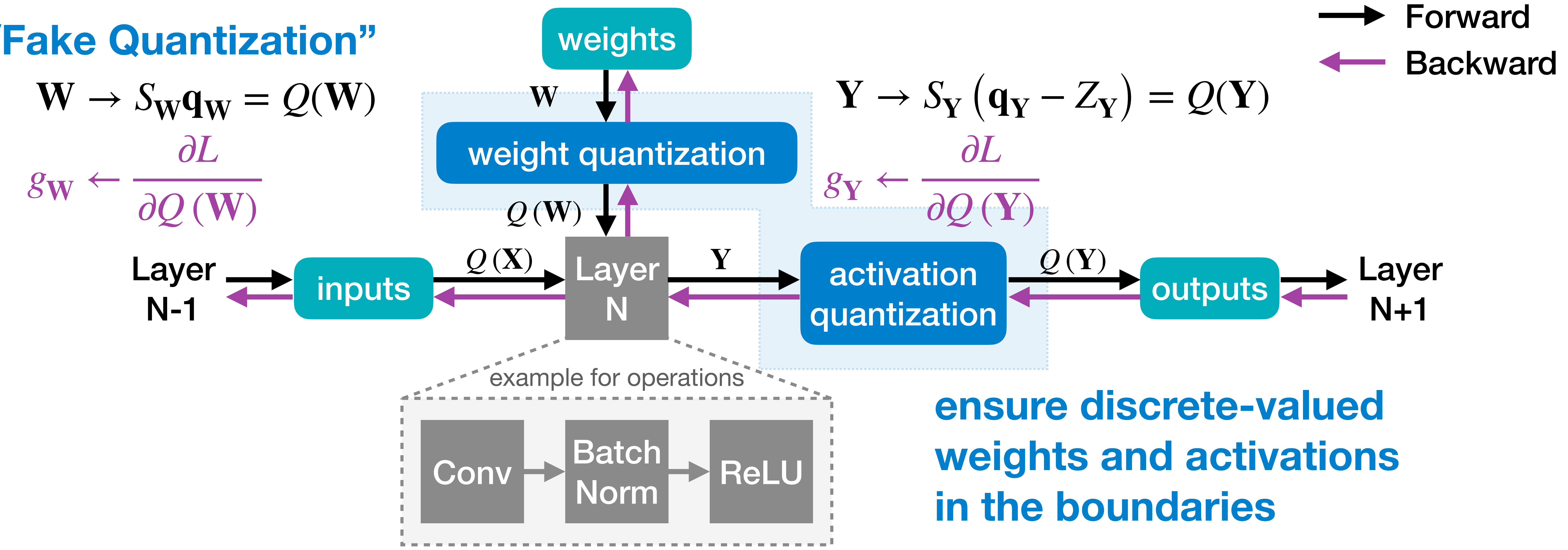
Neural Networks for Machine Learning [Hinton *et al.*, Coursera Video Lecture, 2012]
Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [Bengio, arXiv 2013]

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.

"Simulated/Fake Quantization"



these operations still run in full precision

ensure discrete-valued weights and activations in the boundaries

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]

Neural Network Quantization

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*

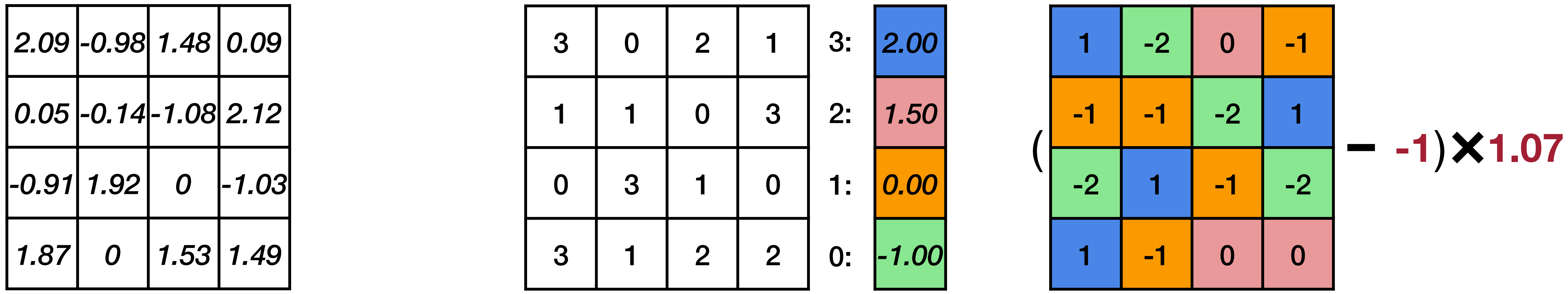
3	0	2	1	3:	*2.00*
1	1	0	3	2:	*1.50*
0	3	1	0	1:	*0.00*
3	1	2	2	0:	*-1.00*

$$\left(\begin{matrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{matrix} - -1 \right) \times 1.07$$

		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

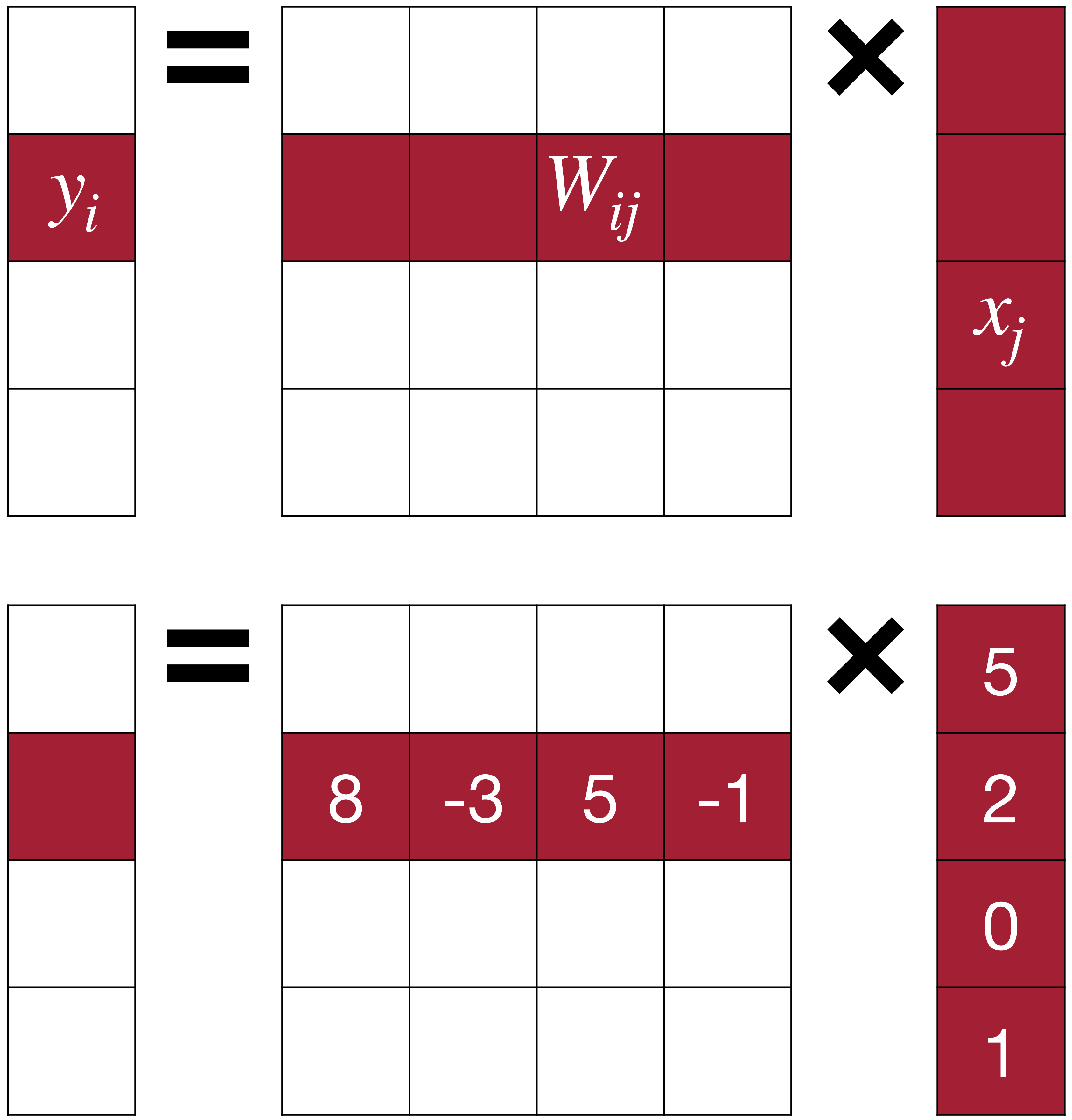
Binary/Ternary Quantization

Can we push the quantization precision to 1 bit?

Can quantization bit width go even lower?

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

$$= 8 \times 5 + (-3) \times 2 + 5 \times 0 + (-1) \times 1$$

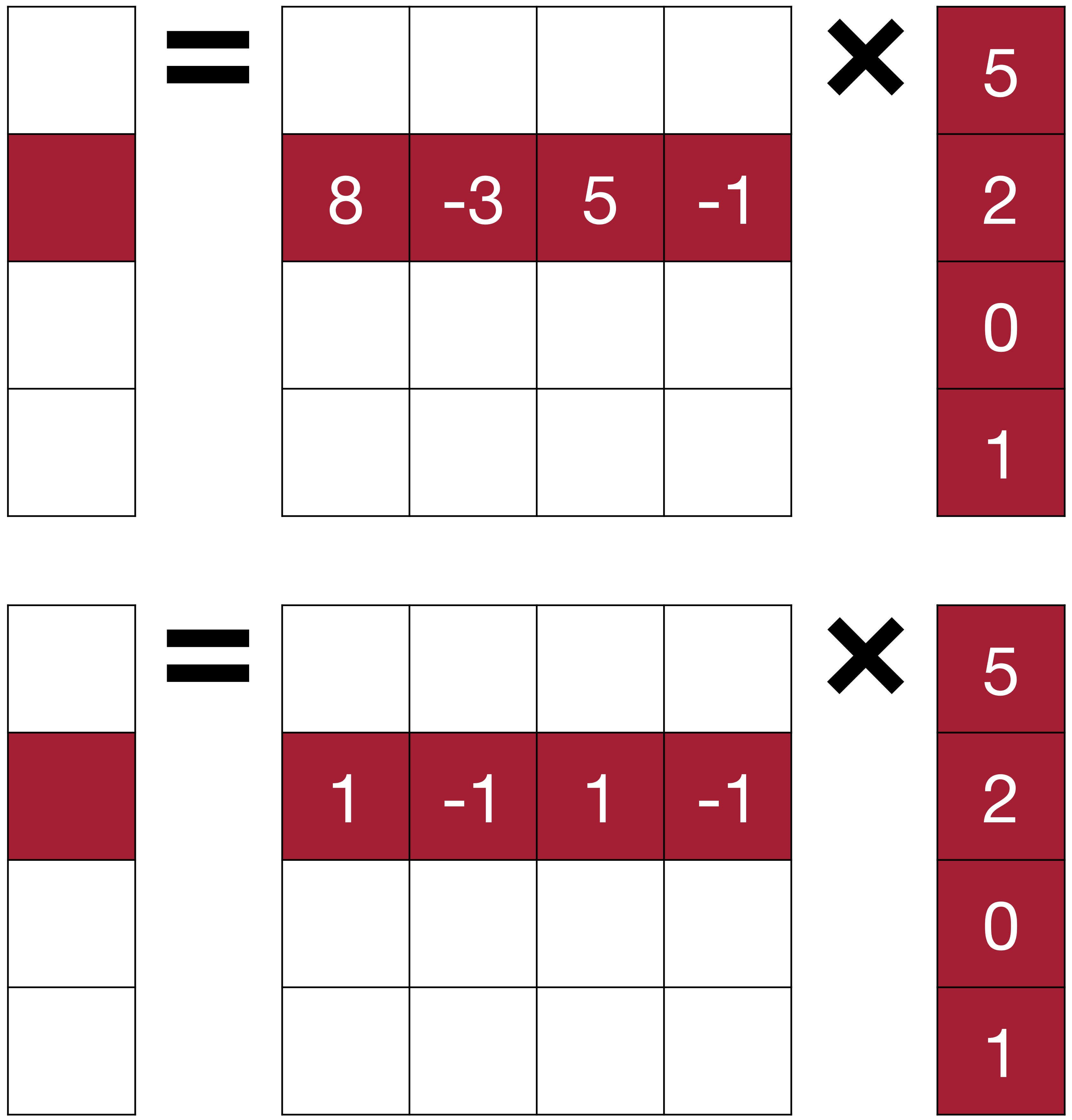


input	weight	operations	memory	computation
$\mathbb{R}$	$\mathbb{R}$	+ ×	1×	1×

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]

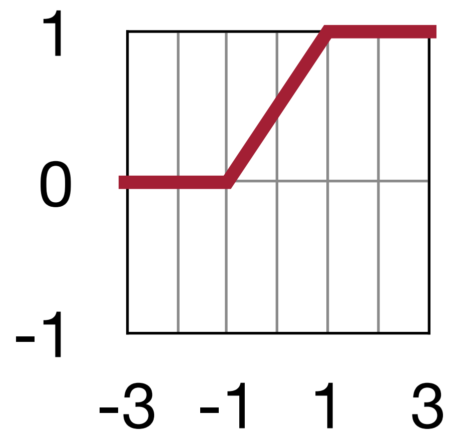
Binarization

- **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*

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

binary weights
(1-bit)

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

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

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

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

scale
(32-bit float)

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

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

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

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

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

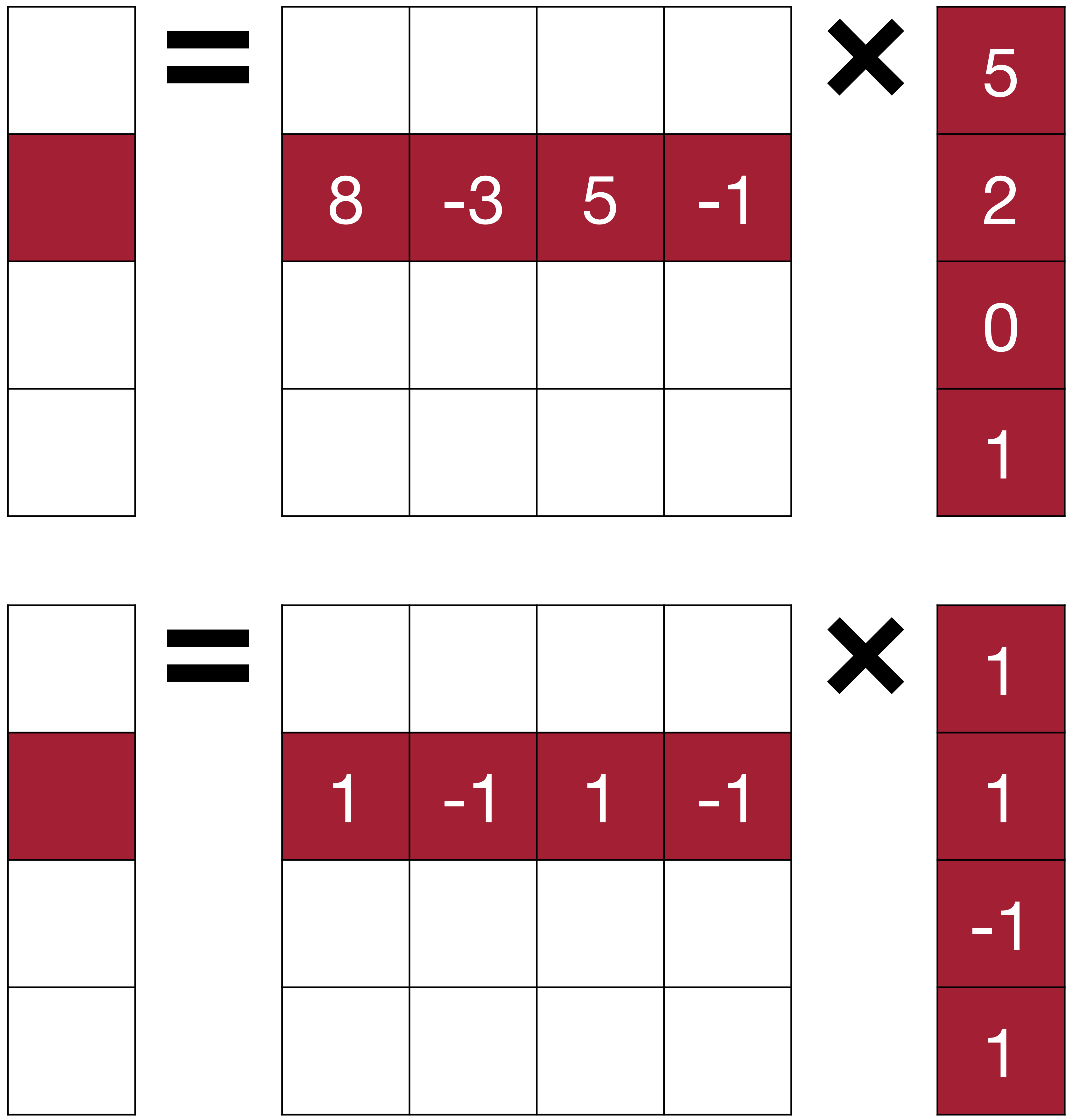
XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

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

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

$$= \boxed{1 \times 1 + (-1) \times 1 + 1 \times (-1) + (-1) \times 1}$$

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

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

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

$= 1{\times}1 + (-1){\times}1 + 1{\times}(-1) + (-1){\times}1$ $\quad = \boxed{1 \text{ xnor } 1 + 0 \text{ xnor } 1 + 1 \text{ xnor } 0 + 0 \text{ xnor } 1}$

$= 1 + (-1) + (-1) + (-1) = -2$ $\quad = 1 + 0 + 0 + 0 \ = \ 1$

?

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

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

$= 1×1 + (-1)×1 + 1×(-1) + (-1)×1$

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

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

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

$= 1 + 0 + 0 + 0 = \boxed{\begin{matrix} 1 \times 2 \\ + \\ -4 \end{matrix}} = -2$

+2

Assuming -1 -1 -1 -1 → -4

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

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

$$= -4 + 2 \times (1 \text{ xnor } 1 + 0 \text{ xnor } 1 + 1 \text{ xnor } 0 + 0 \text{ xnor } 1)$$

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

→ 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

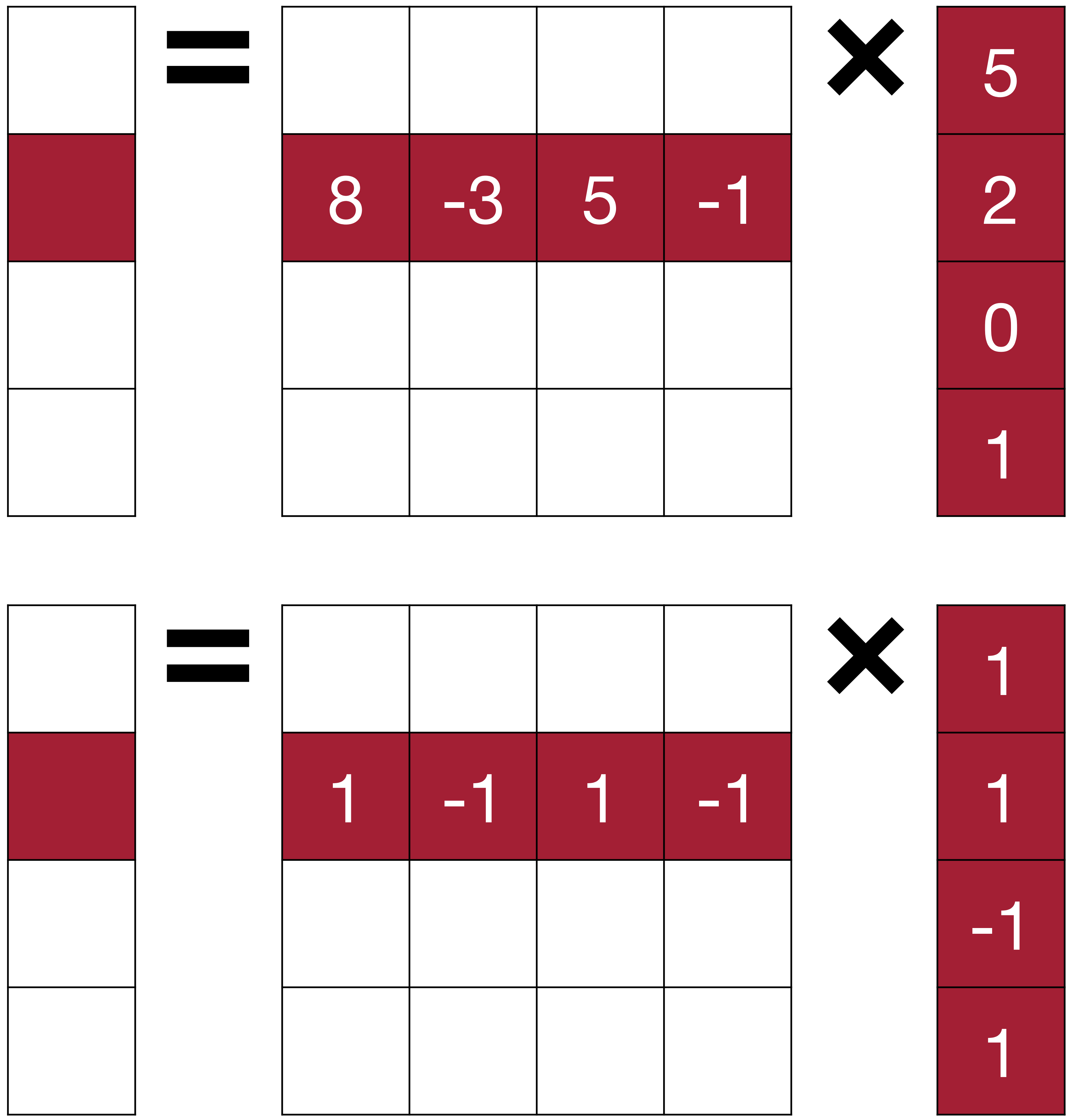
XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari *et al.*, ECCV 2016]

If both activations and weights are binarized

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

$$= -4 + \text{popcount}(1010 \text{ xnor } 1101) \ll 1$$

$$= -4 + \text{popcount}(1000) \ll 1 = -4 + 2 = -2$$



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]

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

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

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

$\longrightarrow$

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

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

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

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

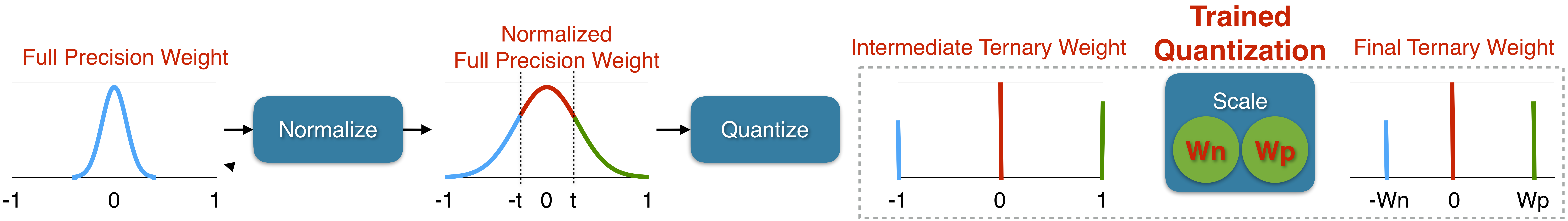
ImageNet Top-1 Accuracy	Full Precision	1 bit (BWN)	2 bit (TWN)
ResNet-18	69.6	60.8	65.3

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

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}$$

K-Means-based Quantization

$$\begin{bmatrix} 3 & 0 & 2 & 1 \\ 1 & 1 & 0 & 3 \\ 0 & 3 & 1 & 0 \\ 3 & 1 & 2 & 2 \end{bmatrix} \qquad \begin{array}{ll} 3: & 2.00 \\ 2: & 1.50 \\ 1: & 0.00 \\ 0: & -1.00 \end{array}$$

Linear Quantization

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1 \right) \times 1.07$$

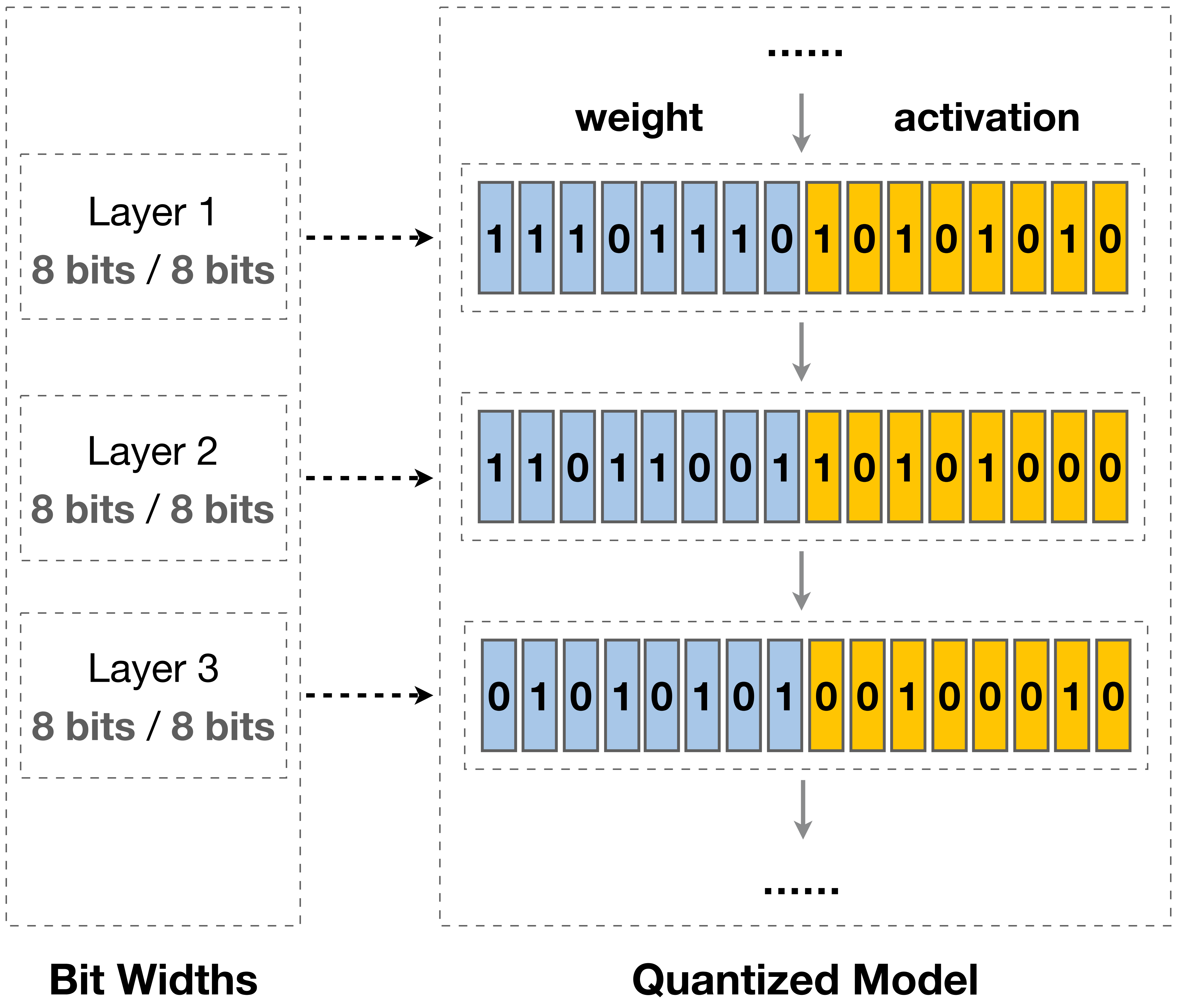
Binary/Ternary Quantization

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

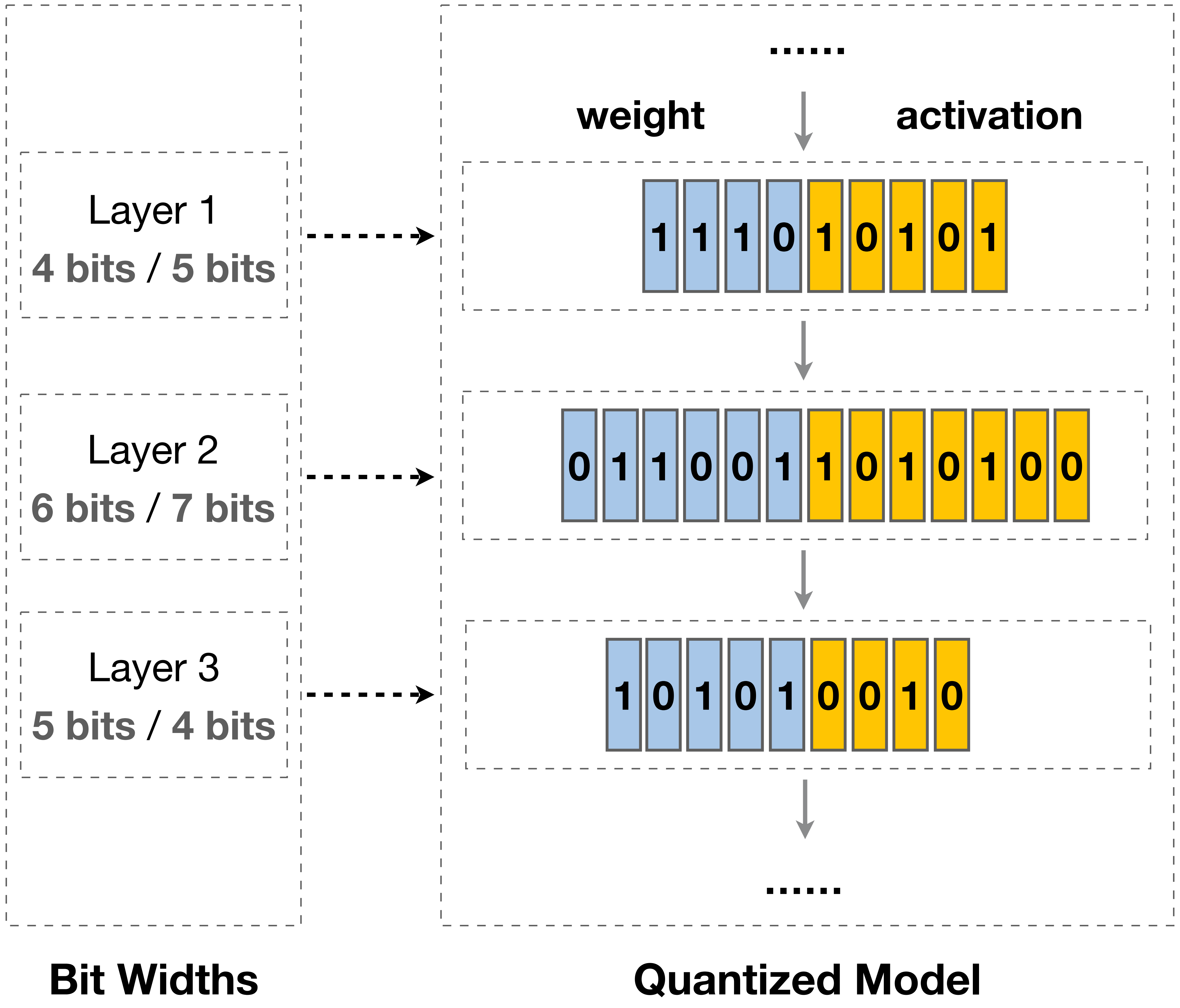
		K-Means-based Quantization	Linear Quantization	Binary/Ternary Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights	Binary/Ternary Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic	Bit Operations

Mixed-Precision Quantization

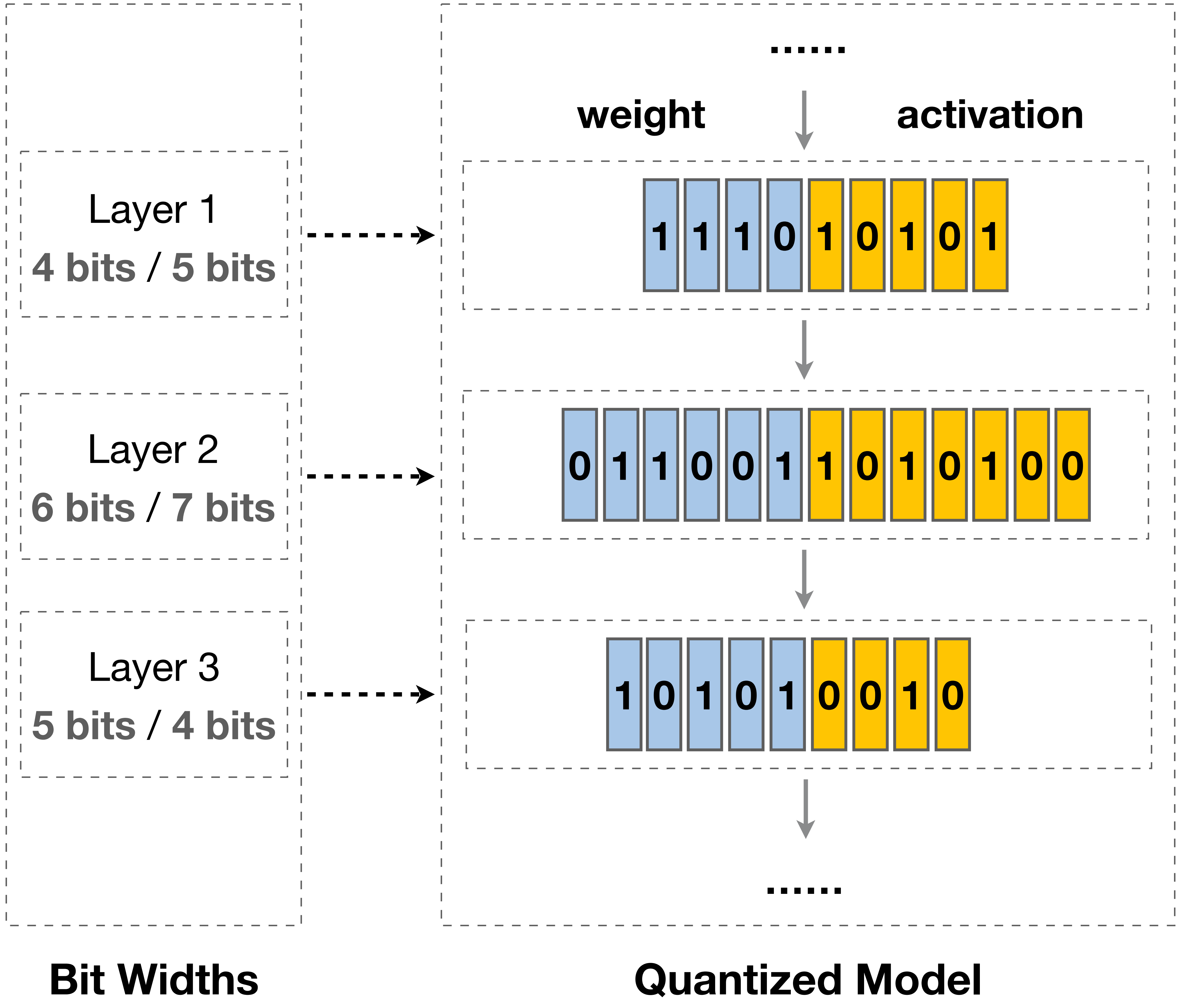
Uniform Quantization



Mixed-Precision Quantization



Challenge: Huge Design Space



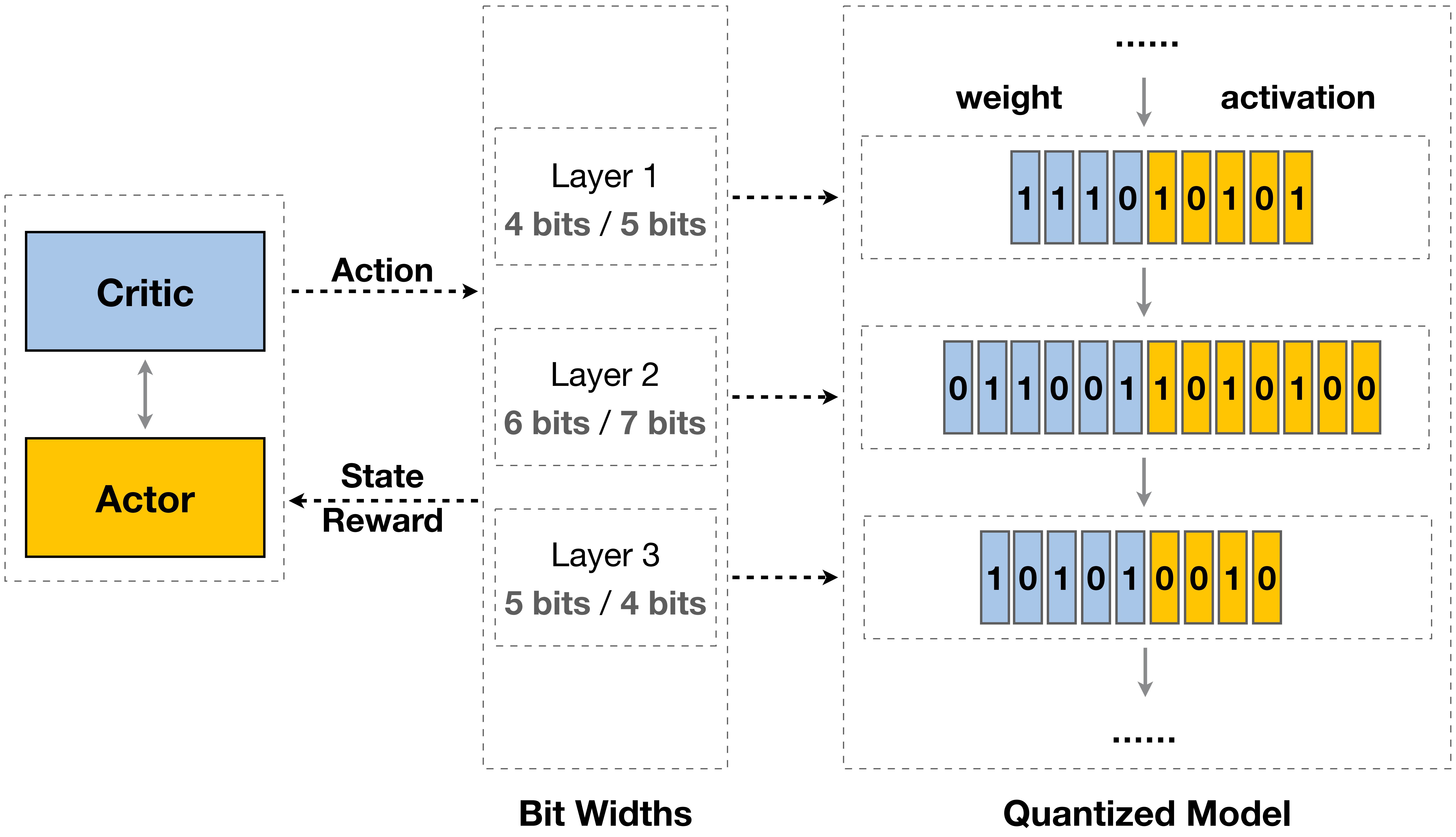
Choices: **8 x 8 = 64**

Choices: **8 x 8 = 64**

Choices: **8 x 8 = 64**

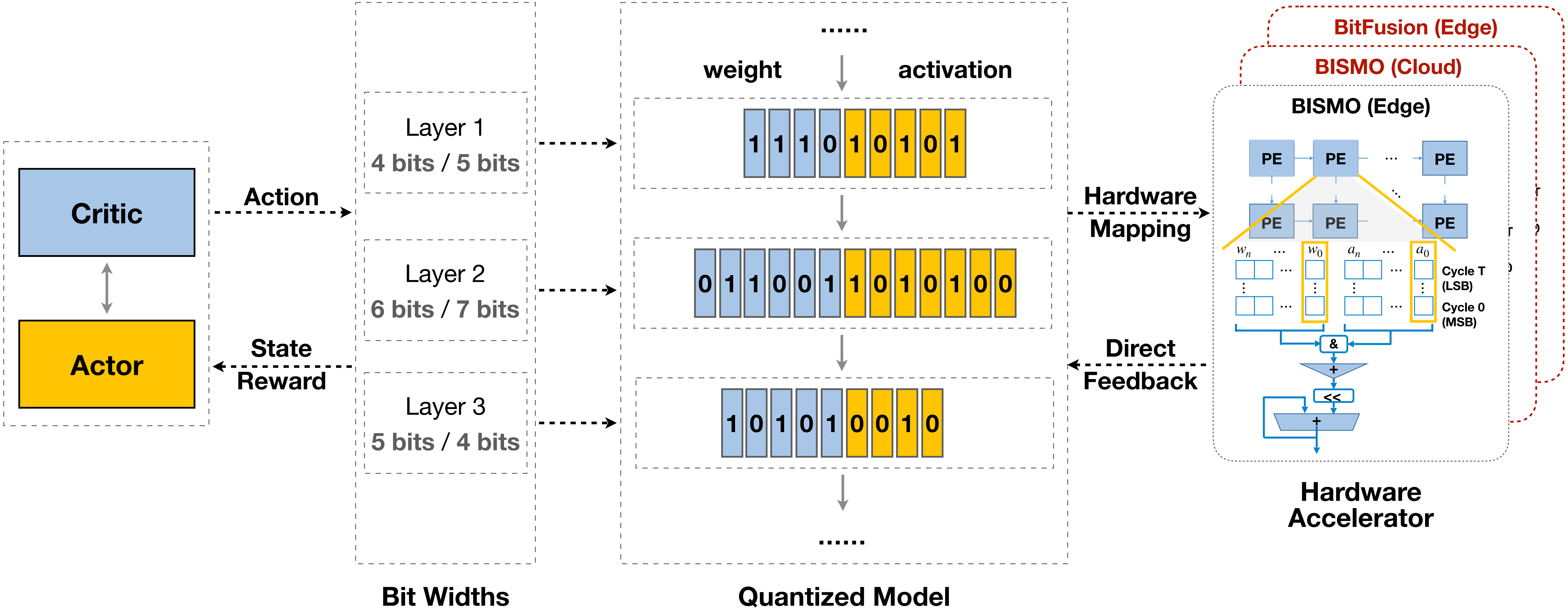
Design Space: 64^n

Solution: Design Automation



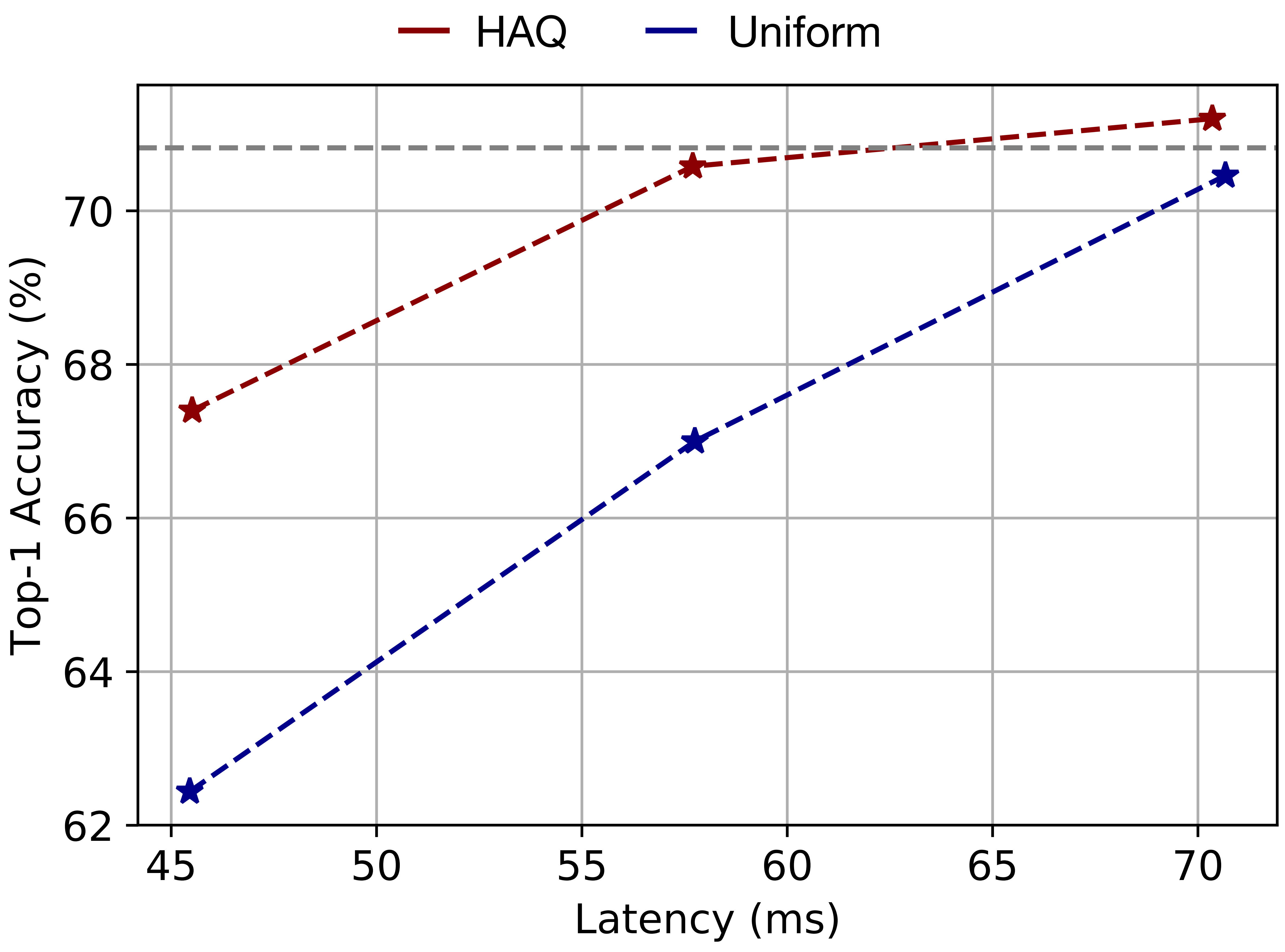
HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

Solution: Design Automation



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

HAQ Outperforms Uniform Quantization

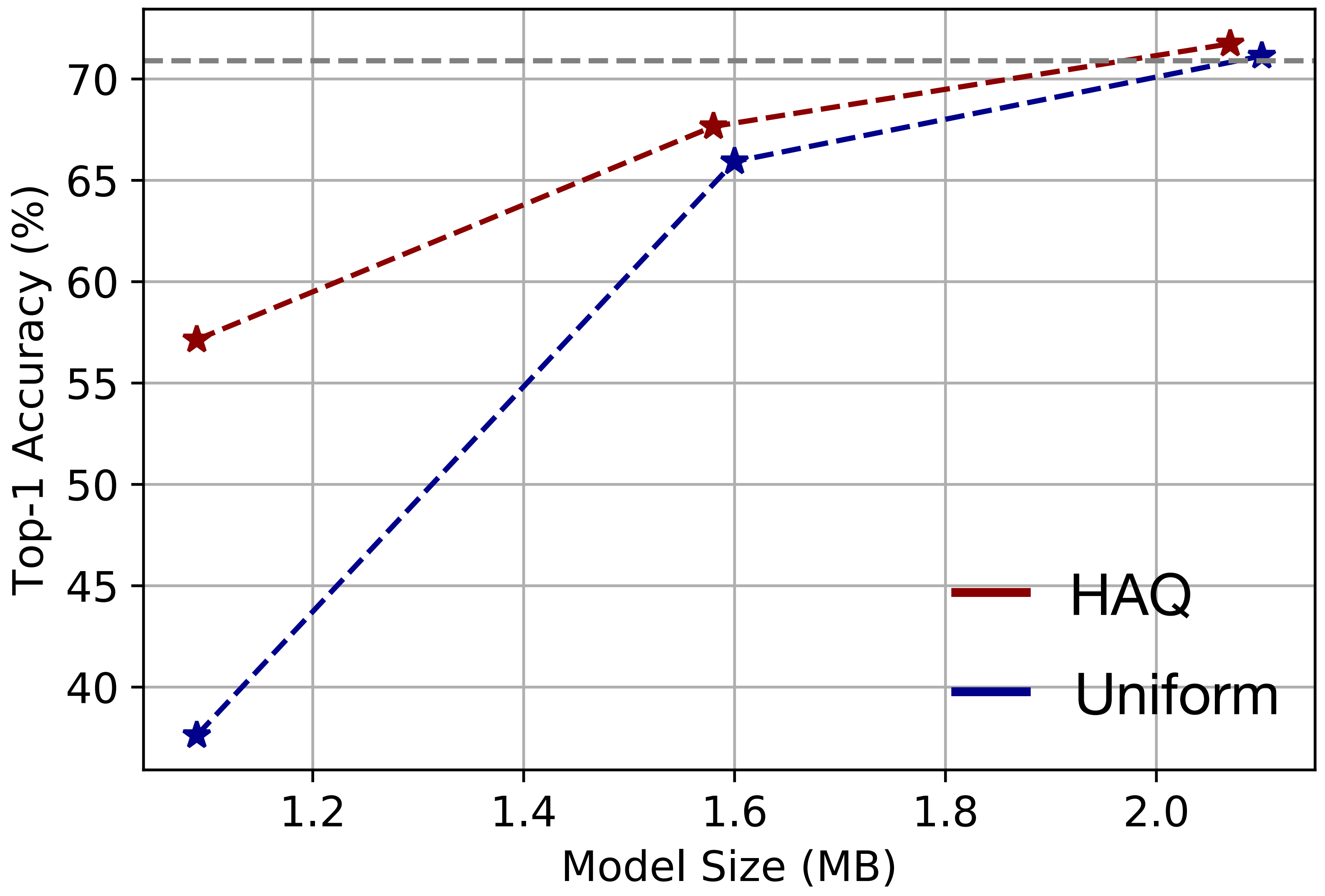


Mixed-Precision Quantized MobileNetV1

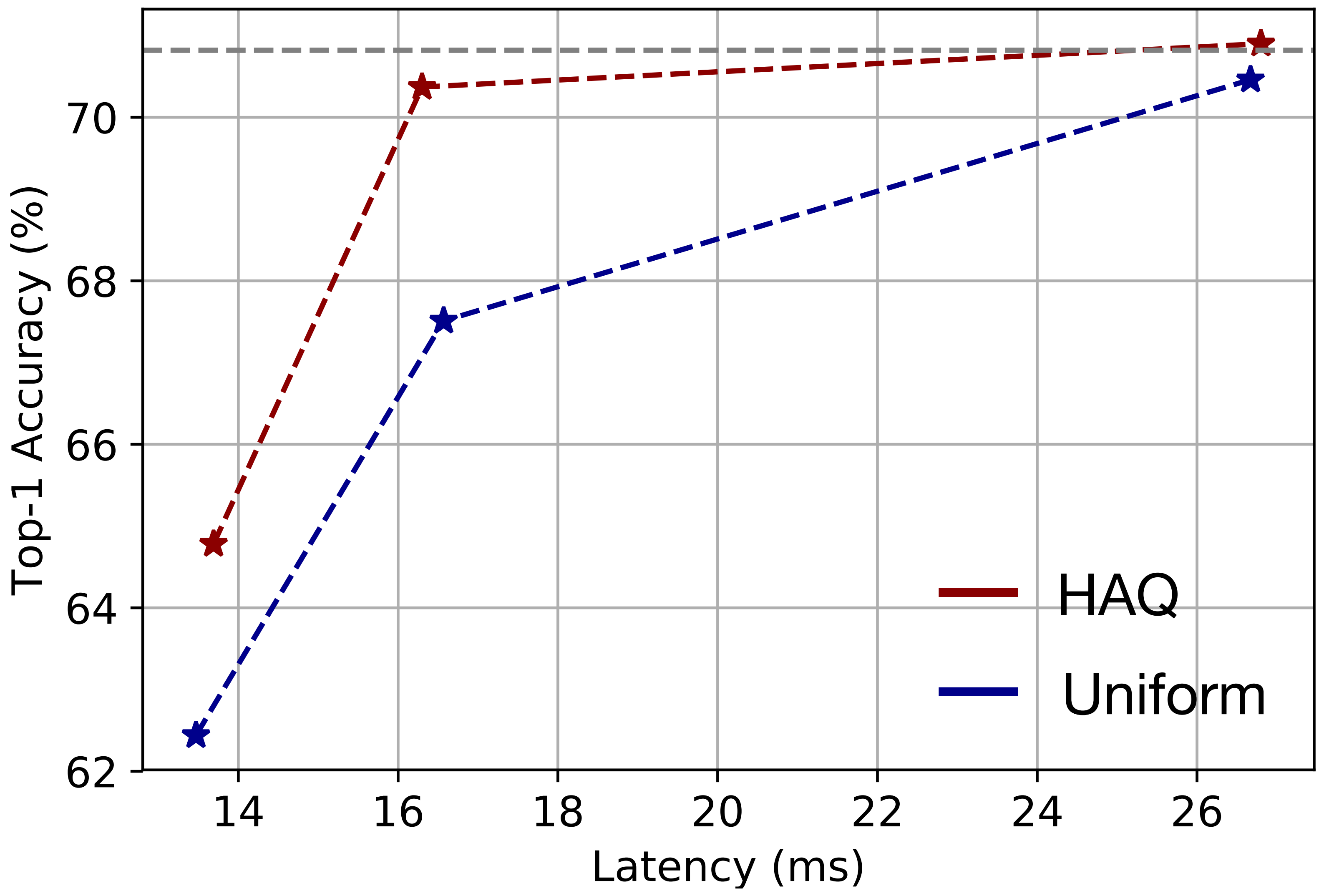
HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

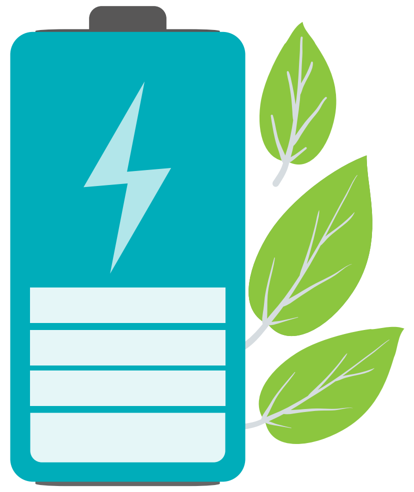
HAQ Supports Multiple Objectives

Model Size Constrained

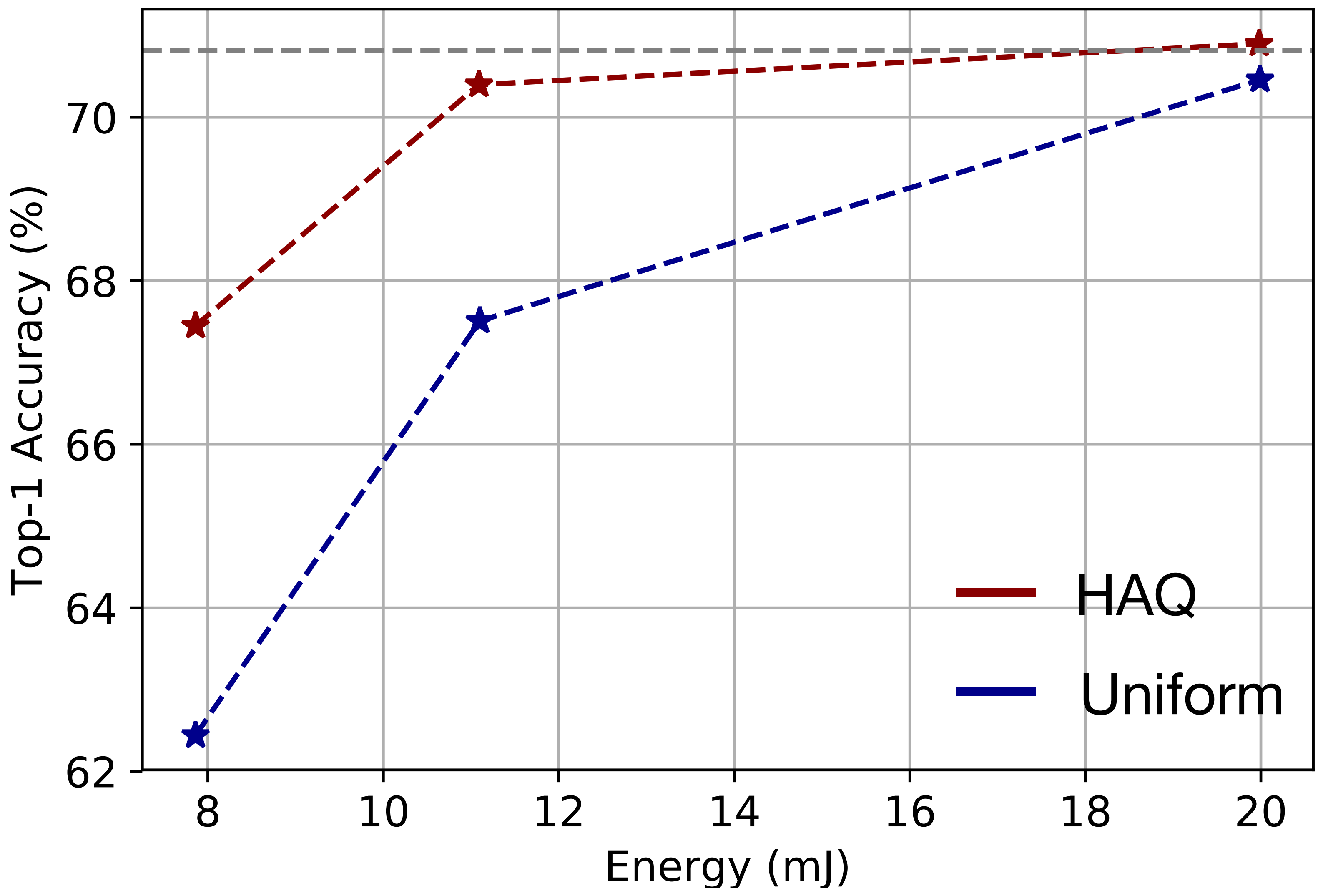


Latency Constrained





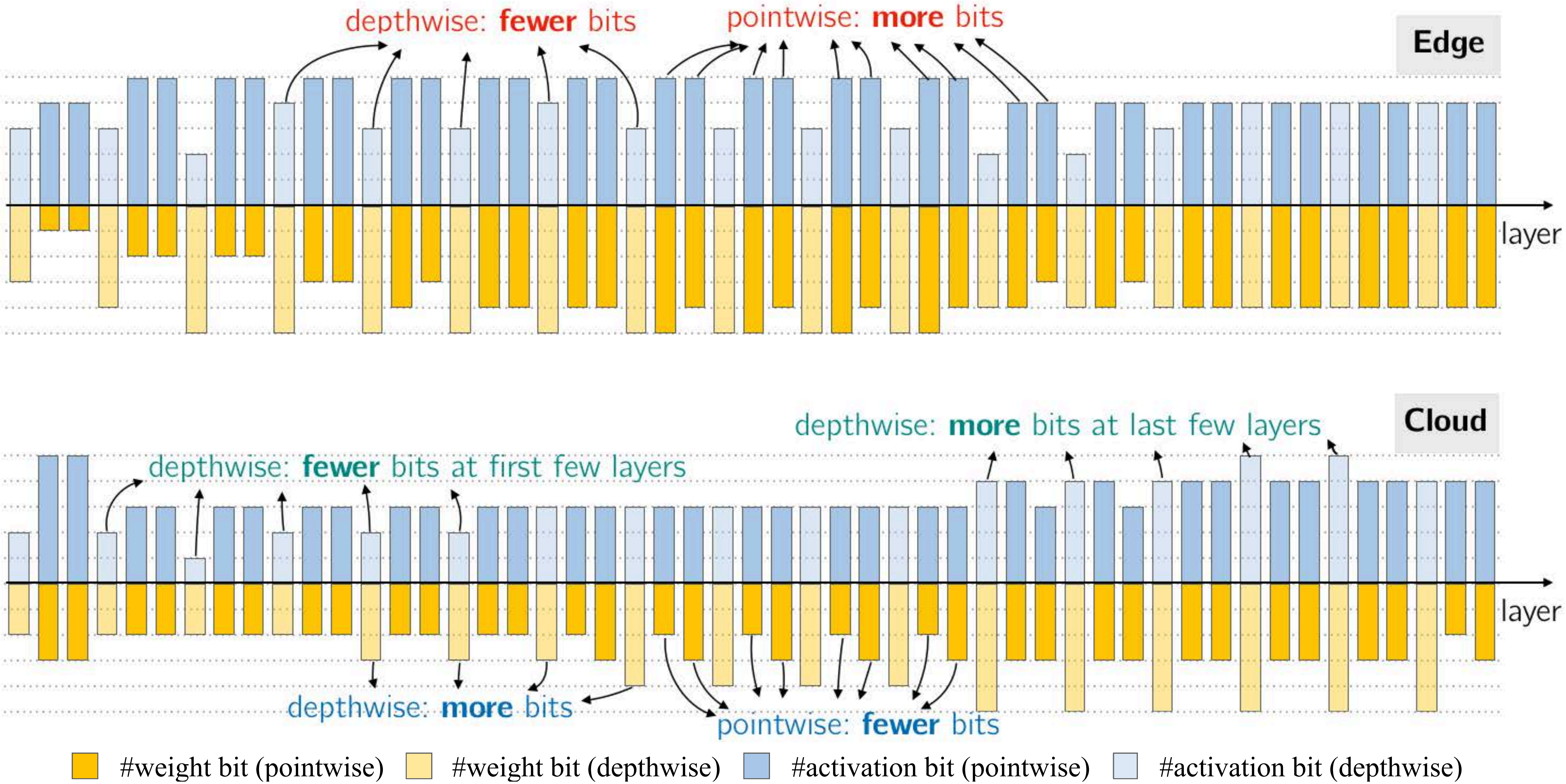
Energy Constrained



Mixed-Precision Quantized MobileNetV1

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

Quantization Policy for Edge and Cloud



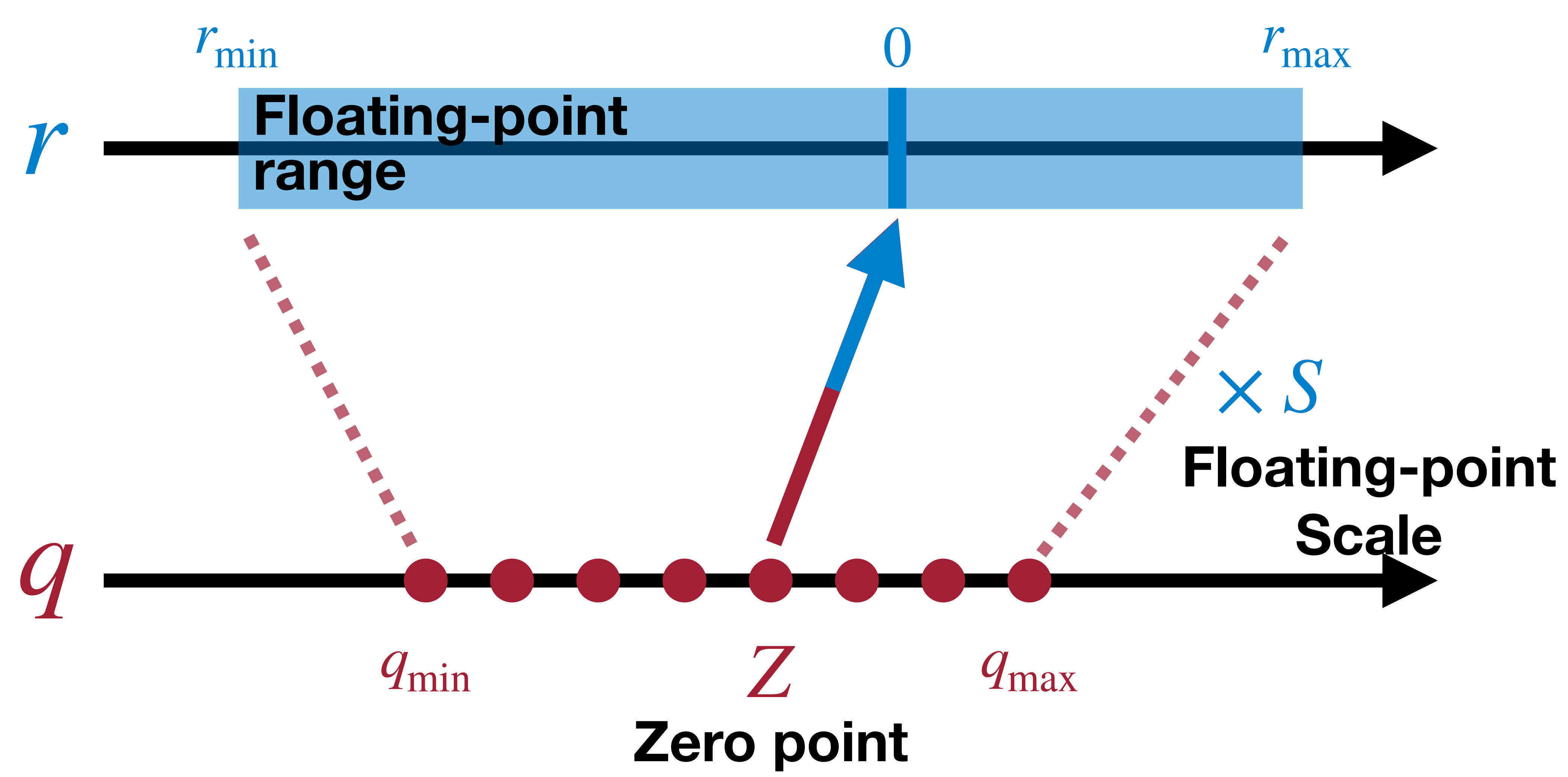
Mixed-Precision Quantized MobileNetV2

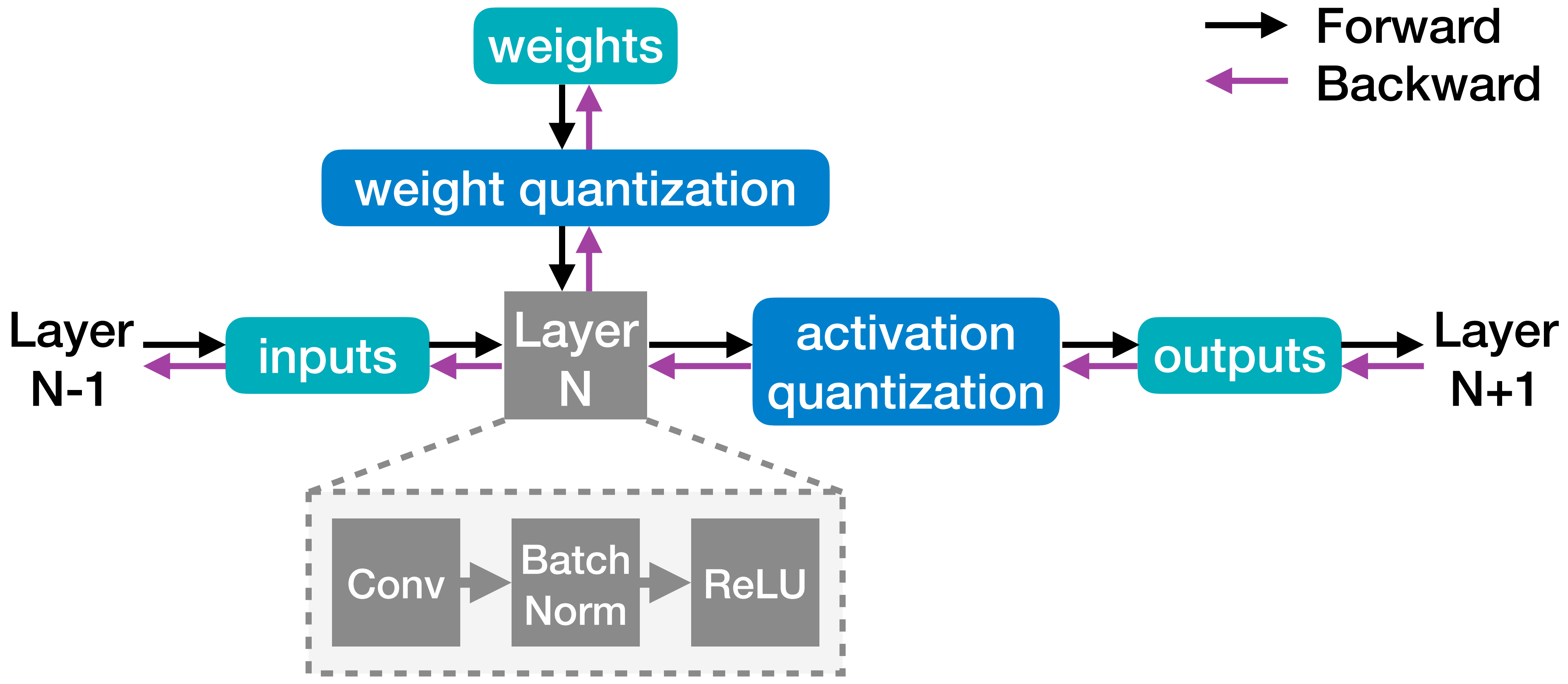
HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang *et al.*, CVPR 2019]

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 *vs.* group quantization
   - How to determine dynamic range for quantization
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]