Foundations of Edge AI

Lecture09

Neural Network Quantization

Lanyu (Lori) Xu

Email: lxu@oakland.edu
Homepage: https://lori930.github.io/
Office: EC 524

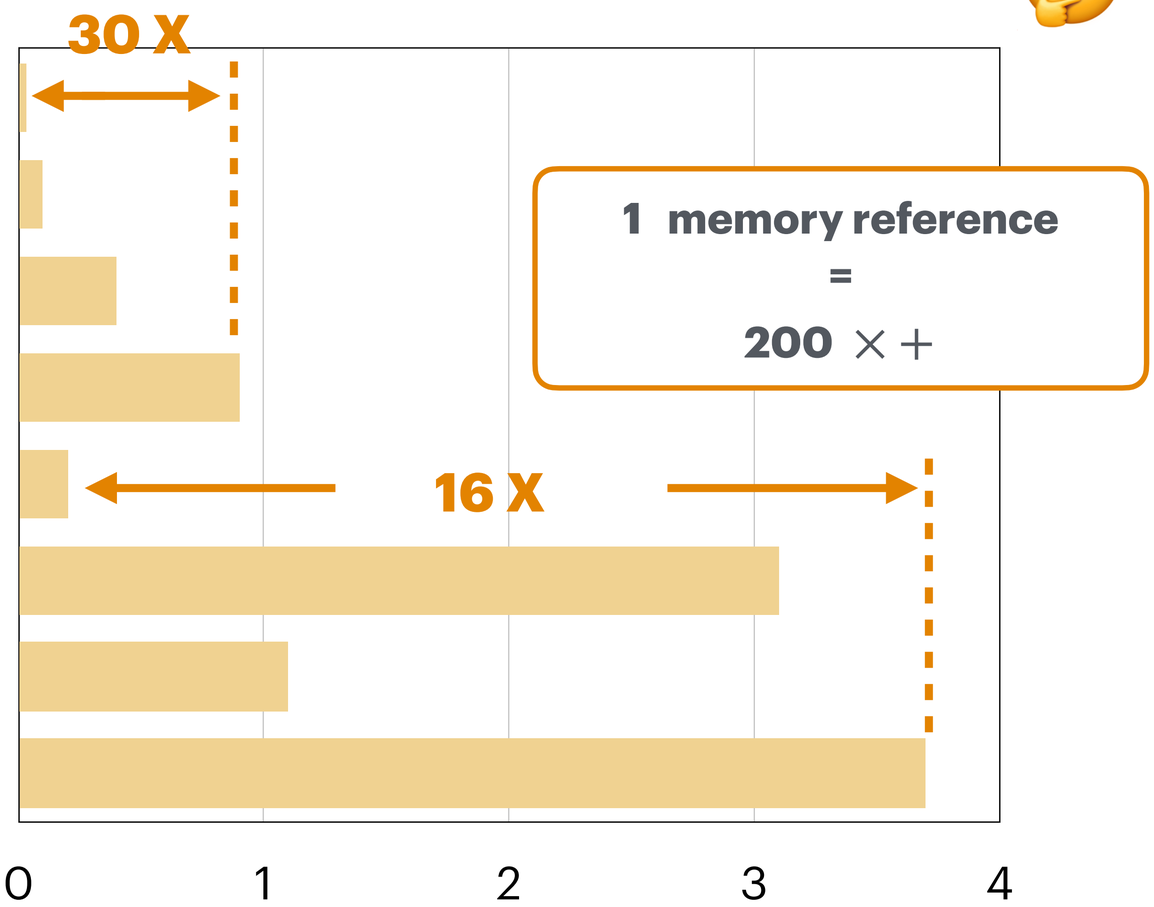
Low Bit-Width Operations are Cheaper

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

How should we make deep learning more efficient?



Horowitz, M. (2014, February). 1.1 computing's energy problem (and what we can do about it). In 2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC) (pp. 10-14). IEEE.

Model Size

Measures the storage for the weights of the given neural network.

Common units: MB, KB, bits

- In general, if the whole neural network uses the same data type (e.g., floating-point)
  - Model size = #Parameters · Bit Width
- AlexNet has 61M parameters
  - If all weights are stored with 32-bit numbers, total storage will be about
    - $61M \times 4\text{Bytes}\ (32\text{bits}) = 244MB(244 \times 10^6\text{Bytes})$
  - If all weights are stored with 8-bit numbers, total storage will be about
    - $61M \times 1\text{Bytes}(8\text{bits}) = 61MB$

Lecture Plan

Today we will

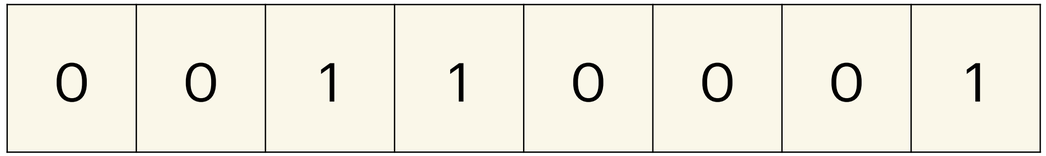
- Review the numeric data types used in the modern computing systems, including integers and floating-point numbers (FP32, FP16, INT4, etc.)
- Learn the basic concept of neural network quantization
- Learn common neural network quantization
  - K-Means-based Quantization
  - Linear Quantization (very widely used)

Numeric Data Types

How is numeric data represented in modern computing systems?

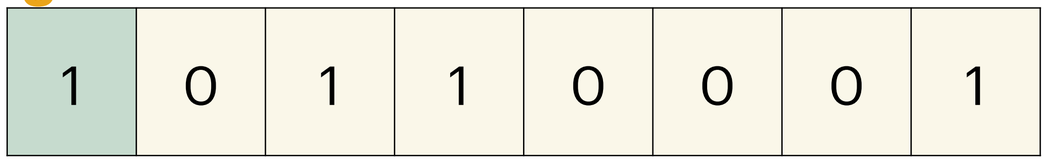
Integer

- **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}

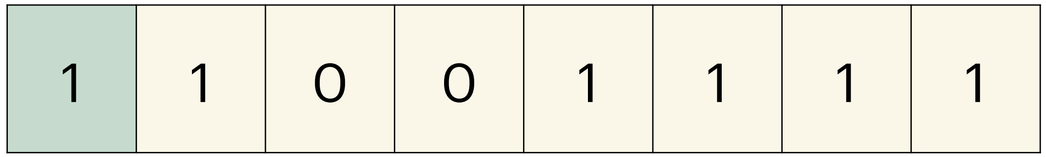


$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 49$$

Sign Bit



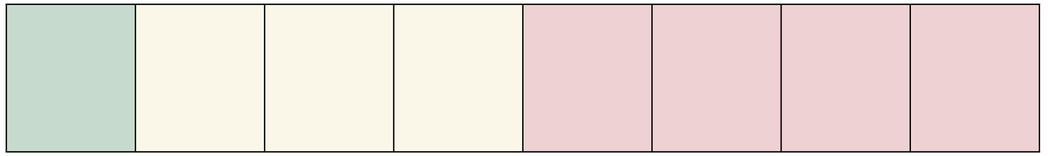
$$-\ 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$



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

Fixed-Point Number

Using 2's complement representation



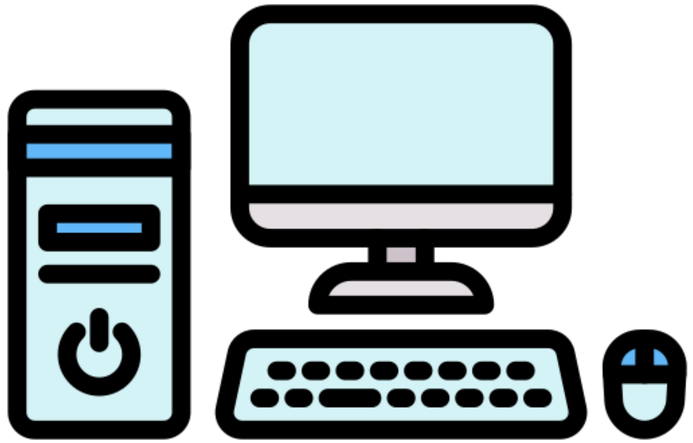
0	0	1	1	0	0	0	1

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

0	0	1	1	0	0	0	1

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

Decimal?
Floating-point!



Why “floating point”?

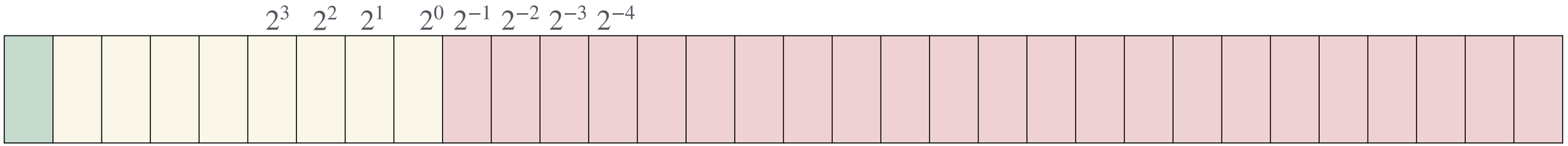
The term floating point refers to the fact that the number's radix point can "float" anywhere to the left, right, or between the significant digits of the number. This position is indicated by the exponent.

Is there any standard to represent “floating point”?

IEEE Standard for Floating-Point Arithmetic (IEEE 754).

Floating-Point Number

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

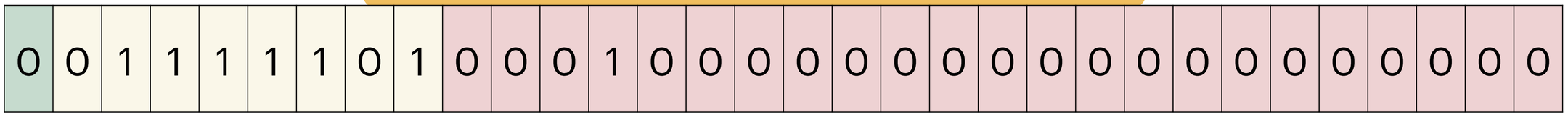


Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

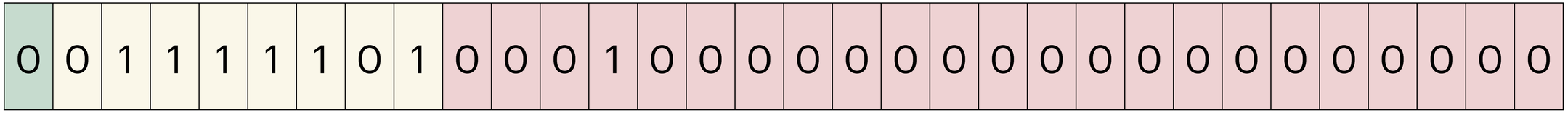
$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \leftarrow \text{Exponent Bias} = 127 = 2^{8-1} - 1$$

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

What is the FP representation for 0.265625?



What is the FP representation for 0.265625?



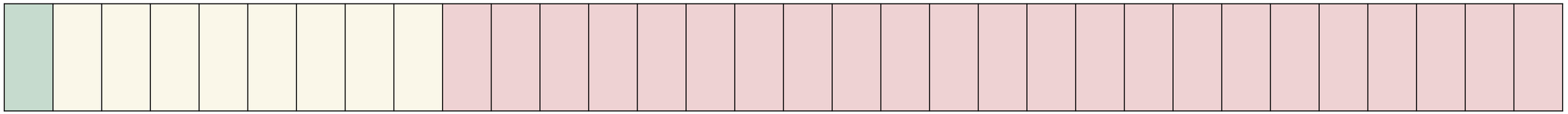
- Convert the decimal number to binary
  - $0.265625 \times 2 = 0.53125 \rightarrow 0$
  - $0.53125 \times 2 = 1.0625 \rightarrow 01$
  - $0.0625 \times 2 = 0.125 \rightarrow 010$
  - $0.125 \times 2 = 0.25 \rightarrow 0100$
  - $0.25 \times 2 = 0.5 \rightarrow 01000$
  - $0.5 \times 2 = 1 \rightarrow 010001$
  - The factional part is now 0.0, the process is complete
  - $(0.265625)_{10} = (0.010001)_2$
- Normalize: shift the binary point so that there's only one nonzero digit to the left of the binary point
  - $(0.010001)_2 = (1.0001)_2 \times 2^{-2}$

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

  - $1 = (-1)^0 \rightarrow \text{sign} = 0$
  - Exponent $= 127 - (-2) = 125 = (01111101)_2$
  - Fraction $= 0001$, trailing 0s to be 23 bits long

Floating-Point Number

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



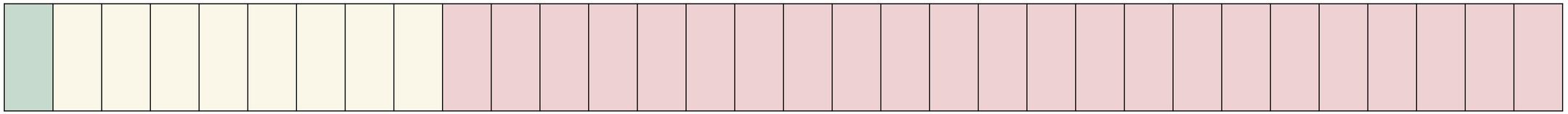
Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \leftarrow \text{Exponent Bias} = 127 = 2^{8-1} - 1$$

How to represent 0?

Floating-Point Number

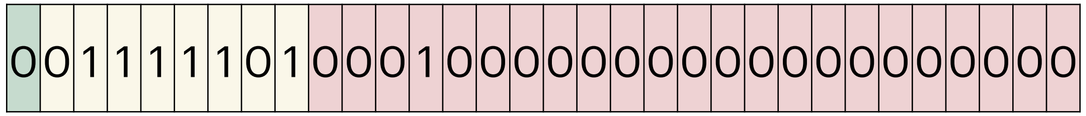
Example: 32-bit floating-point number in IEEE 754



Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

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

Normal Numbers, Exponent $\neq 0$

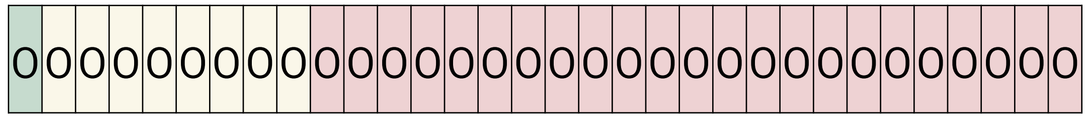


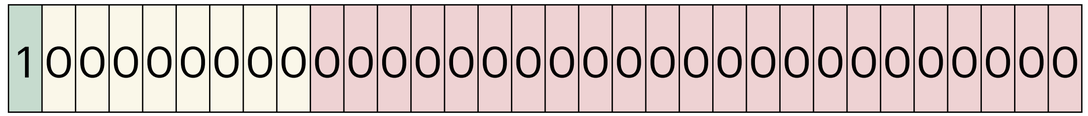
$$0.265625 = 1.0625 \times 2^{-2} = (1 + 0.0625) \times 2^{125-127}$$

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

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

Subnormal Numbers, Exponent $= 0$

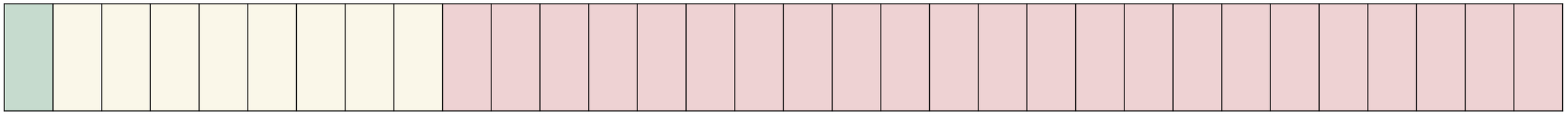




$$0 = 0 \times 2^{-126}$$

Floating-Point Number

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



Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

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

Normal Numbers, Exponent $\neq 0$

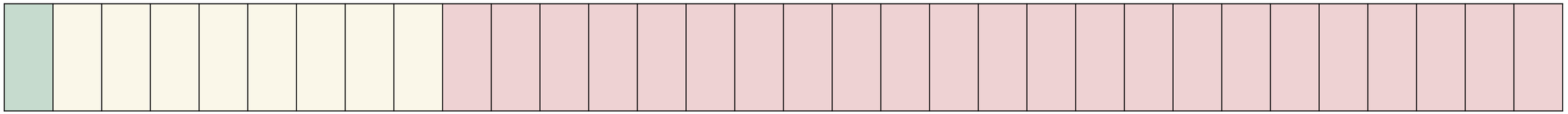
$$(-1)^{\text{sign}} \times \text{Fraction} \times 2^{1-127}$$

Subnormal Numbers, Exponent $= 0$

What is the smallest positive subnormal value?

Floating-Point Number

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



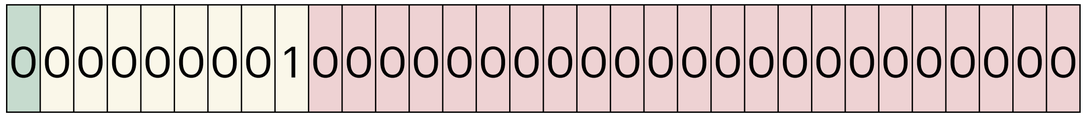
Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

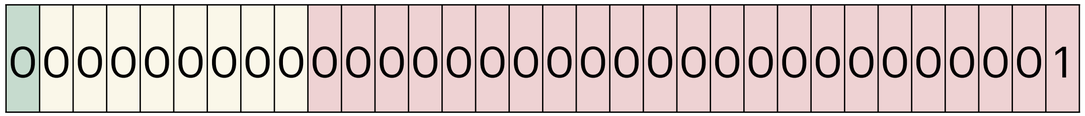
$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127}$$

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

Normal Numbers, Exponent $\neq 0$

Subnormal Numbers, Exponent $= 0$





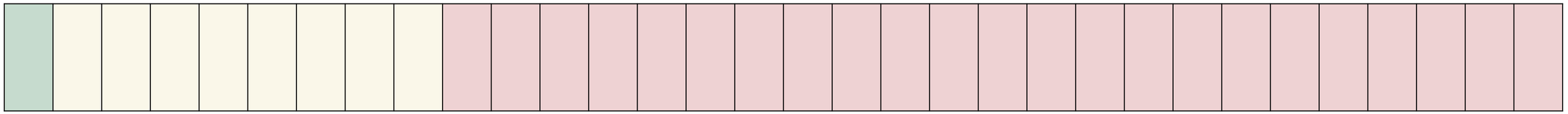
2^{-23}

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

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

Floating-Point Number

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



Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

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

Normal Numbers, Exponent $\neq 0$

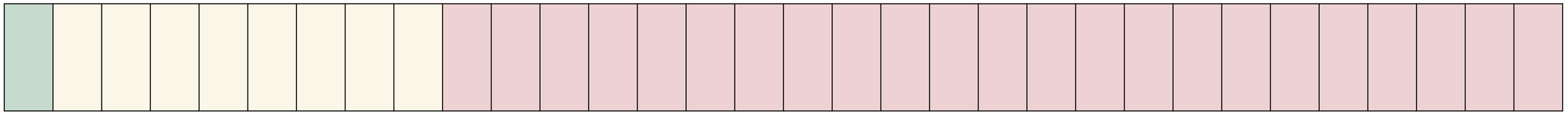
$$(-1)^{\text{sign}} \times \text{Fraction} \times 2^{1-127}$$

Subnormal Numbers, Exponent $= 0$

What is the largest positive subnormal value?

Floating-Point Number

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

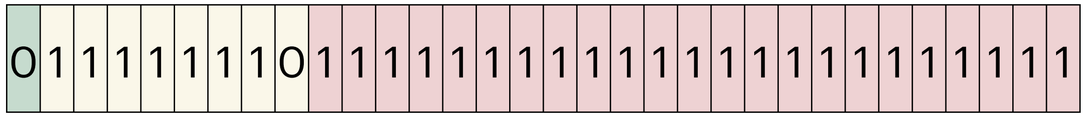


Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

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

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

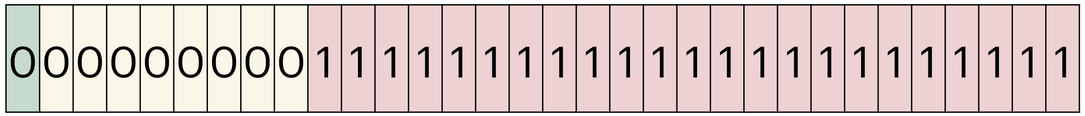
Normal Numbers, Exponent $\neq 0$



$$2^{-23} + 2^{-22} + \ldots + 2^{-1} = 1 - 2^{-23}$$

$$(1 + 1 - 2^{-23}) \times 2^{254-127} = (2 - 2^{-23}) \times 2^{127}$$

Subnormal Numbers, Exponent $= 0$

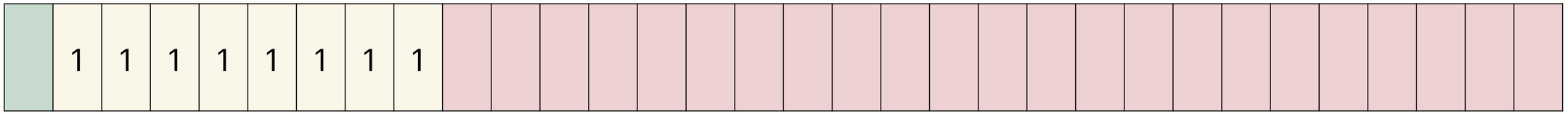


$$2^{-23} + 2^{-22} + \ldots + 2^{-1} = 1 - 2^{-23}$$

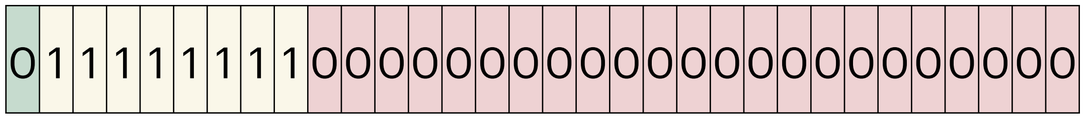
$$(1 - 2^{-23}) \times 2^{-126} = 2^{-126} - 2^{-149}$$

Floating-Point Number

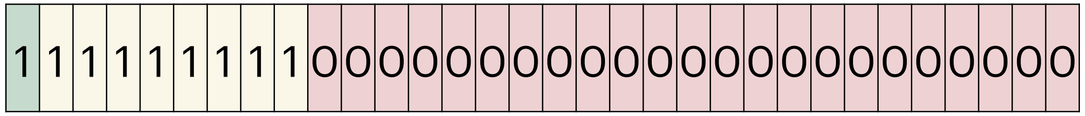
Example: 32-bit floating-point number in IEEE 754



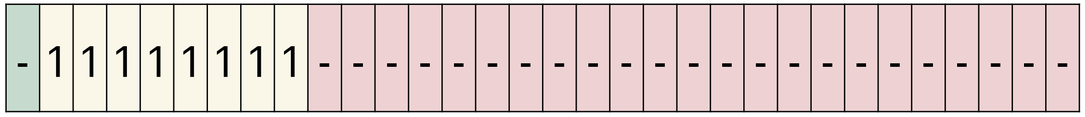
Sign **8 bit Exponent** **23 bit Fraction/Mantissa**



$+\infty$ (positive infinity)



$-\infty$ (negative infinity)

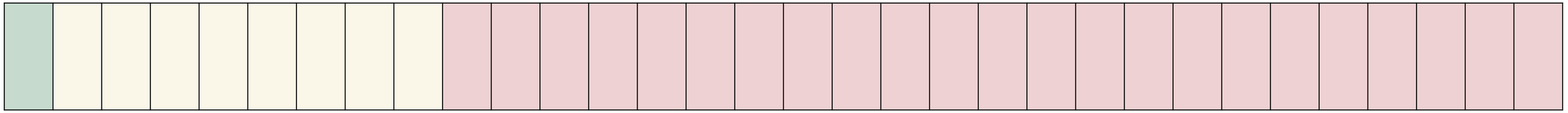


NaN (Not a Number)

Much waste! How to make it efficient?

Floating-Point Number

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



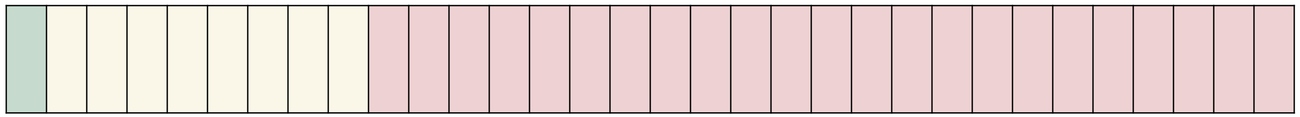
Sign **8 bit Exponent** **23 bit Fraction/Mantissa**

Exponent	Fraction=0	Fraction $\neq$ 0	Equation
$00_H = 0$	± 0	Subnormal	$(-1)^{\text{sign}} \times \text{Fraction} \times 2^{1-127}$
$01_H \ldots FE_H = 1 \ldots 254$	Normal		$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127}$
$FF_H = 255$	$\pm$INF	NaN	

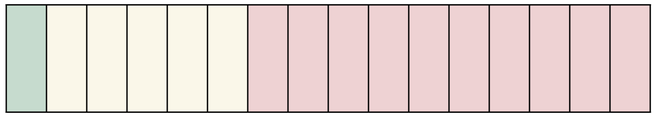
Floating-Point Number

Exponent Width → Range; Fraction Width → Precision

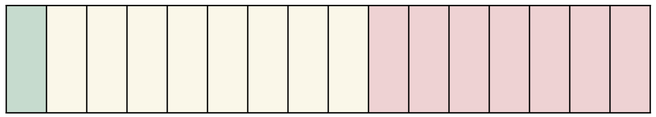
- IEEE 754 Single Precision 32-bit Float (IEEE FP32)



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



- Google Brain Float (BF16)

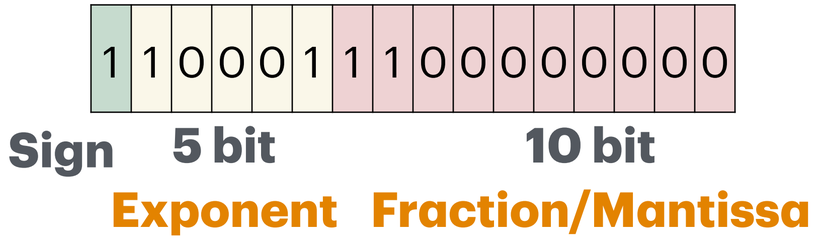


If FP16 leads to instability (such as divergence) during training, switching to BF16 can be a good solution.

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

IEEE FP16

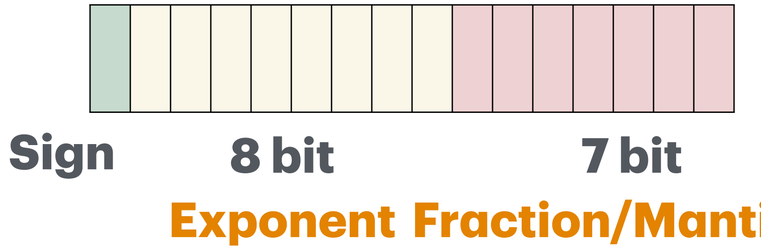
What is the following IEEE half precision (IEEE FP16) number in decimal?



$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-15} \leftarrow \text{Exponent Bias} = 15 = 2^{5-1} - 1$$

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

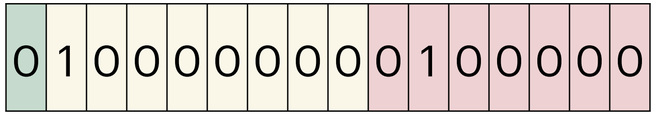
Google BF16



What is the decimal 2.5 in Brain Float (BF16)?

$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \leftarrow \text{Exponent Bias} = 127 = 2^{8-1} - 1$$

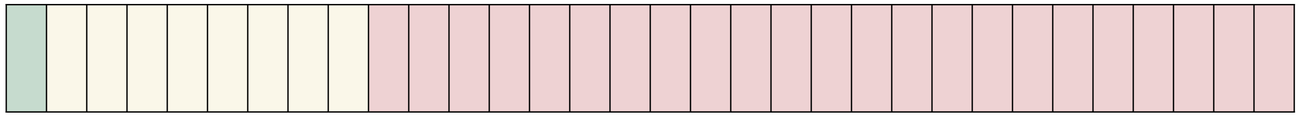
- $2.5_{10} = 10.1_2 = 1.01_2 \times 2^1 = (-1)^0 \times (1 + 0.01)_2 \times 2^1$
- Sign: +
- Exponent binary: $1_{10} + 127_{10} = 128_{10} = 10000000_2$
- Fraction binary: 0100000



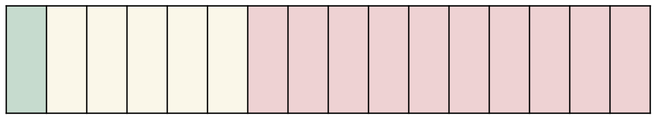
Floating-Point Number

Exponent Width → Range; Fraction Width → Precision

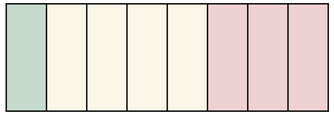
- IEEE 754 Single Precision 32-bit Float (IEEE FP32)



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



- Nvidia FP8 (E4M3)



FP8 E4M3 does not have INF, $S.1111.111_2$ is used for NaN

Largest FP8 E4M3 normal value is $S.1111.110_2 = 448$

- Nvidia FP8 (E5M2) for gradient in the backward

FP8 E4M3 has INF ($S.11111.00_2$) and NaN ($S.11111.XX_2$)

Largest FP8 E4M3 normal value is $S.1111.110_2 = 448$

Exponent (Bits)	Fraction (Bits)	Total (Bits)
8	23	32
5	10	16
4	3	8
5	2	8

INT4 and FP4

Exponent Width → Range; Fraction Width → Precision

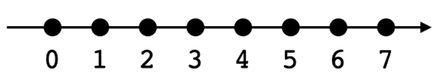
INT4

S			
0	0	0	1
0	1	1	1

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

=1

=7



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

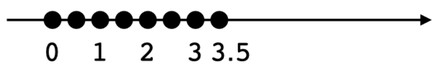
FP4 (E1M2)

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

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

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

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



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

FP4 (E2M1)

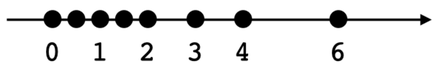
S	E	E	M
0	0	0	1
0	1	1	1

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

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

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

no inf, no NaN



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

FP4 (E3M0)

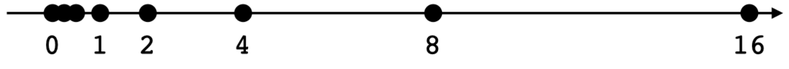
S	E	E	E
0	0	0	1
0	1	1	1

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

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

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

no inf, no NaN



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

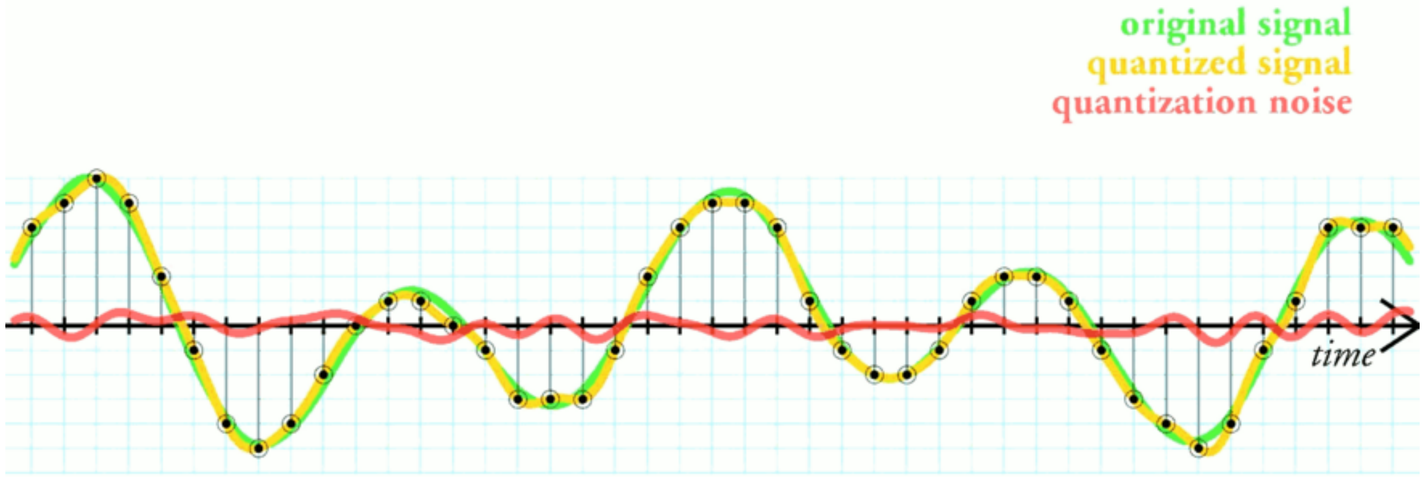
Figure source

What is Quantization?

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

Quantization

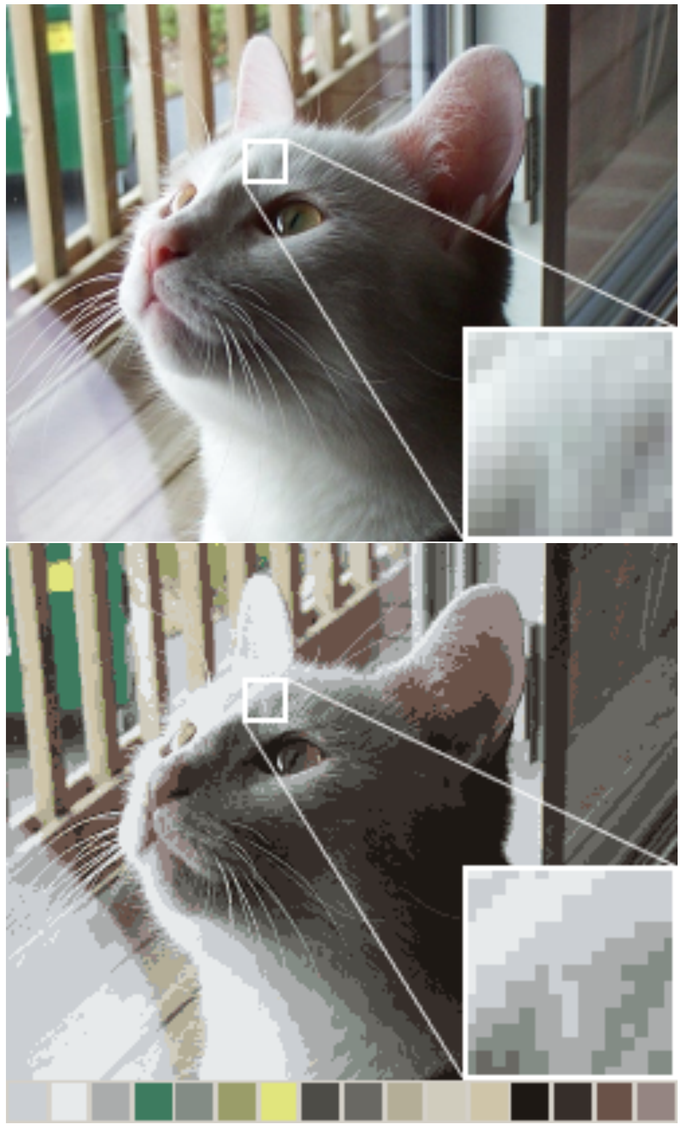
Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



Quantization (Wiki)

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

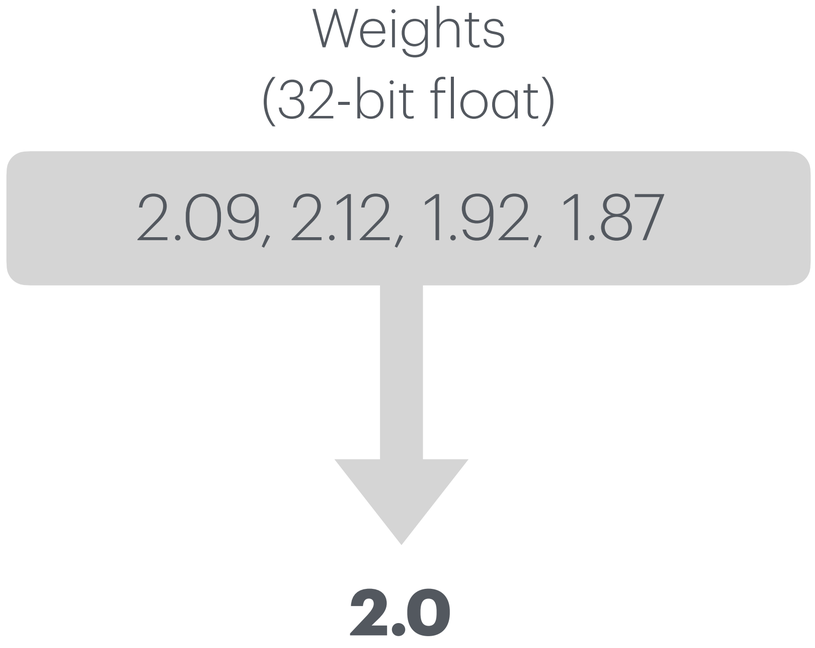
Original image vs. 16-Color image 👉



Color quantization (Wiki)

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



Storage	Floating-Point Weights
Computation	Floating-Point Arithmetic

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

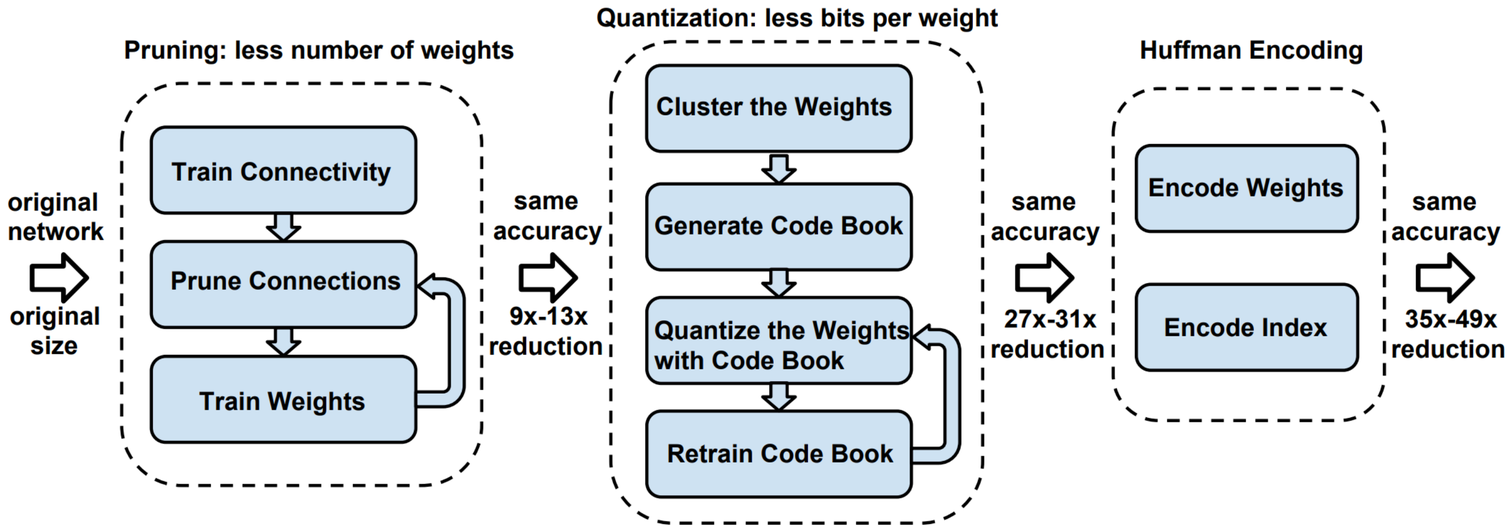
3	0	2	1	**3:**	2.00
1	1	0	3	**2:**	1.50
0	3	1	0	**1:**	0.00
3	1	2	2	**0:**	-1.00

K-Means-based Quantization

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

K-Means-based Weight Quantization

Deep Compression [ICLR'16]



DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING

Song Han
Stanford University, Stanford, CA 94305, USA
songhan@stanford.edu

Huizi Mao
Tsinghua University, Beijing, 100084, China
mhz12@mails.tsinghua.edu.cn

William J. Dally
Stanford University, Stanford, CA 94305, USA
NVIDIA, Santa Clara, CA 95050, USA
dally@stanford.edu

ABSTRACT

Neural networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources. To address this limitation, we introduce "deep compression", a three stage pipeline: pruning, trained quantization and Huffman coding, that work together to reduce the storage requirement of neural networks by 35× to 49× without affecting their accuracy. Our method first prunes the network by learning only the important connections. Next, we quantize the weights to enforce weight sharing, finally, we apply Huffman coding. After the first two steps we retrain the network to fine tune the remaining connections and the quantized centroids. Pruning, reduces the number of connections by 9× to 13×; Quantization then reduces the number of bits that represent each connection from 32 to 5. On the ImageNet dataset, our method reduced the storage required by AlexNet by **35**×, from 240MB to 6.9MB, without loss of accuracy. Our method reduced the size of VGG-16 by **49**× from 552MB to 11.3MB, again with no loss of accuracy. This allows fitting the model into on-chip SRAM cache rather than off-chip DRAM memory. Our compression method also facilitates the use of complex neural networks in mobile applications where application size and download bandwidth are constrained. Benchmarked on CPU, GPU and mobile GPU, compressed network has 3× to 4× layerwise speedup and 3× to 7× better energy efficiency.

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

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

Centroids

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

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

Storage

$32\text{bit} \times 16 = 512\text{bit} = 64\text{B}$

Indexes $2\text{bit} \times 16 = 32\text{bit} = 4\text{B}$

Codebook $32\text{bit} \times 4 = 128\text{bit} = 16\text{B}$

Total: 20B

3.2 X smaller!

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

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

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

Centroids

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

Indexes **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

$32\text{bit} \times 16 = 512\text{bit} = 64\text{B}$

$2\text{bit} \times 16 = 32\text{bit} = 4\text{B}$

$32\text{bit} \times 4 = 128\text{bit} = 16\text{B}$

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

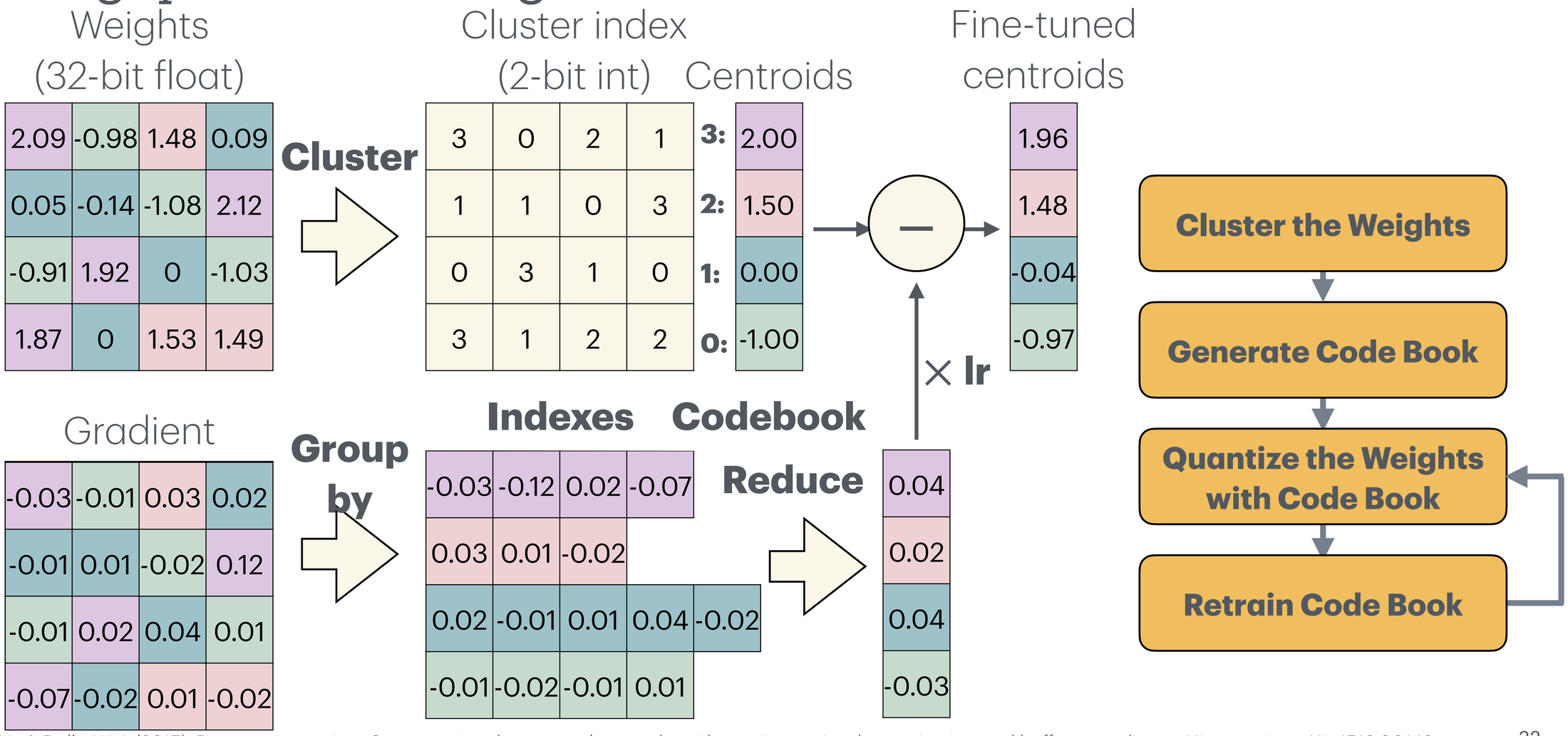
$32\text{bit} \times M = 32M\text{bit}$ $N\text{bit} \times M = NM\text{bit}$ $32\text{bit} \times 2^N = 2^{N+5}\text{bit}$

32/N smaller!

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

K-Means-based Weight Quantization

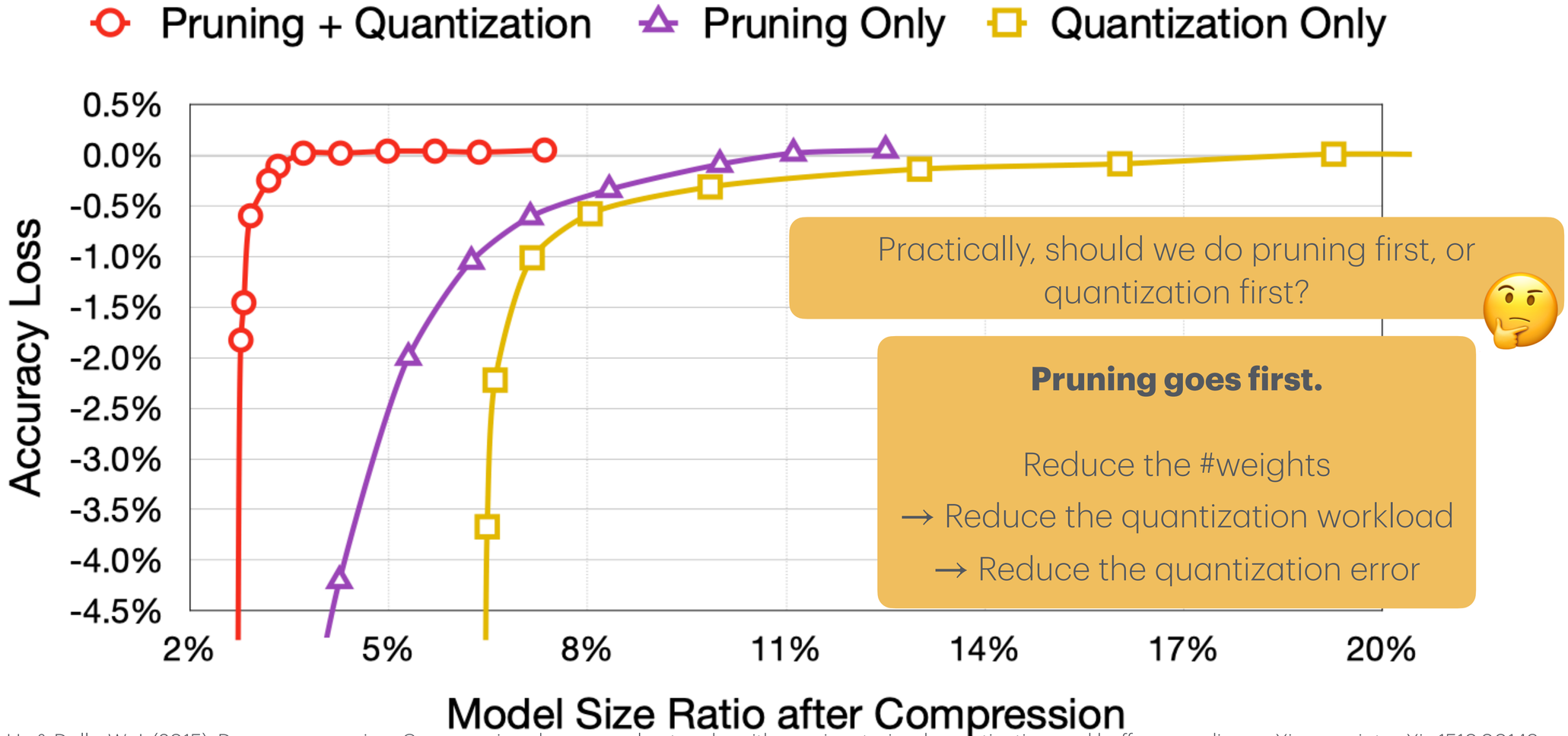
Fine-tuning quantizaed weights



Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

K-Means-based Weight Quantization

Accuracy vs. compression rate for AlexNet on ImageNet dataset



Practically, should we do pruning first, or quantization first?

Pruning goes first.

Reduce the #weights

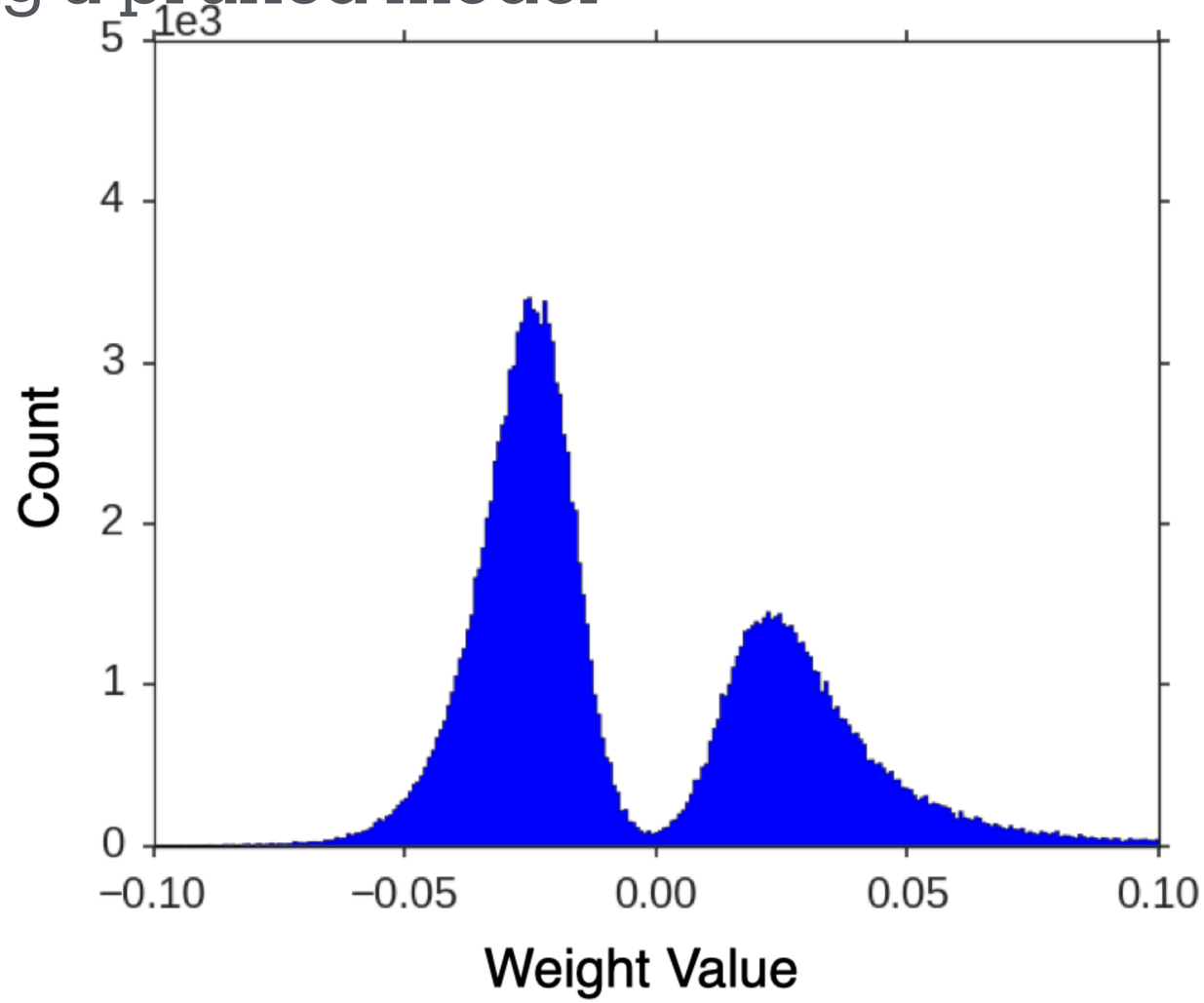
→ Reduce the quantization workload

→ Reduce the quantization error

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

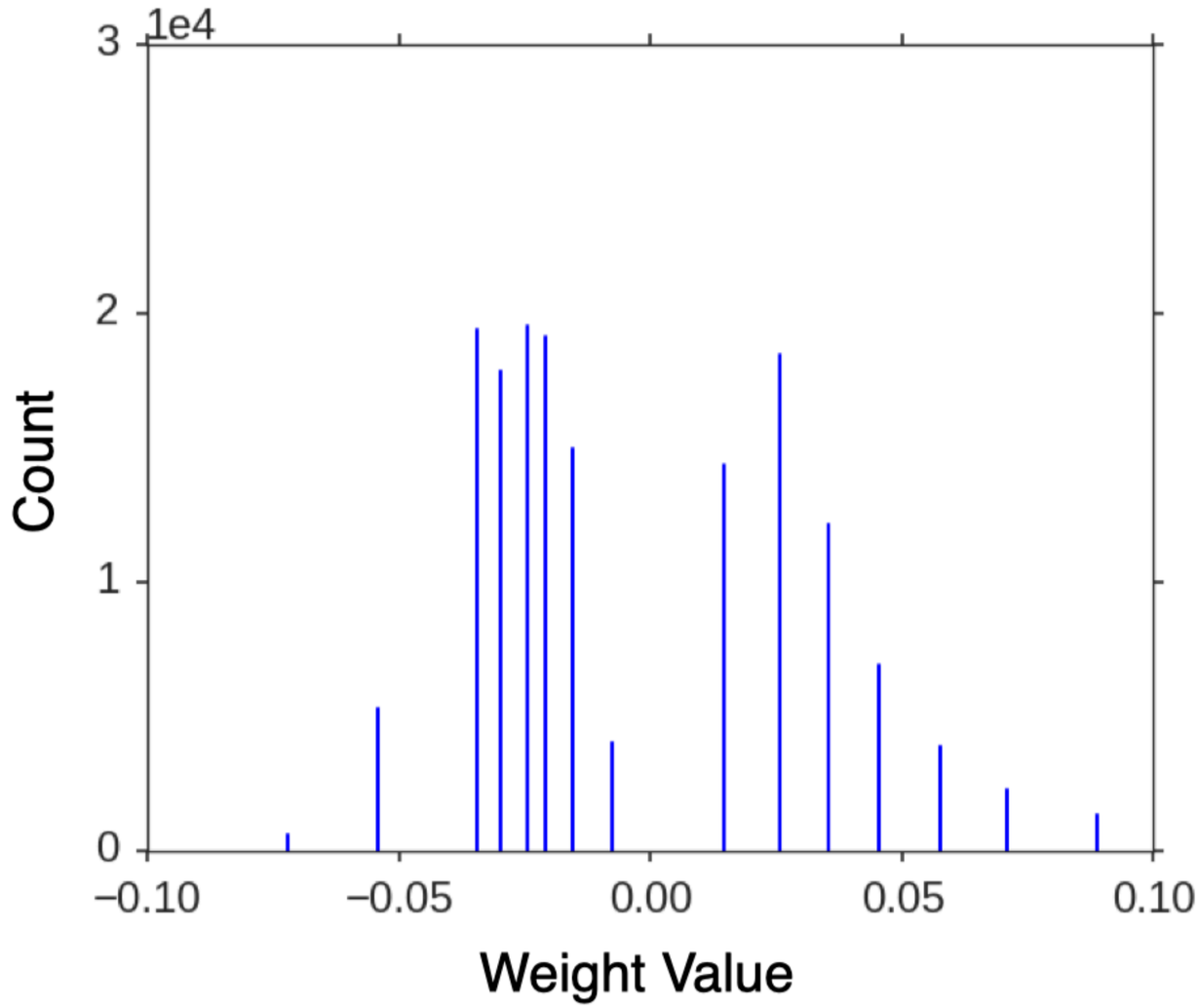
Before Quantization: Continuous Weight

After fine-tuning a pruned model



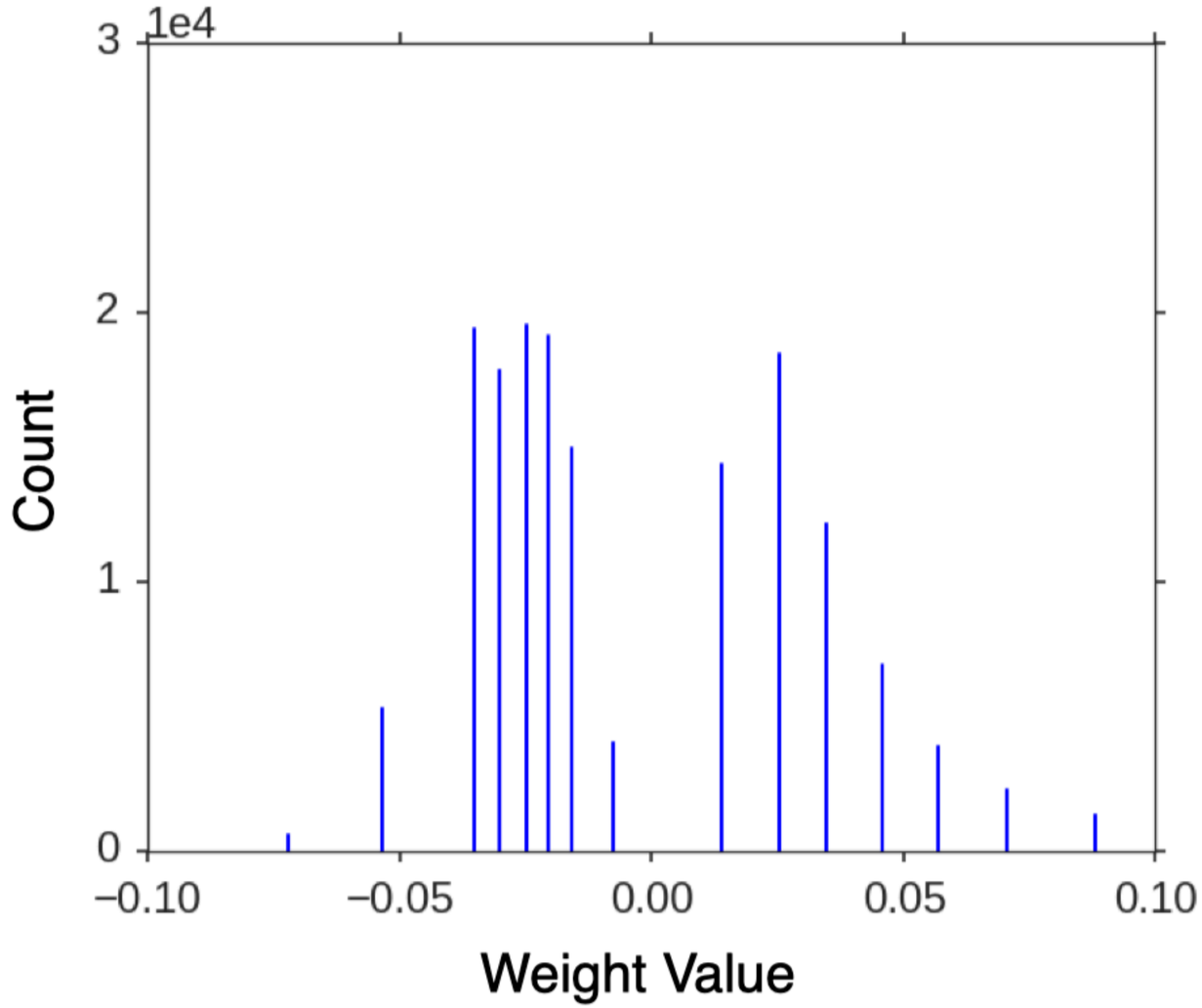
Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

After Quantization: Discrete Weight



Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

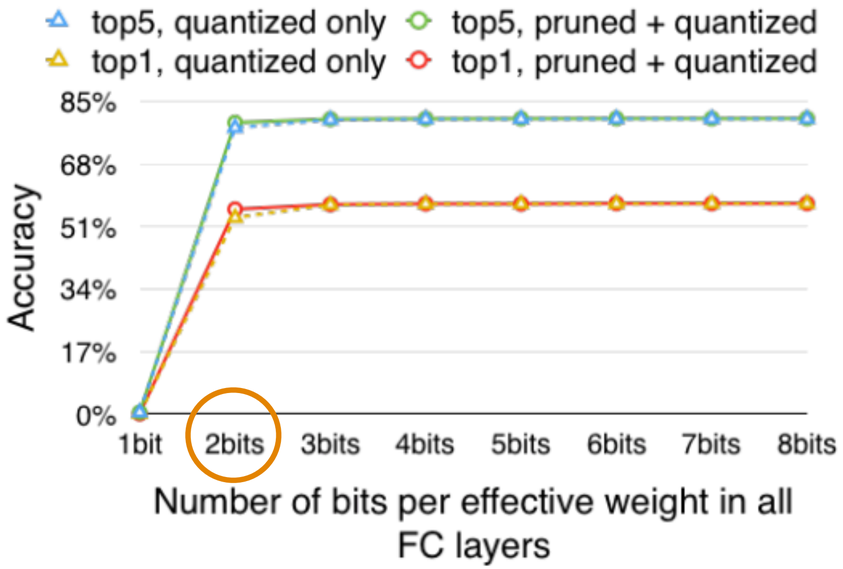
After Quantization: Discrete Weight after Retraining

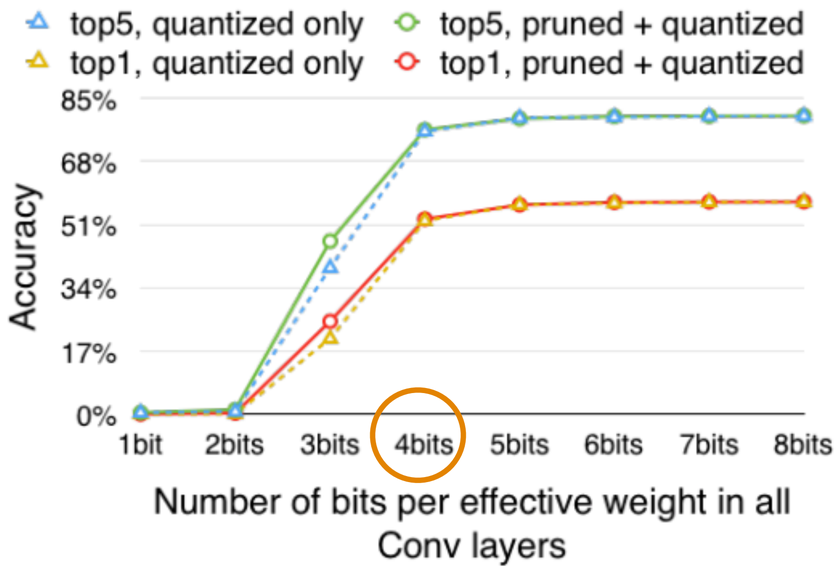


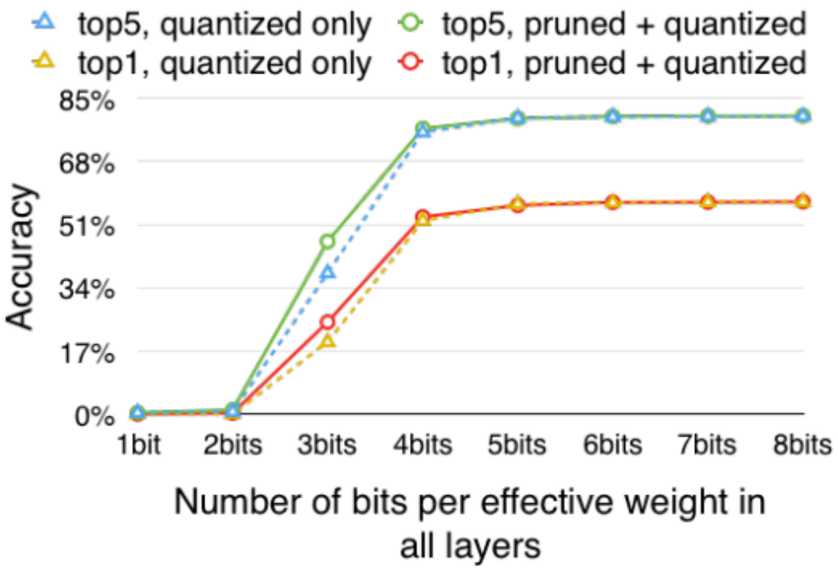
Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

How Many Bits do We Need?

AlexNet







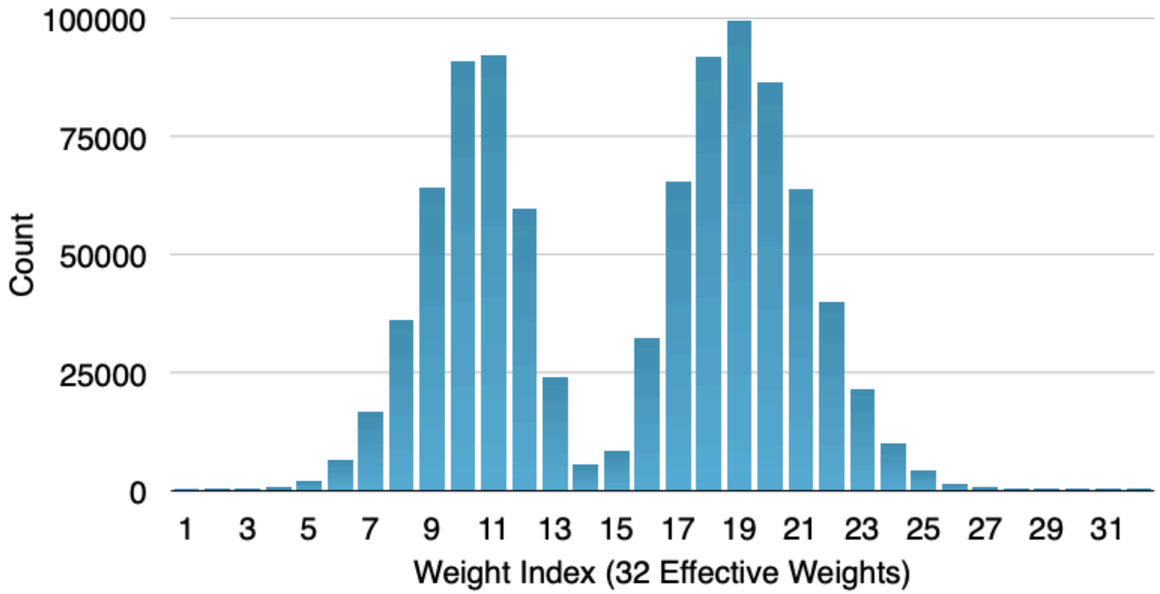
Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

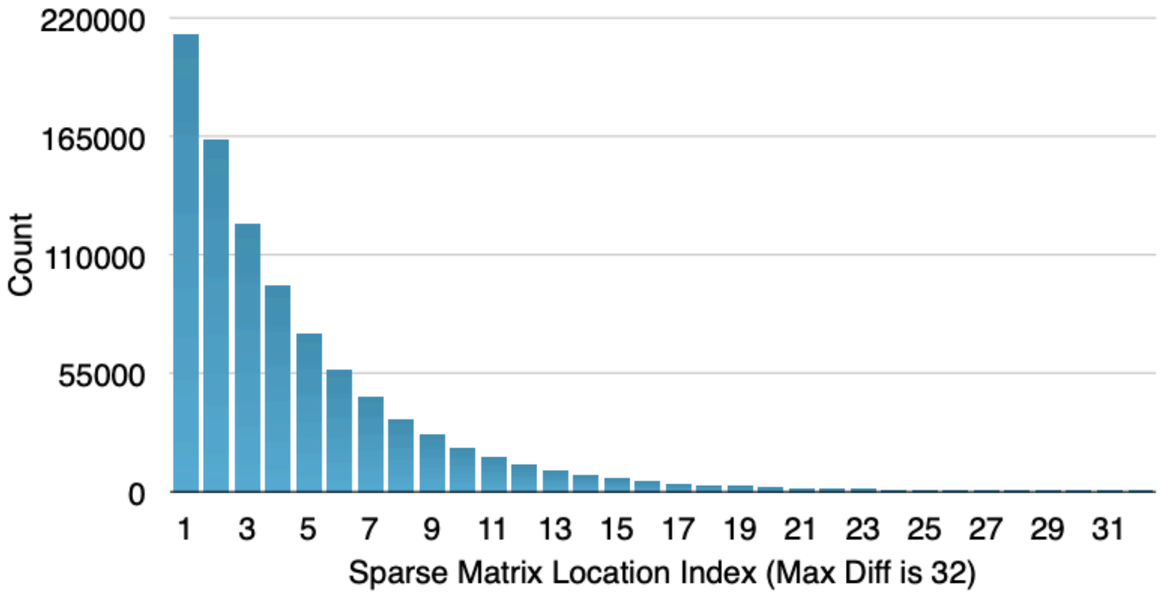
Huffman Coding

Encode Weights

Encode Index

Distribution for weight and index are biased.

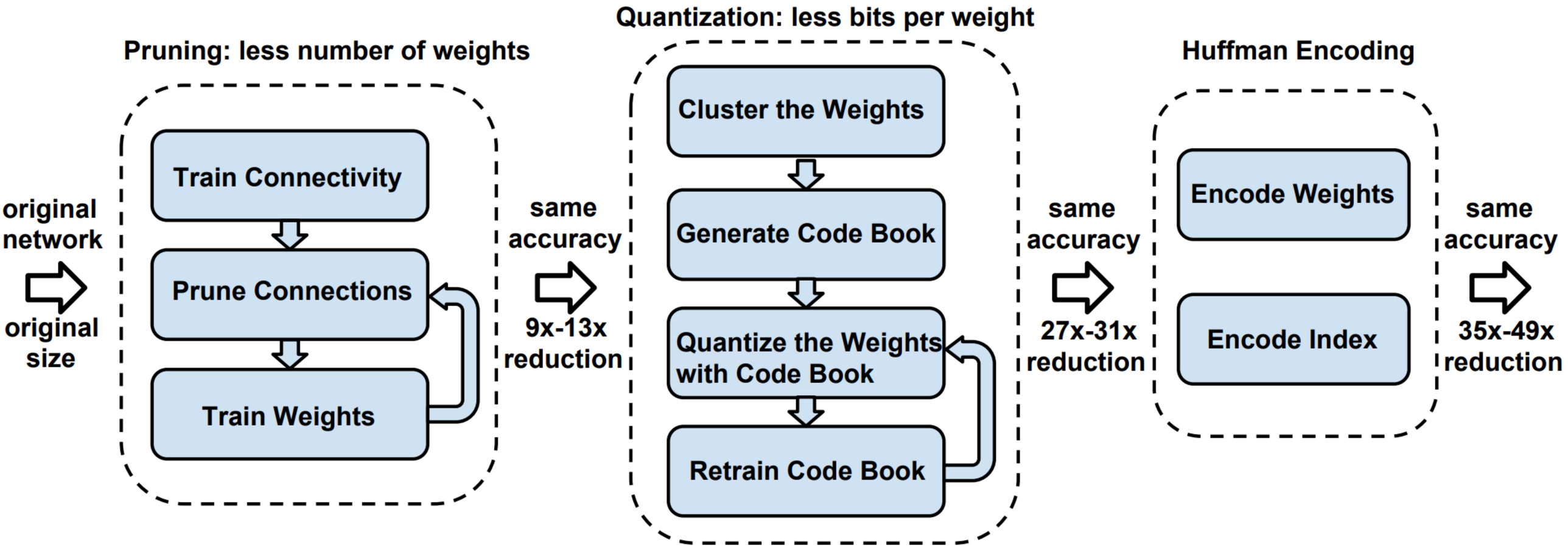




- In-frequent weights: use more bits to represent
- Frequent weights: use less bits to represent

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

Summary of Deep Compression



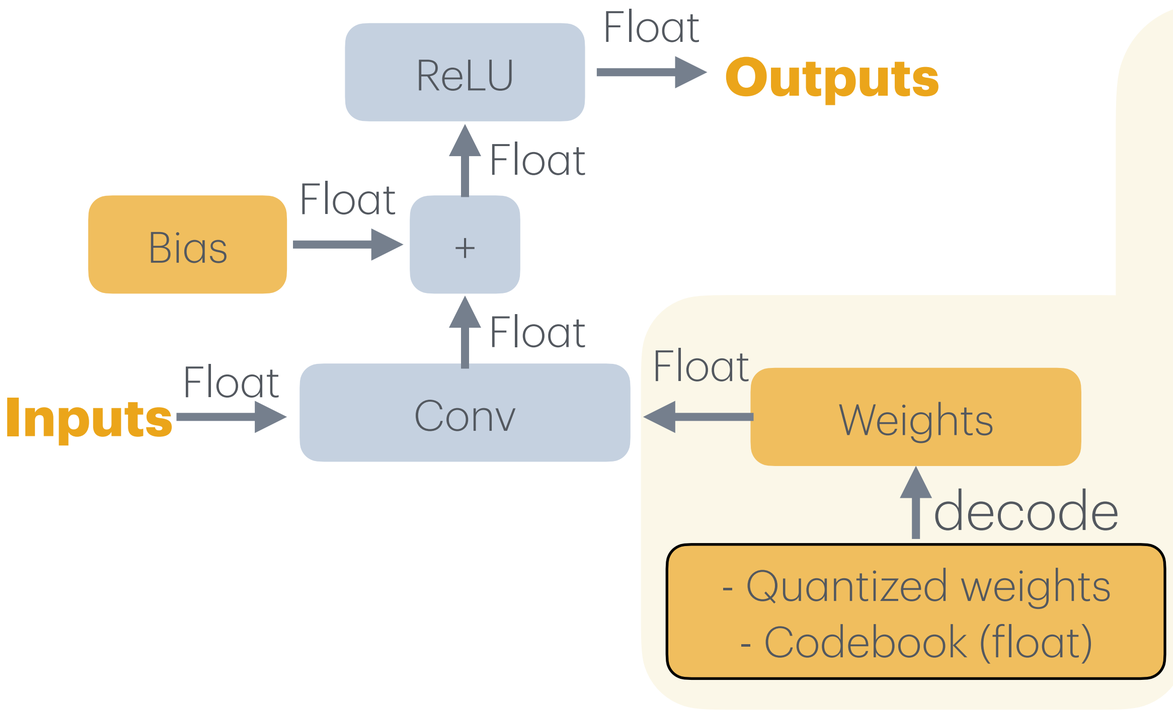
Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

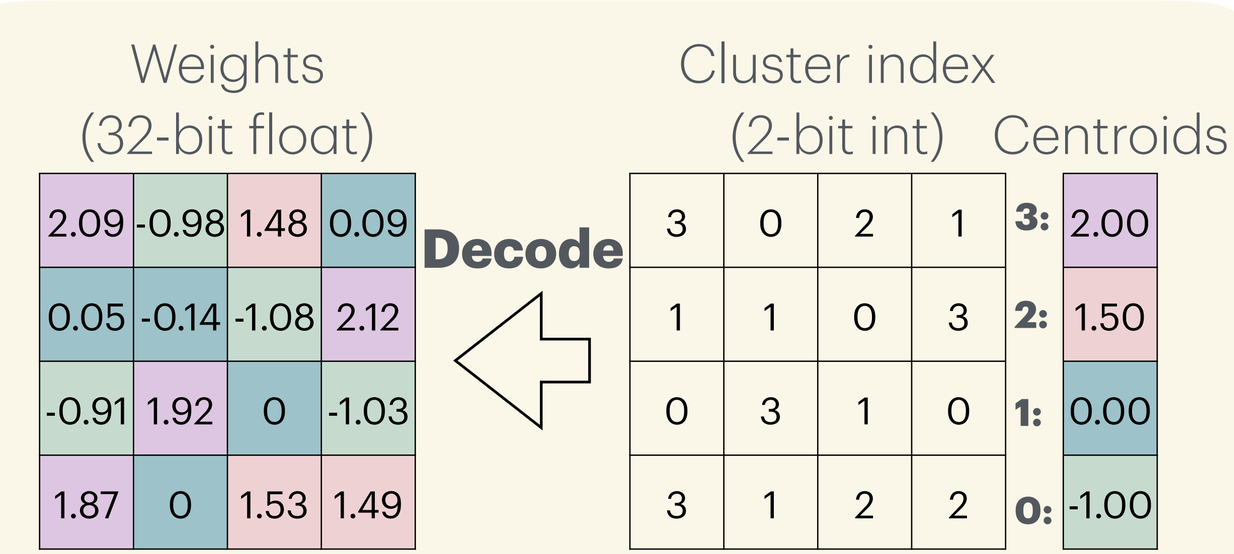
Deep Compression Results

	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%

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

K-Means-based Weight Quantization

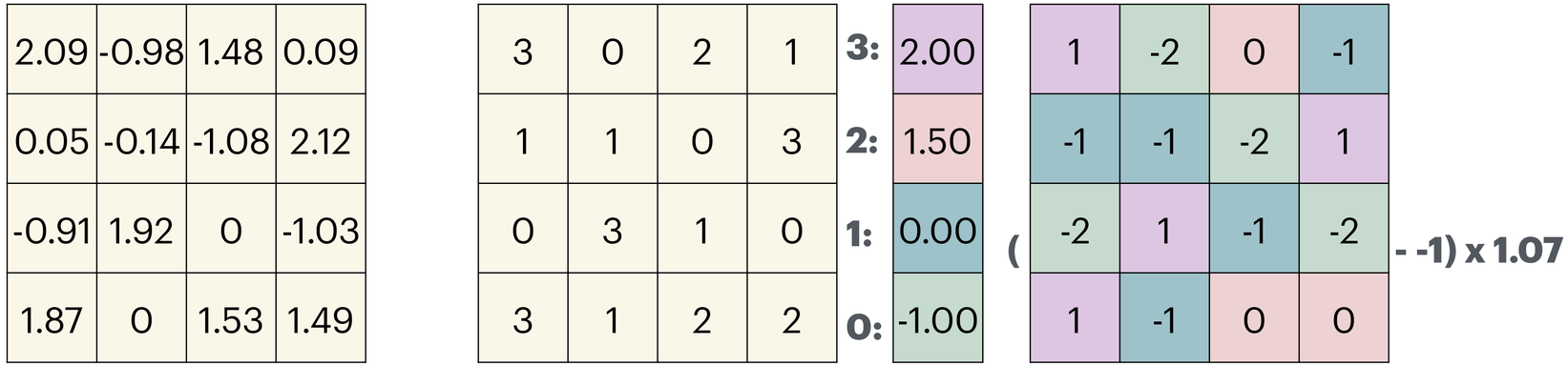




- The weights are decompressed using a **lookup table** (i.e., codebook) during runtime inference
- K-Means-based weight quantization **only saves storage cost** of a neural network model
  - All the **computation** and **memory** access are still **floating-point**

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

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

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

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)

Scale (32-bit float)

$$(\quad - \quad -1) \times 1.07 =$$

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

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

How to determine these parameters?

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

Linear Quantization

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

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

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)

Scale (32-bit float)

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

$$r = (\ q\ - \mathbf{-1}) \times \mathbf{1.07}$$

Quantization parameters

Floating-point **Integer** **Integer** **Floating-point**

Allow real number $r = 0$ be exactly representable by a quantized integer Z

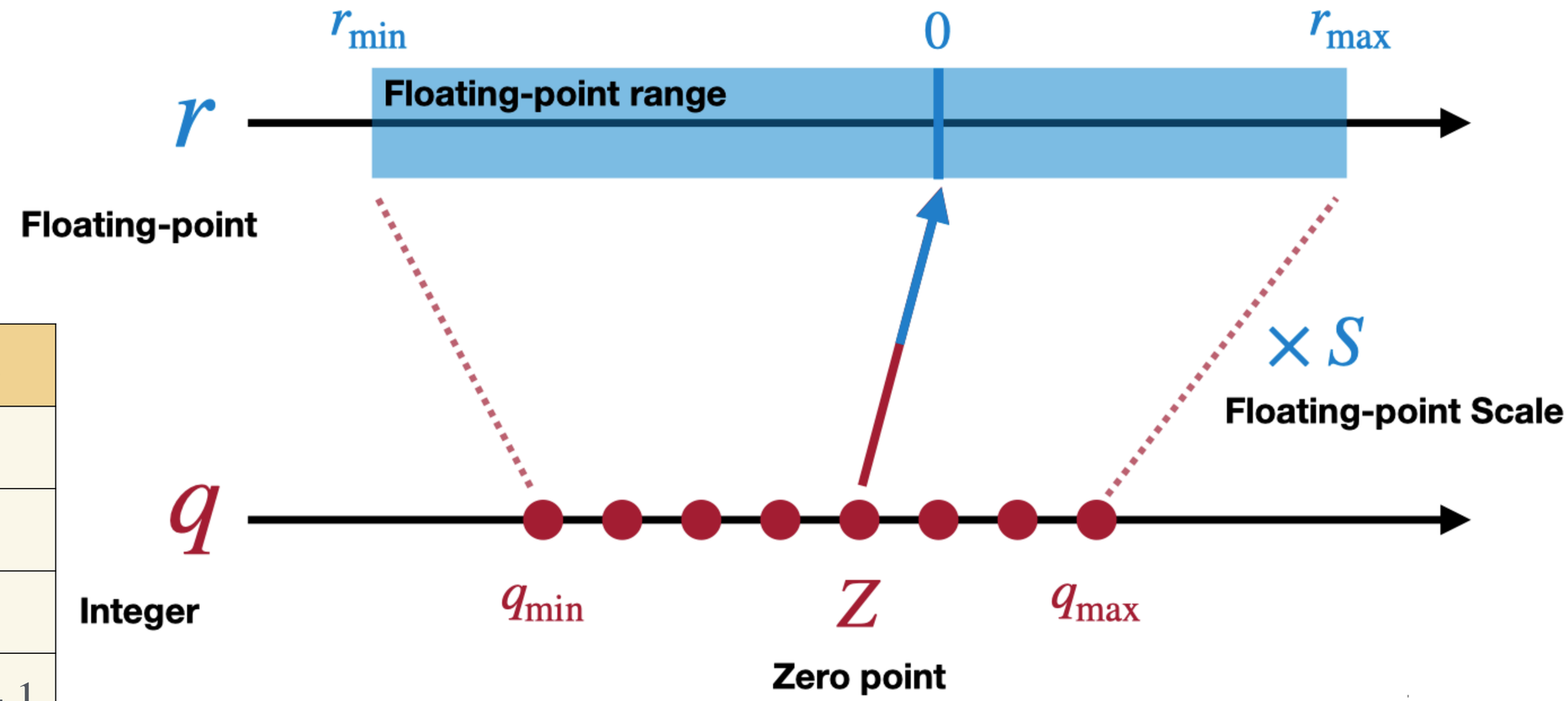
Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

Linear Quantization

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

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

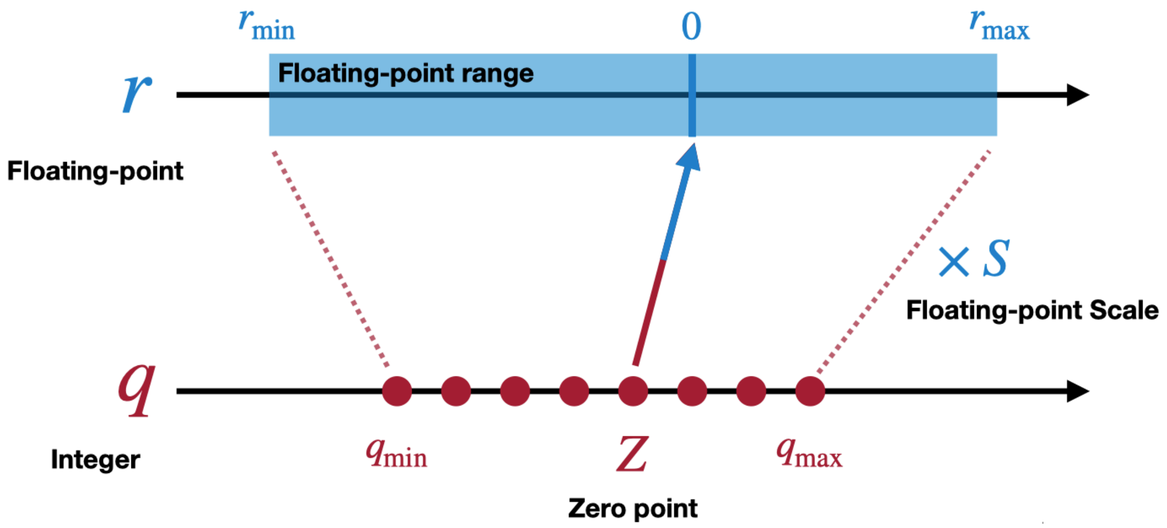
Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1} - 1$



Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

Scale of Linear Quantization

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



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

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

$$\downarrow$$

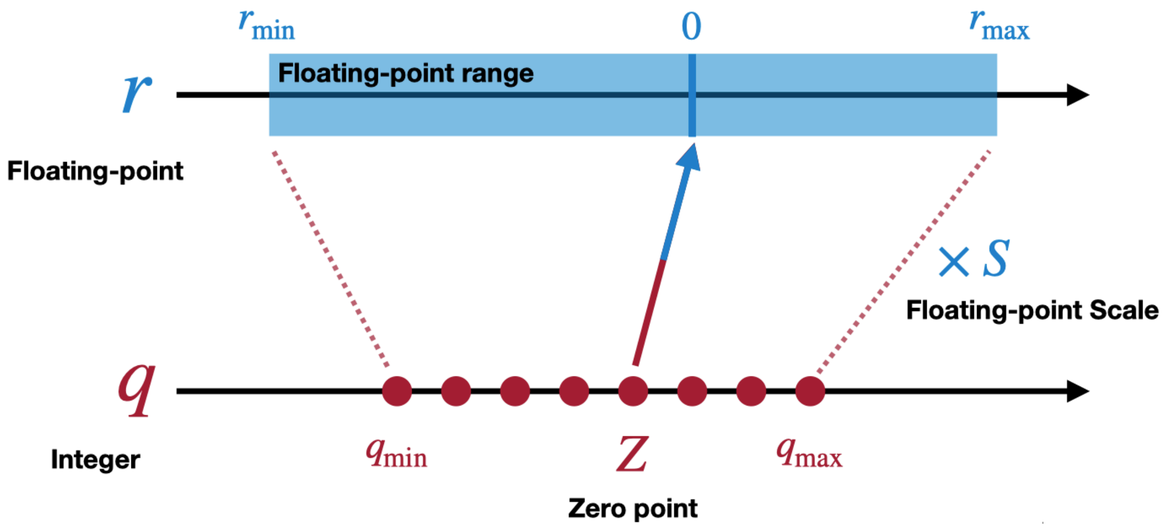
$$r_{max} - r_{min} = S(q_{max} - q_{min})$$

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

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

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

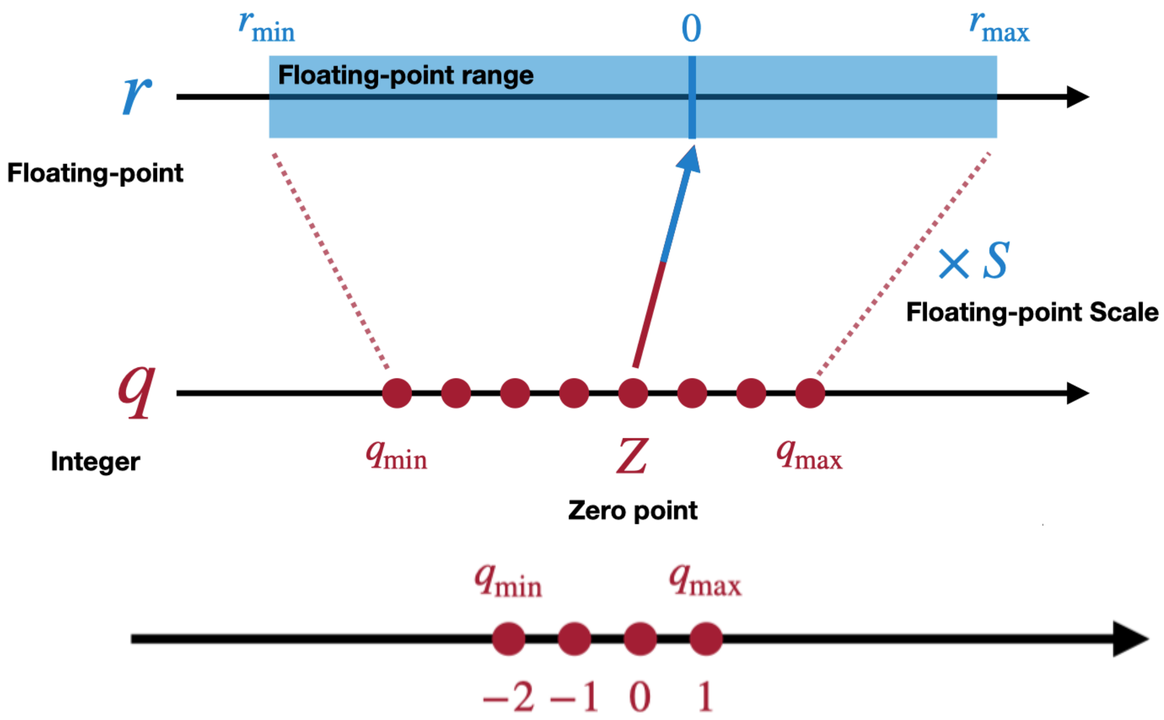
Can you calculate the scale S for the given weight when quantize to 2-bit signed int?

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

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

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

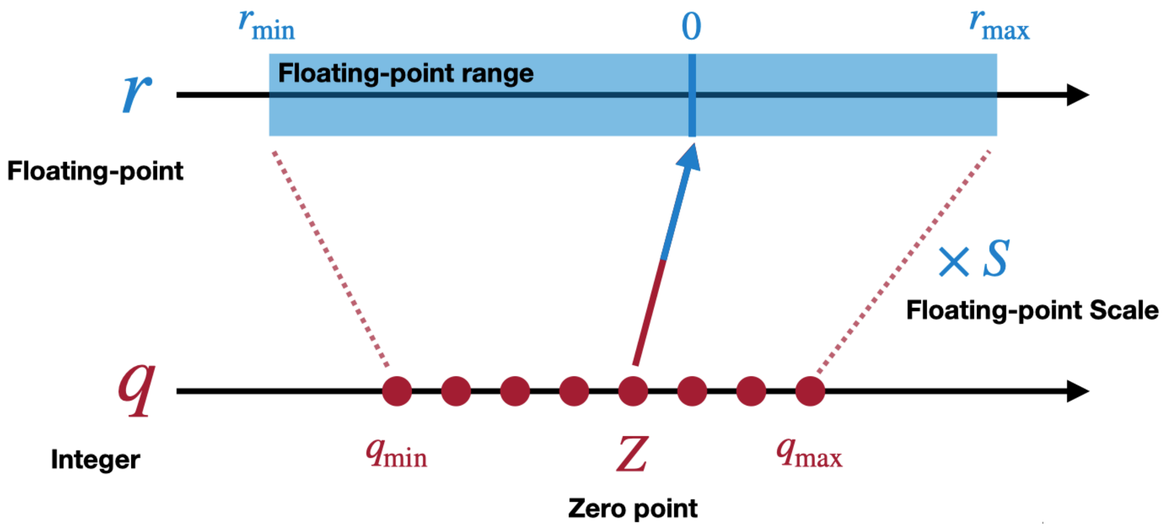
Binary	Decimal
01	1
00	0
11	-1
10	-2

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} = \frac{2.12 - (-1.08)}{1 - (-2)} = 1.07$$

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

Zero Point of Linear Quantization

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



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

$$\downarrow$$

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

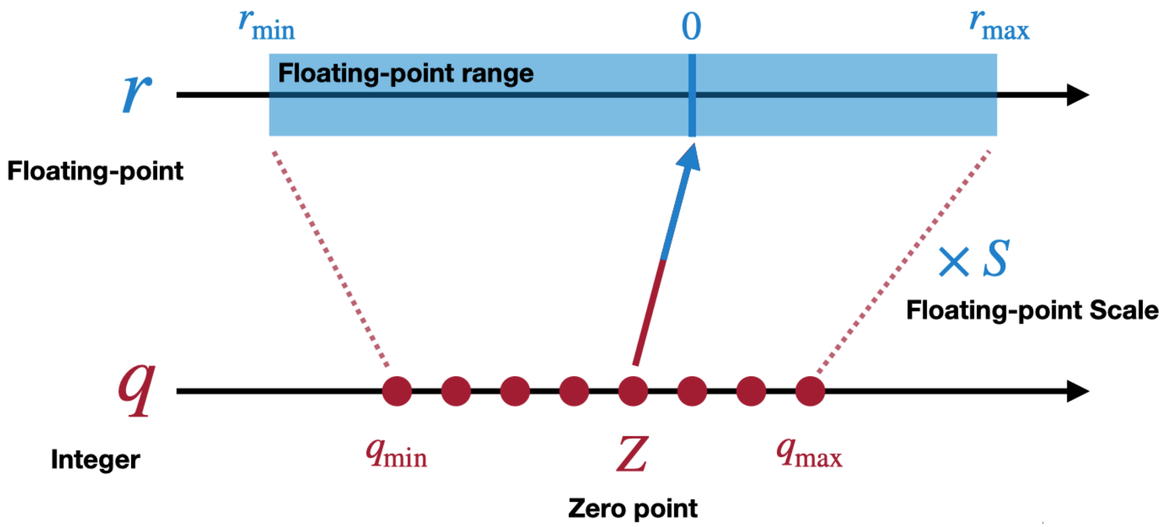
$$\downarrow$$

$$Z = \text{round}(q_{min} - \frac{r_{min}}{S})$$

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

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

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

Can you calculate the zero point Z for the given weight when quantize to 2-bit signed int?

$$Z = q_{min} - \frac{r_{min}}{S} \equiv \text{round}(q_{min} - \frac{r_{min}}{S}) = \text{round}(-2 - \frac{-1.08}{1.07}) = -1$$

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

How to use integer arithmetic to perform matrix multiplication/FC layers/Conv layers?

More math come in,
Ready?

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication, how to compute quantized Y $(i.e., q_Y)$?

$$Y = WX$$

$$S_Y(q_Y - Z_Y) = S_W(q_W - Z_W) \cdot S_X(q_X - Z_X)$$

$$q_Y = \frac{S_W S_x}{S_Y}(q_W - Z_W)(q_X - Z_X) + Z_Y$$

$$q_Y = \frac{S_W S_x}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication

$$Y = WX$$

$$q_Y = \frac{S_W S_x}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Precompute

N-bit integer multiplication
32-bit integer addition/subtraction

N-bit integer addition

- Empirically, the scale $\frac{S_W S_X}{S_Y}$ is always in the interval (0, 1).

$$\frac{S_W S_X}{S_Y} = 2^{-n} M_0, \text{where} M_0 \in [0.5, 1)$$

Fixed-point multiplication

Bit shift

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication

$$Y = WX$$

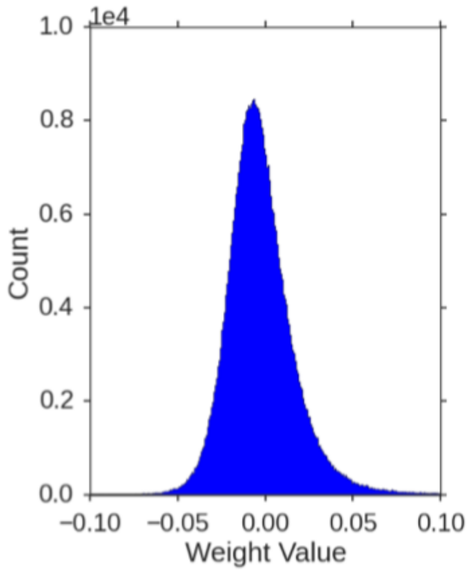
$$q_Y = \frac{S_W S_x}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Precompute

Rescale to N-bit integer

N-bit integer multiplication
32-bit integer addition/subtraction

N-bit integer addition



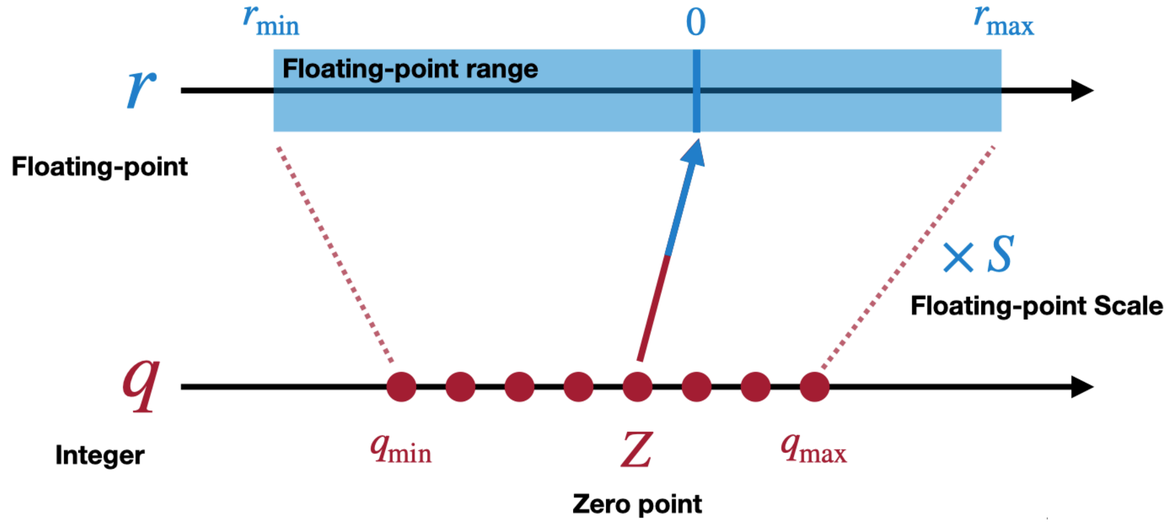
What pattern do you observe from this weight distribution?

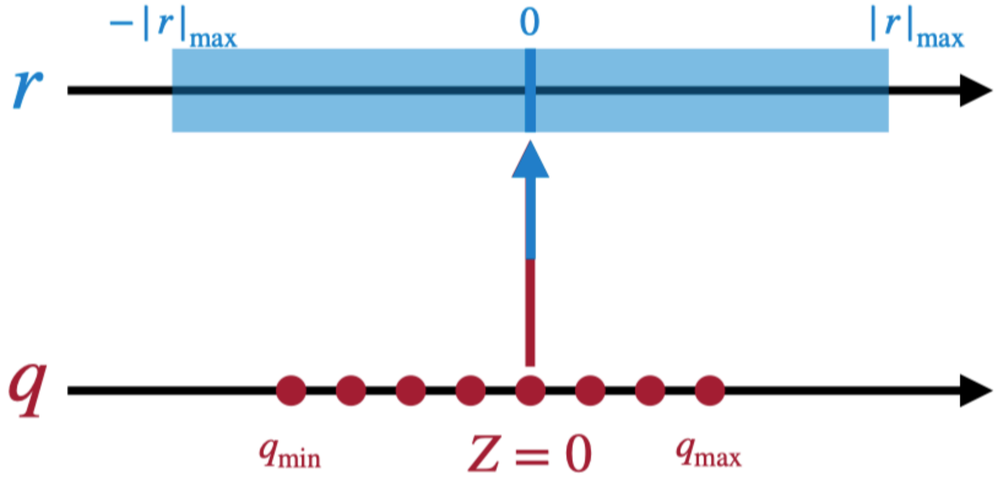
$$Z_W = 0$$

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

Symmetric Linear Quantization

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





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

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

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

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

Symmetric Linear Quantization on Weights

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

- Commly used for weight quantization
  - Weights tend to have symmetric distributions centered around zero
- Simpler and more efficient for hardware acceleration
- No need for a zero point

- Activation oeftn have a non-symmetric distritbution, and therefore use asymmetric quantization more often

Linear Quantized Matrix Multiplication

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

- Consider the following matrix multiplication

$$Y = WX$$

$$q_Y = \frac{S_W S_x}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Precompute

Rescale to N-bit integer

N-bit integer multiplication
32-bit integer addition/subtraction

N-bit integer addition

$Z_W = 0$

$$q_Y = \frac{S_W S_x}{S_Y}(q_W q_X - Z_X q_W) + Z_Y$$

Precompute

Rescale to N-bit integer

N-bit integer addition

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

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.

$$Y = WX + b$$

$$S_Y(q_Y - Z_Y) = S_W(q_W - Z_W) \cdot S_X(q_X - Z_X) + S_b(q_b - Z_b)$$

$$\downarrow Z_W = 0$$

$$S_Y(q_Y - Z_Y) = S_W S_X(q_W q_X - Z_X q_W) + S_b(q_b - Z_b)$$

$$\downarrow Z_b = 0,\ S_b = S_W S_X$$

$$S_Y(q_Y - Z_Y) = S_W S_X(q_W q_X - Z_X q_W + q_b)$$

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.

$$Y = WX + b$$

$$\downarrow Z_W = 0 \quad \downarrow Z_b = 0,\ S_b = S_W S_X$$

$$S_Y(q_Y - Z_Y) = S_W S_X(q_W q_X - Z_X q_W + q_b)$$

$$\downarrow$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X + \underbrace{q_b - Z_X q_W}_{\text{Precompute}}) + Z_Y$$

$$\downarrow q_{bias} = q_b - Z_X q_W$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X + q_{bias}) + Z_Y$$

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.

$$Y = WX + b$$

$$Z_W = 0$$

$$Z_b = 0,\ S_b = S_W S_X$$

$$q_{bias} = q_b - Z_X q_W$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X + q_{bias}) + Z_Y$$

Rescale to N-bit int

N-bit int Mult
32-bit int Add

N-bit int add

Note: both q_b and 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

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

$$Z_W = 0$$

$$Z_b = 0,\ S_b = S_W S_X$$

$$q_{bias} = q_b - \textbf{Conv}(q_W, Z_X)$$

$$q_Y = \frac{S_W S_X}{S_Y}\left(\text{Conv}(q_W, q_X) + q_{bias}\right) + Z_Y$$

Rescale to N-bit int

N-bit int Mult
32-bit int Add

N-bit int add

Note: both q_b and 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

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

$$Z_W = 0$$

$$Z_b = 0,\ S_b = S_W S_X$$

$$q_{bias} = q_b - \textbf{Conv}(q_W, Z_X)$$

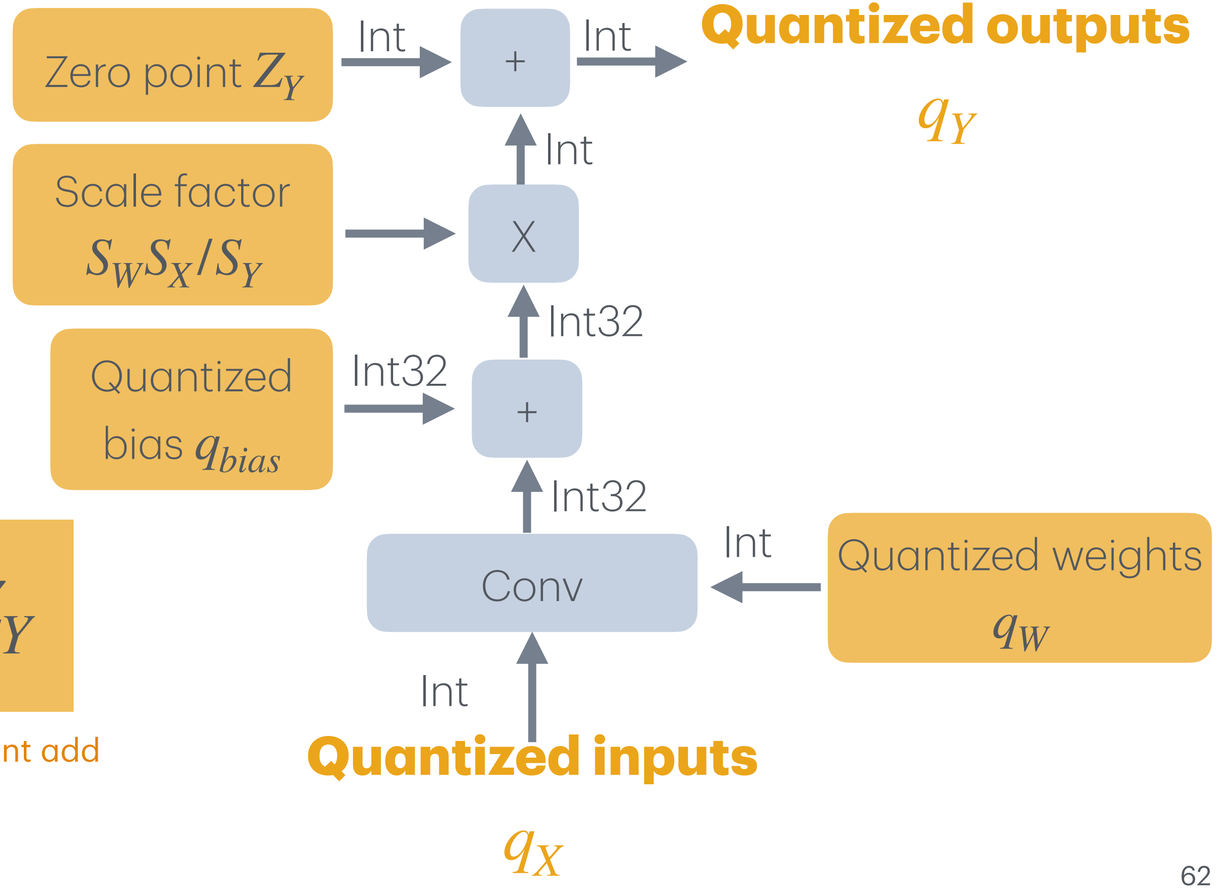
$$q_Y = \frac{S_W S_X}{S_Y}(\text{Conv}(q_W, q_X) + q_{bias}) + Z_Y$$

Rescale to N-bit int

N-bit int Mult
32-bit int Add

N-bit int add

Note: both q_b and q_{bias} are 32 bits

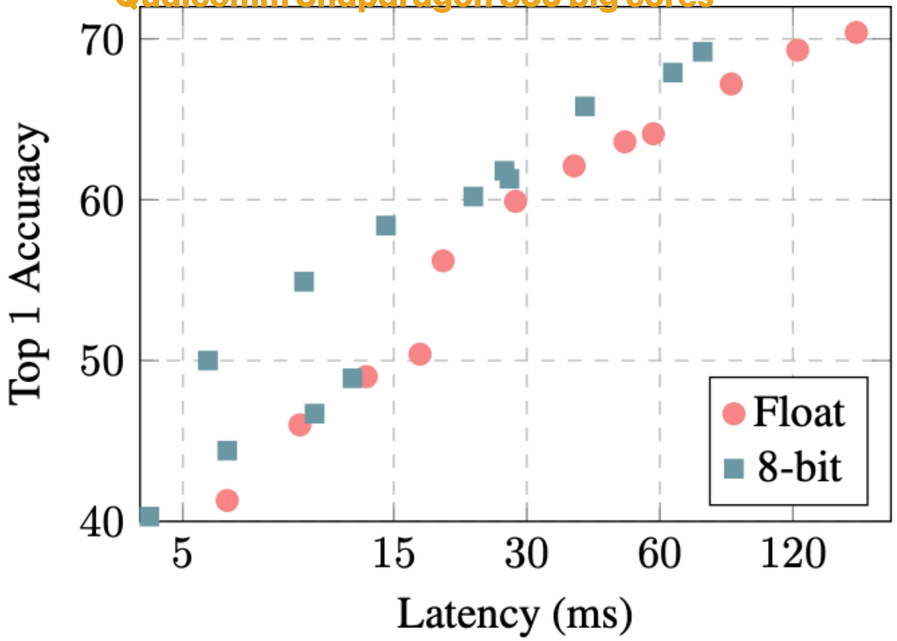


INT8 Linear Quantization

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 accuracy	74.9%	75.4%

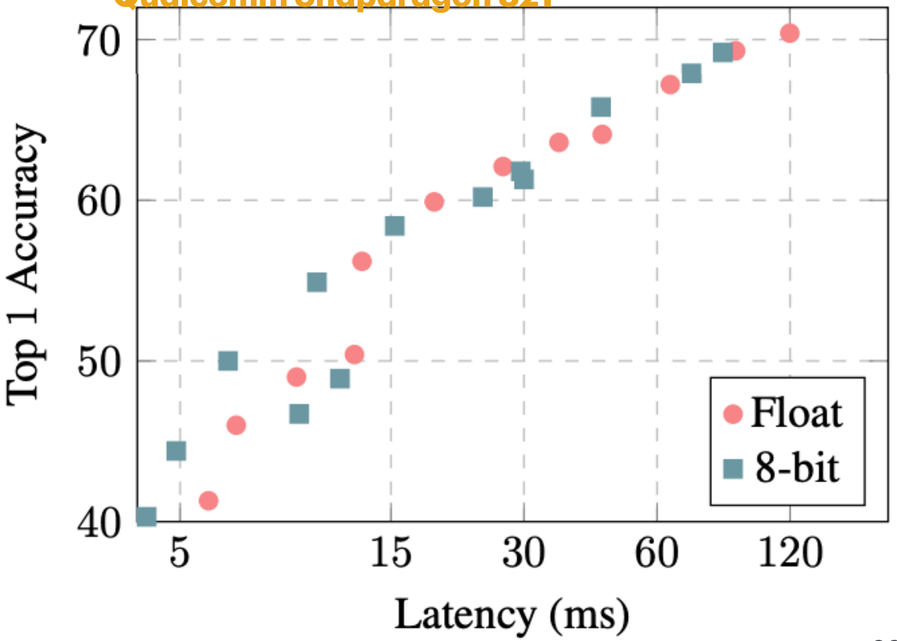
Qualcomm Snapdragon 835 big cores



Latency-vs-accuracy tradeoff of floating-point and integer-only MobileNets

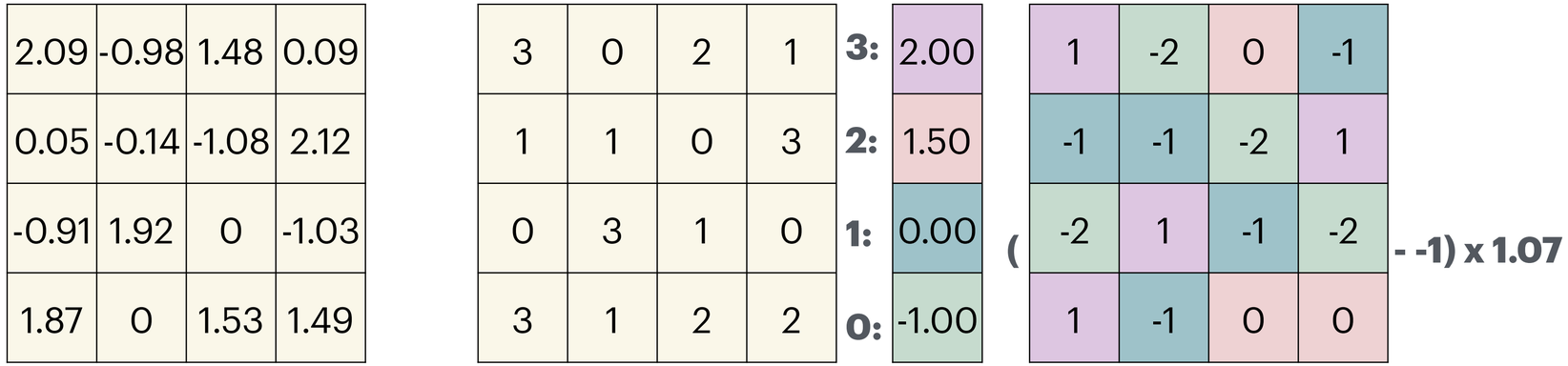
Floating-point computation is better optimized in the Snapdragon 821

Qualcomm Snapdragon 821



Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

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

Reference

- Horowitz, M. (2014, February). 1.1 computing's energy problem (and what we can do about it). In 2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC) (pp. 10-14). IEEE.

- Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

- Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

- Quantization [MIT 6.5940]