Accelerator Architectures for Machine Learning (AAML)

Lecture 3: Quantization

Tsung Tai Yeh
Department of Computer Science
National Yang-Ming Chiao Tung University

Acknowledgements and Disclaimer

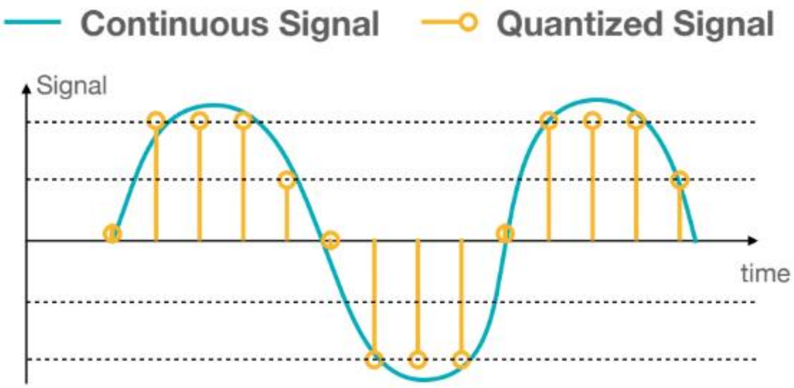
- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020
- 6.5940, TinyML and Efficient Deep Learning Computing, MIT
- NVIDIA, Precision and performance: Floating point and IEEE 754 Compliance for NVIDIA GPUs, TB-06711-001_v8.0, 2017

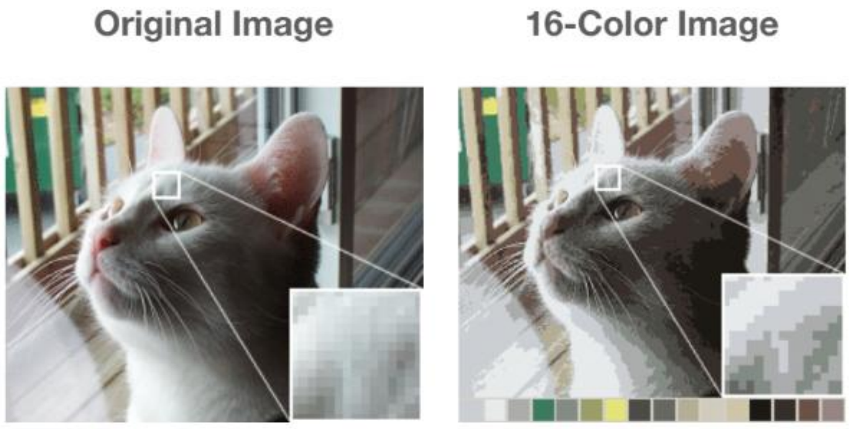
Outline

- K-Means-based Quantization
- Linear Quantization
- Binary and Ternary Quantization

What is Quantization ?

- **Quantization**
  - A process that reduces the precision of a digital signal by converting high-precision data into a lower-precision format

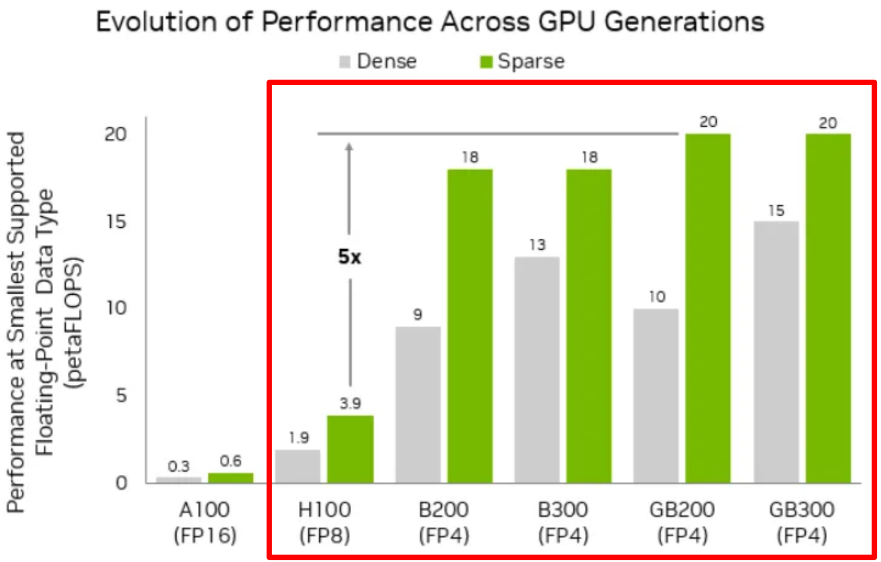




Images are in the public domain.

Benefits of Quantization

- **Reduced memory burden**
  - Reduce pressure on memory bandwidth which can improve output token throughput
- **Simplified compute operations**
  - Improve overall end-to-end latency performance as a result of simplified attention layer computations



https://reurl.cc/gYok94

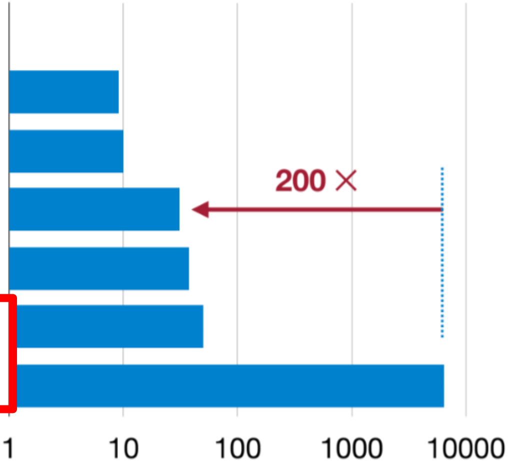
Memory is Expensive !!

- **Data movement -> Move memory reference -> More energy**

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V

Relative Energy Cost



This image is in the public domain

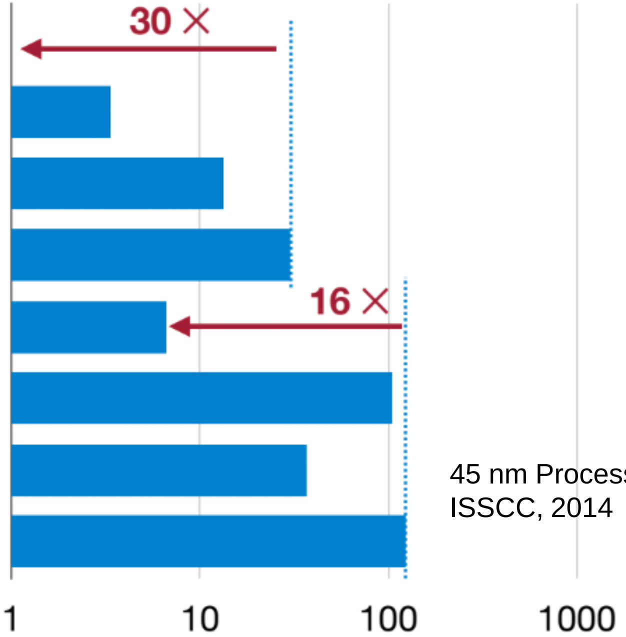
Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]

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
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



45 nm Process, Horowitz, ISSCC, 2014

Energy and Area Cost

Could we make the deep learning efficient by lowering the precision of data ?

Operation	Energy (pJ)	Area(um^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	
32b DRAM Read	640	

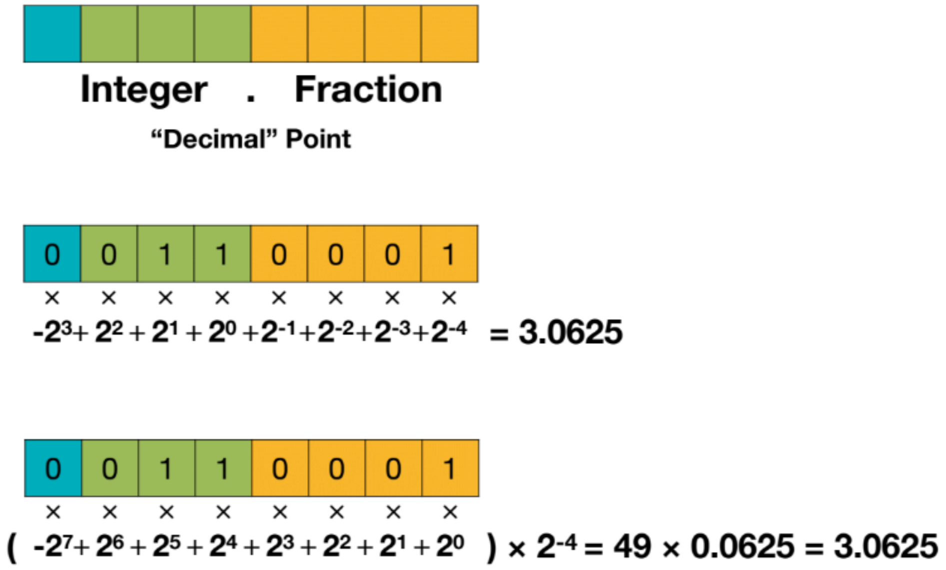
4.7X

173X

45 nm Process, Horowitz, ISSCC, 2014

Numeric Data Types

- Fixed-point number



(using 2's complement representation)

IEEE 765 Single Precision Float Point

- **Sign** determines the sign of the number
- **Exponent** (8 bit) represent -127 (all 0s) and +128 (all 1s)
- **Significand** (23 fraction bits), total precision is 24 bits (23 + 1 implicit leading bit) $\log_{10}(2^{24}) \approx 7.225$ digital bit

Sign | Exponent (8 bits) | Mantissa/Fraction (23 bits)

0	0	1	1	1	1	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

$$value = (-1)^{sign} \times 2^{(e-127)} \times \left(1 + \sum_{i=1}^{23} b_{(23-i)} 2^{-i}\right)$$

IEEE 765 FP32

Sign | Exponent (8 bits) | Mantissa/Fraction (23 bits)

0	0	1	1	1	1	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

$$value = (-1)^{sign} \times 2^{(e-127)} \times (1 + \sum_{i=1}^{23} b_{(23-i)} 2^{-i})$$

Sign = b31 = 0 ; $(-1)^0 = 1$

e =120; $2^{(120 - 127)} = 2^{-7}$

$$1.b_{22}b_{21}...b_0 = (1 + \sum_{i=1}^{23} b_{(23-i)} 2^{-i}) = 1 + 2^{-2} = 1.25$$

Value = 1 x 2^{-7} x 1.25 = 0.009765625

Numeric Data Type

- **Question**: What is the decimal **“11.375”** in FP32 format ?

11.375
$= 11 + 0.375$
$= (1011)_2 + (0.011)_2$
$= (1011.011)_2$
$= (1.011011)_2 \times 2^3$

$0.375 \times 2 = 0.750 = 0 + 0.750 \Rightarrow b_{-1} = 0$
$0.750 \times 2 = 1.500 = 1 + 0.500 \Rightarrow b_{-2} = 1$
$0.500 \times 2 = 1.000 = 1 + 0.000 \Rightarrow b_{-3} = 1$

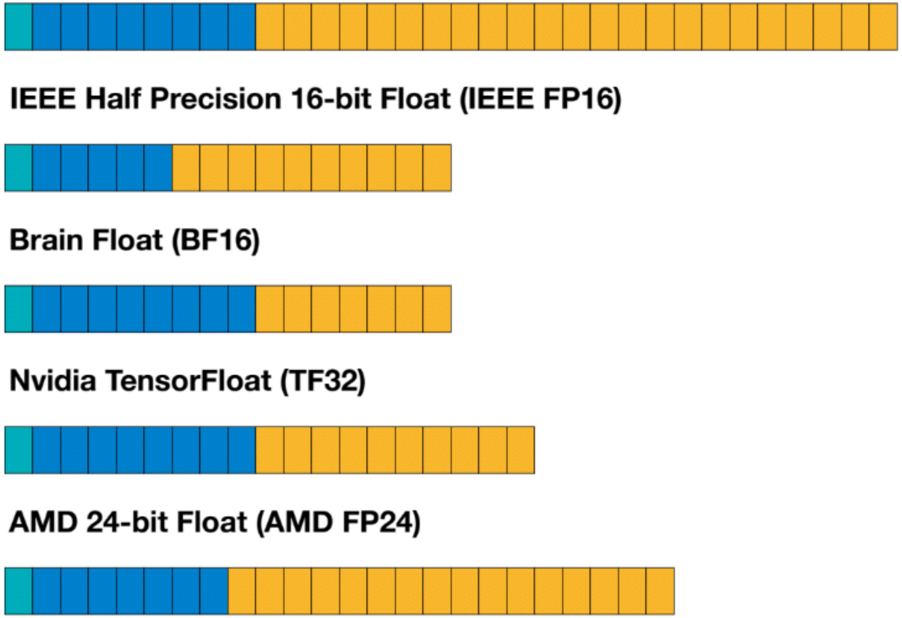
- The exponent is 3 and biased form
  = (3 + 127) = 130 = 1000 0010

Sign Exponent (8 bits) Mantissa/Fraction (23 bits)

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

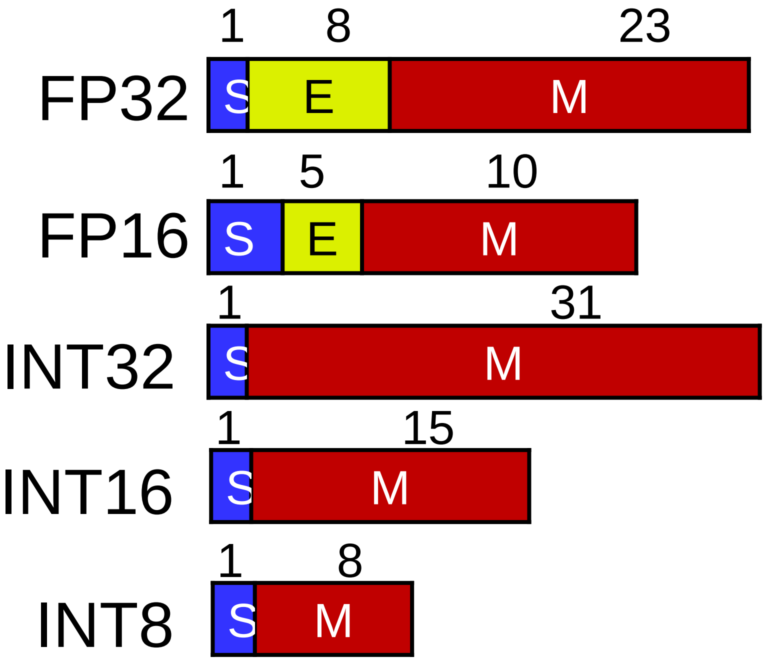
Floating-Point Number

- **Exponent Width -> Range; Fraction Width-> Precision**



Format	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE 754 Single Precision 32-bit Float (IEEE FP32)	8	23	32
IEEE Half Precision 16-bit Float (IEEE FP16)	5	10	16
Brain Float (BF16)	8	7	16
Nvidia TensorFloat (TF32)	8	10	19
AMD 24-bit Float (AMD FP24)	7	16	24

Number Representation



Range

1.2E-38 to 3.4E+38

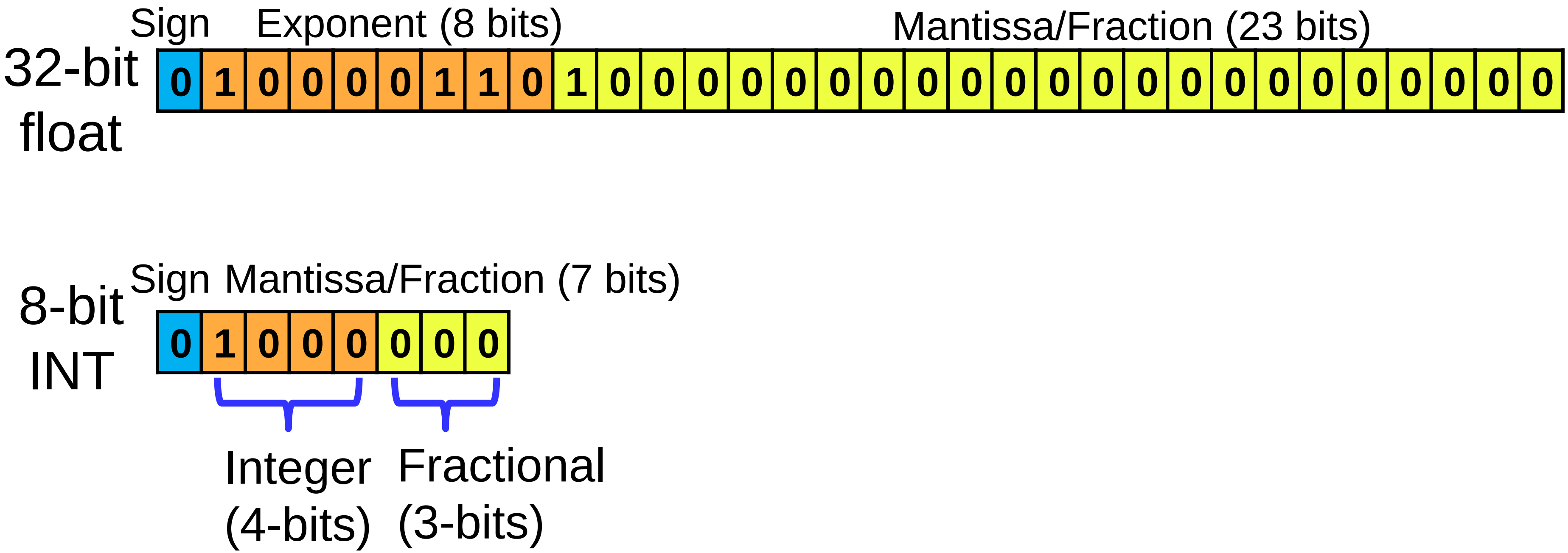
6.1E-5 to 6.6E+4

2147483648 to 2147483647

−32,768 to 32,767

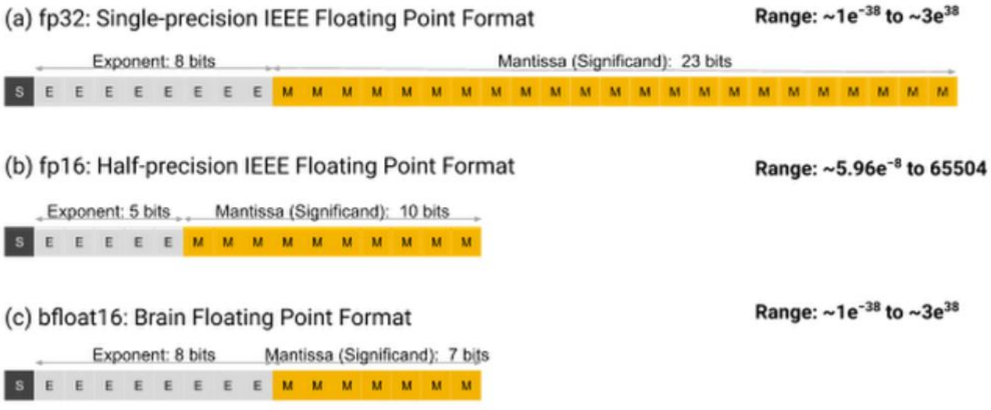
-128 ~ 127

Reduced Bit Width



FP32 vs FP16 vs BF16

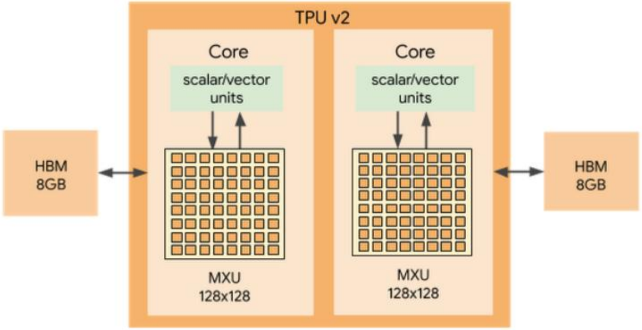
- **FP32 – single precision**
  - With 6-9 significant decimal digits precision
- **FP16 – half precision**
  - Uses in some neural network applications
  - With 4 significant decimal digits precision
- **BF16**
  - A truncated FP32
  - Allow for fast conversion to and from an FP32
  - With 3 significant decimal digits



Format	Bits	Exponent	Fraction
FP32	32	8	23
FP16	16	5	10
BF16	16	8	7

https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

Choosing bFloat16

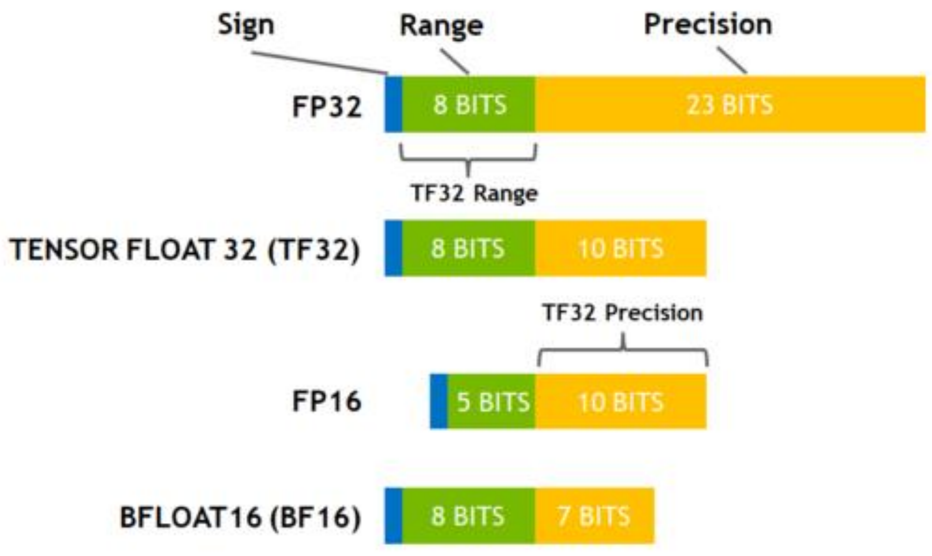


- **Motivation**
  - The physical size of a hardware multiplier scales with the square of the mantissa width
  - Mantissa bit length – FP32-> 23 bits, FP16-> 10 bits, BF16:->7 bits
- **BF16**
  - 8 X smaller than an FP32 multiplier
  - <u>Has the same exponent size as FP32</u>
  - No require special handling (loss scaling) in the FP16 conversion
  - XLA compiler's automatic format conversion
  - In side the MXU, multiplications are performed in BF16 format
  - Accumulations are performed in full FP32 precision

https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

Nvidia's TF32

- **Nvidia's TF32**
  - 19-bit (BF19)
  - 1-bit sign, 8-bit exponent
    10-bit fraction
  - Fuse BF16 and FP16
    - BF16: 8-bit exponent +
    - FP16: 10-bit fraction
  - Nvidia A100 Tensor Core
    - TF32: 156 TFLOPS
    - FP16/BF16: 312 TFLOPS



https://reurl.cc/Omo1dv

FP8 and Tesla CFloat

- **FP8 (1-5-2)**
  - Large loss in MobileNet v2
  - Hybrid FP8 (HFP8)
    - Use FP(1-4-3) in forward
    - Use FP(1-5-2) in backward
- **Tesla Dojo Cfloat (configurable float)**
  - Configurable exponent and mantissa
  - Use software to choose appropriate Cfloat format
    - CF16
    - CF8 (1-4-3), CF8 (1-5-2)

c. Trans-precision Inference Accuracy of FP32 models in FP8 1-5-2 precision

FP32 Model	Baseline	FP8 1-5-2
MobileNet_v2 ImageNet	71.81	**52.51**
ResNet50 ImageNet	76.44	75.31
DensetNet121 ImageNet	74.76	73.64
MaskRCNN COCO†	33.58 29.27	32.83 28.65

† Box and Mask average precision

https://proceedings.neurips.cc/paper/2019/file/65fc9fb4897a89789352e211ca2d398f-Paper.pdf

Nvidia's NVFP4

- **Nvidia's NVFP4**
  - 1 sign bit, 2 exponent bits, and 1 mantissa bit (E2M1)
  - The value in the format ranges approximately -6 to 6
  - The values in the range could include 0.0, 0.5, 1.0, 1.5, 2, 3, 4, 6 (same for the negative range)

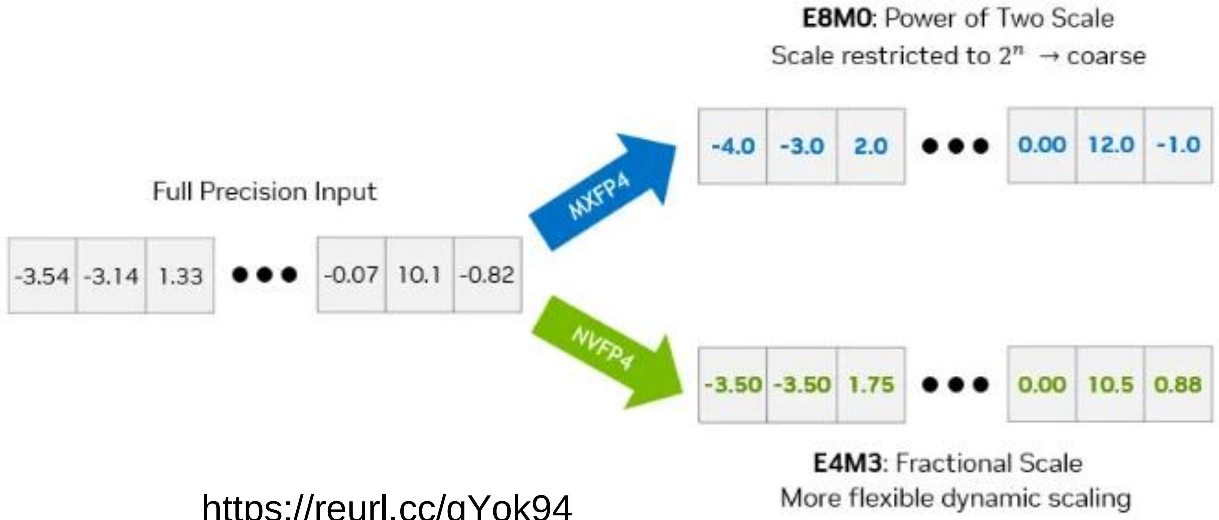
Table 1: Numbers represented in FP4-E2M1 with NaN and Inf (IEEE 754 standard) and Numbers represented in FP4-E2M1 without NaN and Inf (Our design).

UINT4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FP4 w/ NaN&Inf	0	0.5	1	1.5	2	3	Inf	NaN	-0	-0.5	-1	-1.5	-2	-3	Inf	NaN
FP4 w/o NaN&Inf	0	0.5	1	1.5	2	3	4	6	-0	-0.5	-1	-1.5	-2	-3	-4	-6

https://reurl.cc/gYok94

Nvidia’s NVFP4

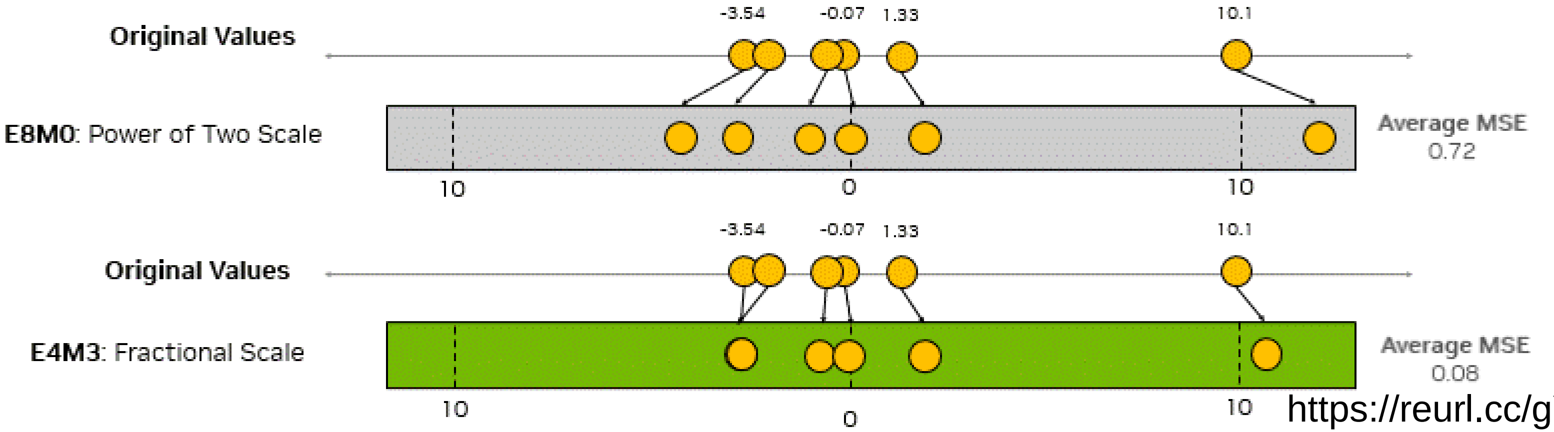
- **High-precision scaling**
  - NVFP4 encodes blocks using E4M3 FP8 precision
  - Enables non-power-of-two scaling factors with fractional precision



https://reurl.cc/gYok94

Nvidia's NVFP4

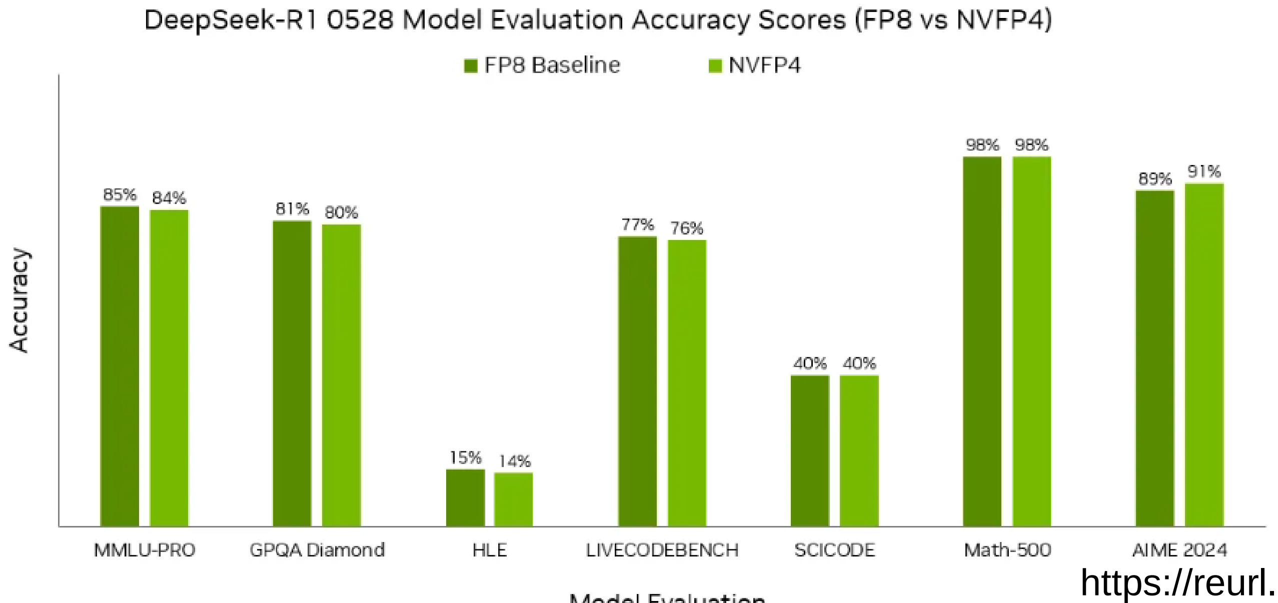
- **High-precision scaling**
  - E8M0 = Snaps the scale factor to nearest 2^n
  - E4M3 = Finds one scale factor that makes the block errors collectively as small as possible



https://reurl.cc/gYok94

Nvidia’s NVFP4

- **Quantize model weights to 4-bits**
  - The analysis showcases the 1% or less accuracy degradation



https://reurl.cc/gYok94

How to Determine Bit Width on DNN ?

- For accuracy, DNN operations decide bit width to achieve sufficient precision
- Which DNN operations affect the accuracy ?
  - **For inference:** weights, activations, and partial sums
  - **For training:** weights, activations, partial sums, gradients, and weight update
    - post-training quantization (PTQ)
      - A model compression technique that converts a pre-trained, full-precision model into a lower-precision model without needing to retrain or fine-tune it

Takeaway Questions

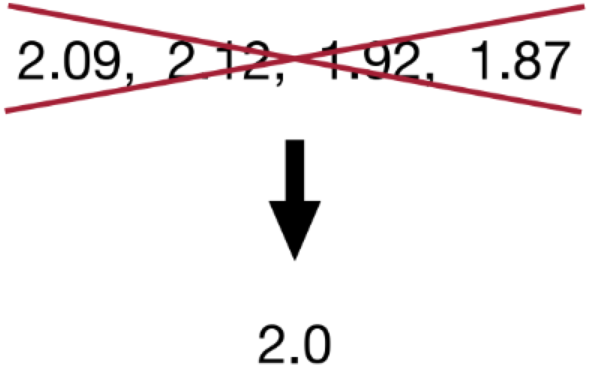
- What are advantages to use BF16 instead of FP16 ?
    - (A) Fast conversion from FP32
    - (B) Get more precise value
    - (C) Represent few different values
- What are benefits to use lower precision data type on neural network ?
    - (A) Reduce the latency of DNN models
    - (B) Save the memory space
    - (C) Lower the power consumption of the accelerator

K-Means-based Weight Quantization

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

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*



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

- weights: 32 bit × 16 = 512 bit = 64 B
- indexes: 2 bit × 16 = 32 bit = 4 B
- codebook: + 32 bit × 4 = 128 bit = 16 B
- total: = 20 B

3.2 × smaller

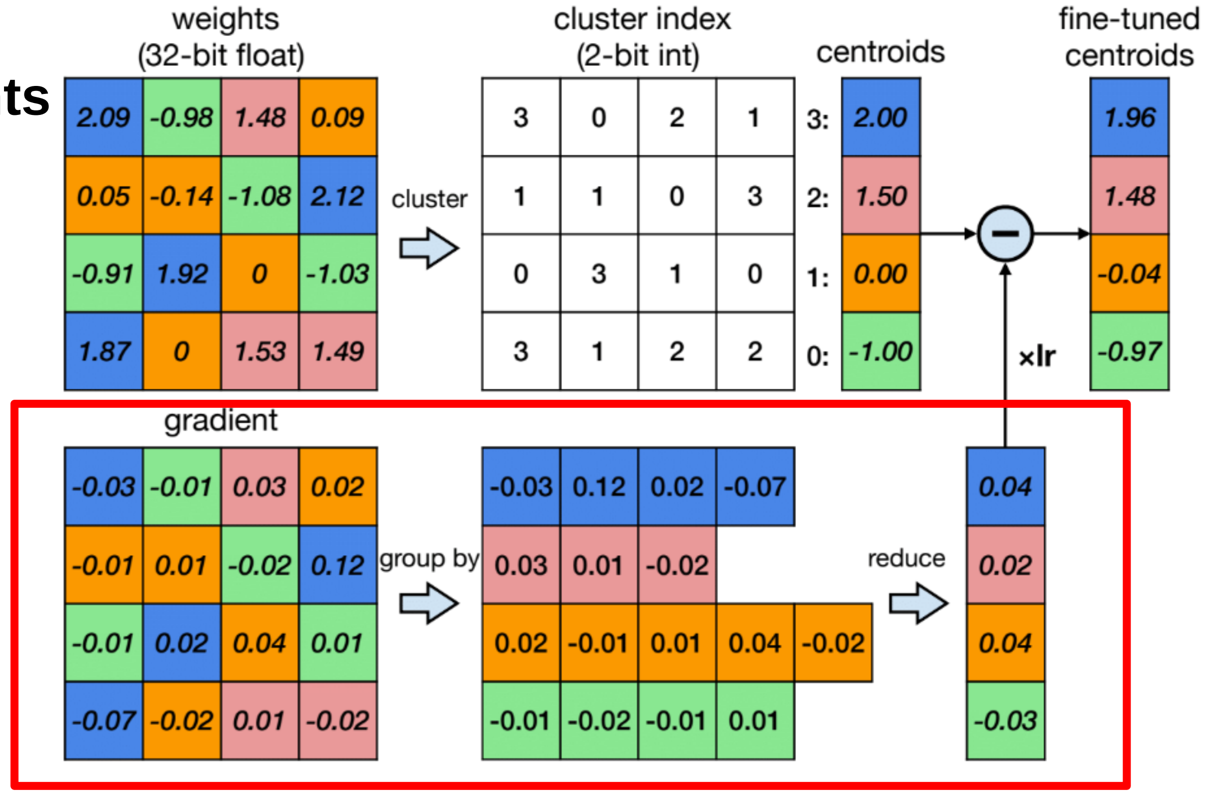
Assume *N*-bit quantization, and #parameters = $M >> 2^N$.

- 32 bit × M = $32M$ bit
- N bit × M = NM bit
- ~~32 bit × 2^N = 2^{N+5} bit~~

32/*N* × smaller

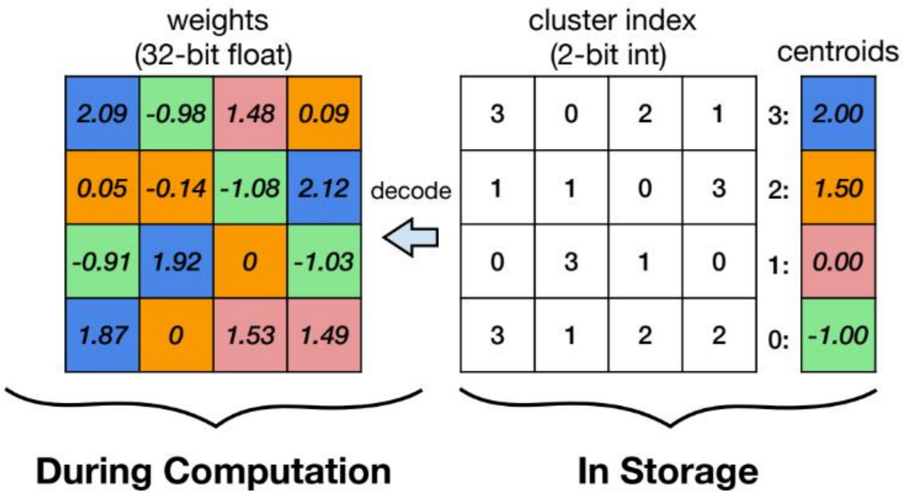
K-Means-based Weight Quantization

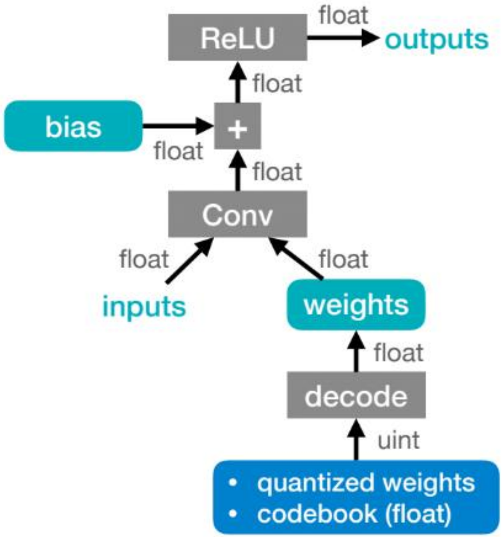
- **Fine-tuning Quantized Weights**
  - Reduce the quantization error



K-Means-based Weight Quantization

- Weights are decompressed using a lookup table during runtime inference
- Only saves storage cost of a neural network model
- All the computation and memory access are still floating-point





What is Linear Quantization ?

- An affine mapping of integers to real numbers
- **Storage:** Integer Weights; **Computation:** Integer Arithmetic

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) − 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

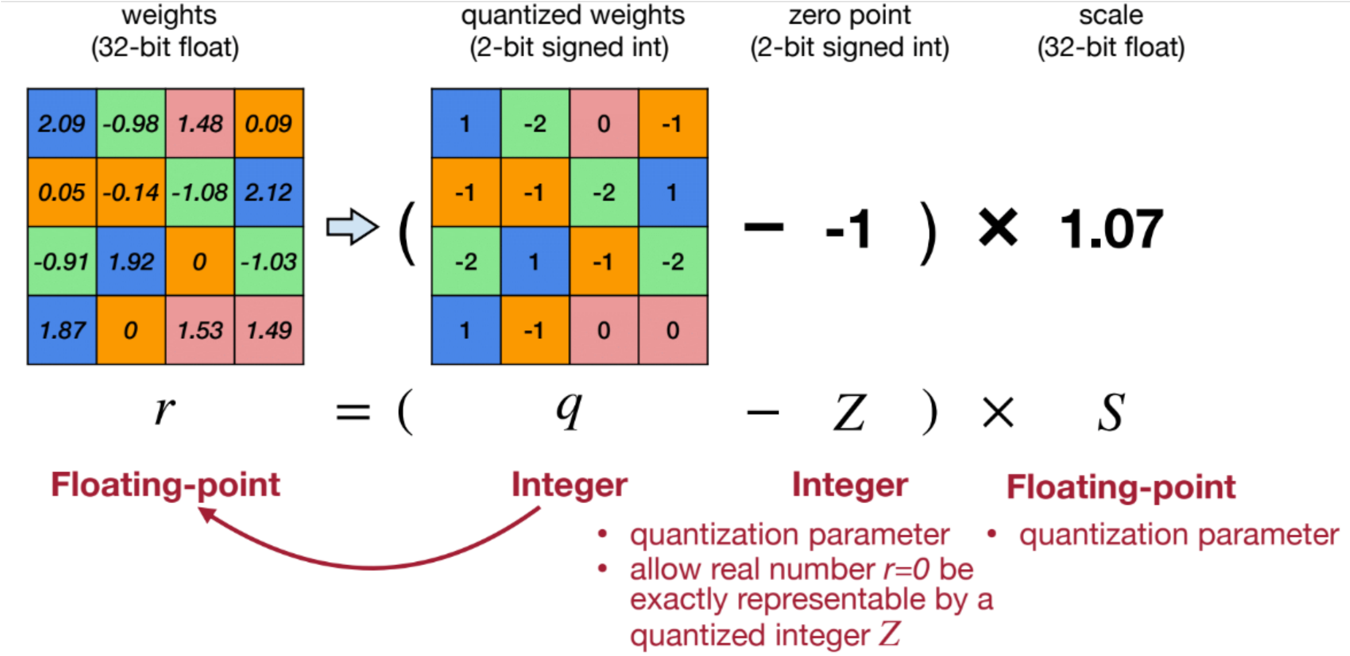
Binary	Decimal
01	1
00	0
11	-1
10	-2

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

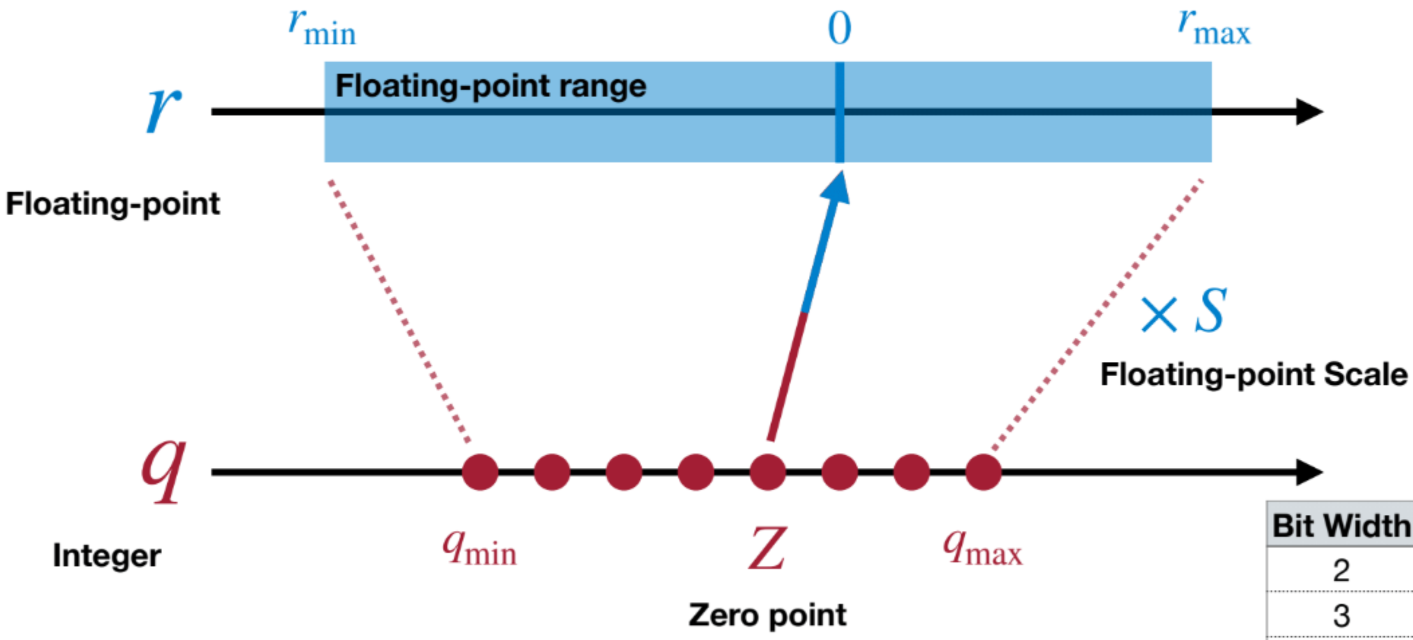
- An affine mapping of integers to real numbers (r = S(q - Z))



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

Linear Quantization

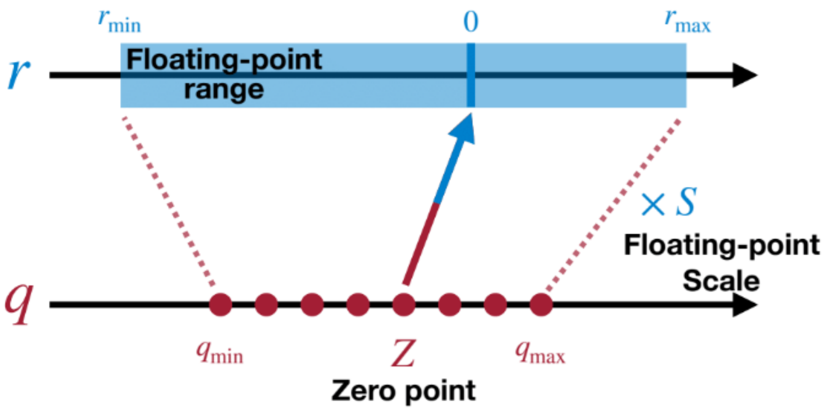
- An affine mapping of integers to real numbers (r = S(q - Z))



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

Scale of Linear Quantization

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



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

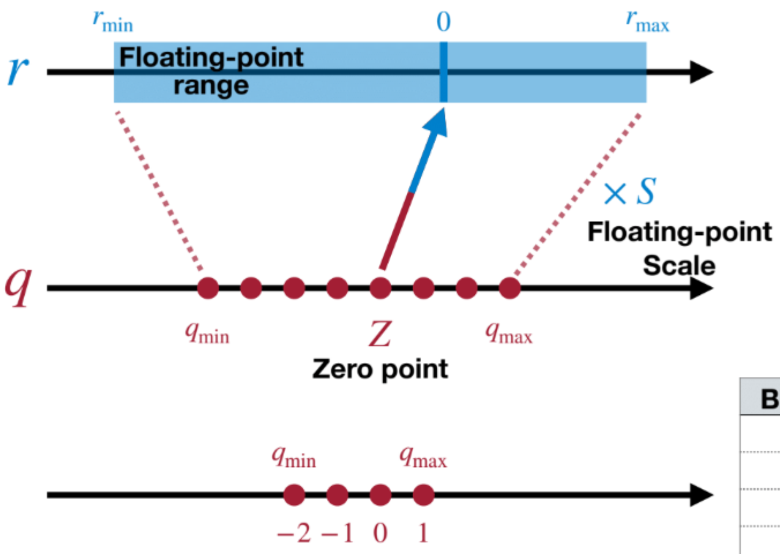
$$r_{min} = S(q_{min} - Z)$$

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

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

Scale of Linear Quantization

- 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

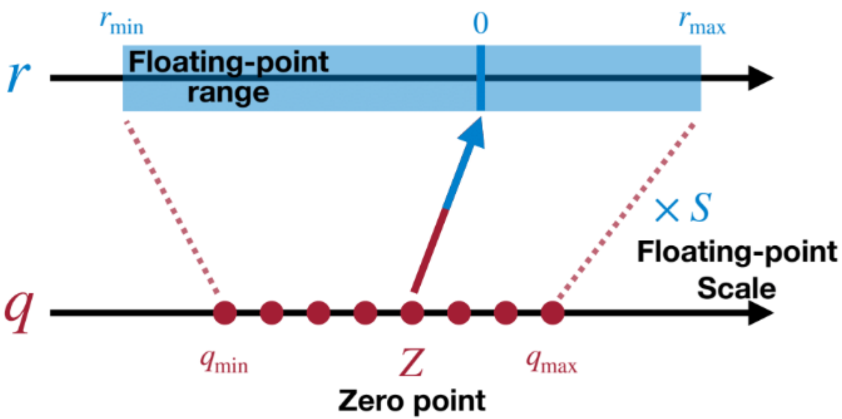
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

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

$$= 1.07$$

Zero Point of Linear Quantization

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



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

$$\downarrow$$

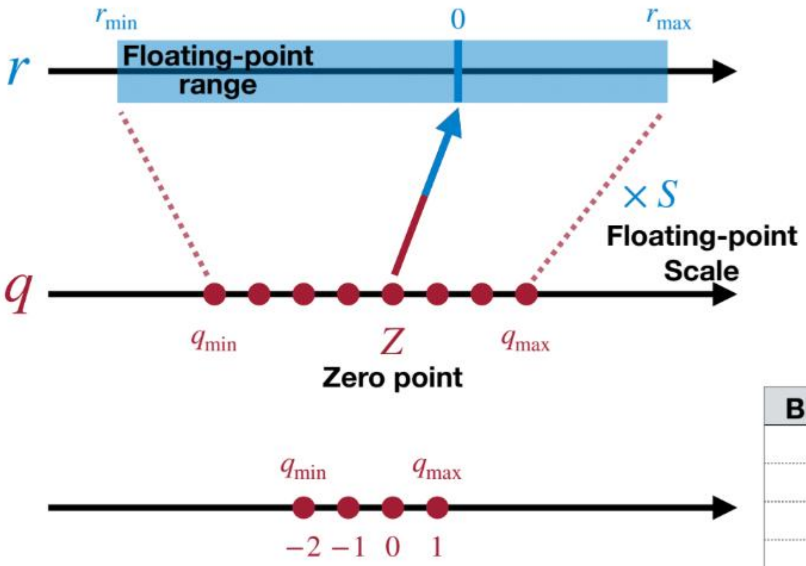
$$Z = q_{min} - \frac{r_{min}}{S}$$

$$\downarrow$$

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

Zero Point of Linear Quantization

- 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

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

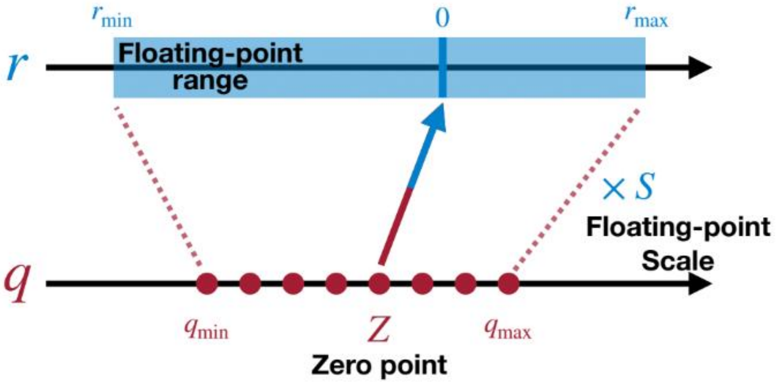
Binary	Decimal
01	1
00	0
11	-1
10	-2

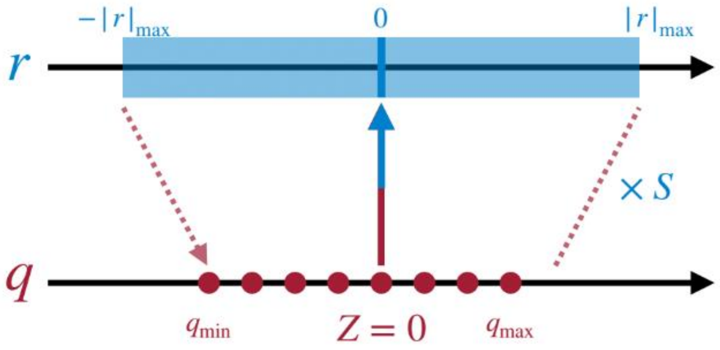
$$= \text{round}(-2 - \frac{-1.08}{1.07})$$

$$= -1$$

Asymmetric Linear Quantization

- Full range mode





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

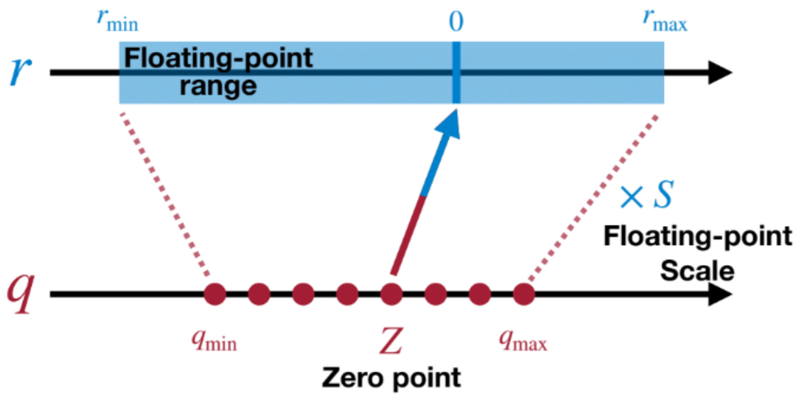
$$S = \frac{r_{\min}}{q_{\min} - Z} = \frac{-|r|_{\max}}{q_{\min}} = \frac{|r|_{\max}}{2^{N-1}}$$

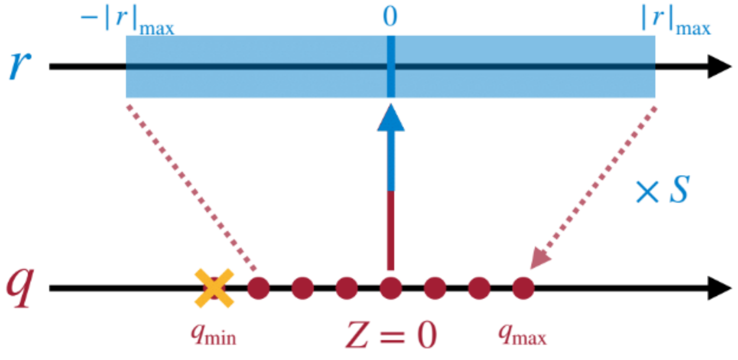
- use full range of quantized integers
- example: PyTorch's native quantization, ONNX

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

- Restricted range mode





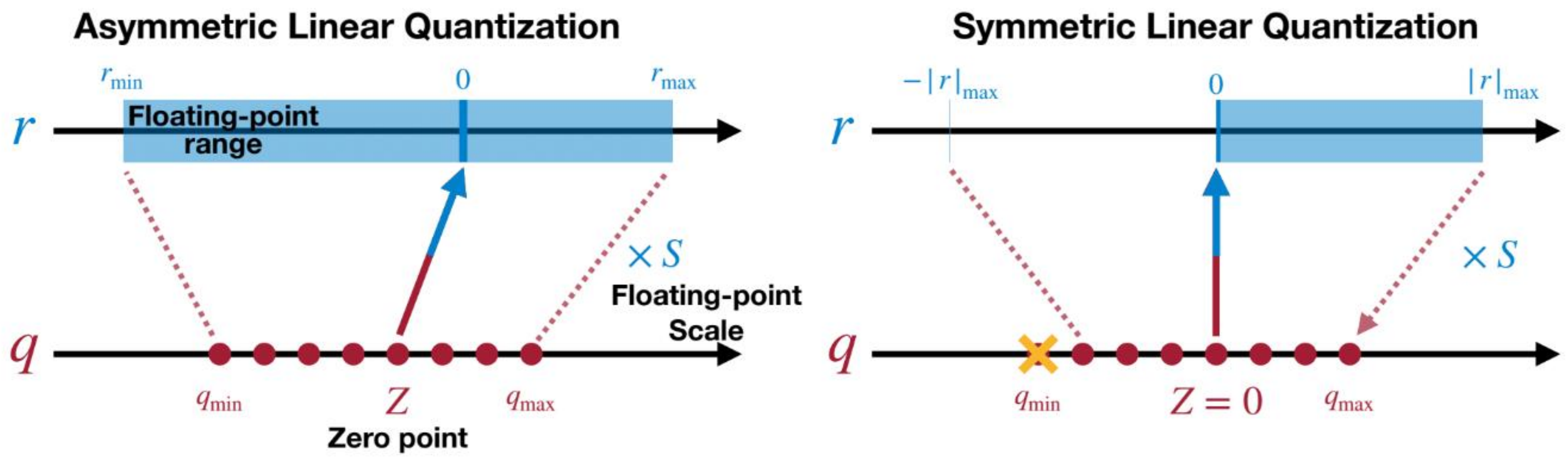
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

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

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

- example: TensorFlow, NVIDIA TensorRT, Intel DNNL

Asymmetric vs. Symmetric



- The quantized range is fully used.
- The implementation is more complex, and zero points require additional logic in hardware.

- The quantized range will be wasted for biased float range.
  - Activation tensor is non-negative after ReLU, and thus symmetric quantization will lose 1 bit effectively.
- The implementation is much simpler.

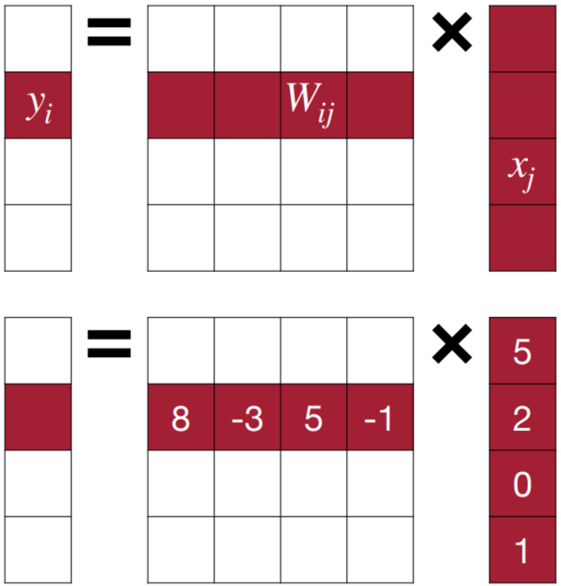
Neural Network Distiller

Binary/Ternary Quantization

- Could we push the quantization precision to 1 bit?

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

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



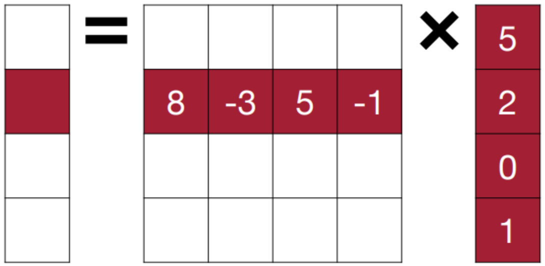
input	weight	operations	memory	computation
$\mathbb{R}$	$\mathbb{R}$	+ ×	1×	1×

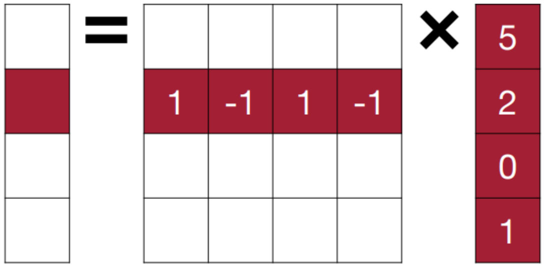
Binary/Ternary Quantization

- 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]

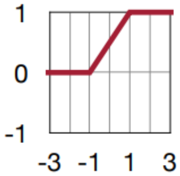
Binarization

- **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.

Minimizing Quantization Error in Binarization

weights
(32-bit float)

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

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

binary weights
(1-bit)

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

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

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

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

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

scale
(32-bit float)

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

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

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

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

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

Binary Net

- **Binary Connect**
  - Weights {-1, 1} (Bipolar binary), Activation 32-bit float
  - Accuracy loss: 19 % on AlexNet
- **Binarized Neural Networks**
  - Weights {-1, 1}, Activations {-1, 1}
  - Both of operands are binary, the multiplication turns into an XNOR
  - Accuracy loss: 29.8 % on AlexNet

XNOR

A	B	Out
0	0	1
1	0	0
0	1	0
1	1	1

Popcount (110010001) = 4

```
for each i in width:
    C += A[row][i] * B[i][col]
```

→

```
for each i in width:
    C += popcount(XNOR(A[row][i], B[i][col]))
```

Courbariaux., NeurIPS, 2015

Case Study: Binary Multiplication

- A = 10010, B = 01111 (0 is really -1 here)
- **Dot product**:
  - A * B = (1 * -1) + (-1 * 1) + (-1 * 1) + (1 * 1) + (-1 * 1) = **-3**
- P = XNOR (A, B) = 00010, popcount(P) = 1
- Result = 2 * P – N, where N is the total number of bits
- 2 * P – N = 2 * 1 – 5 = **-3**

https://sushscience.wordpress.com/2017/10/01/understanding-binary-neural-networks/

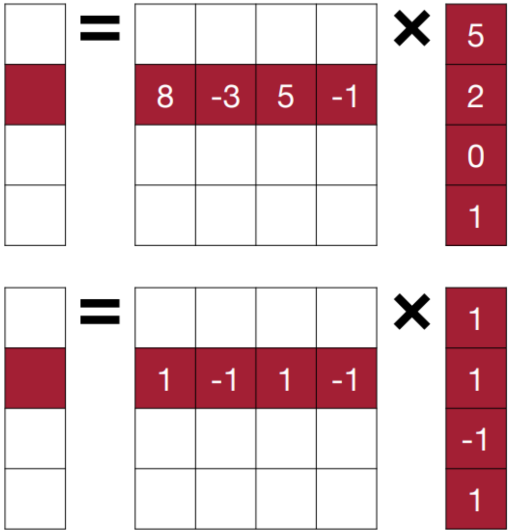
XNOR-Net

- 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]

XNOR-Net

- If both activations and weights are binarized

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

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

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

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

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

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

XNOR-Net

- If both activations and weights are binarized

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

$$= 1\times1 + (-1)\times1 + 1\times(-1) + (-1)\times1 \qquad = 1 \text{ xnor } 1 + 0 \text{ xnor } 1 + 1 \text{ xnor } 0 + 0 \text{ xnor } 1$$

$$= 1 + (-1) + (-1) + (-1) = -2 \qquad = 1 + 0 + 0 + 0 \;=\; 1$$

?

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]

XNOR-Net

- If both activations and weights are binarized

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

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

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

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

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

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

↑+2

Assuming -1 -1 -1 -1 → -4

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

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

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

XNOR-Net

- If both activations and weights are binarized

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

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

= -4 + 2 × (1 + 0 + 0 + 0) = -2

→ popcount: return the number of 1

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

bw	bx	XNOR(bw, bx)
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]

XNOR-Net

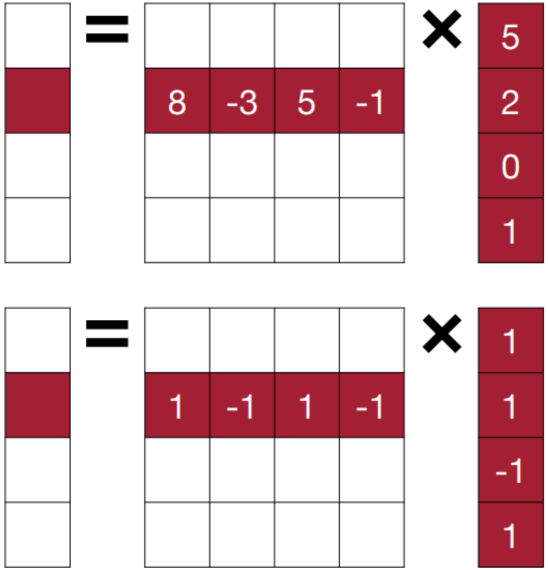
- If both activations and weights are binarized

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

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

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

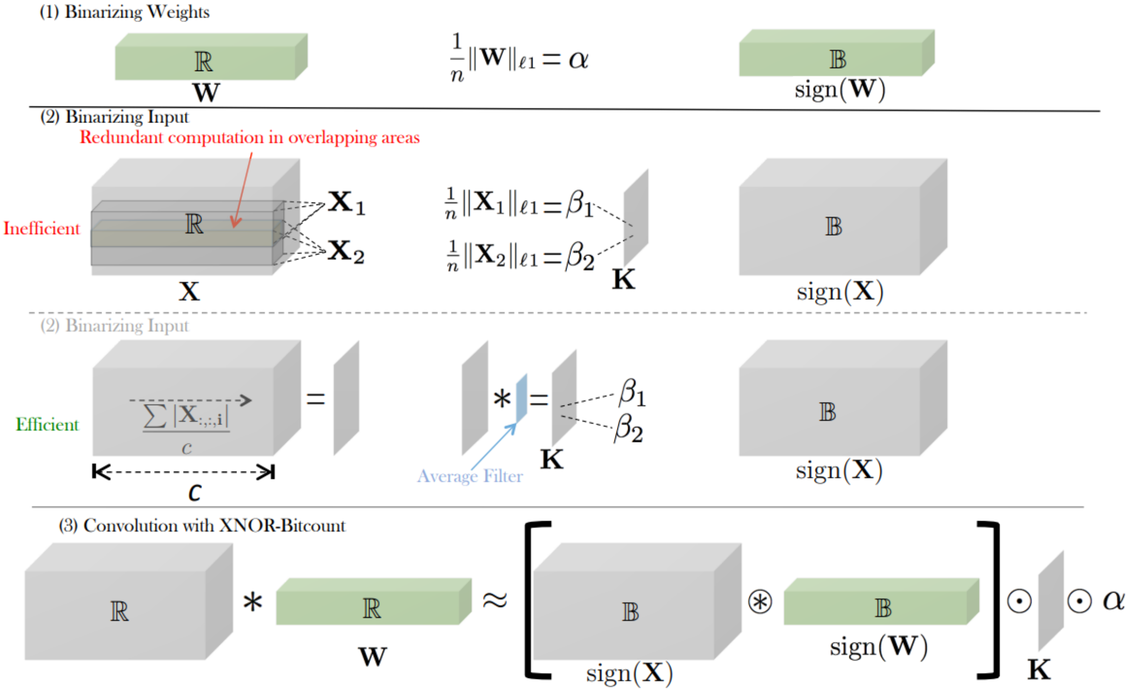
input	weight	operations	memory	computation
$\mathbb{R}$	$\mathbb{R}$	+ ×	1×	1×
$\mathbb{R}$	$\mathbb{B}$	+ -	~32× less	~2× less
$\mathbb{B}$	$\mathbb{B}$	xnor, popcount	~32× less	~58× less



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

XNOR-Net

- Minimizing quantization error in binarization



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

XNOR-Net

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%

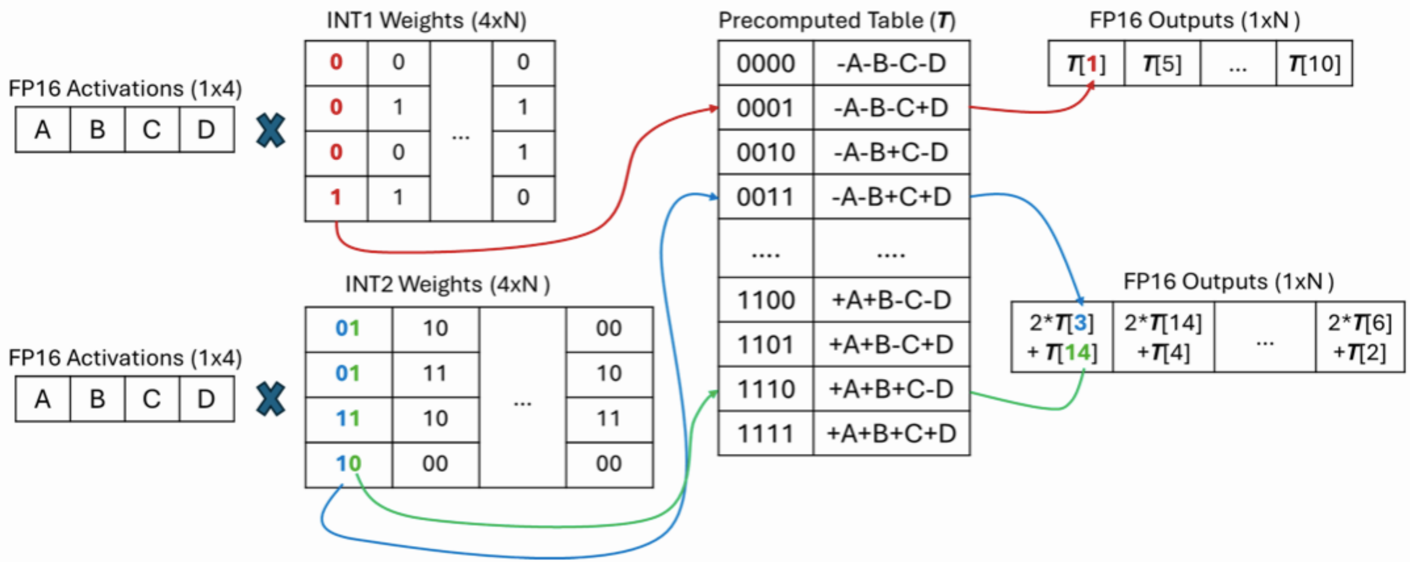
* BWN: Binary Weight Network with scale for weight binarization
* BNN: Binarized Neural Network without scale factors
* XNOR-Net: scale factors for both activation and weight binarization

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]

BitNet

- FP16 activation and 1.58 bit weights – Transformer-based model
- Lookup table (LUT) calculations

https://www.arxiv.org/pdf/2407.00088



What do we Learn from Quantization?

- **Quantization** can improve DNN computational throughput while maintaining accuracy
- Layers on DNN models can be offered with **different bit widths**
- Varying bit width requires **the support of the hardware**
- **No systematic approach** to figure out the proper bit width in layers of DNN models
- What else ?

Takeaway Questions

- What are purposes of data quantization ?
  - (A) Constrain the value of inputs to a set of discrete values
  - (B) Create more values
  - (C) Improve the degree of parallelism on DNN training
- Why training requires large bit width ?
  - (A) The training needs to compute more data
  - (B) Avoid the value underflow and overflow
  - (C) Gradient and weight update have a larger range