CS5120 VLSI System Design, Spring 2025

Number Representation and Quantization

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
NATIONAL TSING HUA UNIVERSITY

資訊工程學系
Computer Science

Lecture 04

聲明

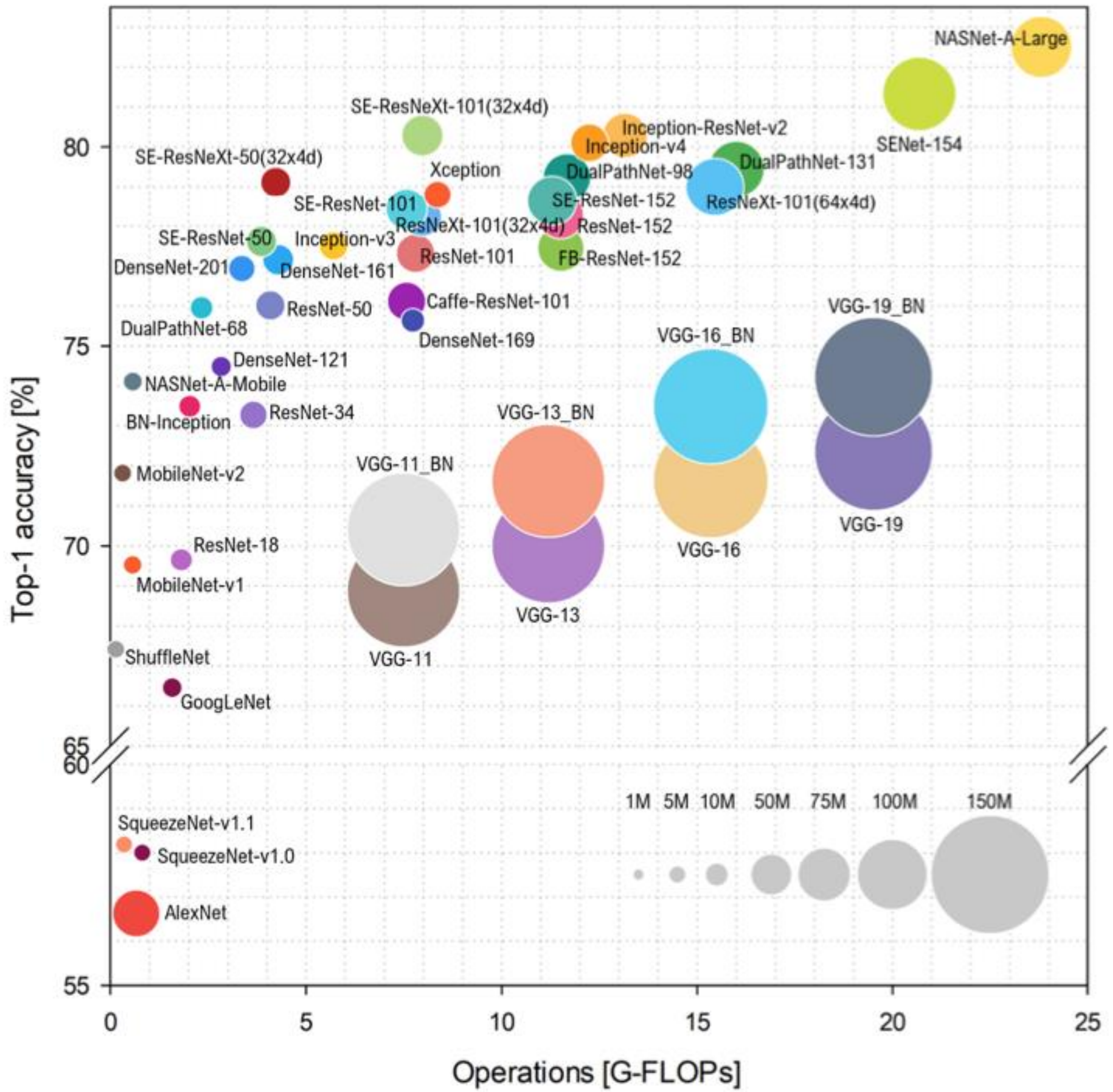
- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

- Floating-point representations
  - IEEE 754 Format
  - Google Brain Floating-Point Format
  - NVIDIA TensorFloat32
  - Nervana Flexpoint
  - Microsoft Floating-Point Format
- Fixed-point representation
- Hardware Arithmetic Consideration
- Quantization

DNNs

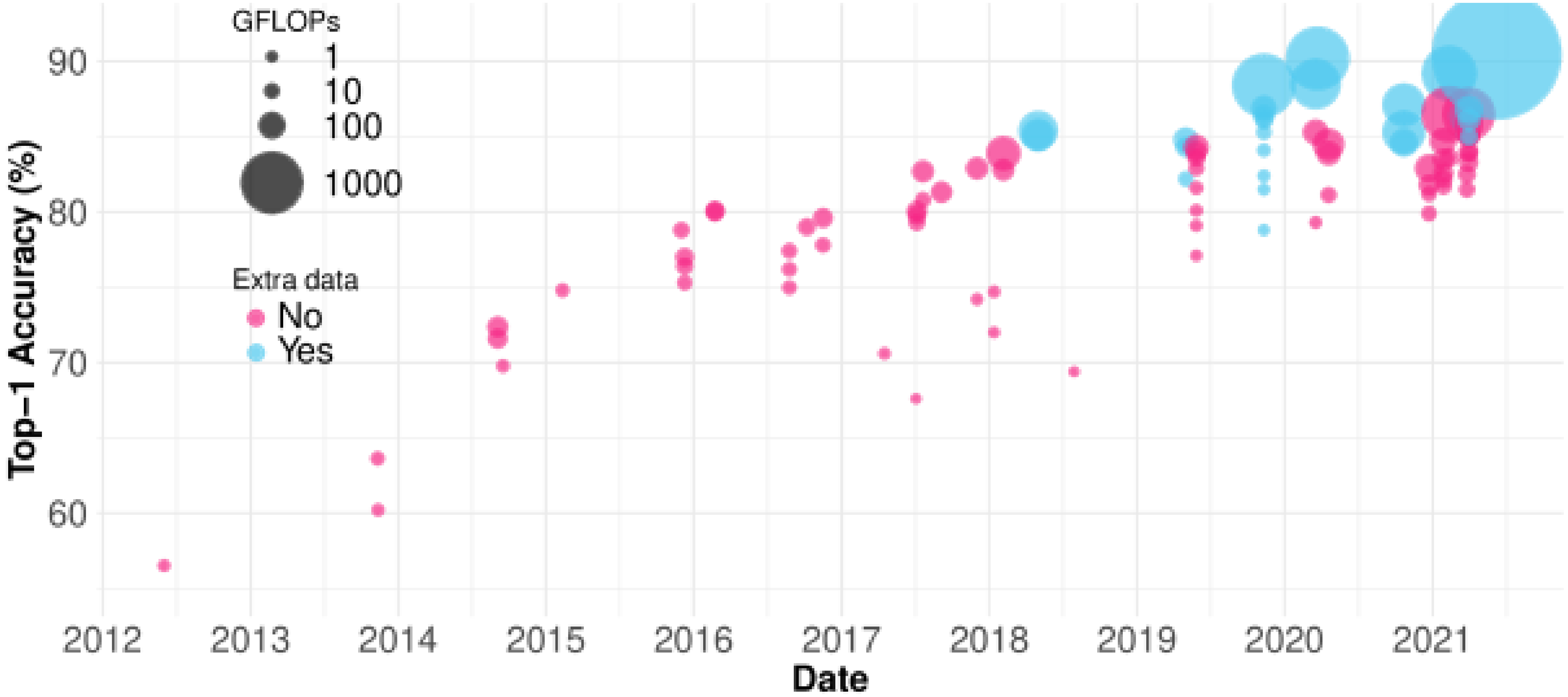
- High accuracy still implies high complexity even when the models become more efficient!



https://arxiv.org/abs/1810.00736

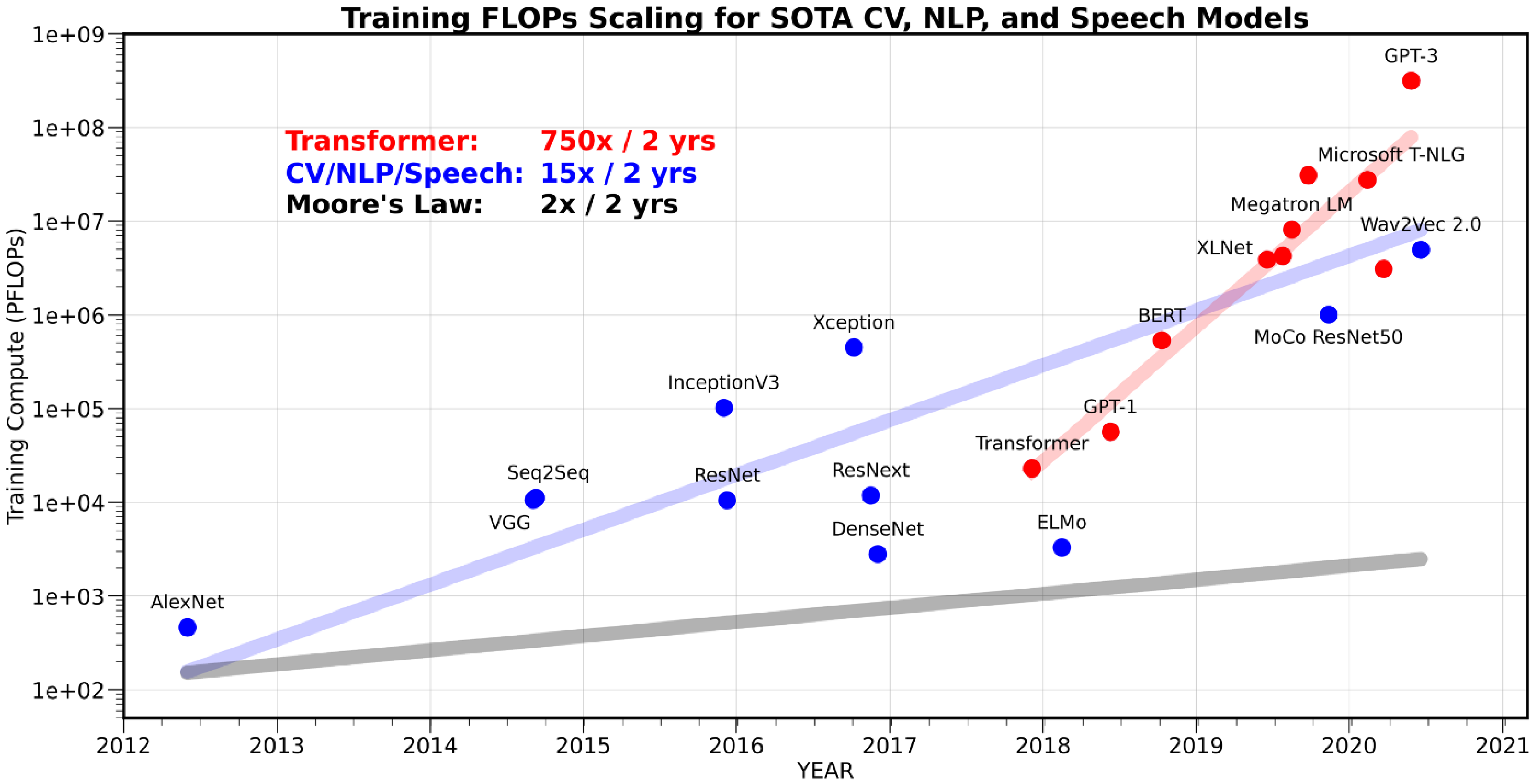
DNN Accuracy Evolution

https://deepai.org/publication/compute-and-energy-consumption-trends-in-deep-learning-inference



Training FLOPs Scaling for AI Models

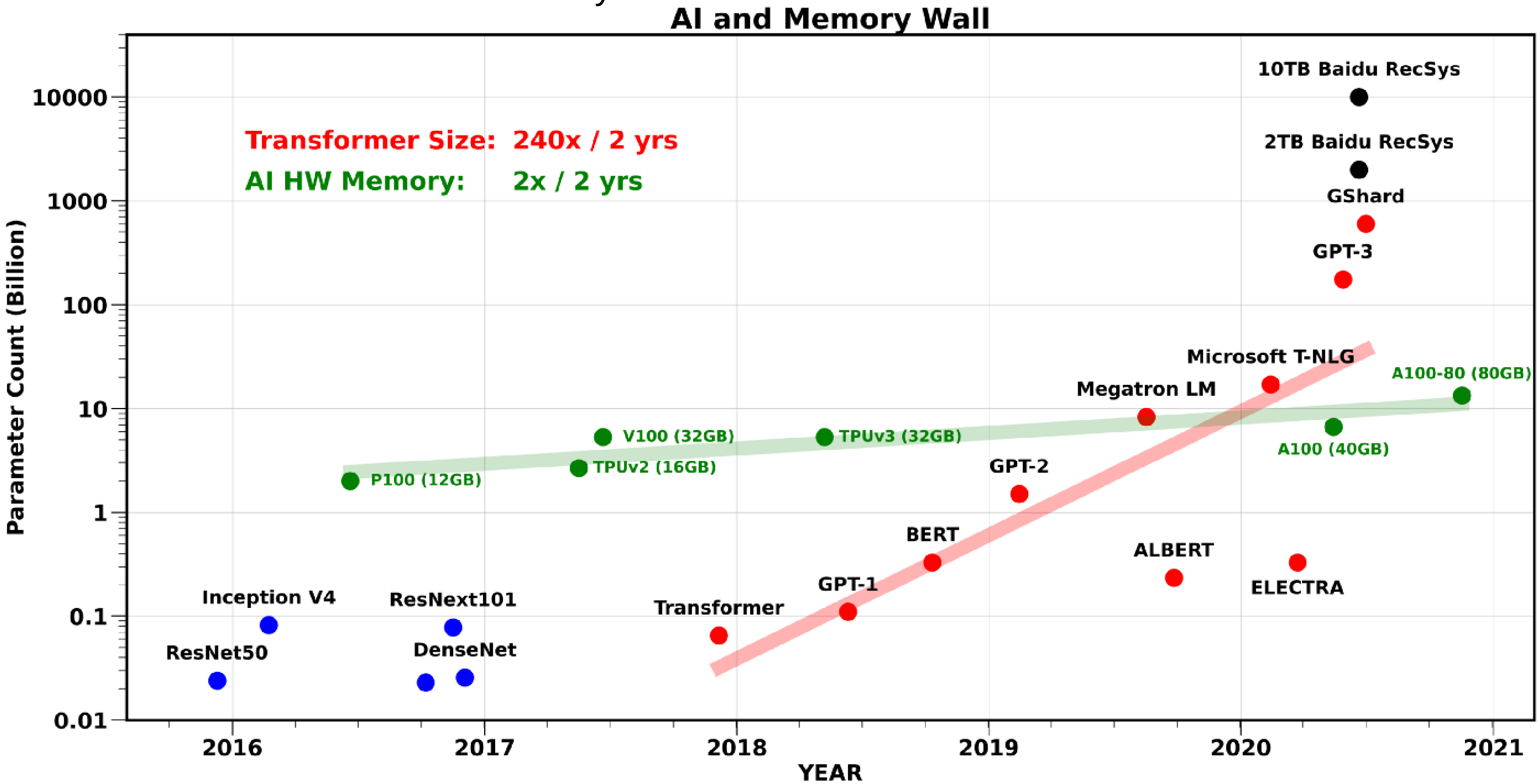
https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8



The amount of compute, measured in Peta FLOPs, needed to train SOTA models, for different CV, NLP, and Speech models, along with the different scaling of Transformer models (750x/2yrs); as well as the scaling of all of the models combined (15x/2yrs).

AI and Memory Wall

https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8



The evolution of the number of parameters of SOTA models over the years, along with the AI accelerator memory capacity (green dots). The number of parameters in large Transformer models has been exponentially increasing with a factor of 240x every two years, while the single GPU memory has only been scaled at a rate of 2x every 2 years

Floating-Point Representations

- Some slides adopted from
  Lectures by Prof. Juinn-Dar Huang, NYCU

Memory Issues

- Domain-specific computation usually suffers from heavy data movement
  - Large memory size and heavy memory traffic
  - Especially, DRAM access is time- and power-consuming
- DRAM bandwidth and capacity is limited (even in GPU servers)
  - System performance bottleneck
- Simply won't work in tiny edge/AIoT devices
- Light-weight number representation
  - Less memory bandwidth/capacity
  - Less computation complexity
  - Less energy/power consumption

Scientific Notation for Numbers

- In decimal
  - Normalized form

mantissa exponent

$$3\,.\,14_{10} \times 10^{9}$$

decimal point (radix point) — base (radix)

  - Non-normalized form:

$$0.314 \times 10^{10}$$

$$31.4 \quad \times 10^{8}$$

- In binary
  - Normalized form

(significand; fraction)
mantissa exponent

$$1\,.\,101_{2} \times 2^{-7}$$

binary point — base (radix)

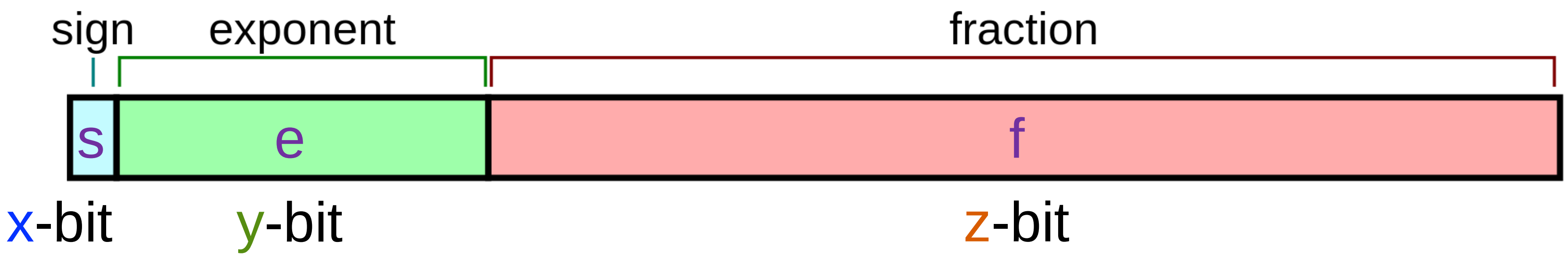
- Also called floating-point numbers in computer arithmetic
  - Binary point is not fixed

Floating-Point Number Formats

- **IEEE-754**: a standard for floating-point (FP) formats
  - Including several formats with different bit lengths

- Most commonly used three formats today
  - **FP64:** 64-bit double-precision format (1985)
  - **FP32:** 32-bit single-precision format (1985)
  - **FP16:** 16-bit half-precision format (2008)

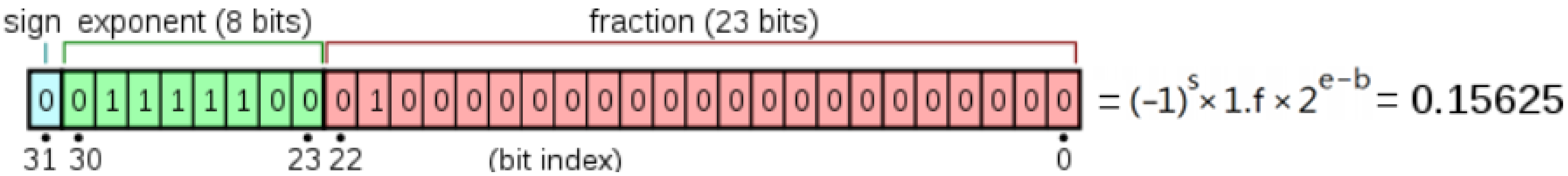
IEEE-754 Floating-Point Formats



Definition (simplified, incomplete)

- x is always 1; y and z are format-dependent
- An implicitly defined exponent *bias* b: $2^{y-1} - 1$
  - i.e., b is solely dependent on y
- Normal value: $(-1)^s \times 1.f \times 2^{e-b}$, $0 < e < 2^y - 1$
- Denormal value: $(-1)^s \times 0.f \times 2^{1-b}$, $e = 0$ and $f \neq 0$
- ± zero: ± 0 , $e = 0$ and $f = 0$

Single-Precision FP Format (FP32)



FP32 format (i.e., $x + y + z = 32$)

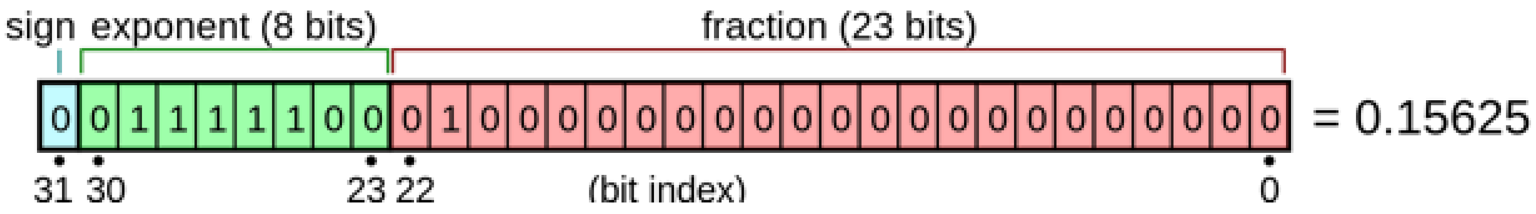
- $x = 1$, $y = 8$, $z = 23$, $b = 2^{y-1} - 1 = 2^{8-1} - 1 = 127$
- Norm value: $(-1)^s \times 1.f \times 2^{e-127}$, $0 < e < 255$
- Denorm value: $(-1)^s \times 0.f \times 2^{-126}$, $e = 0$ and $f \neq 0$
- Min/Max norm value: 2^{-126} / $(2-2^{-23}) \times 2^{127} \approx 2^{128}$
- Min denorm value: $2^{-23} \times 2^{-126} = 2^{-149}$

FP32 vs. FP16

- Use Biased Notation, where bias is number subtracted to get real **exponent**
  - Double precision (fp64): bias = 1023
  - Single precision (fp32): bias = 127
  - Half precision (fp16): bias = 15

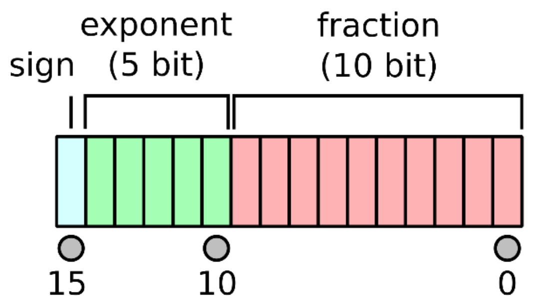
$$(-1)^{\text{sign}} \times (1 + \text{fraction}) \times 2^{(\text{exponent}-\text{bias})}$$

fp32 (float32)



Range: ~1.18e-38 … ~3.40e38 with
6–9 significant decimal digits precision.

fp16 (float16)

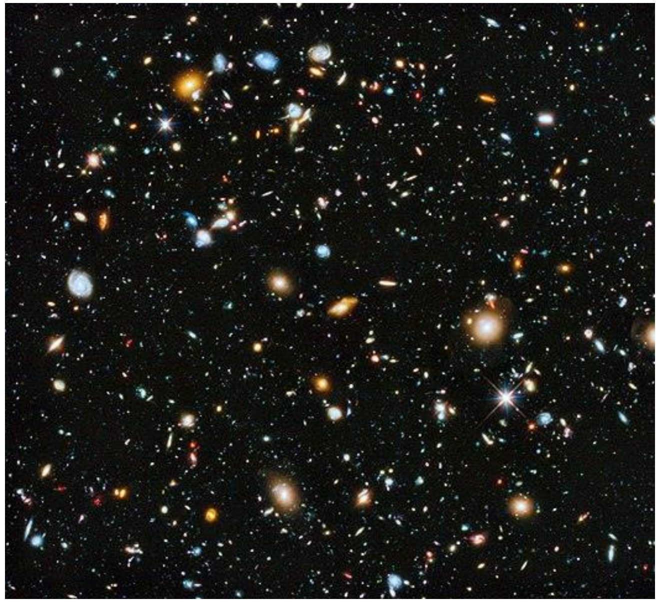


Range: ~5.96e−8 (6.10e−5) … 65504 with
4 significant decimal digits precision.

https://en.wikipedia.org/wiki/Single-precision_floating-point_format

Value Range of FP32

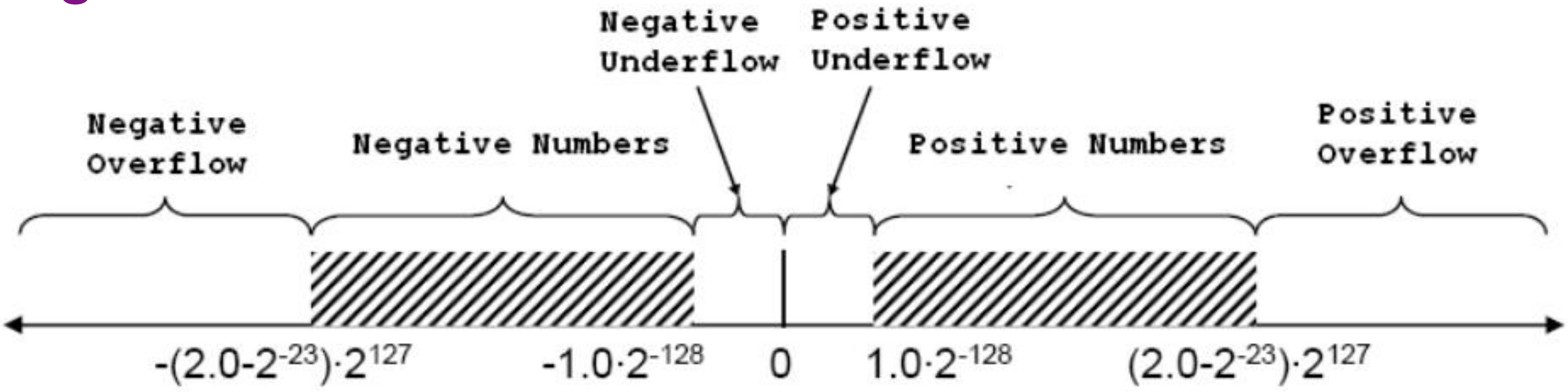
- Range of normal values
  - binary: $2^{-126} \sim 2^{128}$ (24-bit precision)
  - decimal: $1.18 \times 10^{-38} \sim 3.40 \times 10^{38}$ (~7-digit precision)
- A trillion (10^{12} or 2^{40}) times the diameter of the observable universe in meters
- 10 billionth (10^{-8}) the weight of an electron in KGs



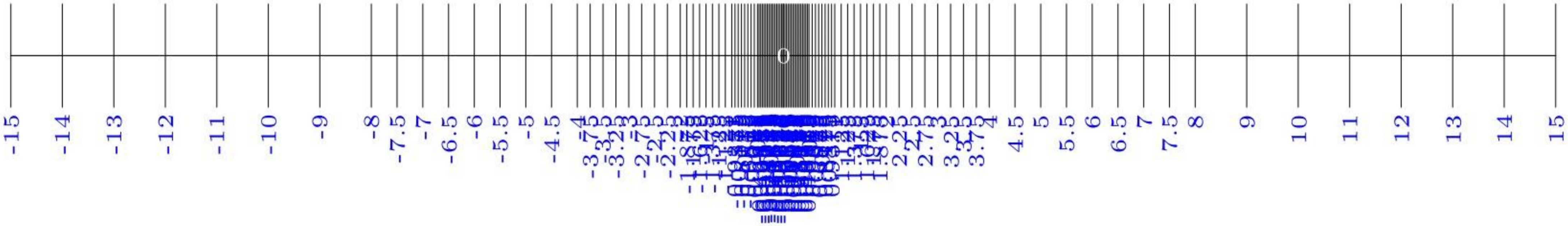
- FP32 is the most commonly used FP format in today's machine learning frameworks ➔ **WASTE!**

Discontinuous Range and Non-Uniform Distribution of Floating-Point Values

Dynamic Range



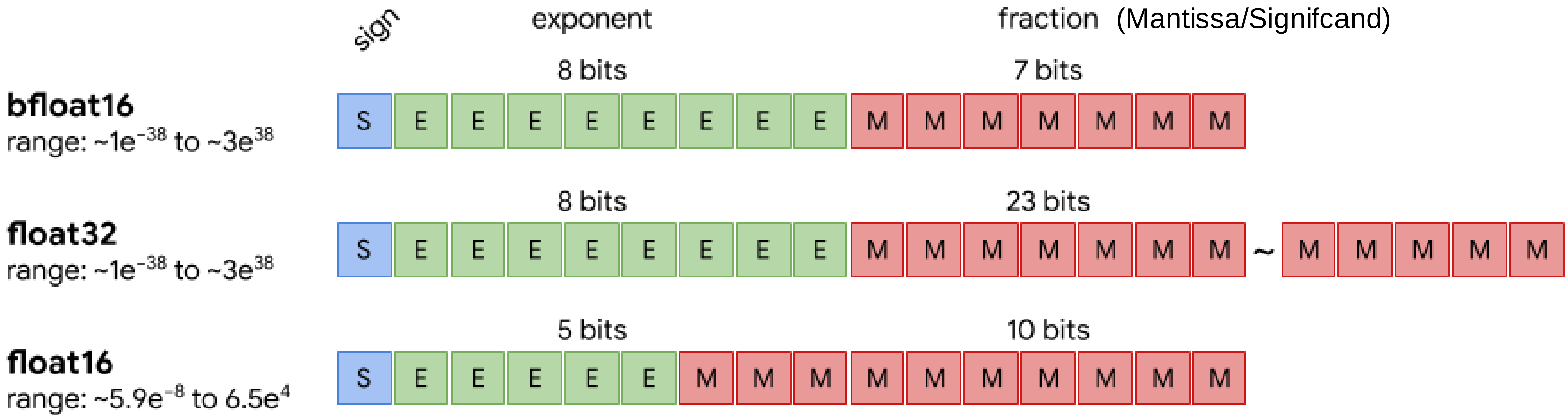
Precision



Non uniform distribution of IEEE 754 tiny floating-point (normal and subnormal) numbers.
https://anniecherkaev.com/the-secret-life-of-nan

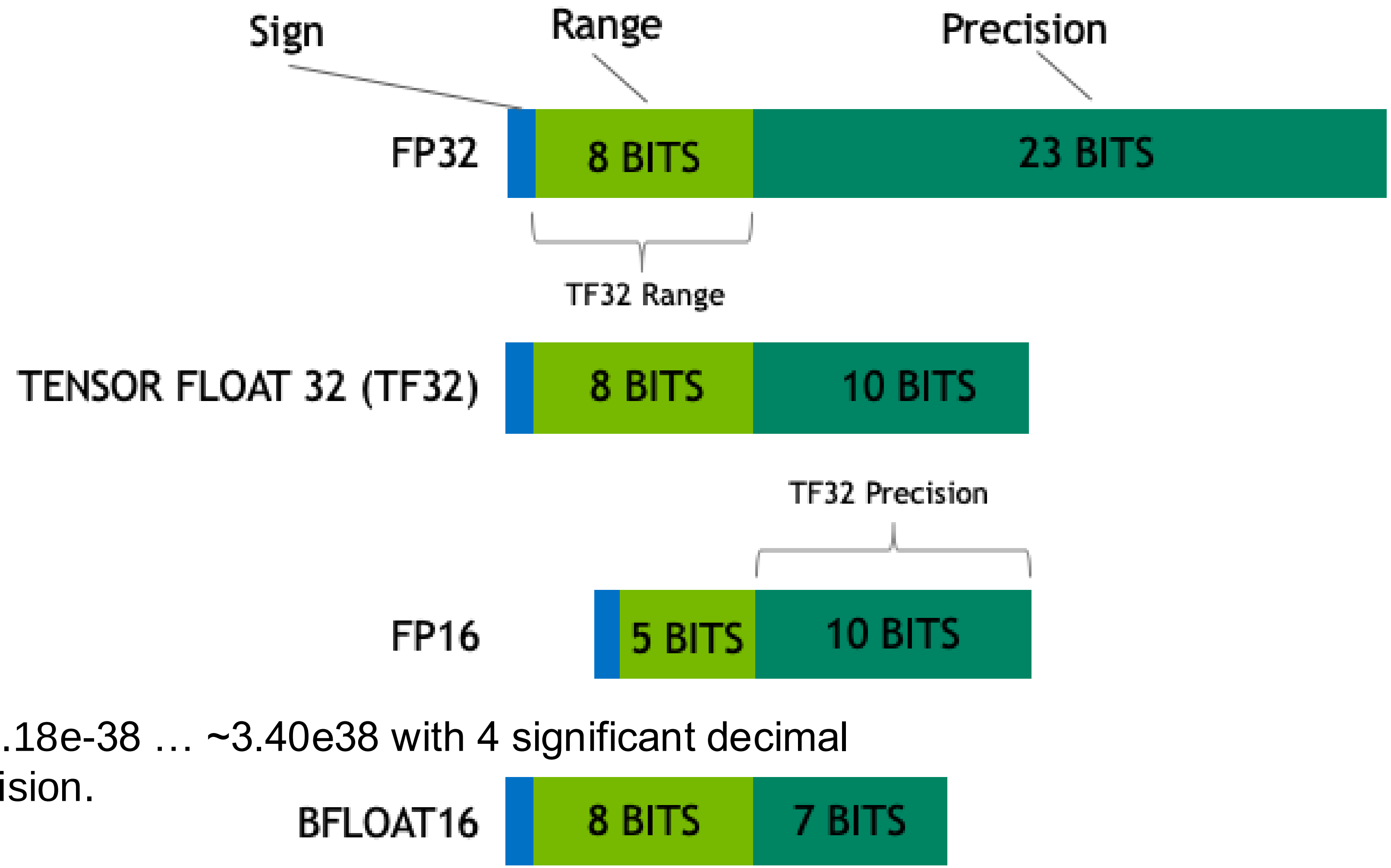
IEEE Floating-Point Format vs.
Google Brain Floating-Point Format (bfloat)

- Dynamic range of bfloat16 is greater than that of fp16
- Replacing fp16 with faster conversion to/from fp32
- Range: ~1.18e-38 … ~3.40e38 with 3 significant decimal digits.



https://cloud.google.com/tpu/docs/bfloat16

NVIDIA's TensorFloat32 (TF32)



Range: ~1.18e-38 … ~3.40e38 with 4 significant decimal digits precision.

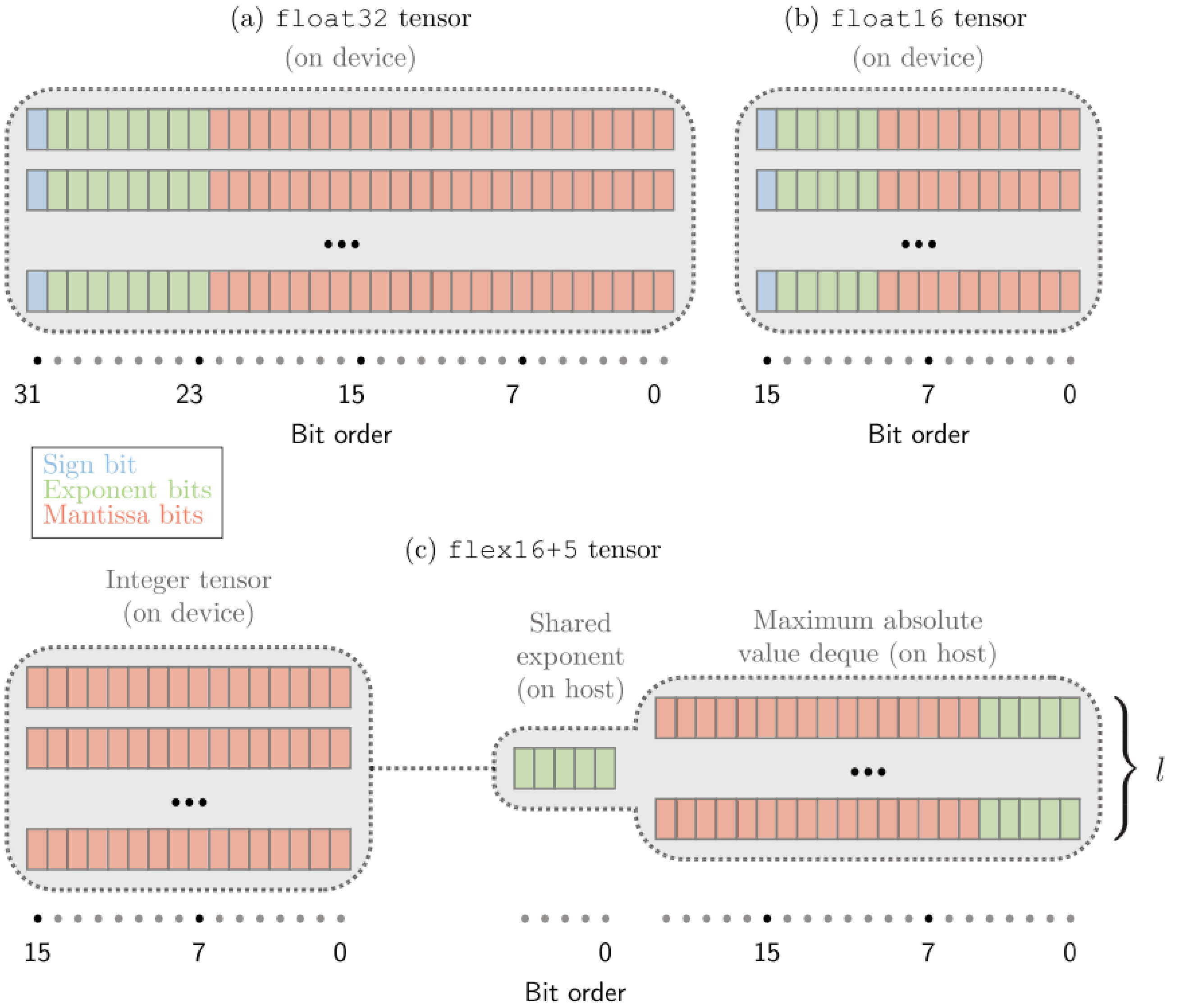
https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/

NVIDIA TF32 Advantage

- Extended-precision over BFLOAT16 or reduced-precision over FP32
- SW can support TF32 easily (i.e., only inside CUDA compiler)
  - FP32 with less precision
  - Instead, fp16/bloat16 require further conversion
- NVIDIA A100's peak performance
  - FP32 without tensor core: 19.5 TFLOPS
  - TF32 tensor core: 156 TFLOPS
  - FP16/BF16 tensor core: 312 TFLOPS

Flexpoint from Nervana (Intel)

- Blocked floating-point
  - Shared exponent across tensors
  - Dynamically adjusted to minimize overflows and maximize available dynamic range
  - Reduce memory capacity and bandwidth requirements

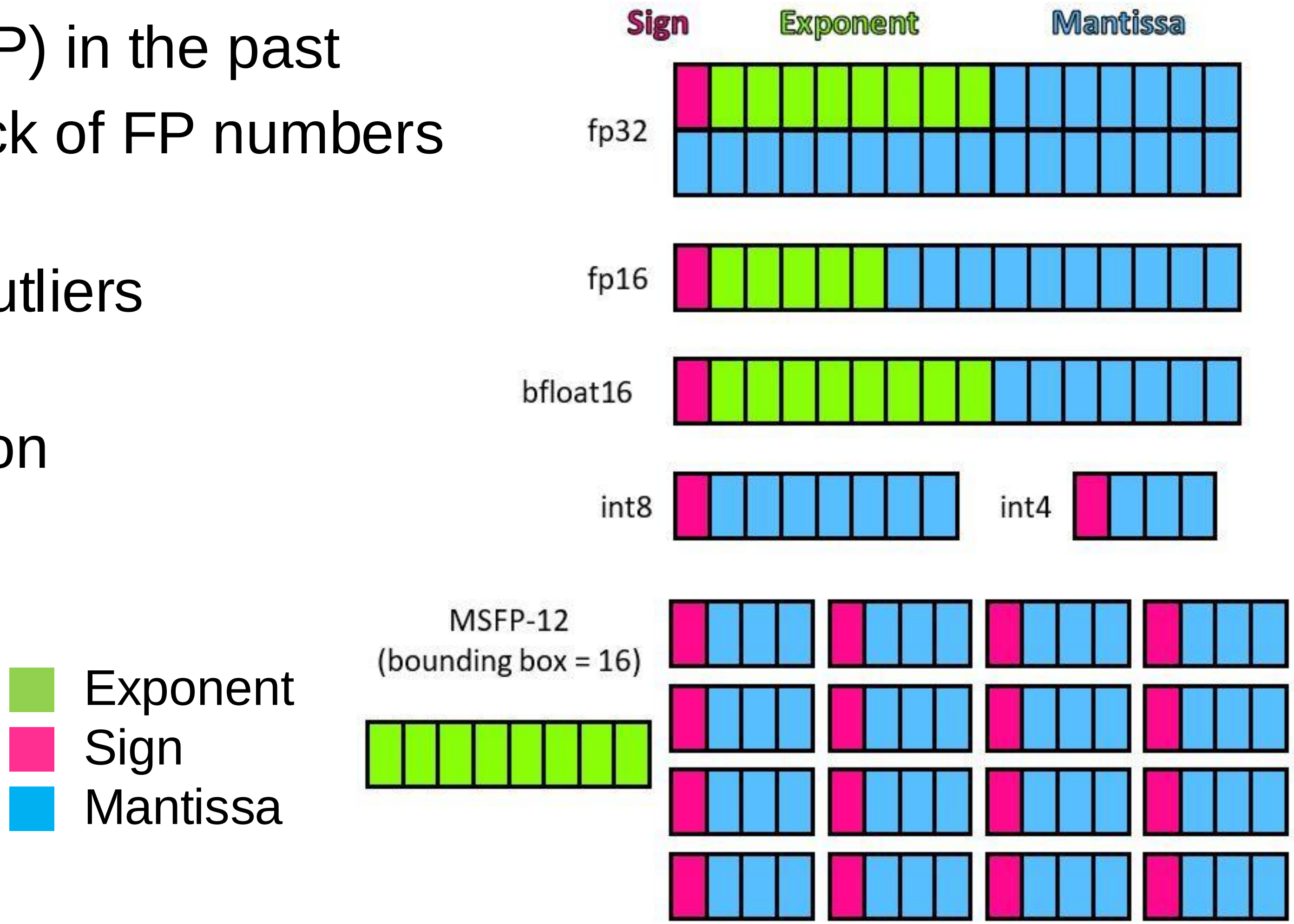


https://papers.nips.cc/paper/2017/file/a0160709701140704575d499c997b6ca-Paper.pdf

Microsoft Floating-Point Format (MSFP) [NIPS2020]

https://www.microsoft.com/en-us/research/blog/a-microsoft-custom-data-type-for-efficient-inference/

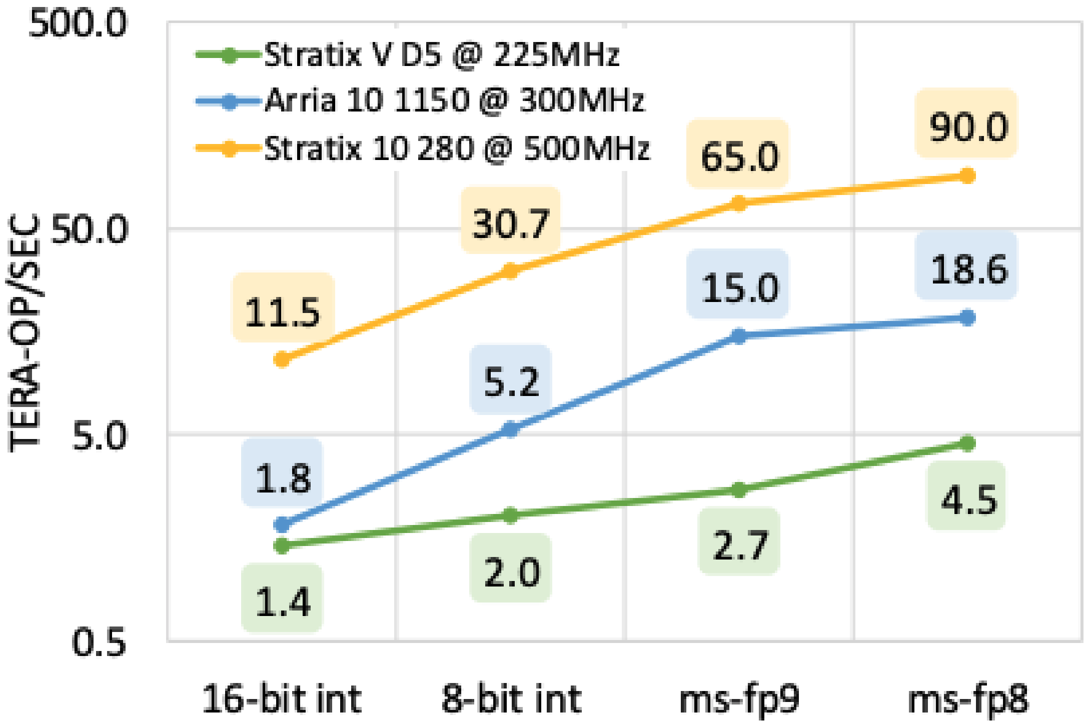
- A.k.a. block floating-point (BFP) in the past
- Shared exponent within a block of FP numbers
  - ➔ Shorter format on average
- Collapsing the exponent for outliers
  - ➔ data degradation
- No implicit leading bit in fraction
  - ➔ accuracy loss
- No denorm
  - ➔ smaller dynamic range



MSFP-X: X = 8+N ; 8 bits for exponent,1 bit for sign, N-1 bits for Mantissa

MS-FP in Project Brainwave

> Proprietary "neural"-optimized data formats based on 8- and 9-bit floating point, where mantissas are trimmed to 2 or 3 bits.
>
> These formats, referred to as ms-fp8 and ms-fp9, exploit efficient packing into reconfigurable resources and are comparable in FPGA area



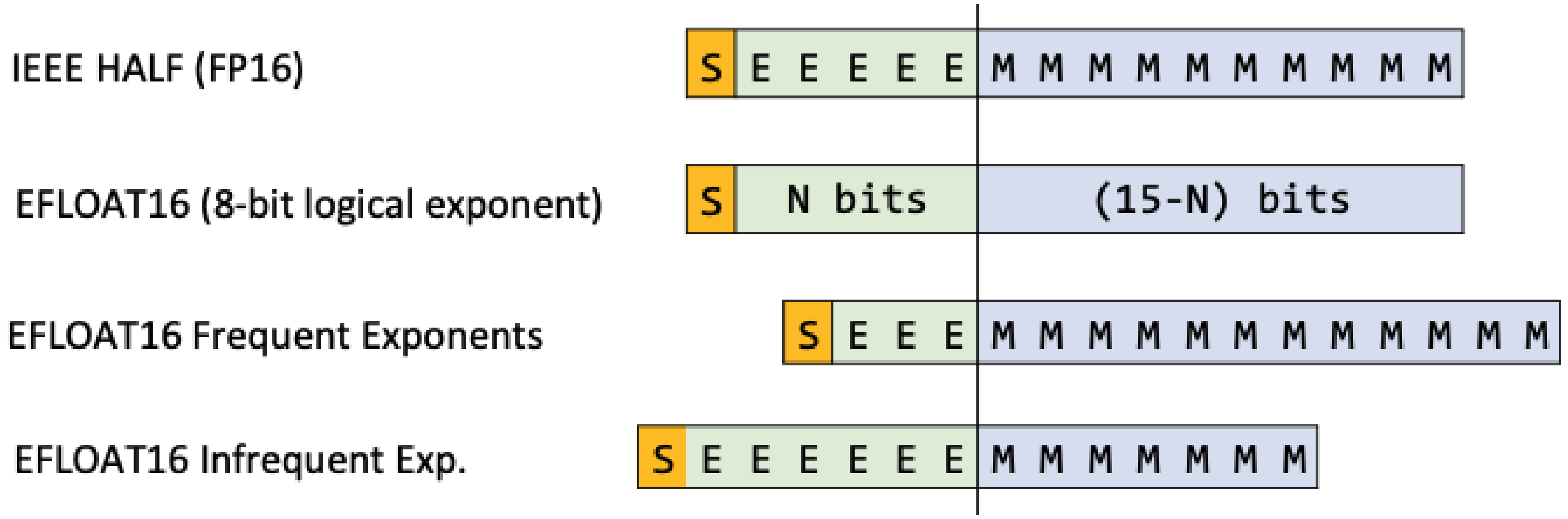
Source: Serving DNNs in Real Time at Datacenter Scale with Project Brainwave

Entropy-coded Floating Point by IBM (Efloat)

https://arxiv.org/pdf/2102.02705.pdf

Format for Inference

- More average precision (Huffman encoding on exponent)
- Large multiplier required
- Fewer precision bits for infrequent big values



Fixed-Point Representation

Fixed-Point Arithmetic

- Integers with a binary point and a bias
  - "slope and bias": $y = s \times x + z$ (affine mapping)
  - Qm.n: m (# of integer bits) n (# of fractional bits)

x_2	x_1	x_0

s = 1, z = 0

2^2	2^1	2^0	y
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

s = 1/4, z = 0

2^0	2^{-1}	2^{-2}	y
0	0	0	0
0	0	1	1/4
0	1	0	2/4
0	1	1	3/4
1	0	0	1
1	0	1	5/4
1	1	0	6/4
1	1	1	7/4

s = 4, z = 3

2^4	2^3	2^2	y
0	0	0	0+3
0	0	1	4+3
0	1	0	8+3
0	1	1	12+3
1	0	0	16+3
1	0	1	20+3
1	1	0	24+3
1	1	1	28+3

s = 1.5, z =10

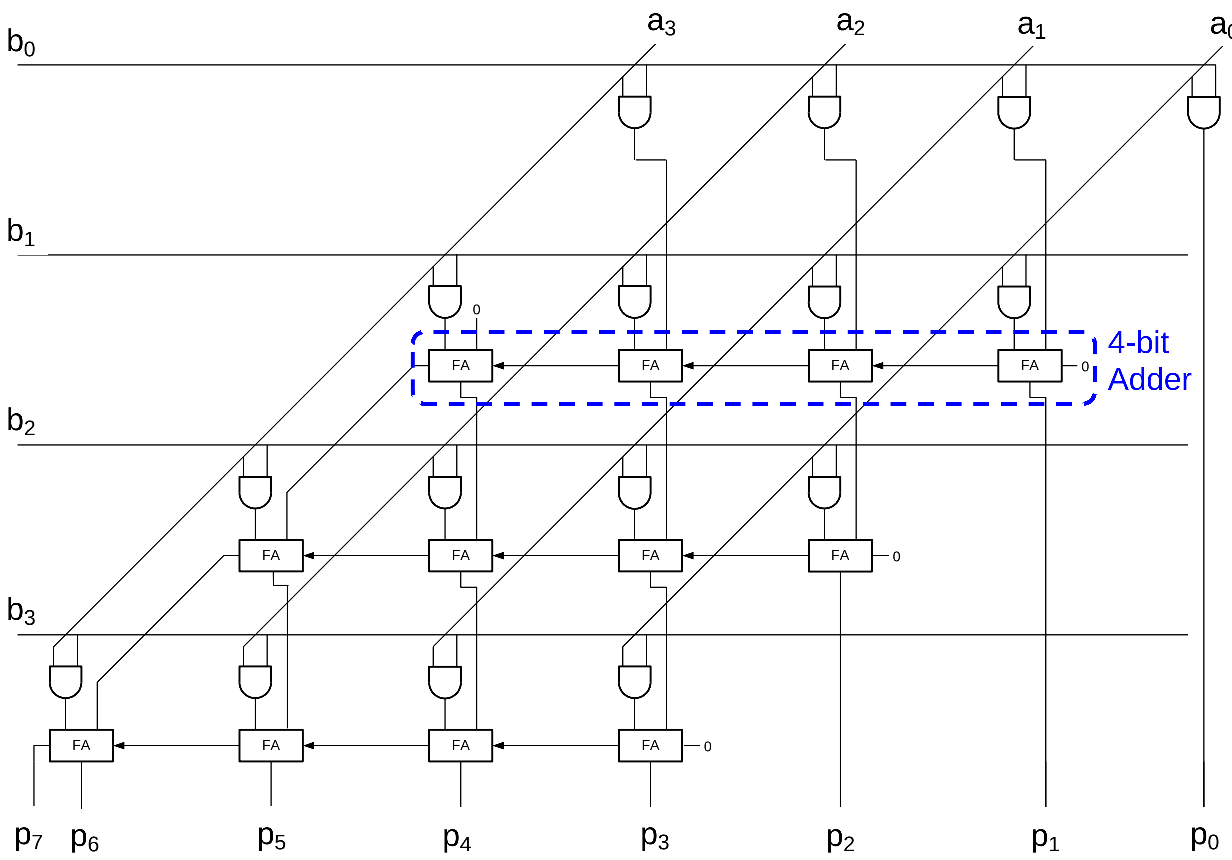
2^2	2^1	2^0	y
0	0	0	1.5*0 +10
0	0	1	1.5*1 +10
0	1	0	1.5*2 +10
0	1	1	1.5*3 +10
1	0	0	1.5*4 +10
1	0	1	1.5*5 +10
1	1	0	1.5*6 +10
1	1	1	1.5*7 +10

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

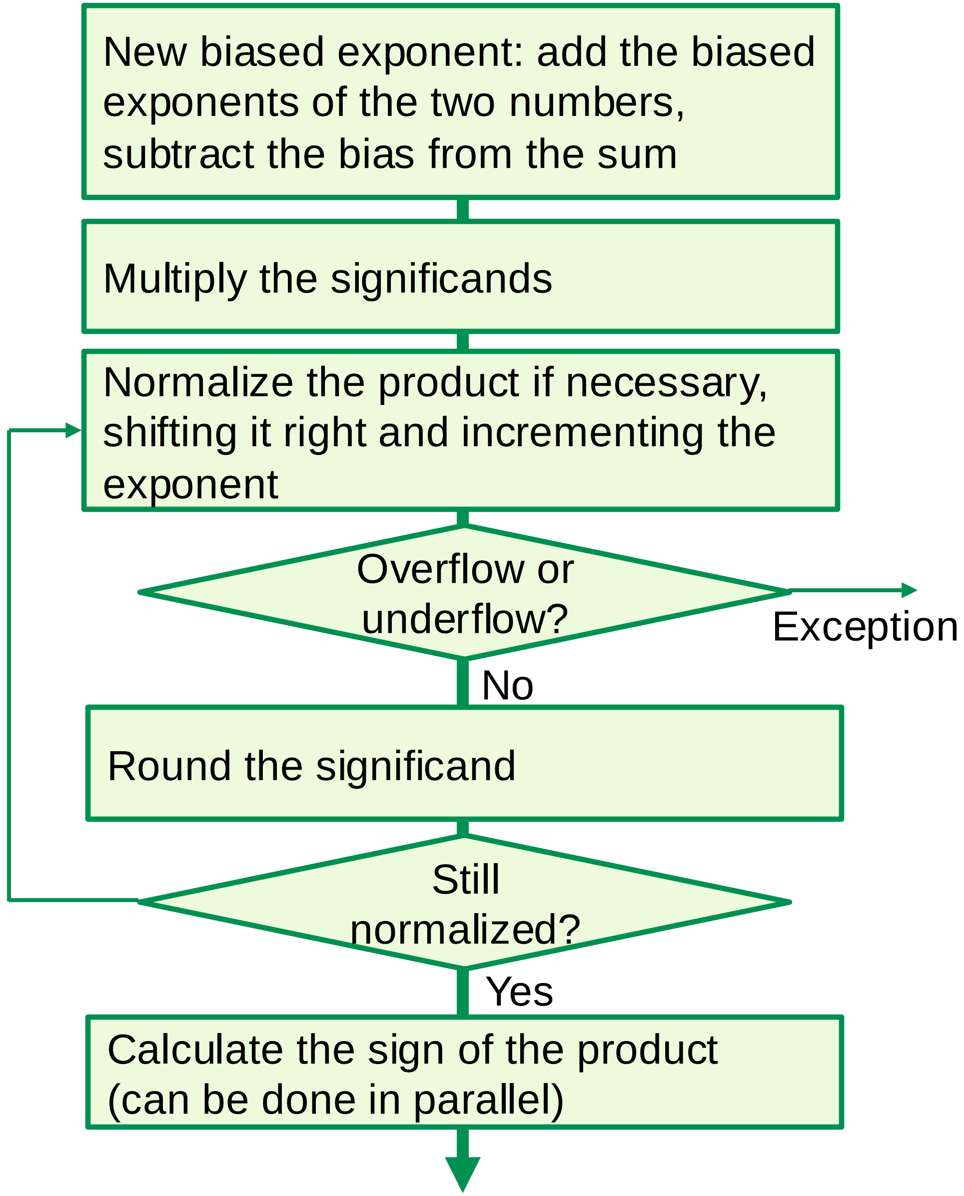
Hardware Arithmetic Consideration

Multiplication

Fixed-Point Array Multiplier



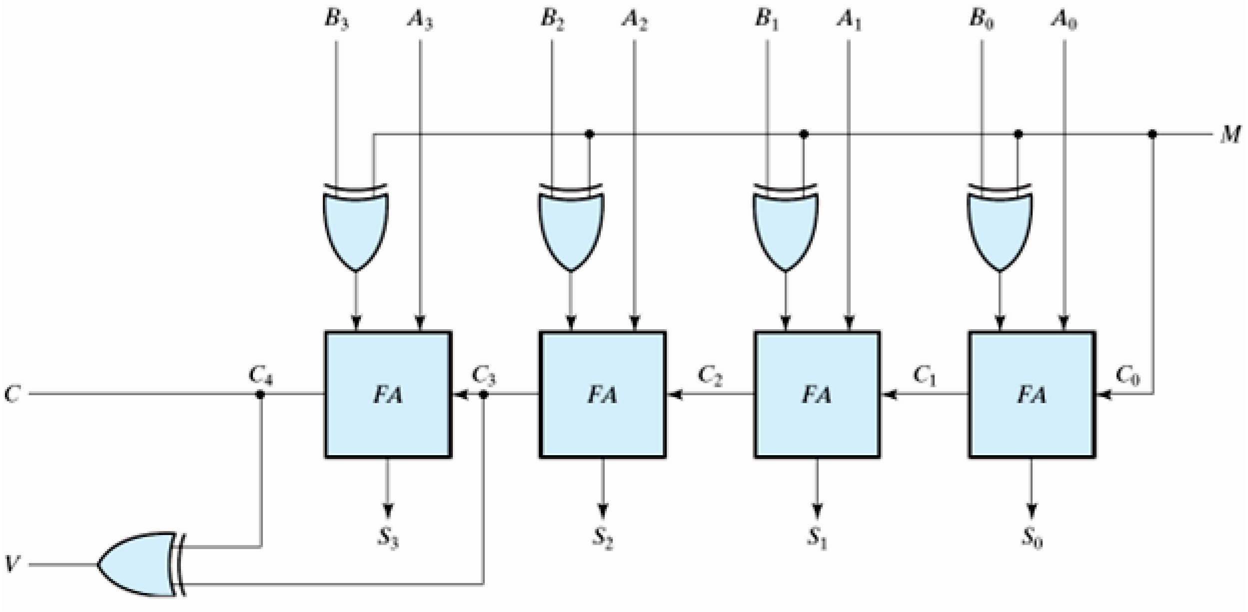
Floating-Point Multiplier



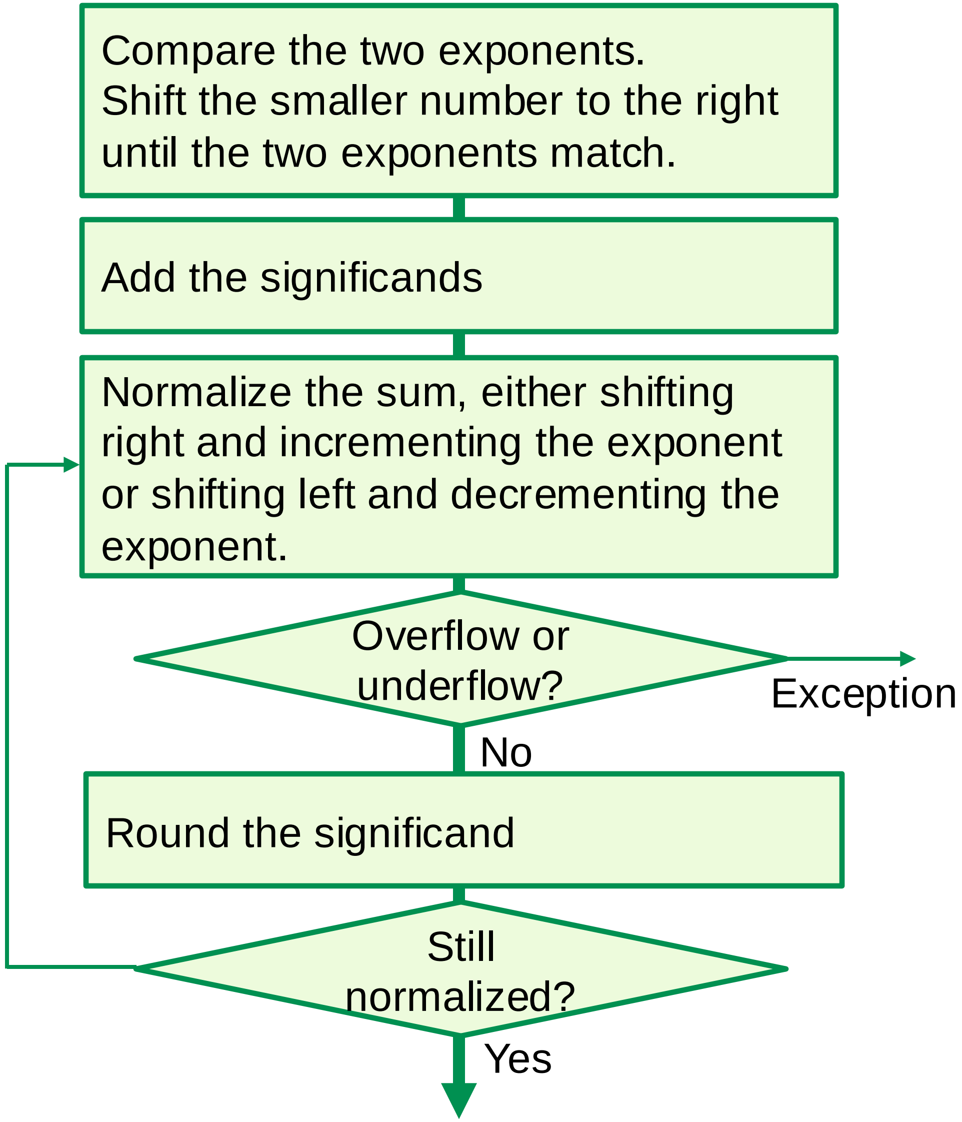
[Logic Design or Computer Architecture or Computer Arithmetic]

Addition

Fixed-Point Ripple Carry Adder



Floating-Point Adder

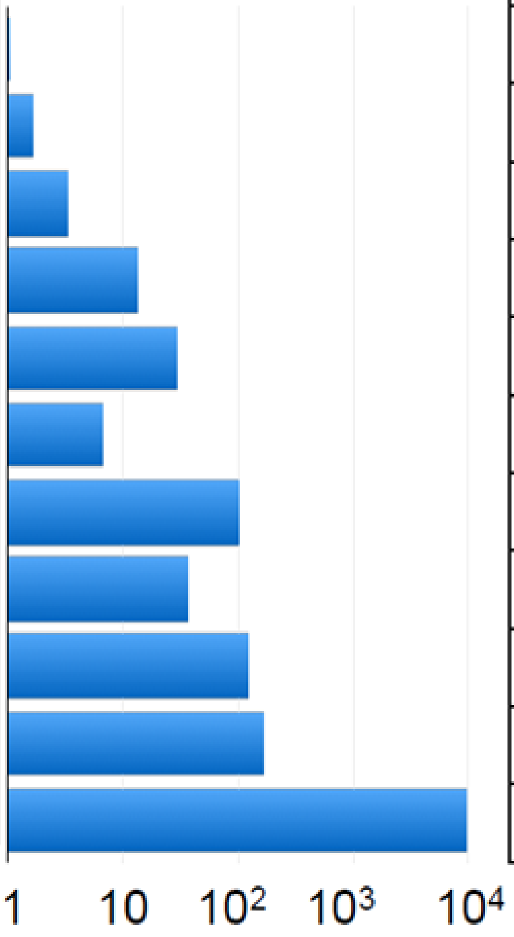


[Logic Design or Computer Architecture or Computer Arithmetic]

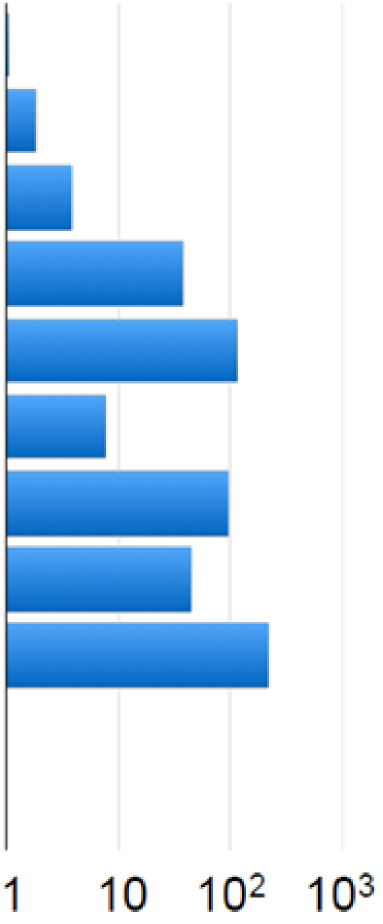
Energy Table for a 45nm CMOS Process

Operation:	Energy (pJ)	Area (μm^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
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

Relative Energy Cost



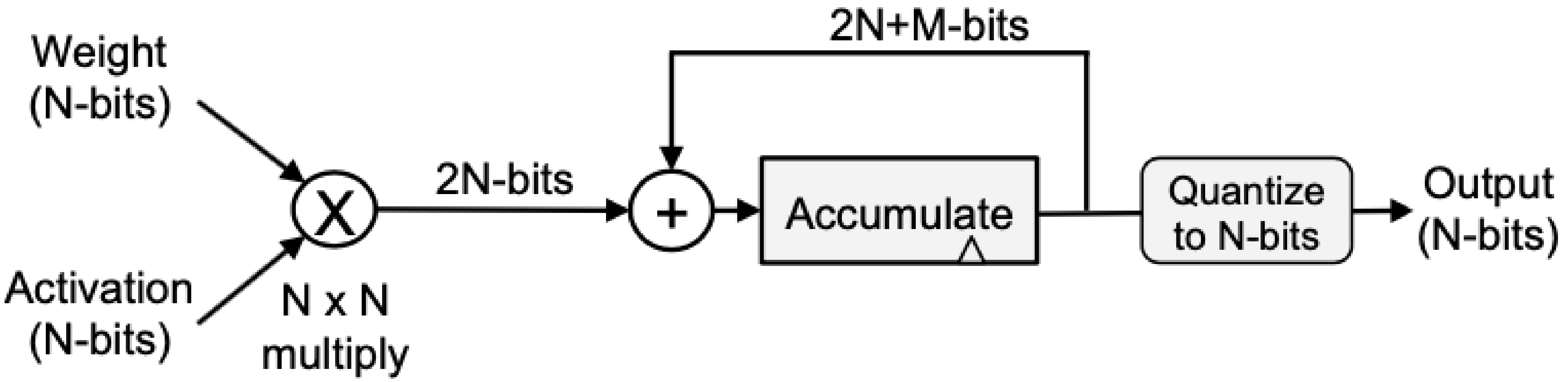
Relative Area Cost



[Horowitz, “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014]

Precision for Multiplication and Accumulation (MAC)

- Accumulation requires higher precision than inputs
- Partial sums



[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Quantization

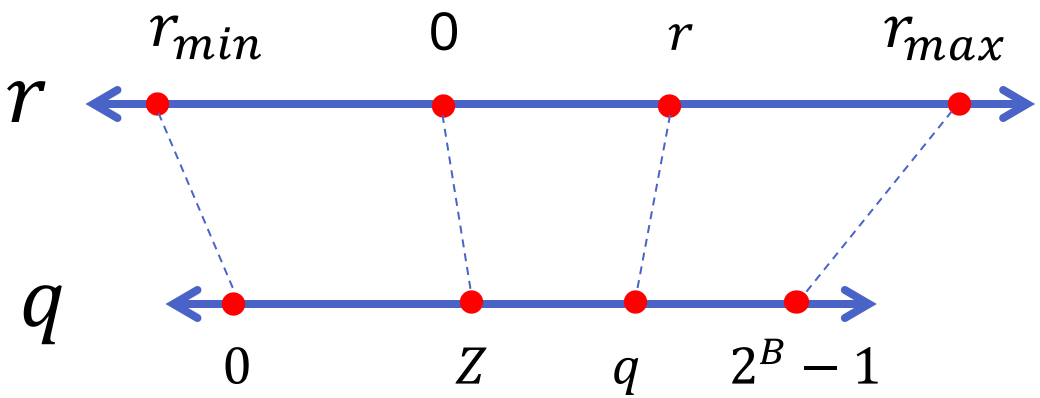
Why Quantization?

- Performing computations and memory accesses with lower precision data
  - Advantage
    - Less computation complexity and less memory accesses
      - Faster
      - Less energy consumption
      - Less hardware area
  - Disadvantage
    - Less accuracy
- E.g., using int8 compared to fp32
  - 4x reduction in model size
  - 2-4x reduction in memory bandwidth
  - 2-4x faster inference (due to faster compute and less memory requirement)
    - The exact speedup depending on the hardware and model
  - Multiplier area becomes order of magnitude smaller

Practical Quantization Methods

- Post-training quantization (PTQ)
  - Mapping the weights and activations from floating point numbers to fixed-point numbers based on the distributions
- Quantization-aware training (QAT)
  - All weights and activations are “fake quantized” in forward/backward passes
    - Floating-point values are rounded to mimic fixed-point numbers
    - But all computations are still done with floating point numbers
  - All the weight adjustments are aware of the later quantization
  - Higher accuracy

Quantization Scheme



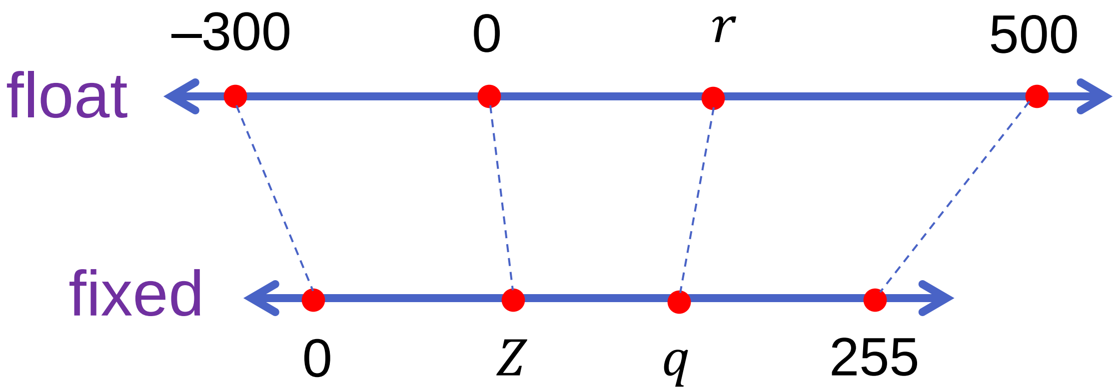
$$S = \frac{r_{max} - r_{min}}{2^B - 1}$$

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

- Floating-point value: r
- Quantized B-bit fixed-point value: q
- Scale (scaling factor): S
- Zero point (bias, offset): Z

- Zero-point: $Z = round(\frac{-r_{min}}{S})$
- Quantized value: $q = \text{round}\left(\frac{r}{S}\right) + Z$
- De-quantized value: $r = S(q - Z)$

Quantization Error



$$S = \frac{r_{max} - r_{min}}{2^B - 1} = \frac{500 - (-300)}{256 - 1} = \frac{800}{255}$$

$$Z = 96$$

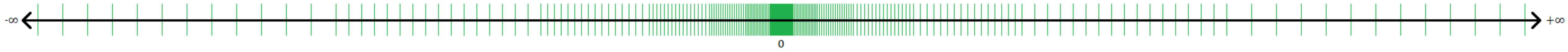
$$r = S(q - Z)$$

Quantization

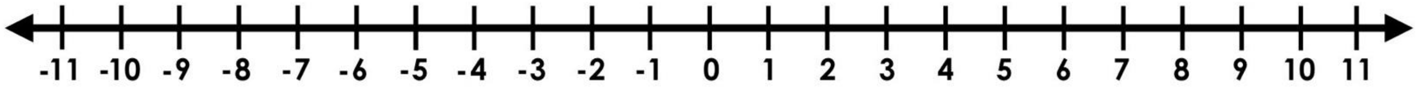
- $Z = \mathrm{round}(\frac{-r_{min}}{S})$
- Quant: $q = \mathrm{round}\left(\frac{r}{S}\right) + Z$
- De-quant: $r = S(q - Z)$

- Quant: $r = 0 \rightarrow q = 96$
- De-quant: $q = 96 \rightarrow r = \frac{800}{255}(96 - 96) = 0$
- Quant: $r = 100 \rightarrow q = \mathrm{round}\left(\frac{100}{S}\right) + 96 = 128$
- De-quant: $q = 128 \rightarrow r = \frac{800}{255}(128 - 96) = 100.39$

Quantization: Floating-Point vs. Fixed Point

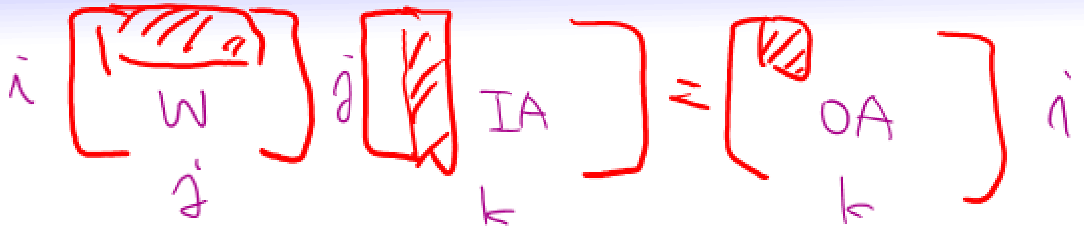


Floating-Point



Fixed Point

Quantization



$r = S(q - Z)$

float $\quad r_{OA}^{(i,k)} = \sum_{j=1}^{N}\left(r_{W}^{(i,j)} \times r_{IA}^{(j,k)}\right)$ for multiplication of two $N \times N$ matrices

Scale $\qquad$ zero point

fixed $\quad S_{OA}\left(q_{OA}^{(i,k)} - Z_{OA}\right) = \sum_{j=1}^{N}\left\{S_W\left(q_W^{(i,j)} - Z_W\right) \times S_{IA}\left(q_{IA}^{(j,k)} - Z_{IA}\right)\right\}$

8b $\qquad$ 8b $\qquad$ 8b

$$q_{OA}^{(i,k)} = Z_{OA} + M\sum_{j=1}^{N}\left((q_W^{(i,j)} - Z_W)(q_{IA}^{(j,k)} - Z_{IA})\right),$$

32b psums

$$M = \frac{S_W S_{IA}}{S_{OA}}, \quad M \in (0,1)$$

$$= 2^{-n} M_0, \quad M_0 \in [0.5,1)$$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference

Quantization Scalar M

$$M = \frac{S_W S_{IA}}{S_{OA}}, \quad M \in (0,1) \quad M \neq 0, M > 0$$

$$= 2^{-n} M_0, \quad M_0 \in [0.5,1)$$

E.g.,

Leading zeros

$$M = 0.\underbrace{0000000}_{\text{7 bits}}\underbrace{11011101010111101 \cdots 010001}_{\text{32 bits}} \cdots$$

fixed point

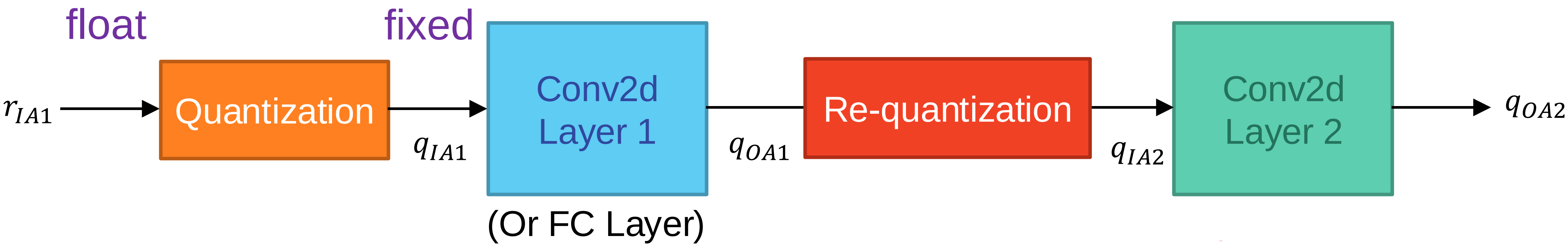
Quantized M_0, e.g., 32 bits

$$= 2^{-7} \times \underbrace{0.11011101010111101 \cdots 010001 \cdots}_{M_0}$$

$M_0, \quad M_0 \in [0.5,1)$

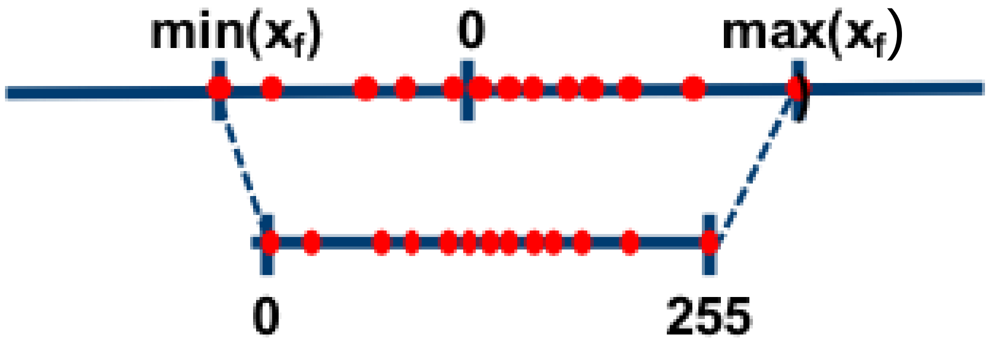
When Cascading Two Layers

- Re-quantization is needed between two layers
  - Because each layer is quantized with different scaling factor and zero point



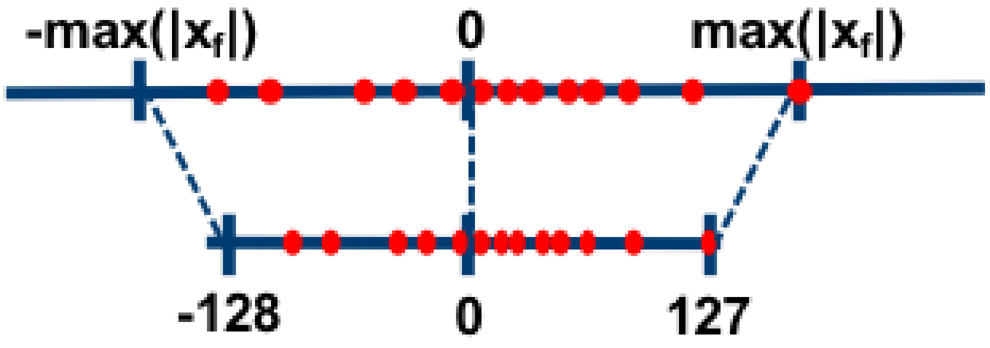
Range-Based Linear Quantization

- Asymmetric



- Unsigned integers
- Dynamic range more efficient
- Quantized result may be more accurate

- Symmetric

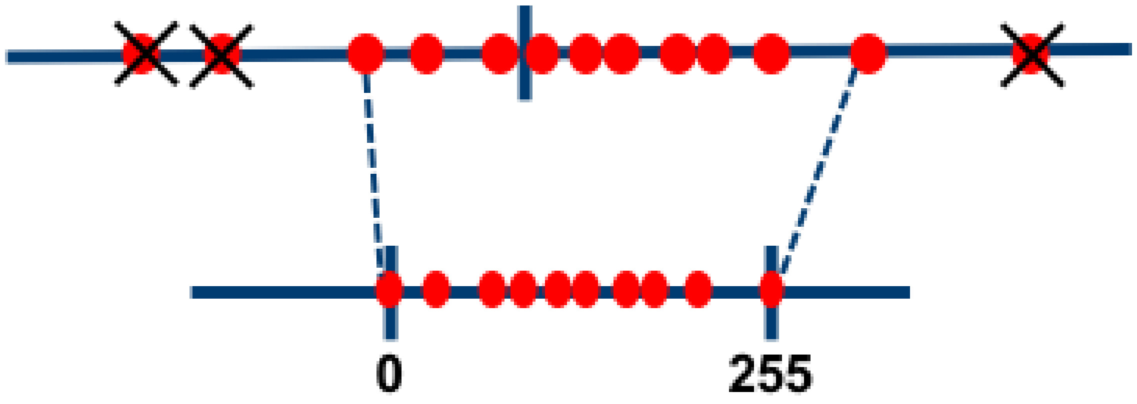


- Signed integers
- No zero point ($Z = 0$)
- Simpler in hardware

Distiller: https://intellabs.github.io/distiller/algo_quantization.html

Clipping of Outliers

- Outliers can be removed for an efficient dynamic range representation
  - Can be saturated to min/max values
  - Good for CNN; bad for Transformer



Distiller: https://intellabs.github.io/distiller/algo_quantization.html

Rounding

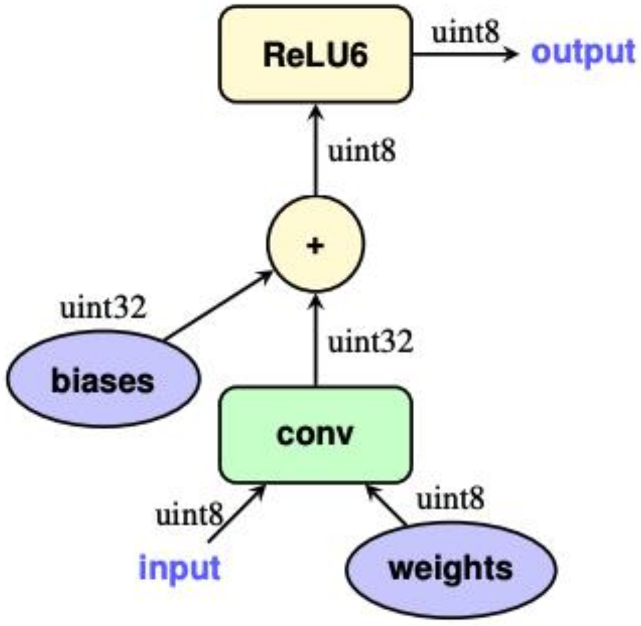
- Different conversions from floating-point numbers to fixed-point representation

	Round-to-Nearest	Round-Down	Round-to-Even
17.5			
18.5			

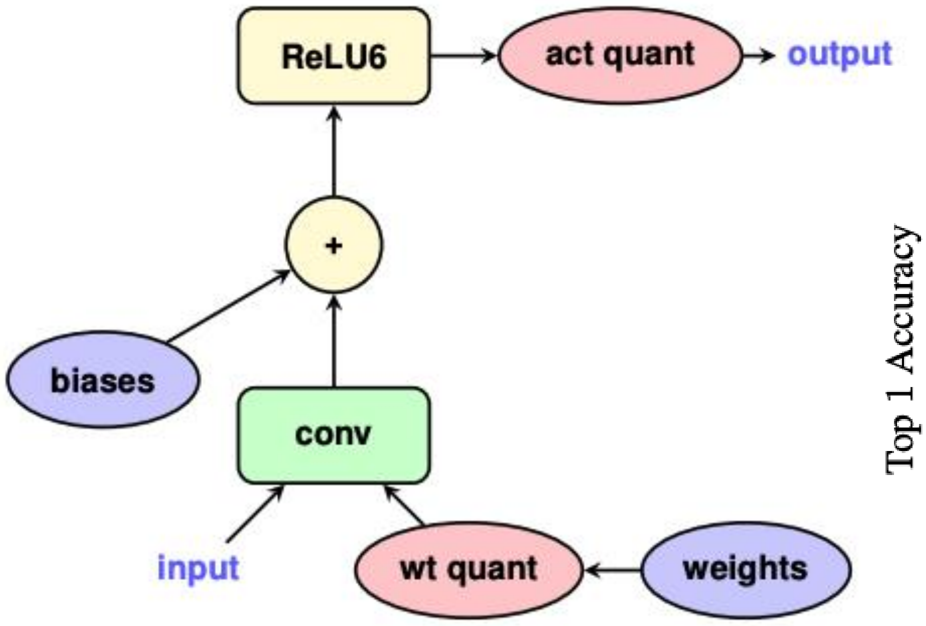
- Round-to-even scheme is common in DNN computing
  - Mean error expected to be 0
    (an equal chance of being even or odd)
  - Otherwise, the predictions will get shifted up

Quantization-Aware Training (QAT)

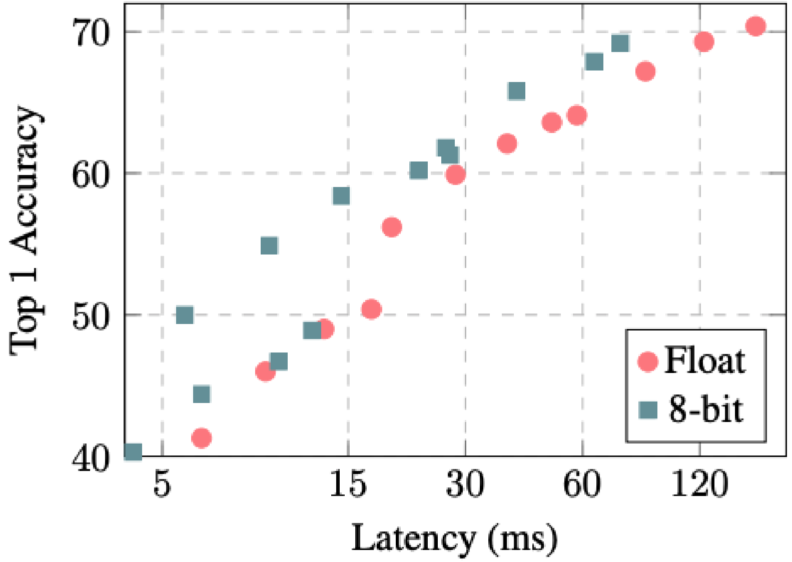
- Typically performs better than post-training quantization
- Emulate quantization effects in the forward pass
- Update weights and biases in floating point during backpropagation



(a) Integer-arithmetic-only inference



(b) Training with simulated quantization



Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference

Summary

- Number representation is more important than you think
  - Number with larger bit widths
    - Multiplication takes longer time (latency)
    - Requires larger memory and larger compute unit (cost)
    - Requires more energy due to more data movement and memory access (energy)
- Float-point number formats
- Different fix-point number formats
- Rounding
- Quantization-aware training vs. Post-training quantization