Low Precision Computation for Deep Neural Networks

Naigang Wang

IBM T. J. Watson Research Center

November 17, 2022

Outline

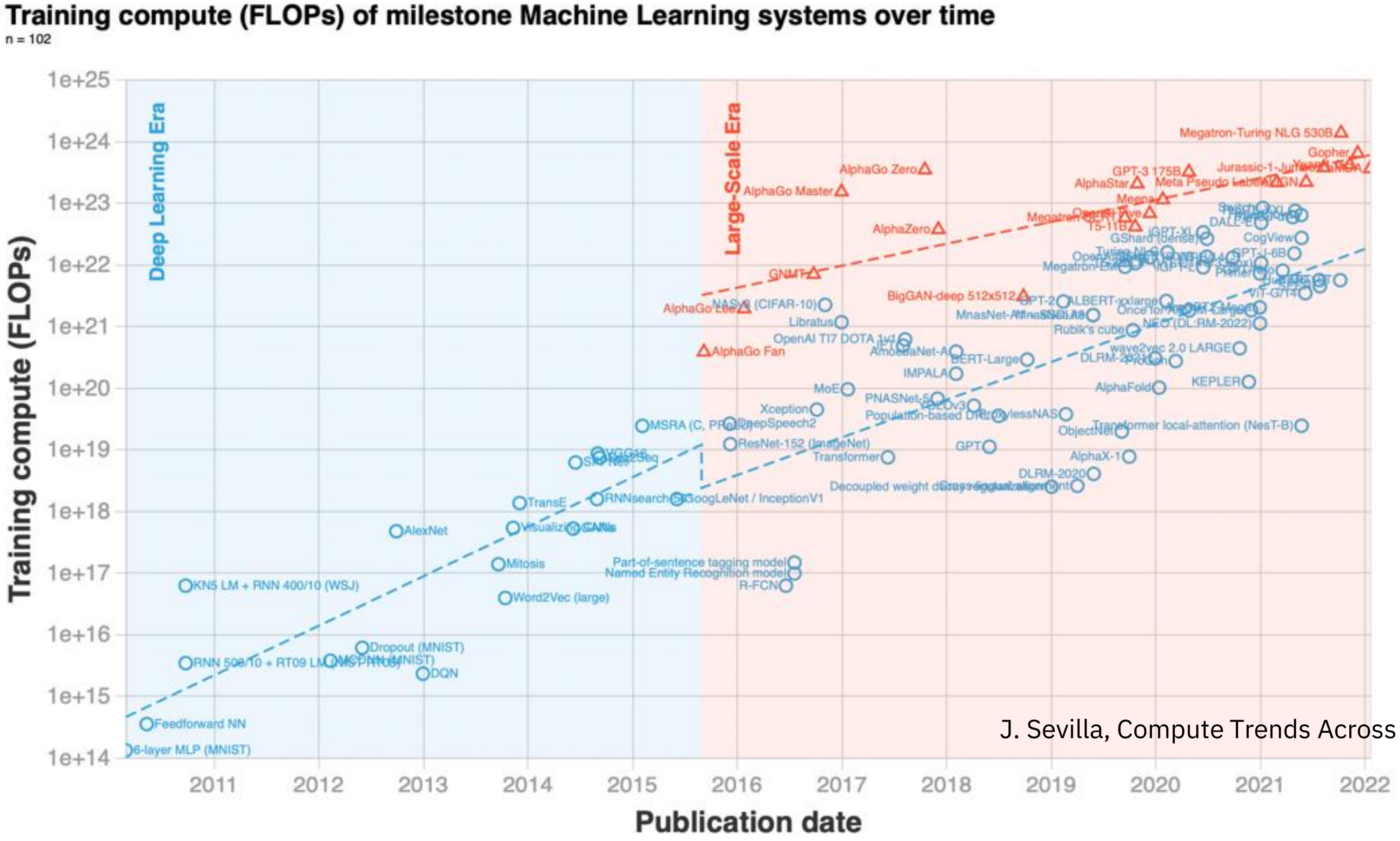
- **Why low-precision approximation?**

- **Data Format**
  - Floating point
  - Fixed point

- **Low precision training**
  - GEMM Multiplier
  - Accumulation

- **Low precision inference**
  - Quantizers
  - Quantization aware training
  - Post training quantization

AI Compute Growth



J. Sevilla, Compute Trends Across Three Eras of Machine Learning, arXiv' 22

Deep learning models grow deeper and larger, necessitating 100s of Zetta-Ops of computations and Tera-bytes of storages for model and data.

AI Compute Growth

- **Extraordinary overheads to train. For example, to train GPT3-like language models with ~170B parameters**
  - **Time**: ~3 months w/ 1000s of GPUs
  - **Cost**: 2-5M $ w/ cloud computing service
  - **Carbon footprint**: 552.1 tCO_2e compared to 180t for plane round trip from NY-SF (D. Patterson, arXiv'21)

- **Challenging to deploy on edge devices.**
  - Limited memory
  - Latency

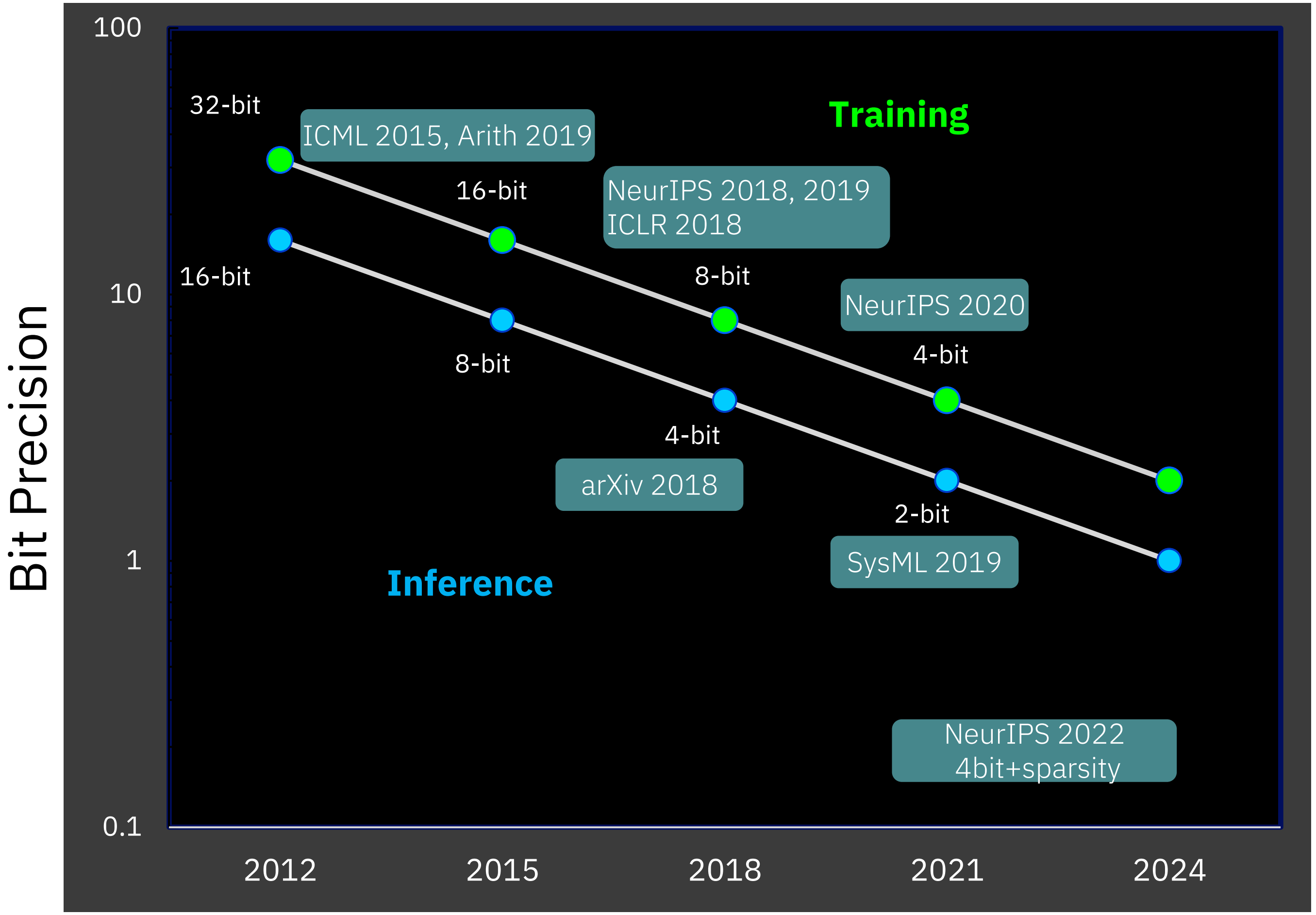
- **Need to make AI compute faster and more efficient!**

AI Acceleration

- **An equivalent smaller model:**
  - Knowledge distillation: distill a large teacher model to a smaller student model
  - Pruning (sparsity): remove less important weights/activations

- **Approximate computing**
  - SVD: low-rank matrix approximation
  - **Reduced precision**: use a subset of numbers to represent a whole distribution.
    - Trading off the inherent resilience of machine learning algorithms for improved computation efficiency.
    - Currently the most widely used technique for DNN acceleration in commercial hardwares.
    - Advantages:
      - Better usage of cache and reduce bandwidth bottleneck
      - **Quadratic improvement** with precision reduction – applies to both training and inference
    - Critical requirement: Maintain model accuracy (vs. single precision FP32)

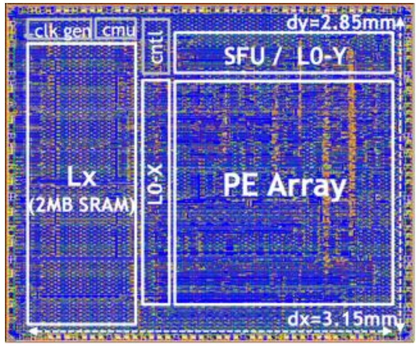
IBM Research Leadership in Reduced Precision

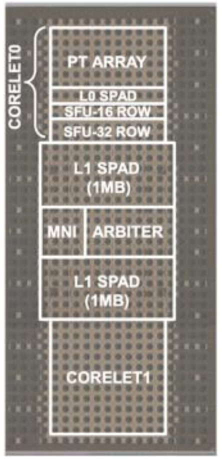


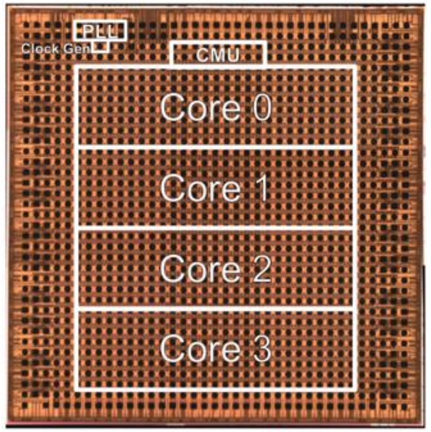
IBM Research has been at the forefront of every major technical advancement on bit-precision scaling.

IBM Research AI Accelerator

IBM







Gen 1
GF 14LPP
VLSI 2018

Gen 2
GF 14LPP
VLSI 2020

Gen 3
Samsung 7LPP
ISSCC 2021

Artificial Intelligence Unit (AIU)
Announced Oct. 2022

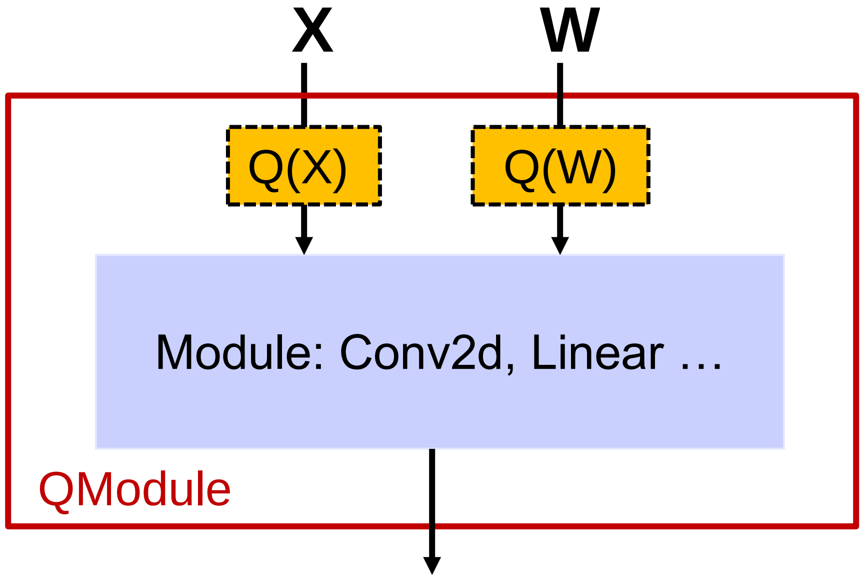
- **Leveraging reduced precision advancements,**
  - **3 generations of AI cores have been developed.**
  - **Recently announced System-on-Chip design.**

Quantization Emulation in ML framework

- **Goal:**
  - Emulate reduced precision compute in ML framework to check the impact on model accuracy.
  - Develop error tolerate algorithms to recover accuracy loss.

- **Implementation**
  - Quantizer: a function to convert a tensor from high precision to low precision.
  - Precision conversion may happens at CUDA level

- **Matrix multiply (gemm) dominates computation**



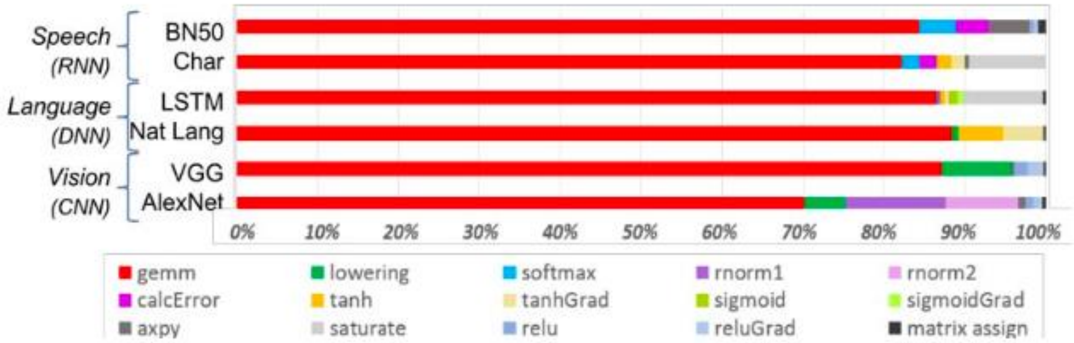
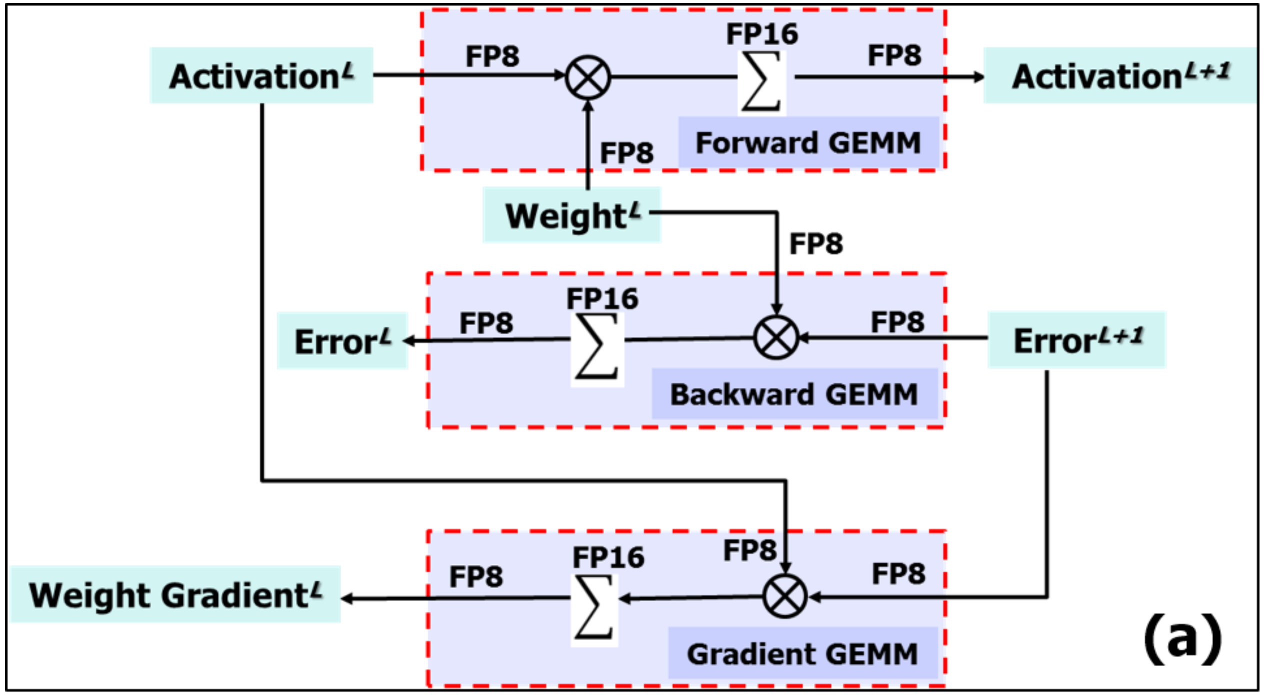


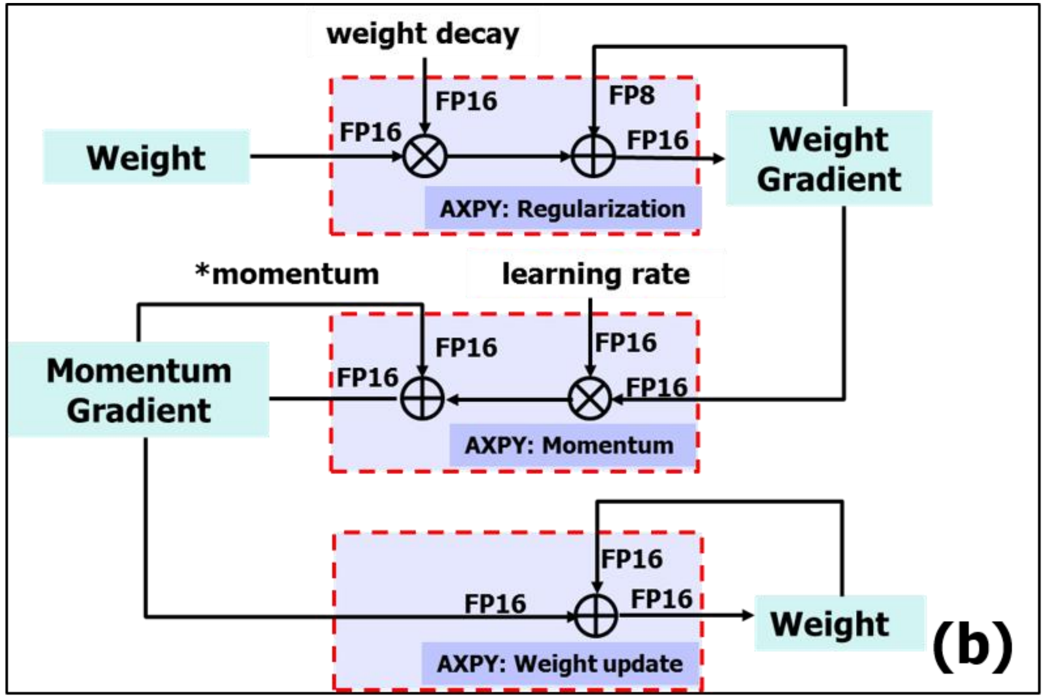
Fig. 1. Profiling result for various NN.

S. Shukla, et al, “A Scalable Multi_TeraOPS Core for AI Training and Inference”” ISSC2018

GEMM

- **Training**
  - FWD/BWD/GRAD
  - AXPY weight update in optimizer
  - Communication for distributed learning
- **Inference**
  - FWD





GEMM:

❑ 3 GEMM functions

$$\begin{aligned} FWD&: \quad x \cdot w = y \\ BWD&: \quad dy \cdot w = dx \\ GRAD&: \quad dy \cdot x = dw \end{aligned}$$

Momentum SGD:

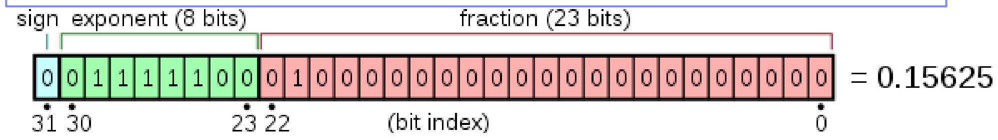
❑ 3 AXPY functions

$$\begin{aligned} dw &= dw + \lambda_w \cdot w \\ dw_{old} &= m \cdot dw_{old} + \eta dw \\ w &= w + dw_{old} \end{aligned}$$

Data Format

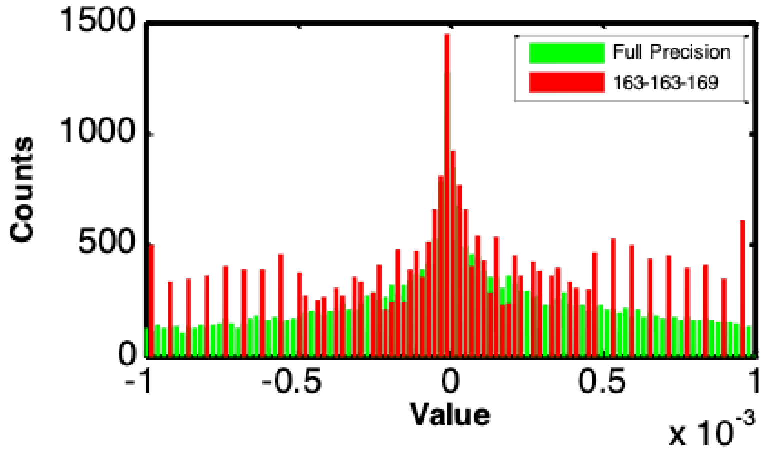
- **Floating point**
  - Format: a signed bit (S), exponent bits (E), and mantissa bits(M), $(-1)^S \times 2^E \times (1 + M)$

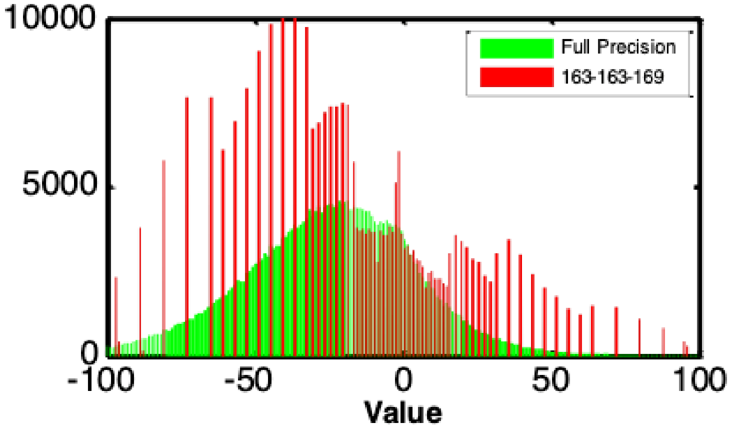
<1,8,23>:   <sign: 1, exponent: 8, mantissa: 23>



https://en.wikipedia.org/wiki/Floating-point_arithmetic

  - Operation: $c \leftarrow c + a \times b$
  - Reduced precision error
    - Give a format, dynamic range is determined
      - Overflow: → largest number
      - Underflow: → 0
    - Rounding: nearest rounding; stochastic rounding
  - Primarily used for training due to large dynamic range



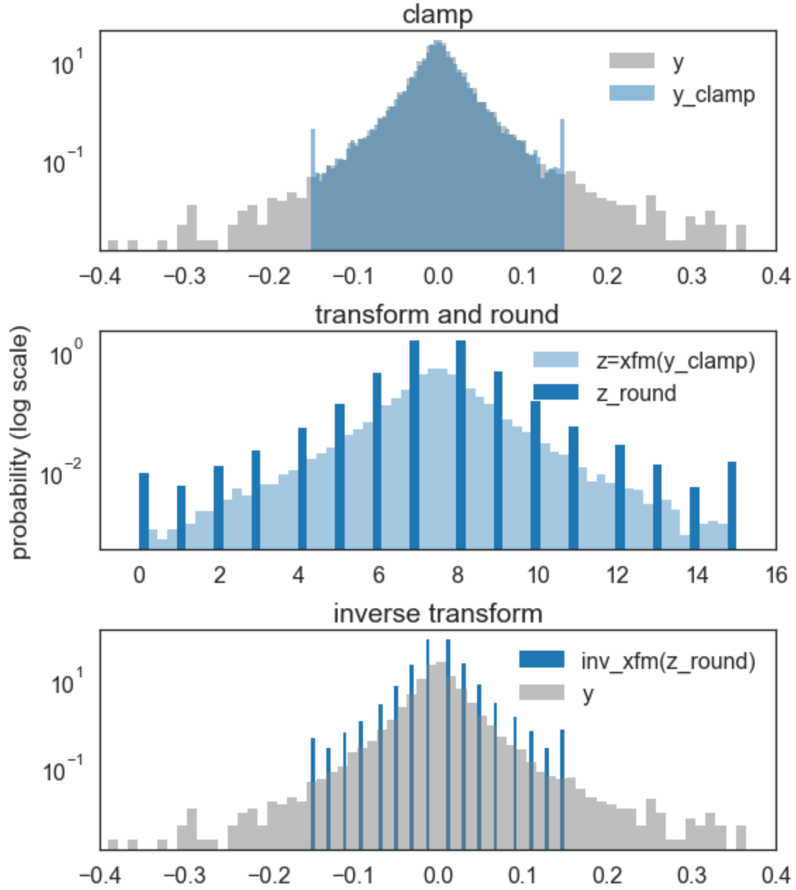


Data Format

- **Integer**
  - Quantize and de-quantize
    - Scale
    - Zero-point
  - Uniform vs. non-uniform
    - Uniform is hardware friendly.
  - Primarily used for inference due to highly efficient hardware design

Operation	MUL	ADD
8bit Integer	0.2pJ	0.03pJ
32bit Integer	3.1pJ	0.1pJ
16bit Floating Point	1.1pJ	0.4pJ
32tbit Floating Point	3.7pJ	0.9pJ

(Horowitz, 2014)



$$z_{INT} = \left\lfloor \frac{clamp(y, \alpha_l, \alpha_u) - zp}{\Delta} \right\rceil$$

$$y_q = z_{INT} * \Delta + zp$$

Low Precision Training

- **Notes and Challenges:**
  - Need to quantize tensors with a large range of dynamic range (weight, activation and gradients)
  - Multi-type of ops (Multiply, Add, AXPY…)
  - Implementation often in CUDA kernels to access bottom ops (slow emulation).

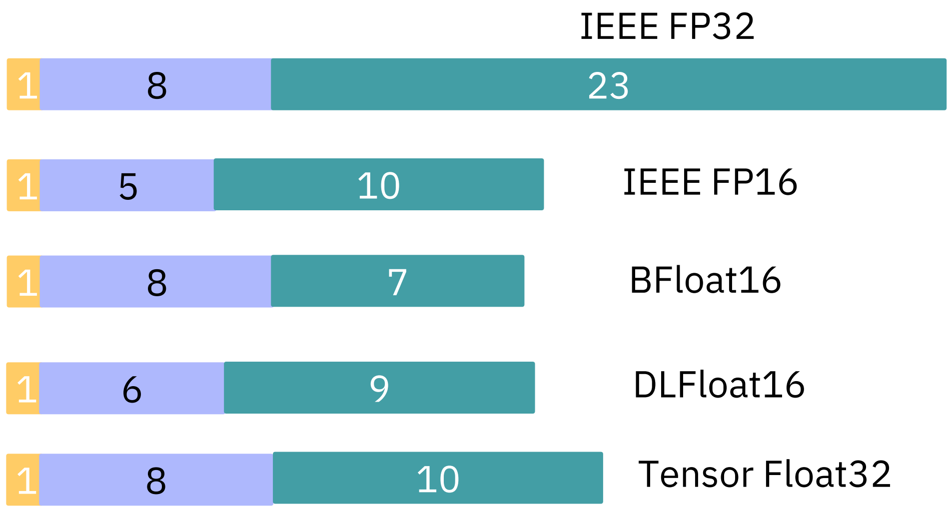
- **Status**
  - FP16 becomes mainstream particularly for large model training
  - FP8 start to emerge in commercial devices (IBM Sentient Core, Nvidia Hopper)
  - 4-bit training has been demonstrated.

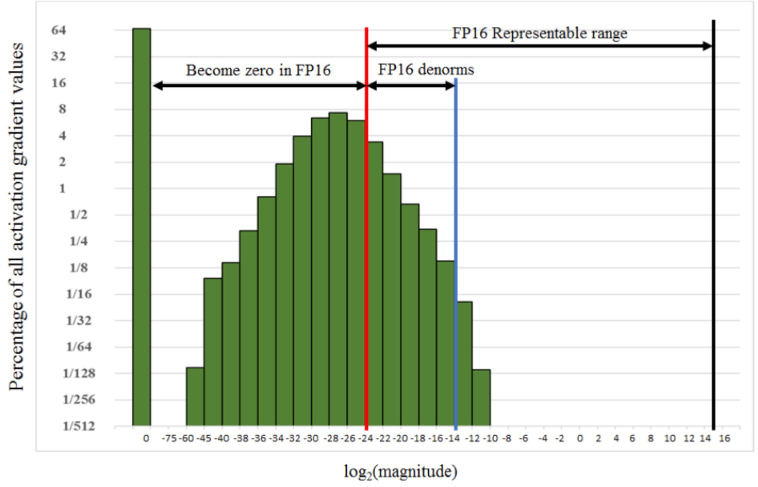
FP16 Training (>4X the performance of FP32)

- **3 Different formats for FP16**
  - IEEE FP16
    - Introduced in NVidia V100 in 2017 for Training & Inference
    - Supports a dynamic range of +/- 2^{-24} to 1.999×2^{15}
    - Challenge: Insufficient dynamic range for gradients: need to ideally represent values 2^{-30}
    - HW Challenge: supporting denormal numbers is more expensive in the FPU
    - Loss scaling techniques

$$(-1)^S \times 2^{-127} \times (0.M)$$

  - BFloat16
    - Introduced by Google in TPUs for Training & Inference
    - Supports a dynamic range of +/- 2^{-127} to $1.9999...... \times 2^{127}$
    - No loss scaling needed
    - Limited mantissa precision implies accumulations need to be in fp32 (more expensive in hardware)
  - DLFloat16
    - Introduced by IBM in AI Hardware (http://www.lab3.kuis.kyoto-u.ac.jp/arith26/slides/session4/4-3.pdf)
    - Allows for accumulations in FP16 while fully preserving accuracy.





https://developer.nvidia.com/automatic-mixed-precision

FP8 Training (2-4X improvement over FP16)

- **FP8**
  - First demonstrated in NeurIPS2018 (https://proceedings.neurips.cc/paper/2018/file/335d3d1cd7ef05ec77714a215134914c-Paper.pdf)
  - Weights, Activations and Gradients (W, A, G) are all represented in FP8 format
  - Demonstrations: AlexNet, ResNet18/50, speech models w/ <0.5% accuracy loss.

1 | 5 | 2 DLFloat8

- **Hybrid FP8**
  - Original FP8 show degradation for challenging model such as MobileNets and Transformers.
  - Hybrid solution
    - NeurIPS2019 (https://proceedings.neurips.cc/paper/2019/file/65fc9fb4897a89789352e211ca2d398f-Paper.pdf)
    - W & A need higher fidelity but less range – represent with (1,4,3)
    - G needs higher range but lesser fidelity – represent with (1,5,2)
  - Demonstrated in IBM hardware @ ISSCC 2021. Recently introduced by GPUs (Hopper)

1 | 5 | 2 Gradients

1 | 4 | 3 Weight&Activations

4-bit Training

- **IBM work on 4-bit training:**
  - https://proceedings.neurips.cc/paper/2020/file/13b919438259814cd5be8cb45877d577-Paper.pdf
  - Key ideas: New 4-bit floating-point representation (FP4) with radix 4 to represent gradients, novel adaptive gradient scaling technique and 2-phase rounding schemes
  - Hardware: **7X** improvement possible over FP16 optimized HW
  - Challenges : Some loss in accuracy (~ 2-3% for many models)
- **Additional work on low-precision training at NeurIPS 2020**
  - https://proceedings.neurips.cc/paper/2020/file/099fe6b0b444c23836c4a5d07346082b-Paper.pdf
  - Presents a statistic framework for analyzing low-precision training.
  - 2 novel gradient quantizers
  - 5-bit gradient : only 0.5% loss in ResNet50
- **In general, low-precision training at 4-bits remains an active area of research**

FP4

Table 2: ImageNet test accuracies using 4-bit training

FWD	FP32	INT4	INT4	INT4	INT4	INT4
BWD	FP32	FP32	TPR FP4	dx(FP4) dW(FP8)	TPR FP4 bot1x1(FP8)	TPR FP4 1x1(FP8)
Alexnet	57.56	57.51	**56.38**	57.11	-	-
ResNet18	69.40	69.43	**68.27**	68.99	-	-
ResNet50	76.48	75.76	74.01	74.92	74.99	**75.51**
MobileNetV2	71.85	69.77	68.85	69.65	-	-

FQT

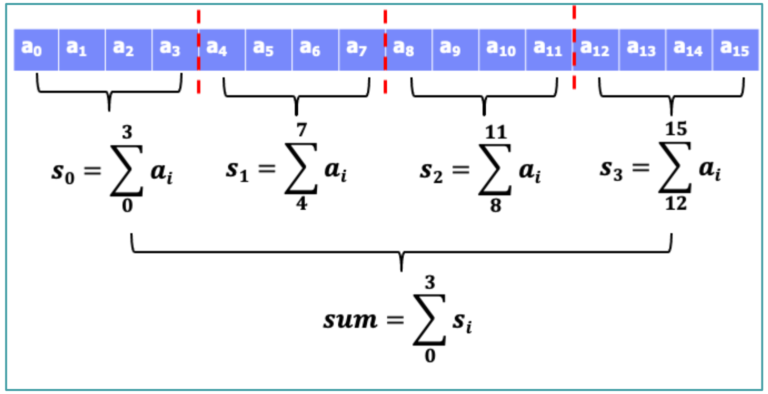
ResNet50

PTQ	PSQ	BHQ
77.09 (1.75)	–	–
77.35 (1.78)	–	–
76.40 (1.81)	**77.40 (1.77)**	77.36 (**1.77**)
76.62 (1.80)	**77.36 (1.77)**	76.96 (1.78)
76.06 (1.84)	76.97 (1.79)	**77.25 (1.78)**
74.62 (1.93)	76.30 (1.85)	**76.83 (1.81)**
diverge	73.78 (2.04)	**74.75 (1.96)**

Low Precision Accumulation

- **<u>Accumulation error (Swamping)</u>:** Floating point addition involves right-shift of the smaller operands by the difference in exponents. In case of large-to-small number addition, small numbers maybe partially or completely truncated causing information loss.

Ex: $1.0000 + 0.0002 = 1.000\cancel{2}$



Chunk-based accumulation

- **Partial Sum ($c \leftarrow c + a \times b$)**
  - Chunk-based accumulation
    - ICLR2019, https://arxiv.org/abs/1901.06588
- **AXPY ($\boldsymbol{W \leftarrow W + lr \times dW}$)**
  - Floating point stochastic rounding can effectively recover the information loss due to accumulating small-magnitude gradients to large-magnitude weights in reduced precision floating point.

$$Round(x) = \begin{cases} s \cdot 2^e \cdot (1 + \lfloor m \rfloor + \epsilon) & \text{with probability } \frac{m - \lfloor m \rfloor}{\epsilon}, \\ s \cdot 2^e \cdot (1 + \lfloor m \rfloor) & \text{with probability } 1 - \frac{m - \lfloor m \rfloor}{\epsilon}, \end{cases}$$

Low Precision Inference

- **Notes and Challenges:**
  - Precision used in training can be used for inference (FP16/8).
  - Can be used for different hardware that support INT engines, e. g. CPUs.
  - Challenging to main the model accuracy at ultra-low precision (INT4 or less).
- **Techniques (for uniform quantization):**
  - QAT and PTQ
  - Quantizers
- **Status**
  - INT8 has been wide deployed.
  - INT4 has been demonstrated of working well for most popular models.
  - 2bit or 1bit is actively studied.
    - Multiply is NOT needed (shift or lookup)
    - However, still incur accuracy loss (can be used in extreme resource constrained devices)

QAT vs. PTQ

- **Post-Training Quantization:**
  - Directly convert the weights and convert activations on-the-fly.
  - Or minimize Mean Square Error (MSE) of a single tensor or output of a layer.
  - Pros and Cons
    - No data needed or only a small amount of unlabeled data.
    - Often incur large accuracy loss.

$$\min(MSE(x, xq));$$
$$\min(MSE(w, wq));$$
$$\min(MSE(xw, wqxq));$$

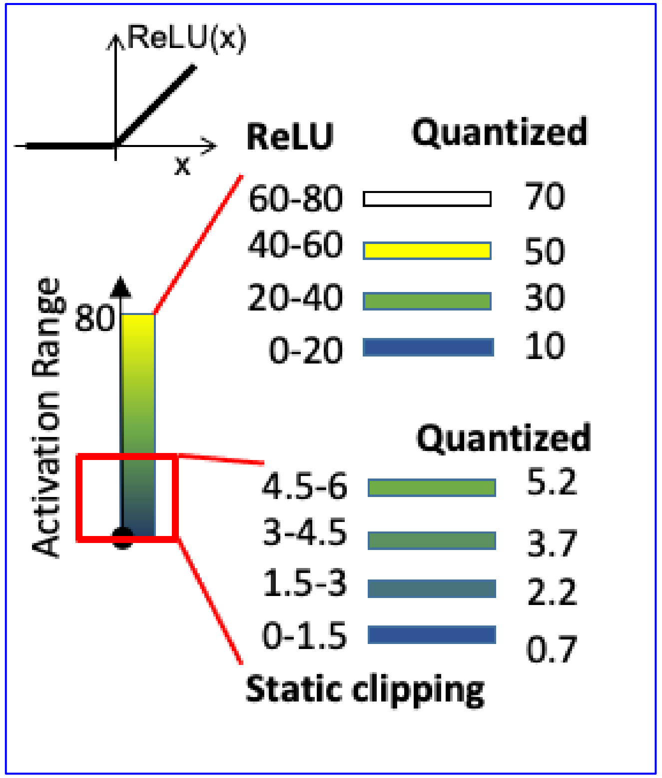
- **Quantization-Aware Training:**
  - Training with weights & activations in INT8 in the forward pass – network absorb quantization error to achieve best performance.
  - Pros and Cons
    - Best performance at low precision
    - Require labeled training dataset

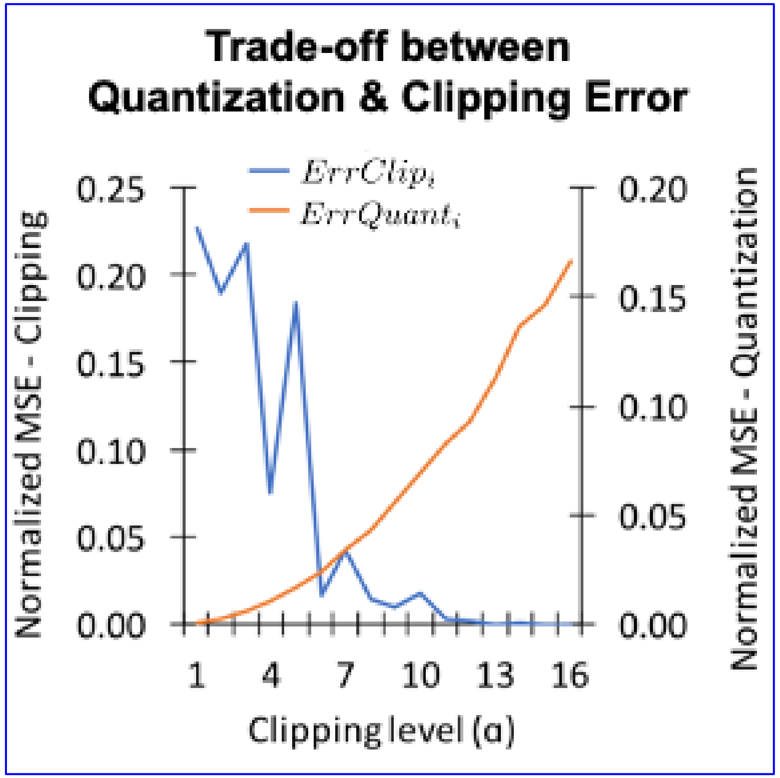
$$f(x + \Delta x, w + \Delta w) \approx f(x, w)$$

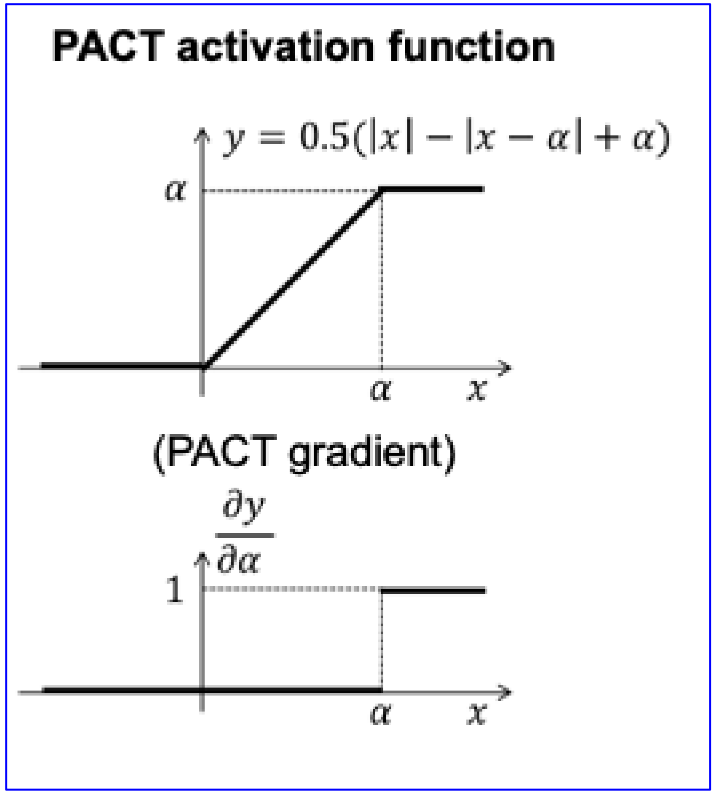
PArameterized Clipping acTivation (PACT) quantizer

- Clipping level (= α) as a *trainable parameter* → Auto-tuned by Backprop
- Improved versions: LSQ, LSQ+, PACT+

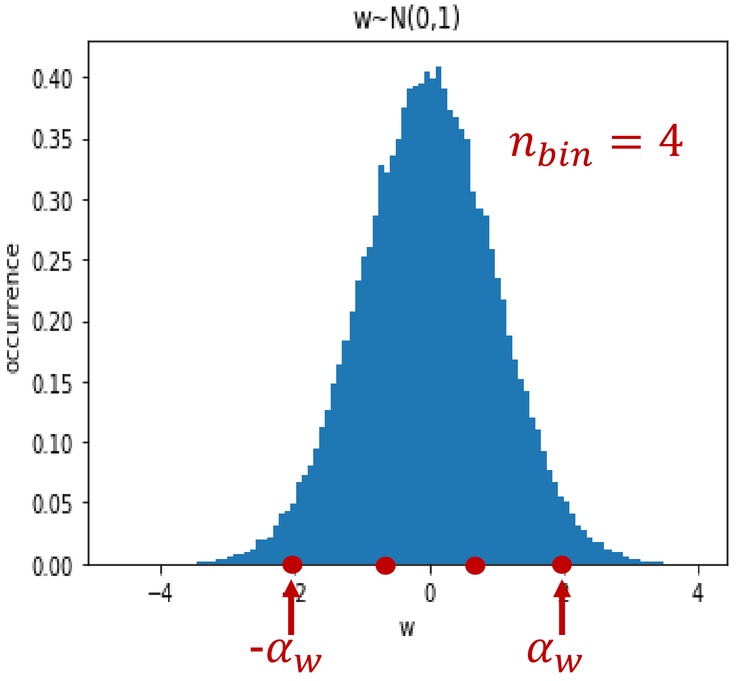
Hard to find sweet spot → *Automatic tuning via training!*

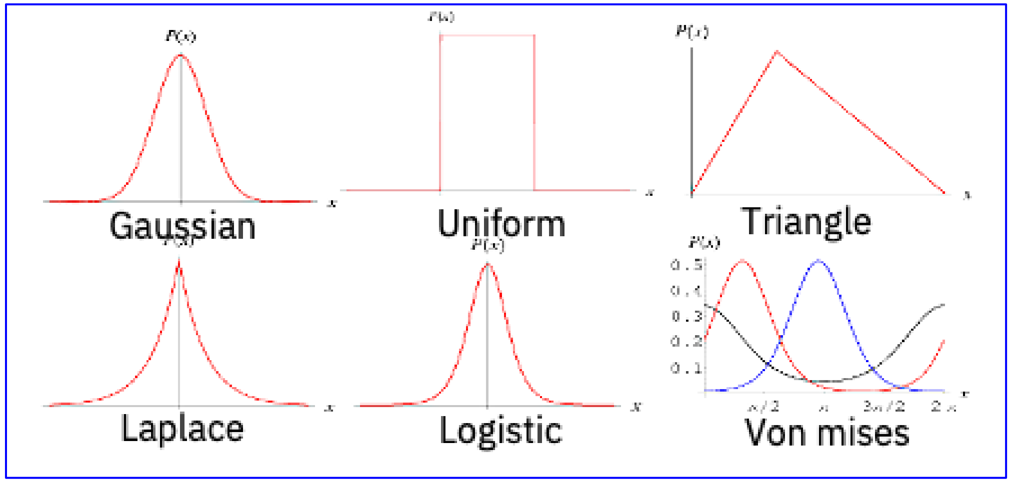


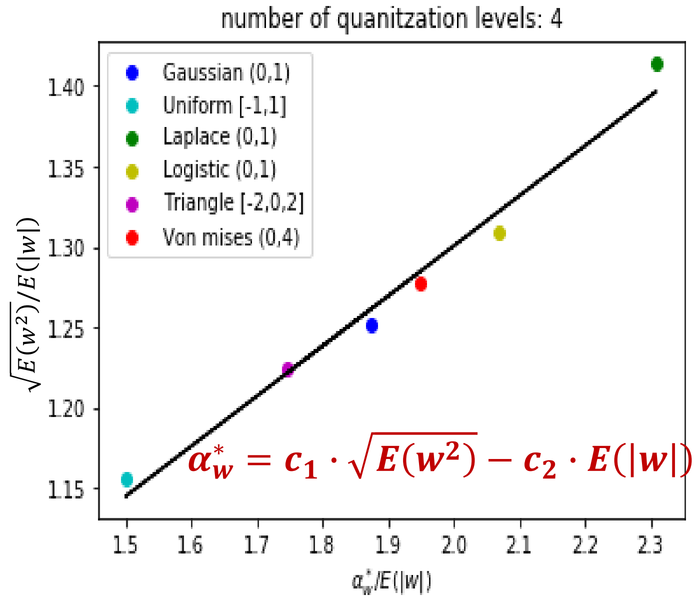




Statistics Aware Weight Binning (SAWB) weight quantizer







How to better capture shape of weight?

- $E(|W|)$ captures the ***representative values***
- $E(W^2)$ captures the ***overall shape***
- Use $E(|W|)$ and $E(W^2)$ sampling from 6 standard distributions to find the best α_w

Summary

- Quantization is one of the most effective techniques to accelerate DNN computation models.

- Low precision floating point (FP16/8/4) for training and how to minimize the impact of Representation error and accumulation errors

- Low precision integer point (INT8/4/2) for inference. PTQ and QAT approaches and PACT and SAWB quantizers.

- Quantitation is still one of the most active research topics, particularly at the age of large-scale models.