6.5930/1

Hardware Architectures for Deep Learning

Co-Design of DNN Models and Hardware: Precision

April 17, 2024

Joel Emer and Vivienne Sze

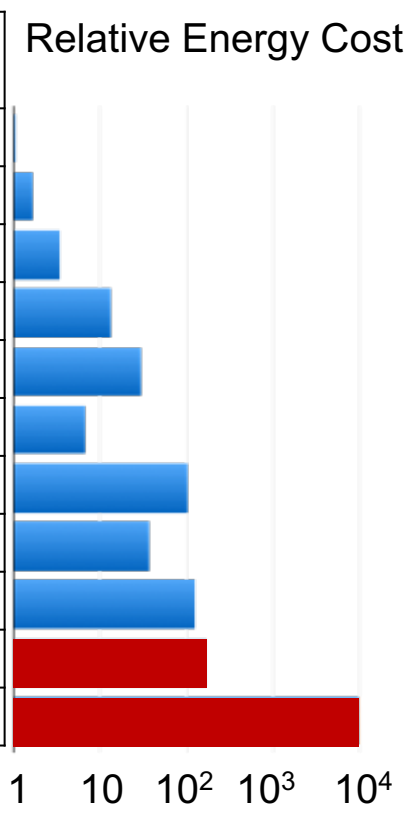
Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Lecture

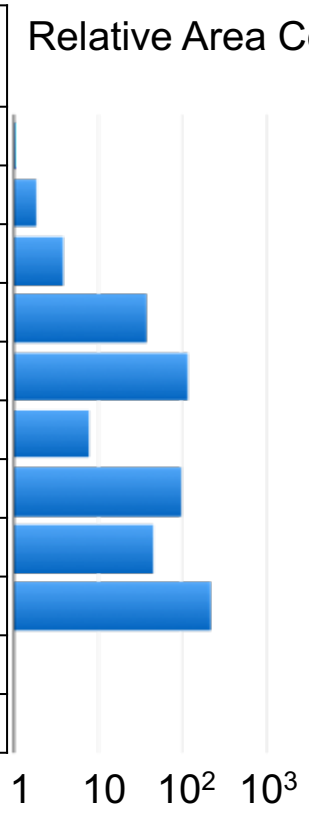
- Lectures on the **co-design of the DNN models and the hardware**
  - Better than what each could achieve alone
  - Unlike previously discussed approaches, these approaches can affect accuracy! → Evaluate tradeoff between accuracy and other metrics
- Co-design approaches can be loosely grouped into two categories:
  - **Reduce *number* of operations** for storage/compute (Sparsity [Ch. 8] and Efficient Network Architectures [Ch. 9])
  - **Reduce *size* of operands** for storage/compute (Reduced Precision [Ch. 7])
- Hardware support required to maximize savings in latency & energy
  - Ensure that overhead of hardware support does not exceed benefits

Benefits of Reduced Precision

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640



Area (μm^2)
36
67
137
1360
4184
282
3495
1640
7700
N/A
N/A



[Horowitz, *ISSCC* 2014]

Benefits of Reduced Precision

- Reduced Precision → Reduced Bit Width
- Reduce data movement and storage cost for inputs and outputs of MAC operation
  - Store more data (e.g., weights, activation) on chip
  - Smaller memory → lower energy
- Reduce cost of MAC operation
  - Cost of multiplication increases with bit width (n)
    - Energy and area by $O(n^2)$
    - Delay by $O(n)$

Fixed Point Arithmetic

Binary Multiplication

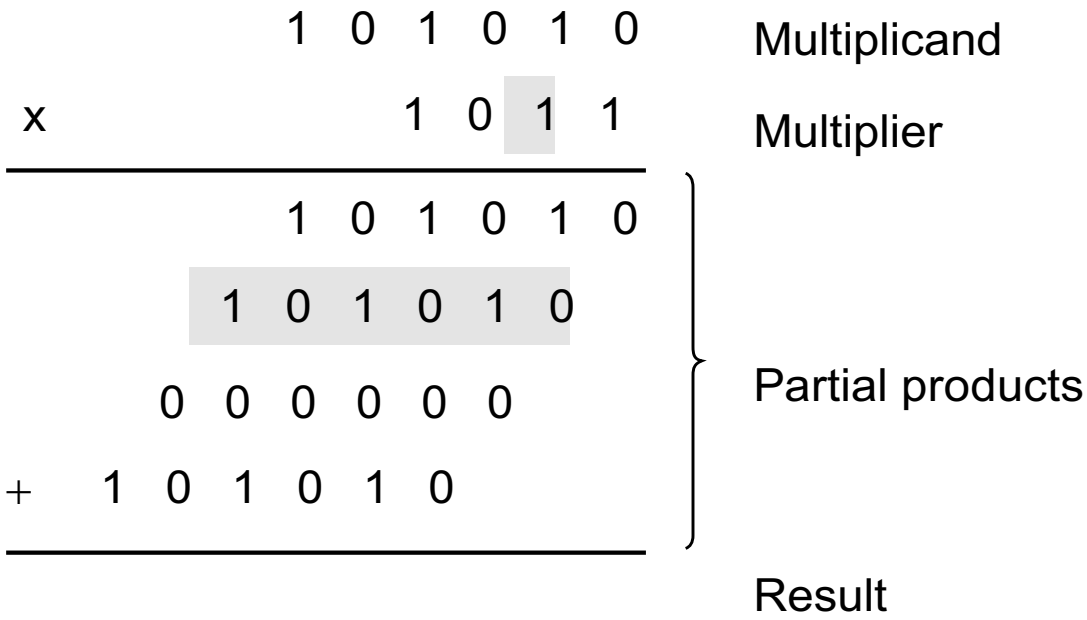


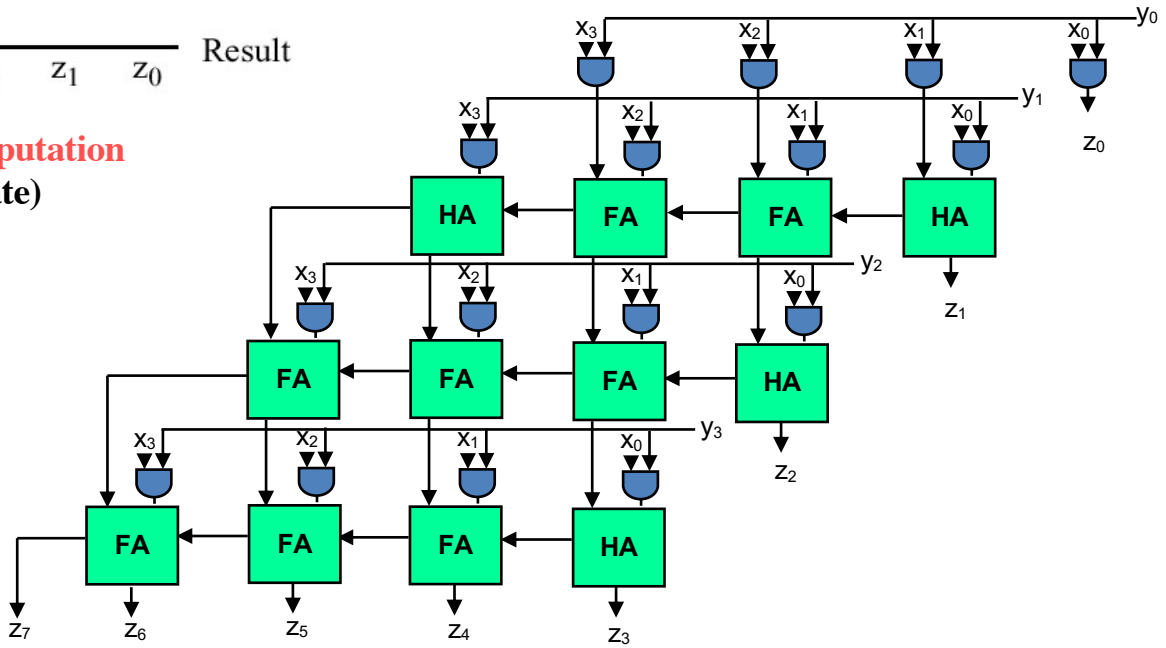
Image source: 6.6010[6.374]

Binary Multiplication

					x_3	x_2	x_1	x_0	Multiplicand
				×	y_3	y_2	y_1	y_0	Multiplier
				x_3y_0	x_2y_0	x_1y_0	x_0y_0		
			x_3y_1	x_2y_1	x_1y_1	x_0y_1			Partial Product
		x_3y_2	x_2y_2	x_1y_2	x_0y_2				
+	x_3y_3	x_2y_3	x_1y_3	x_0y_3					
z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0		Result

$\mathbf{n_1}$-bits
$\mathbf{n_2}$-bits

- **Partial product computation is simple (single and gate)**

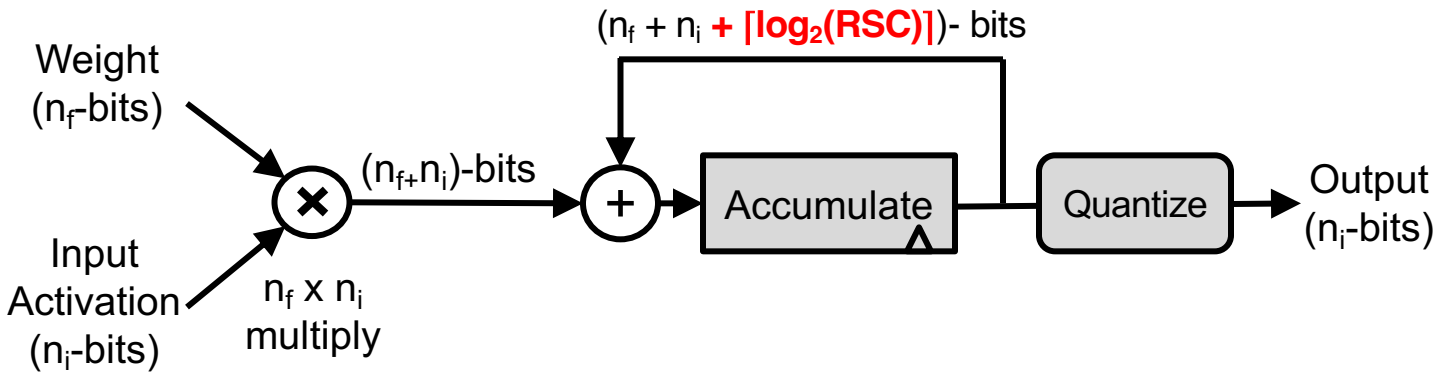


$$t_{mult} \approx [(\mathbf{n_1}-1)+(\mathbf{n_2}-2)]t_{carry}+(\mathbf{n_2}-1)t_{sum}+t_{and}$$

Image source: 6.6010[6.374]

Various Bit Widths Within a MAC

How many additional bits are required for partial sum to ensure no loss in precision?



Precision of internal values of MAC higher than inputs or outputs

Example assumes integer format

Popular DNNs

Metrics	AlexNet	VGG-16	GoogLeNet (v1)
Input Size	227x227	224x224	224x224
# of CONV Layers	**5**	**16**	**21 (depth)**
Filter Sizes	3, 5,11	3	1, 3 , 5, 7
# of Channels	3 - 256	3 - 512	3 - 1024
# of Filters	96 - 384	64 - 512	64 - 384
# of Weights	2.3M	14.7M	6.0M
# of MACs	666M	15.3G	1.43G
# of FC layers	**3**	**3**	**1**
# of Weights	58.6M	124M	1M
# of MACs	58.6M	124M	1M
Max Weights per Filter (RSC)	**9216**	**25088**	**1728**
Minimum additional bits $\lceil \log_2(RSC) \rceil$	**14**	**15**	**11**

For no loss in precision, **$\lceil \log_2(RSC) \rceil$** is determined based on largest filter size

Determining the Bit Width

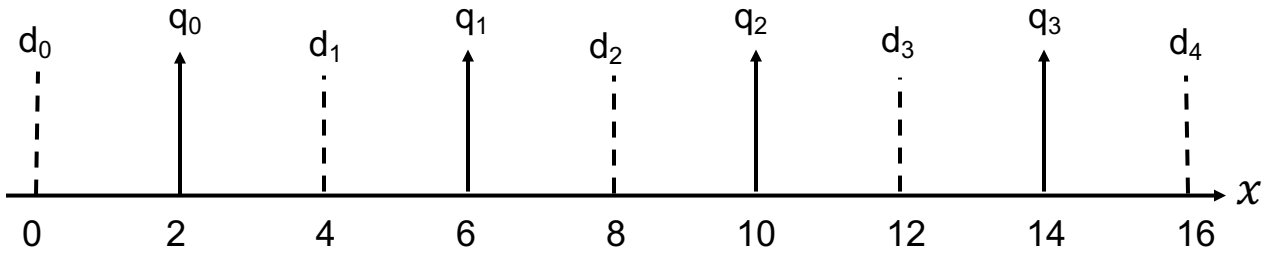
How do we determine the number of bits for the input activations (n_i), weights (n_f), and partial sums?

Quantization

Map data to a small set of quantization levels (e.g., L = 4)

q_i = quantization values

d_i = decision boundaries



x = 1, 3, 7, 8, 9, 15, 6, 2 → Quantization Q(·) → $\hat{x}$ = 2, 2, 6, 10, 10, 14, 6, 2

Quantization

Goal: Minimize the error between the reconstructed data from the quantization levels and the original data.

q_i = quantization values
d_i = decision boundaries

$$x_{min} \leq x \leq x_{max},$$

L= number of quantization levels
$p_x(x_o)$: probability density function of x

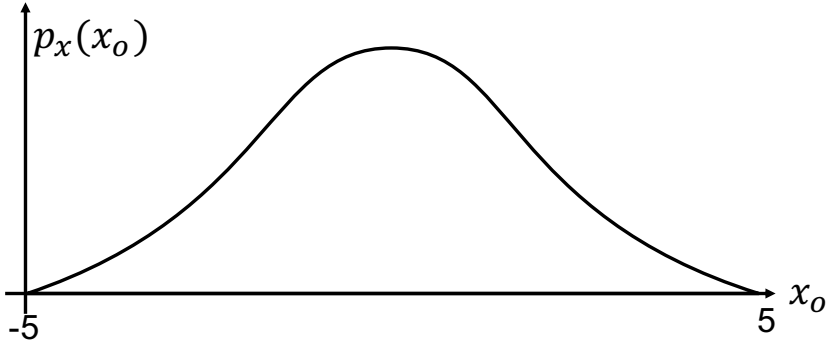
Optimally choose q_i and d_i

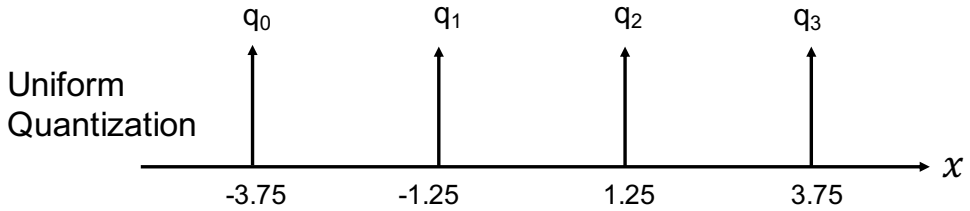
$$\min_{q_i, d_i} \quad E[(\underbrace{x - \hat{x}}_{\text{error}})^2] = \int_{x_o=x_{min}}^{x_{max}} (\hat{x} - x_o)^2 \cdot p_x(x_o) \cdot dx_o$$

error
(quantization noise)

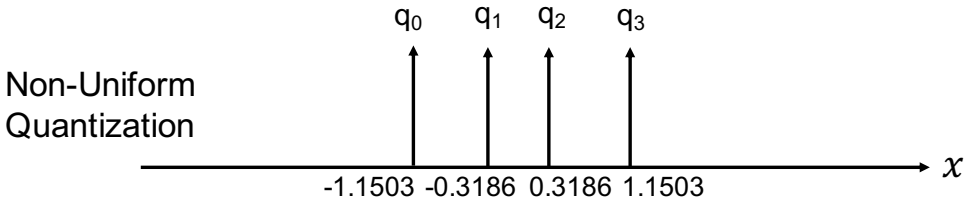
Image source: 6.344

Example





quantization error
0.5477



quantization error
0.1467

Precision

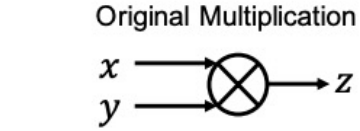
- The ***number of quantization levels*** reflects the precision and ultimately the ***number of bits*** required to represent the data
  - Usually, $\log_2$ of the number of levels
- ***"Reduced precision"*** refers to reducing the number of levels (values), and thus the number of bits
  - The benefits of reduced precision include reduced storage cost and/or reduced computation requirements
  - Can also increase throughput (recall vector lecture)
- **Range of values** also matters
  - Ratio of the largest and smallest non-zero value (magnitude) (i.e., x_{max}/x_{min})
  - To support a wide range, can increase the number of quantization levels or add scale factor (a form of non-uniform quantization)

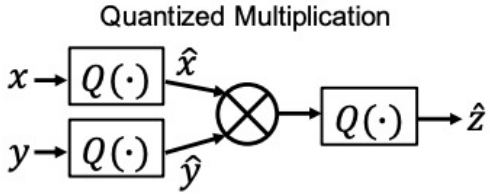
Example: Uniform vs. Scale factor

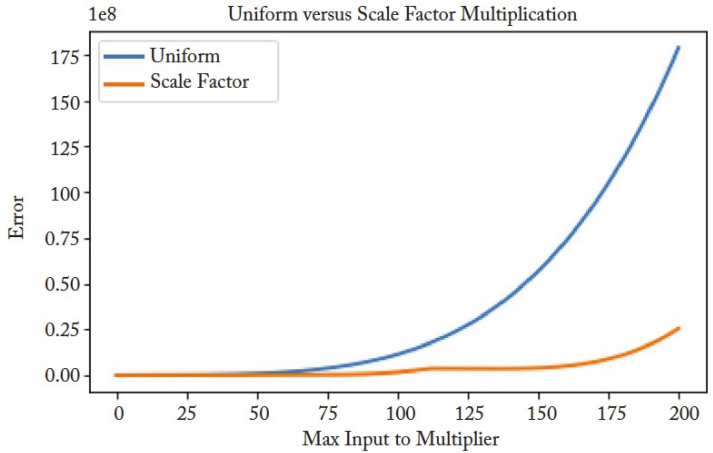
Uniform: $q_i = 44000 \times i/16$

Scale factor: $q_i = (4^{i/2+1} - 4^{i/2}) \times (i\%2)/8 + 4^{i/2} - 1$

i	q_i - uniform	q_i - scale factor
0	0	0
1	2750	1.5
2	5500	3
3	8250	9
4	11000	15
5	13750	39
6	16500	63
7	19250	159
8	22000	255
9	24749	639
10	27499	1023
11	30249	2559
12	32999	4095
13	35749	10239
14	38499	16383
15	41249	40959



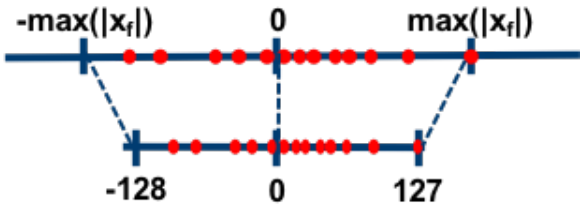




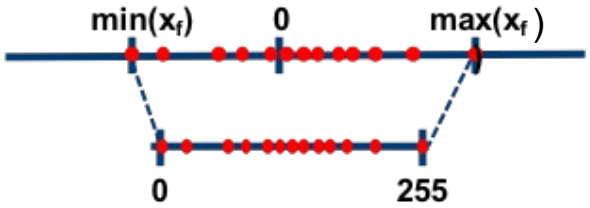
See section 7.2.1. in book for more details

Range Selection

- Symmetric mode vs. Asymmetric mode

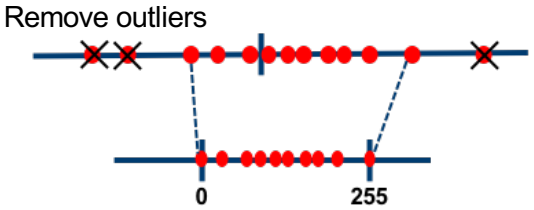


Easier to implement



Better utilize full range

- Clip range (saturation)
  - DoReFa (clip act to [0,1])
  - ReLU6 (clip act to [0,6])
  - PACT (clip act to [0, α], where α is learned)



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

Standard Components of the Bit Width

- Range of values
  - e.g., n_E-bits to scale values by $2^{(E-127)}$
- # of unique values per scale factor
  - e.g., n_M-bits to represent 2^M values
- Signed or unsigned values
  - e.g., signed (S) requires one bit (n_S=1)
- Total bits = $n_S+n_E+n_M$

- Floating point (FP) allows range to change for each value (n_E-bits)
- Fixed point (Int) has fixed range

- Default CPU/GPU is 32-bit float (FP32)

Common Numerical Representations

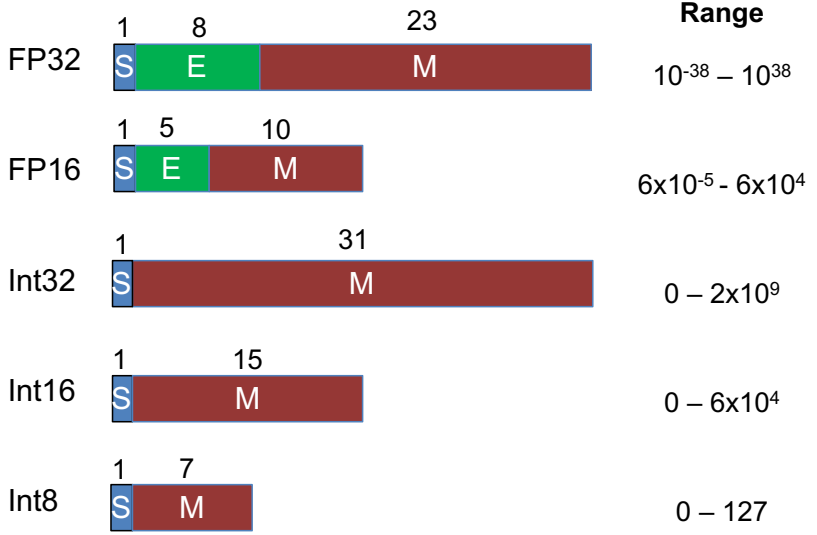


Image Source: B. Dally

Fixed-Point Format

When range is limited, a simple fixed-point format can be used.
For instance, 8-bit fixed (-128 to 127)

Components of a fixed-point number

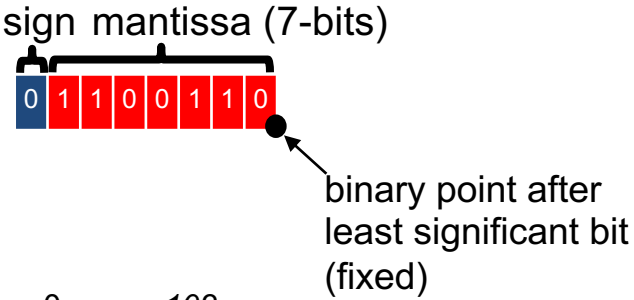
Mantissa (m): number of levels

Sign (s): indicates if number is positive or negative

$$(-1)^s \times m$$

Integer Example

8-bit fixed



102 ***s*** = 0 ***m***=*102*

Fixed-Point Format

When range is limited, a simple fixed-point format can be used.
For instance, 8-bit fixed (-127 to 128)

Components of a fixed-point number

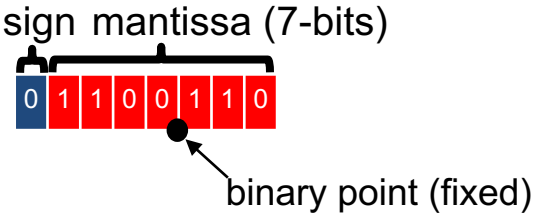
Mantissa (m): number of levels

Sign (s): indicates if number is positive or negative

$$(-1)^s \times m \times 2^{-f}$$ ← fixed

Fixed Point Example

8-bit fixed



12.75 ***s*** = 0 ***m***=*102* ***f*** = 3 (fixed)

Floating-Point Format

To support a wide range, the default format in most CPUs and GPUs is 32-bit float (10^{-38} to 10^{38})

Components of a floating-point number

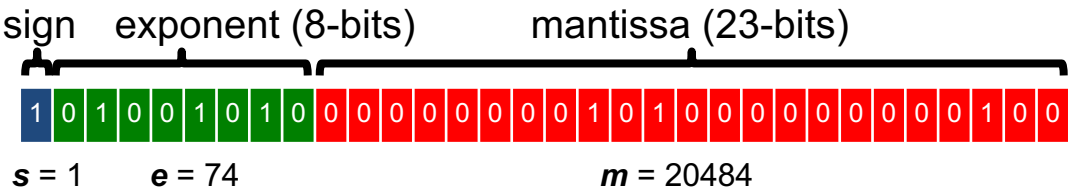
Mantissa (m): number of levels

Exponent (e): scale to a target range (location of binary point ***varies***)

Sign (s): indicates if number is positive or negative

$$(-1)^s \times m \times 2^{(e-127)}$$

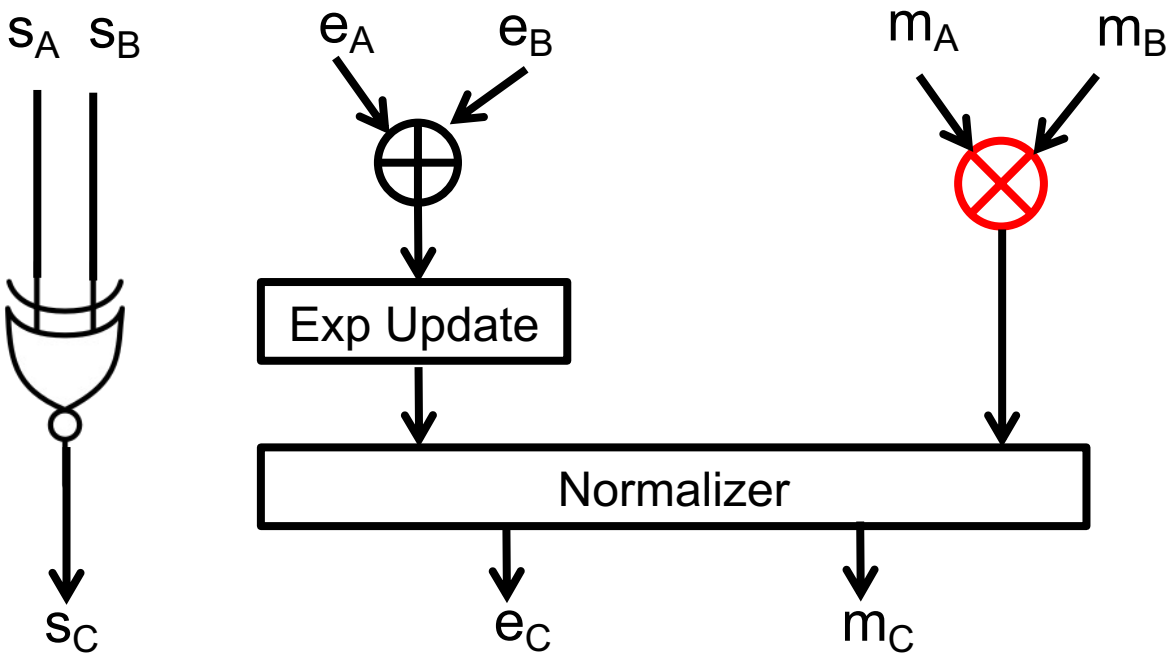
Single Precision Example



IEEE Standard for Floating-Point Arithmetic (IEEE 754)

Floating Point Arithmetic

Multiplier Example: C = A x B

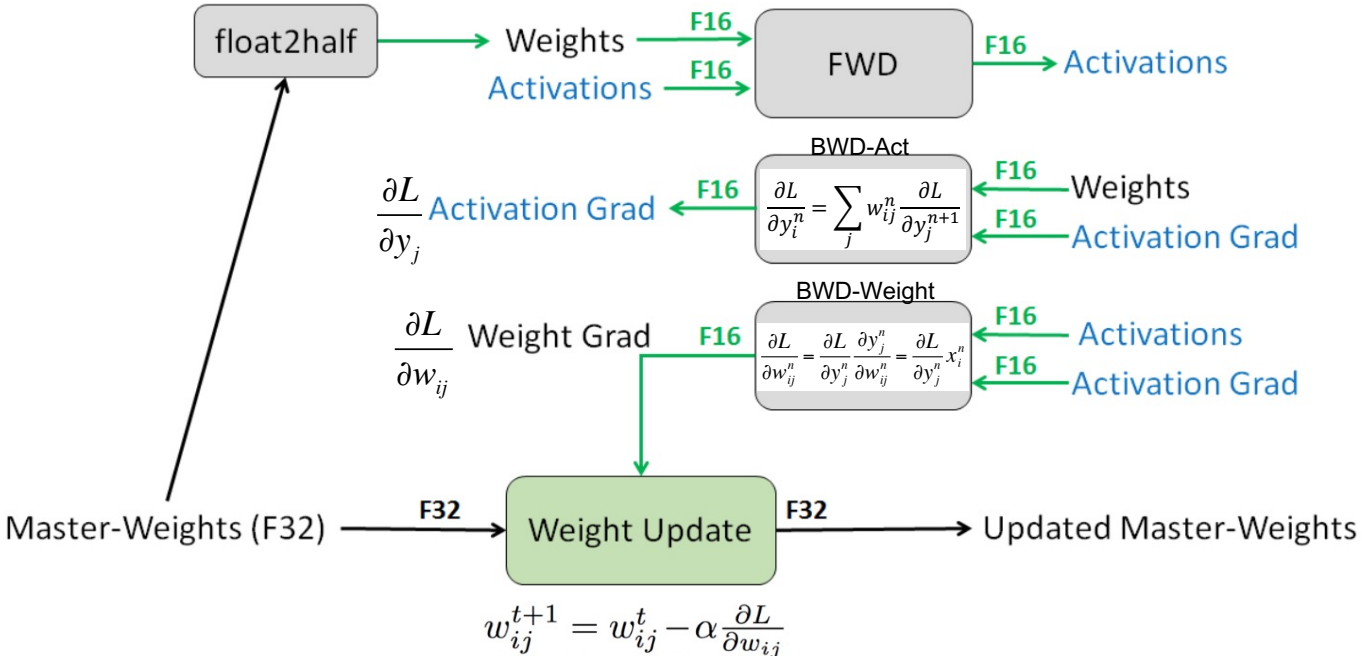


Mixed Precision

- **Different precision for different data types,** since different data types have different distributions
  - **For inference:** weights, activations, and partial sums
  - **For training:** weights, activations, partial sums, *gradients,* and *weight update*

Mixed Precision for Training

Weight update kept a **full precision (FP32)** while other variables are at **half precision (FP16)**



Modified figure from [**Narang**, *ICLR* 2018]

Reduce Mantissa Bits (M)

- **Reduce number of unique values**
- **Uniform quantization** (values are equally spaced out) [default]
- **Non-uniform quantization** (spacing can be computed, e.g., logarithmic, or with look-up-table)
- Fewer unique values can make transforms and compression more effective

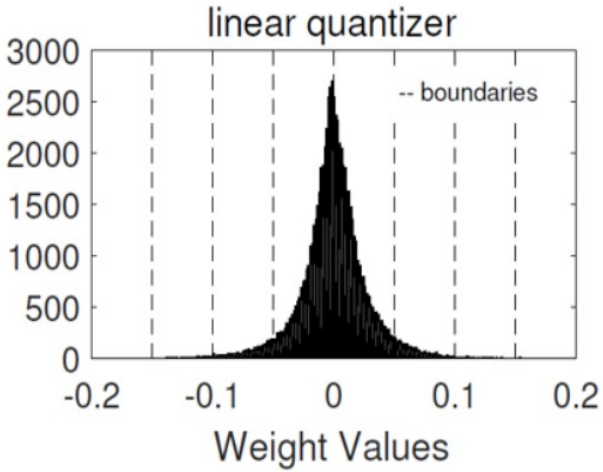
Non-Uniform Quantization

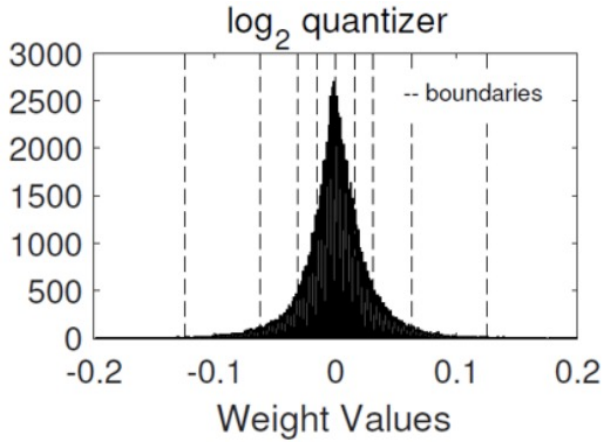
- **Precision** refers to the **number of levels**
  - Number of bits = $\log_2$ (number of levels)
- **Quantization** refers to mapping data to a smaller set of **levels**
  - Uniform, e.g., fixed point
  - Non-Uniform
    - Constrained (computed as a function of binary value) – e.g., log
    - Unconstrained (arbitrary relation between values) – e.g., learned from opt

Objective: Reduce size to improve speed and/or reduce energy while preserving accuracy

Computed Non-Uniform Quantization

Log Domain Quantization





Product = X * W

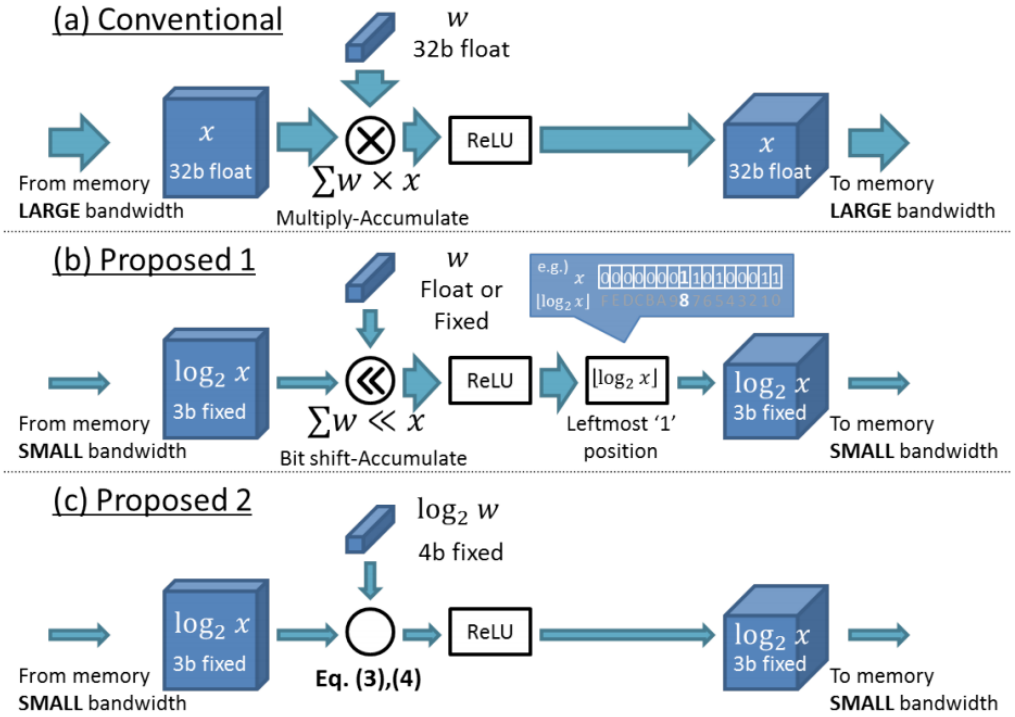
Product = X << W

[**Lee**, LogNet, *ICASSP* 2017]

Log Domain Computation

Only activation in log domain

Both weights and activations in log domain

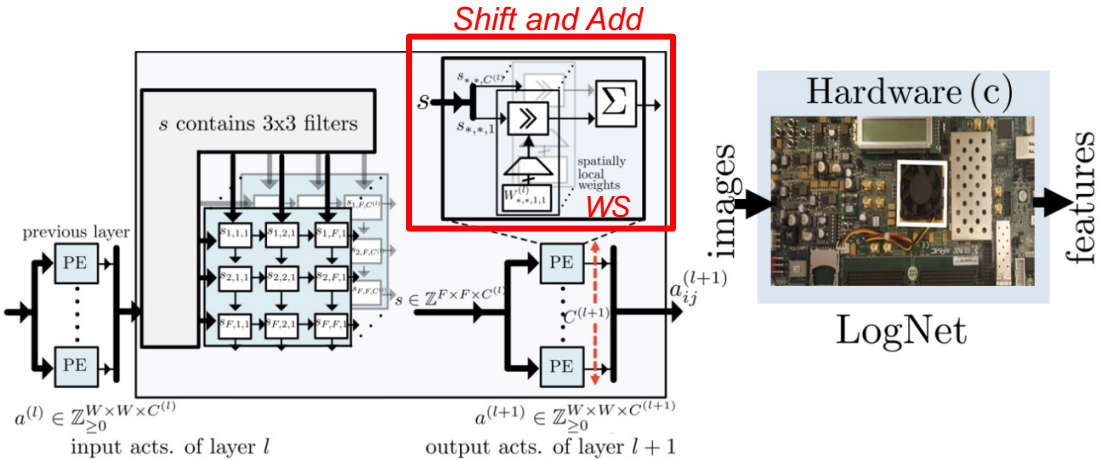


max, bitshifts, adds/subs

[**Miyashita**, *arXiv* 2016]

Log Domain Quantization

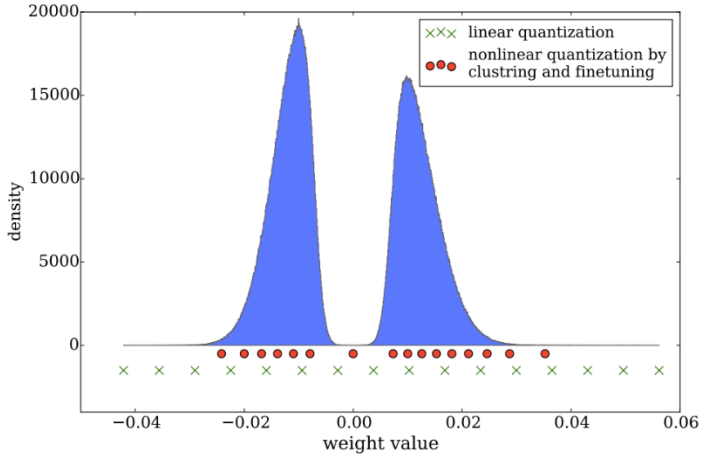
- Weights: 5-bits for CONV, 4-bit for FC; Activations: 4-bits
- Accuracy loss: 3.2% on AlexNet



[**Miyashita**, *arXiv* 2016],
[**Lee**, LogNet, *ICASSP* 2017]

Learned Quantization

- Learned mapping of data to quantization levels (e.g., k-means)



Implement with look up table

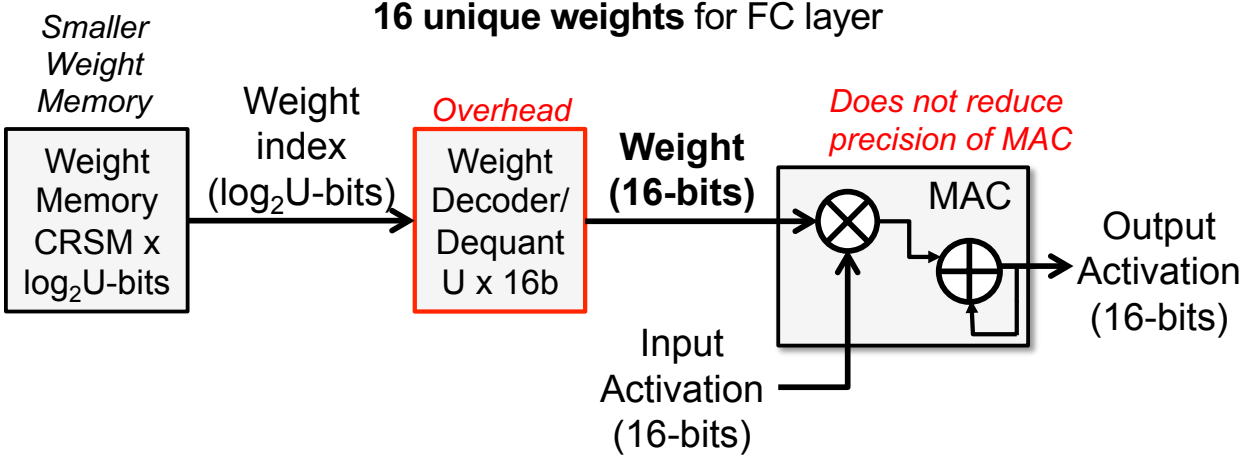
[**Han**, *ICLR* 2016]

- Additional Properties
  - Fixed or Variable (across data types, layers, channels, etc.)

Non-Uniform Quantization Table Lookup

Learned Quantization: Find U weights via k-means clustering to reduce number of unique weights ***per layer*** (weight sharing)

Example: AlexNet (no accuracy loss)
256 unique weights for CONV layer
16 unique weights for FC layer



Consequences: Narrow weight memory and second access from (small) table

Energy savings if reading from: mem(CRSMxlog$_2$U) + mem(Ux16) < mem(CRSMx16b)

Precision Taxonomy

- Uniform Quantization
  - Direct binary value (i.e., integer)
  - Fixed binary point (i.e., fixed point)

- Non-uniform Quantization
  - Constrained (a function of binary value) – e.g., log
  - Unconstrained (arbitrary relation between values) – e.g., learned from opt
  - **Scaled binary value (e.g., floating point)**

Mantissa (M) and Exponent Bits (E)

Tradeoff between number of bits allocated to M-bits and E-bits

fp16 (S=1, E=5, M=10) 

range: ~5.9×10^{-8} to ~6.5×10^{4}

bfloat16 (S=1, E=8, M=7) 

range: ~1×10^{-38} to ~3×10^{38}

Bfloat16 increases number of bits for exponents (equal to FP32) to support wider range (important for gradient) at a cost of fewer unique values

Precision Taxonomy

- Uniform Quantization
  - Direct binary value (i.e., integer)
  - Fixed binary point (i.e., fixed point)
- Non-uniform Quantization
  - Constrained (a function of binary value) – e.g., log
  - Unconstrained (arbitrary relation between values) – e.g., learned from opt
  - Scaled binary value (e.g., floating point)
- “Shared” values and/or hardware
  - **Exponent (e.g., dynamic fixed point)**

Eliminate Exponent Bits (E)

- Share range across group of values (e.g., weights for a layer or channel)
  - Referred to as **block floating point** or **dynamic fixed point**
  - Reduces storage and compute requirements

- The range can change for
  - Different types of data (e.g., activations, weights)
  - Different layers (e.g., CONV, FC)

Something in between fixed point and floating point
→ Dynamic fixed point!

Dynamic Fixed-Point Format

Components of a dynamic fixed-point number

Mantissa (m): number of levels

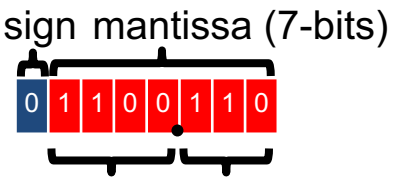
Scale factor (f): location of binary point

Sign (s): indicates if number is positive or negative

$$(-1)^s \times m \times 2^{-f}$$

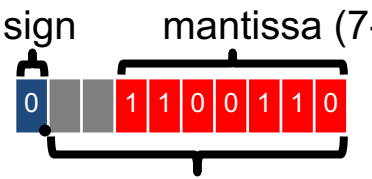
vary across data types, layers, etc.

8-bit dynamic fixed



12.75 ***s*** = 0 ***m***=*102* ***f*** = 3

8-bit dynamic fixed



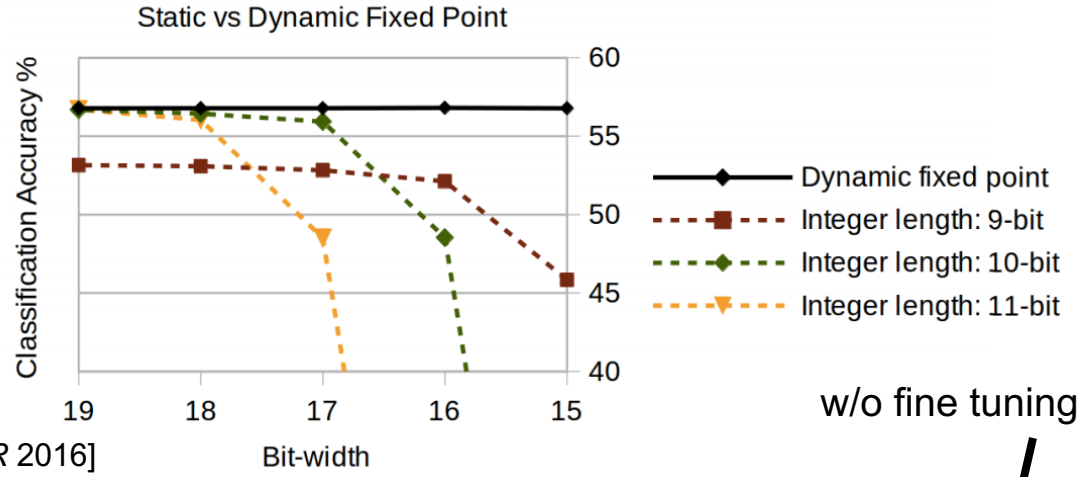
0.19921875 ***s*** = 0 ***m***=*102* ***f*** = 9

Allow ***f*** to be vary for different groups of variables;
f cannot change for each variable like floating point

[D. Williamson, Dynamically scaled fixed point arithmetic. 1991]

Impact on Accuracy

Top-1 accuracy on of CaffeNet on ImageNet



[**Gysel**, Ristretto, *ICLR* 2016]

w/o fine tuning

	Layer outputs	CONV parameters	FC parameters	32-bit floating point baseline	Fixed point accuracy
LeNet (Exp 1)	4-bit	4-bit	4-bit	99.1%	99.0% (98.7%)
LeNet (Exp 2)	4-bit	2-bit	2-bit	99.1%	98.8% (98.0%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.7%	81.4% (80.6%)
SqueezeNet top-1	8-bit	8-bit	8-bit	57.7%	57.1% (55.2%)
CaffeNet top-1	8-bit	8-bit	8-bit	56.9%	56.0% (55.8%)
GoogLeNet top-1	8-bit	8-bit	8-bit	68.9%	66.6% (66.1%)

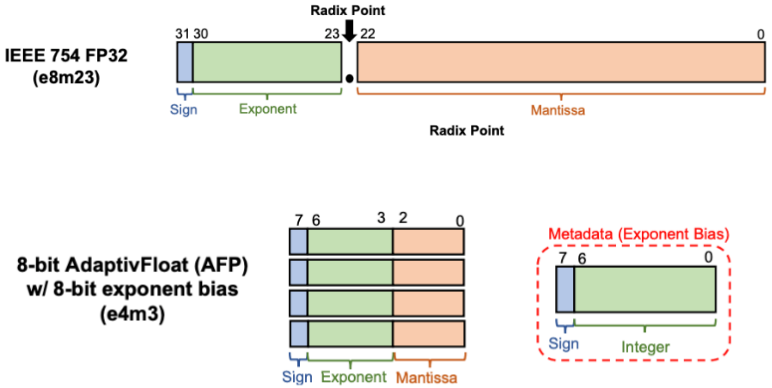
Varying Exponent Bias

- Allow ***exponent bias*** to be configurable

$$(-1)^s \times m \times 2^{(e-127)}$$

exponent bias

- AdaptivFloat [**Tambe**, *DAC* 2020]
  - Divides up exponent scale factor into two parts
    1. 'e' changes per value (i.e., floating point)
    2. **Bias changes per layer** (i.e., dynamic fixed point except at layer granularity)
- Configurable Float (CFloat8 & CFloat16) [Telsa Dojo 2021]
  - Fully configurable exponent bias (6-bits for CFloat8)
  - Two different methods of partitioning bits between mantissa and exponents
- Can repeatedly divide values into shared and unshared values → **Fractal**



Nvidia PASCAL

"New half-precision, **16-bit floating point instructions deliver over 21 TeraFLOPS** for unprecedented training performance. **With 47 TOPS (tera-operations per second) of performance, new 8-bit integer instructions** in Pascal allow AI algorithms to deliver real-time responsiveness for deep learning inference."

– Nvidia.com (April 2016)

Mixed Precision in Nvidia GPUs

A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x		
	IEEE FP64		19.5	1x		

V100→A100
2.5x FLOPS for HPC

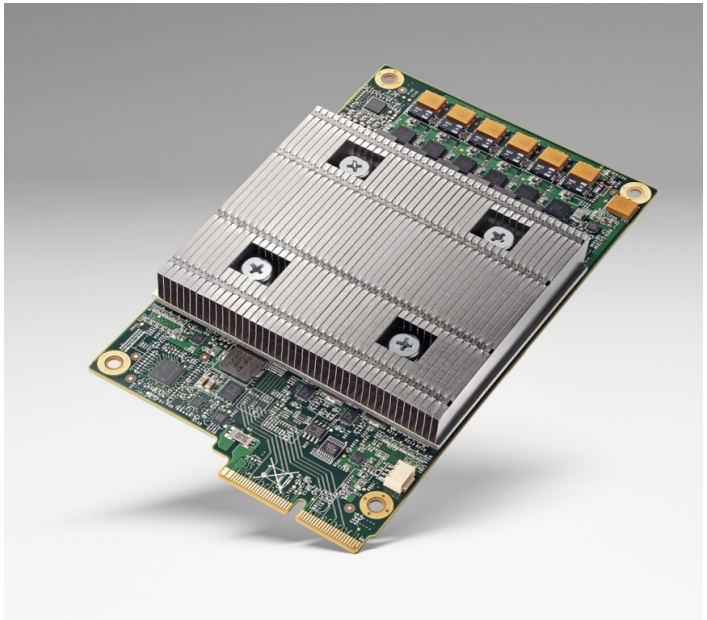
FFMA= floating point fused multiply-add

Source: Nvidia (May 2020)

Google's Tensor Processing Unit (TPU)

" With its TPU Google has seemingly focused on delivering the data really quickly by **cutting down on precision**. Specifically, it doesn't rely **on floating point precision like a GPU**
….
Instead, the chip uses integer math…TPU used **8-bit integer**."

- Next Platform (May 19, 2016)

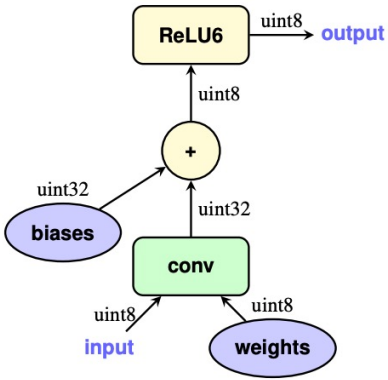


accumulators are 32-bits

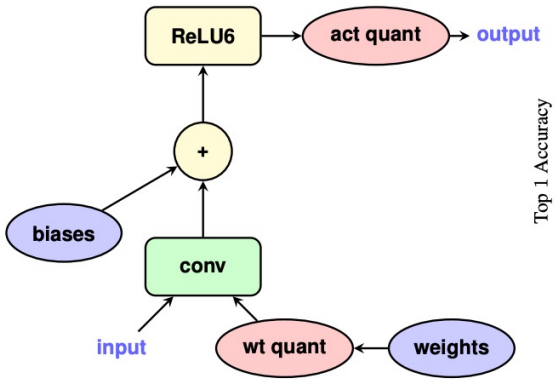
[**Jouppi**, *ISCA* 2017]

Quantization in TensorFlow Lite (TFLite)

TFLite has support for 8-bit quantization

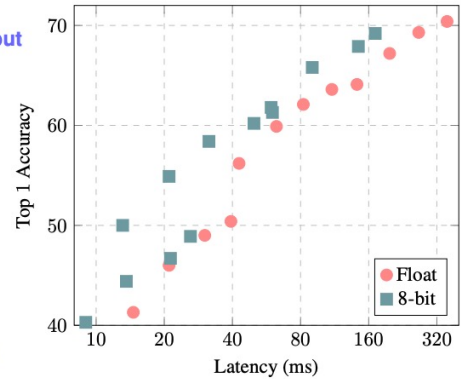


(a) Integer-arithmetic-only inference



(b) Training with simulated quantization

Performance of MobileNet

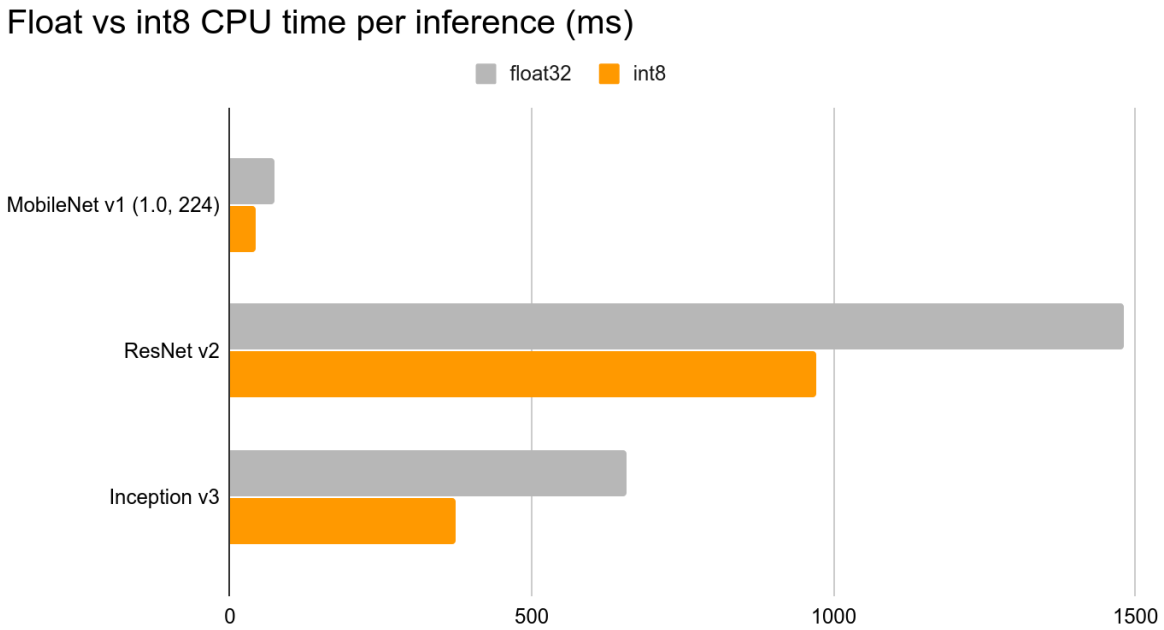


(c) ImageNet latency-vs-accuracy tradeoff

[**Jacob**, *CVPR* 2018]

Quantization in TensorFlow Lite (TFLite)

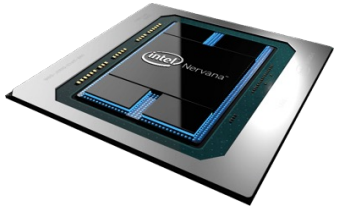
Impact of quantization to 8-bits integer (accuracy within <1%)



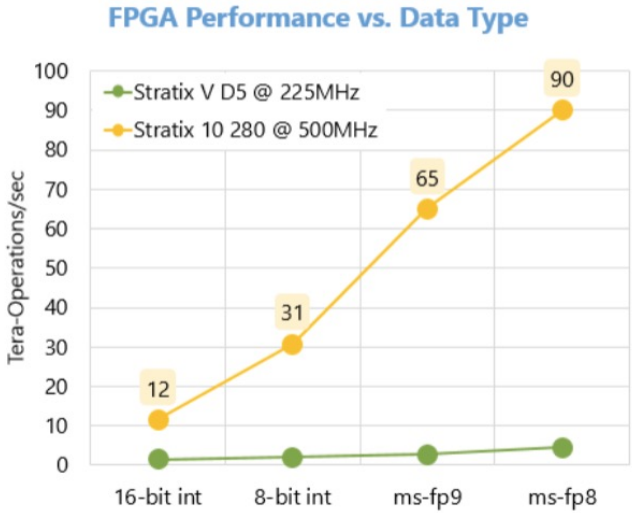
Source: https://blog.tensorflow.org/2019/06/tensorflow-integer-quantization.html

Other Industry Examples

- NVDLA
  - Binary/INT4/INT8/INT16/INT32/FP16 /FP32/FP64
  - For inference
- Microsoft BrainWave
  - Custom 8-bit and 9-bit floating point
  - For inference of RNNs/LSTMs on FPGAs
- Nervana Systems (now Intel)
  - Custom FlexPoint format for training
- TPU v2 & v3
  - bfloat16 for training



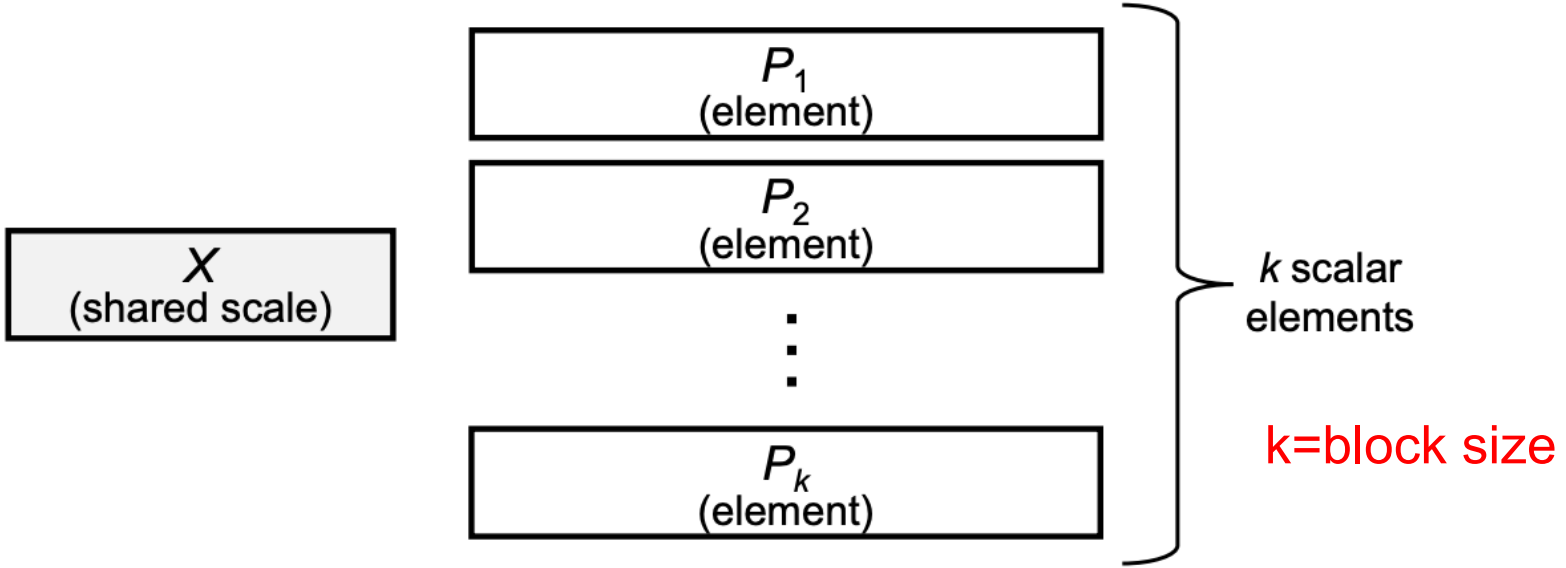
Intel Neural Network Processor



[**Chung**, *Hot Chips* 2017]

Standardize Microscaling (MX) Data Formats

MX data formats share scaling across block of “narrow bit width elements”



Source: https://www.opencompute.org/blog/amd-arm-intel-meta-microsoft-nvidia-and-qualcomm-standardize-next-generation-narrow-precision-data-formats-for-ai **(Oct 2023)**

Standardize Microscaling (MX) Data Formats

Format Name	Block Size	Scale Data Format	Scale Bits	Element Data Format	Element Bit-width
MXFP8	32	E8M0	8	FP8 (E4M3 / E5M2)	8
MXFP6	32	E8M0	8	FP6 (E2M3 / E3M2)	6
MXFP4	32	E8M0	8	FP4 (E2M1)	4
MXINT8	32	E8M0	8	INT8	8

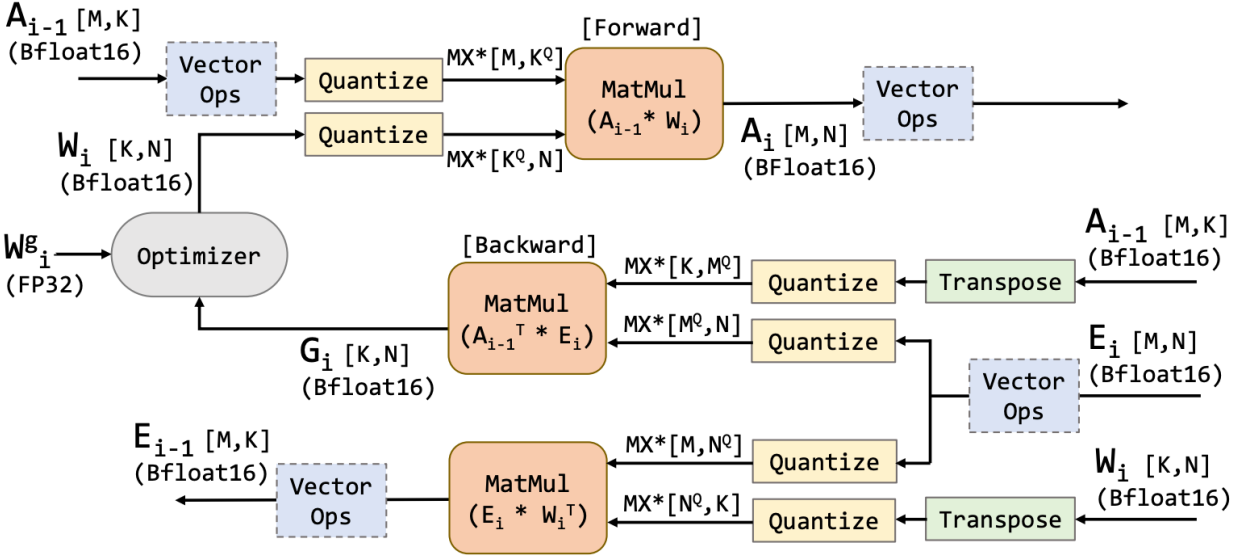
Impact on accuracy

- Inference with MXINT8 and MXFP8 can be used on FP32 pretrained models with minimal loss
- Inference with MXFP6 closely matches FP32 accuracy after quantization-aware finetuning
- Training with MXFP6 weights, activations, and gradients w/ minimal loss (w/o changing training recipe)
- Training with MXFP4 weights and MXFP6 activations and gradients incurs only a minor loss

Source: https://www.opencompute.org/blog/amd-arm-intel-meta-microsoft-nvidia-and-qualcomm-standardize-next-generation-narrow-precision-data-formats-for-ai **(Oct 2023)**

Standardize Microscaling (MX) Data Formats

MX data formats used for matrix multiplication, while vector operations (e.g., layernorm, Softmax, GELU, and residual add) are performed in a scalar floating-point format like Bfloat16 or FP32



Source: https://www.opencompute.org/blog/amd-arm-intel-meta-microsoft-nvidia-and-qualcomm-standardize-next-generation-narrow-precision-data-formats-for-ai **(Oct 2023)**

Varying Precision

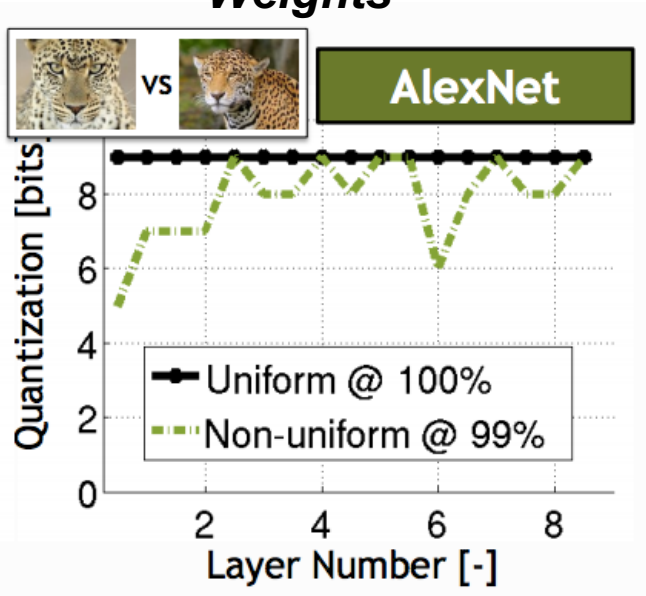
Change precision for different parts of the DNN
(e.g., vary across layers)

Activations

Tolerance	Bits per layer (I+F)
AlexNet (F=0)	
1%	10-8-8-8-8-8-6-4
2%	10-8-8-8-8-8-5-4
5%	10-8-8-8-7-7-5-3
10%	9-8-8-8-7-7-5-3

[**Judd**, *arXiv* 2016]

Weights

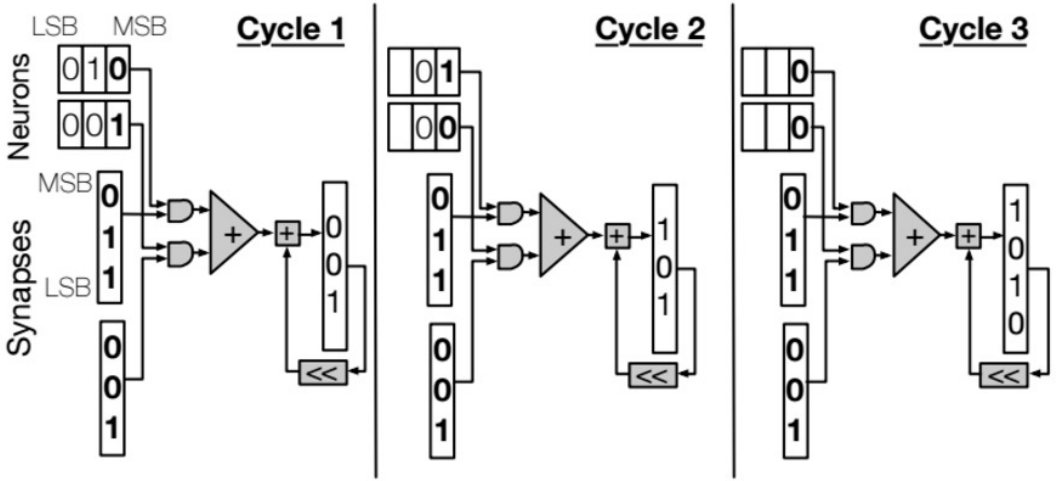


[**Moons**, *WACV* 2016]

Bitwidth Scaling (Speed)

Bit-Serial Processing: Reduce Bit-width → Skip Cycles
Speed up of 1.92x vs. 16-bit fixed

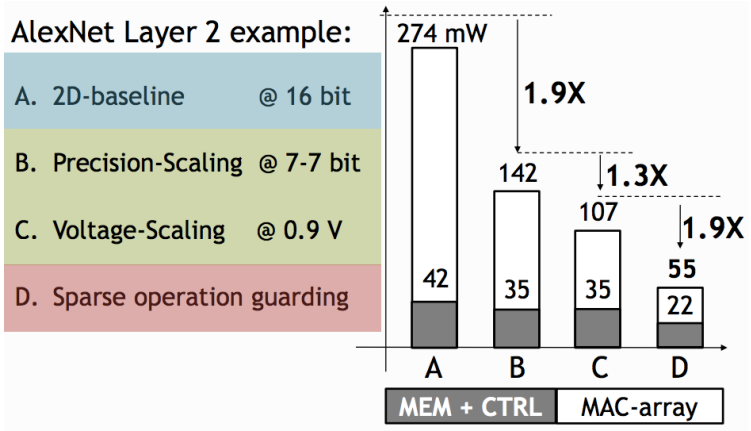
$$\sum_{i=0}^{N_i-1} s_i \times n_i = \sum_{i=0}^{N_i-1} s_i \times \sum_{b=0}^{P-1} n_i^b \times 2^b = \sum_{b=0}^{P-1} 2^b \times \sum_{i=0}^{N_i-1} n_i^b \times S_i$$

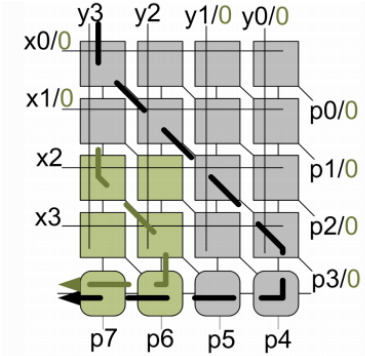


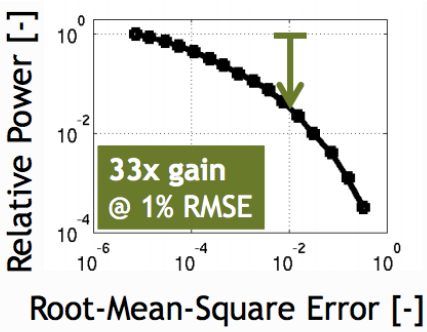
[**Judd**, Stripes, *MICRO* 2016]

Bitwidth Scaling (Power)

Reduce Bit-width → Shorter Critical Path → Reduce Voltage







$$P_{precise} = \alpha C f V^2 \quad \Rightarrow \quad P_{imprecise} = \frac{\alpha}{k_1} C f \left(\frac{V}{k_2}\right)^2$$

Power reduction of 2.5x vs. 16-bit fixed On AlexNet Layer 2

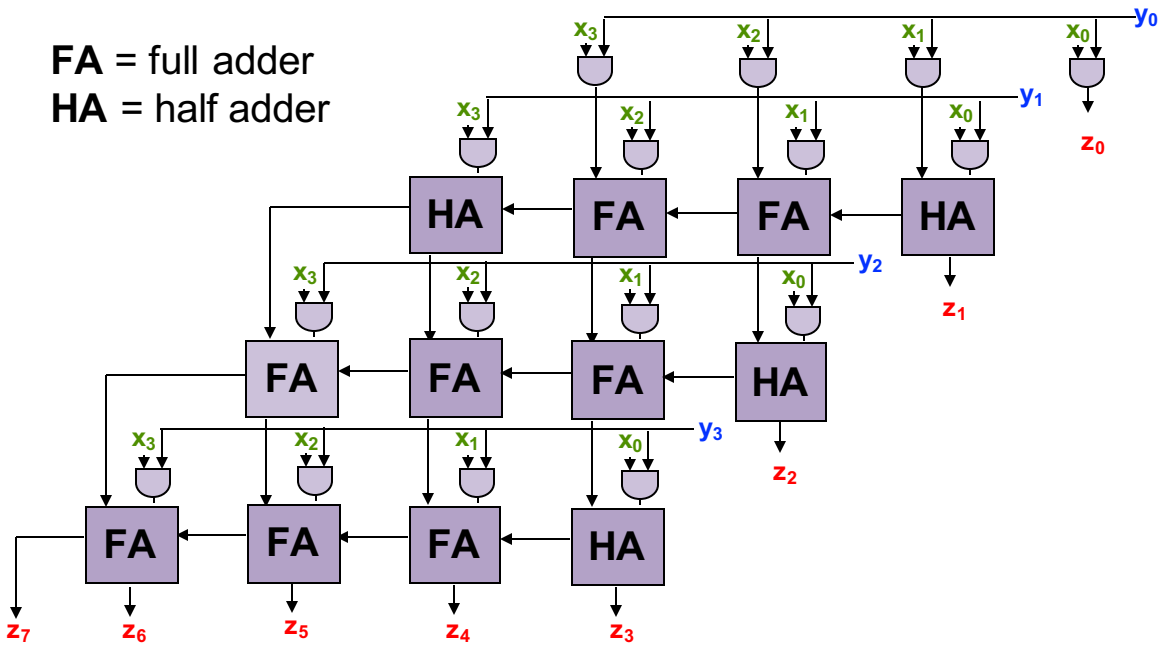
[**Moons**, *VLSI* 2016]

Precision Taxonomy

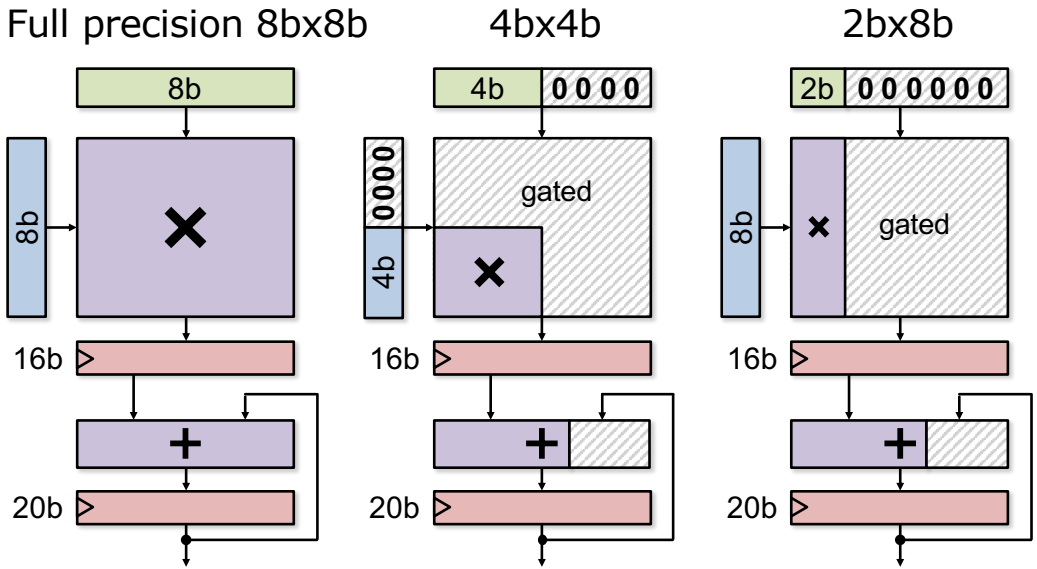
- Uniform Quantization
  - Direct binary value (i.e., integer)
  - Fixed binary point (i.e., fixed point)
- Non-uniform Quantization
  - Function of binary value (e.g., log)
  - Arbitrary relation (i.e., table lookup)
  - Scaled binary value (e.g., floating point)
- “Shared” values and/or hardware
  - Exponent (e.g., dynamic fixed point)
  - **Mantissa (e.g., varying precision hardware)**

Fixed Point Multiplier

FA = full adder
HA = half adder



Precision Scalable MACs for *Varying* Precision



Conventional data-gated MAC

Gate unused logic (e.g., full adders) to reduce energy consumption;
share hardware across different precisions

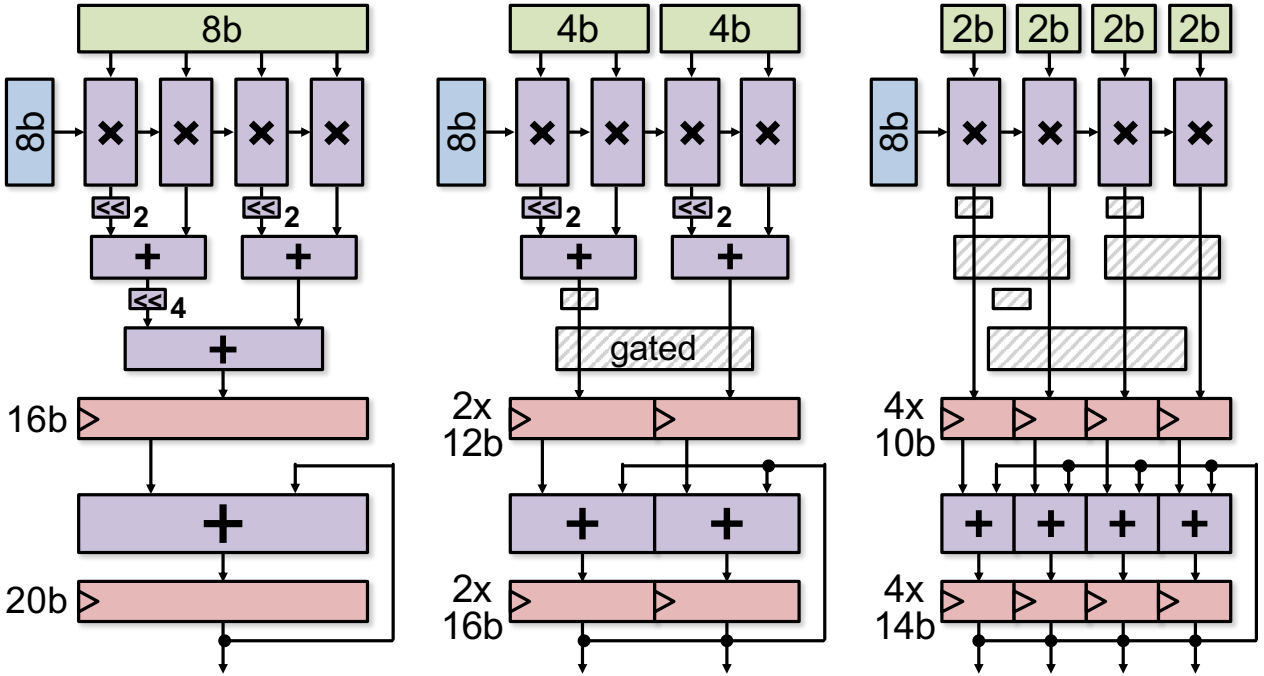
Can we add logic to increase utilization for higher throughput/area?

Many Types of Precision-Scalable MACs

- Many similarities between DNN accelerators and precision-scalable MACs
- DNN accelerators with a spatial architecture contain **multiple PEs within a PE array**, while a spatial precision-scalable MAC contains **multiple full adders within a spatial multiplier**
- The PEs in the PE array **accumulate partial sums**, while the full adders in the multiplier **accumulate partial products**

Use similarities to classify different precision-scalable MAC architectures

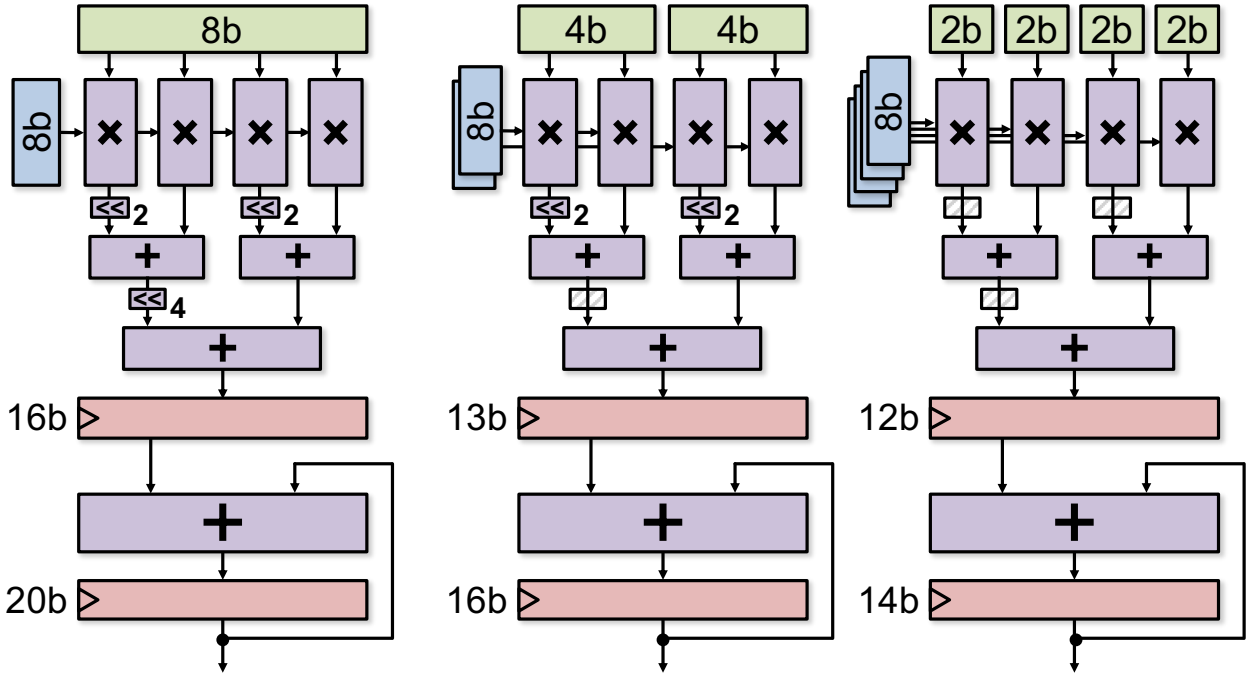
Spatial Precision-Scalable MACs



Temporal accumulation of partial products

[**Camus**, *JETCAS* 2019]

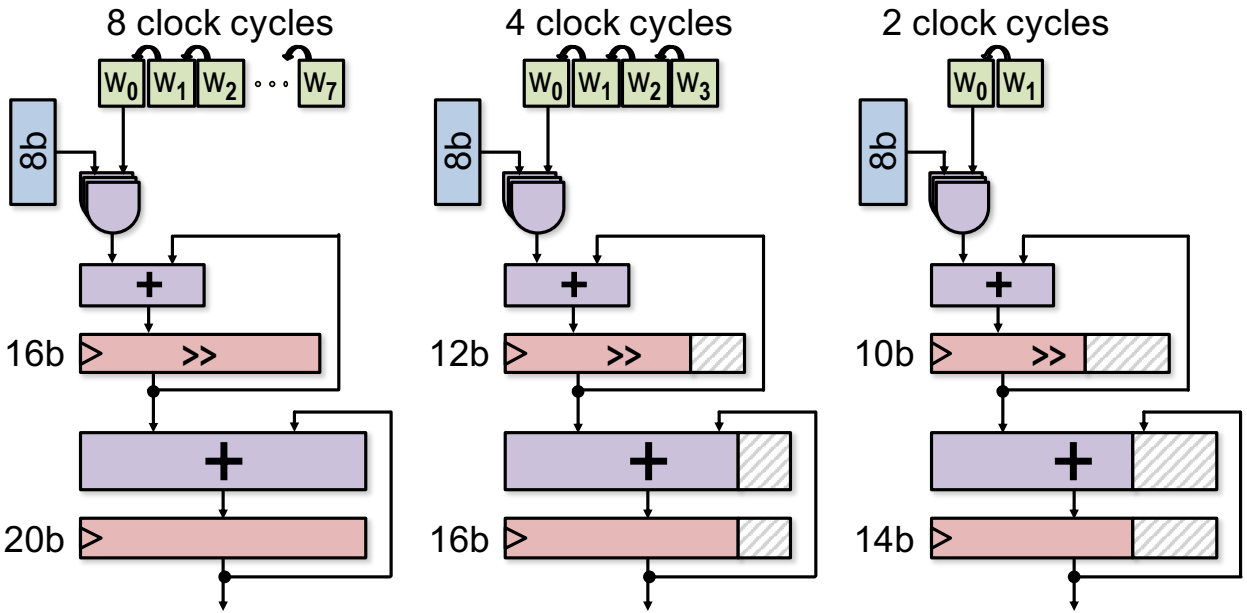
Spatial Precision-Scalable MACs



Spatial accumulation of partial products

[**Camus**, *JETCAS* 2019]

Temporal Precision-Scalable MACs



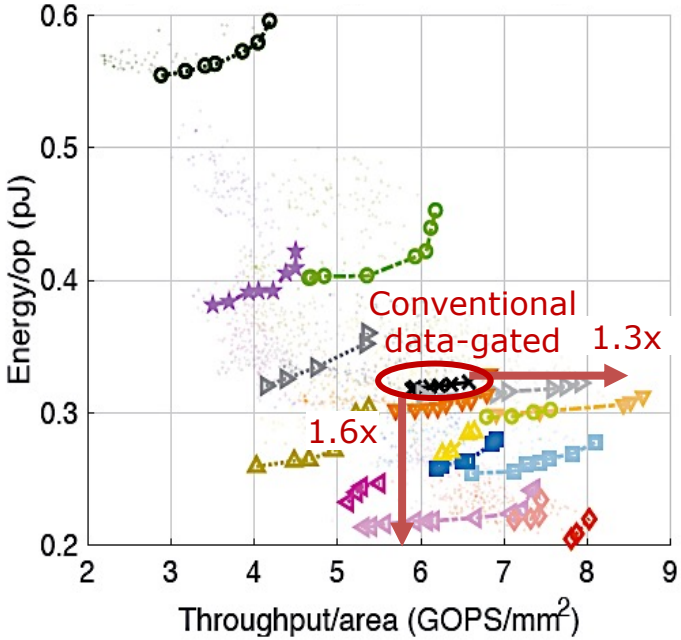
Also referred to as bit-serial processing

[**Camus**, *JETCAS* 2019]

Precision Scalable MACs for *Varying* Precision

Evaluation of **19 precision-scalable MAC designs**

- 5% of values 8bx8b
- 95% of values at 2bx2b and 4bx4b



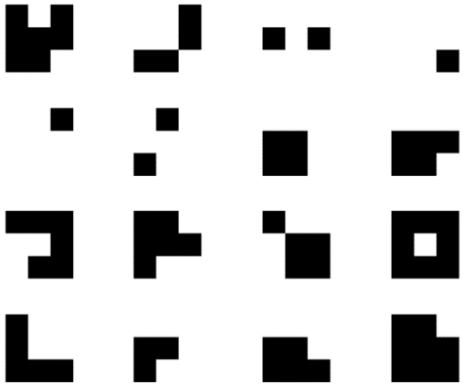
Overhead of additional logic to increase utilization for higher throughput/area can reduce benefits

[**Camus**, *JETCAS* 2019]

Binary Nets

- **Binary Connect (BC)**
  - Weights {-1,1}, Activations 32-bit float
  - MAC → addition/subtraction
  - Accuracy loss: **19%** on AlexNet

Binary Filters



Reduced number of unique filters

[**Courbariaux**, *NeurIPS* 2015]

Binary Connect

Algorithm 1 SGD training with BinaryConnect. C is the cost function for minibatch and the functions binarize(w) and clip(w) specify how to binarize and clip weights. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_b \leftarrow \text{binarize}(w_{t-1})$

For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}

2. Backward propagation:

Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$

For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b

3. Parameter update:

Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{db_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

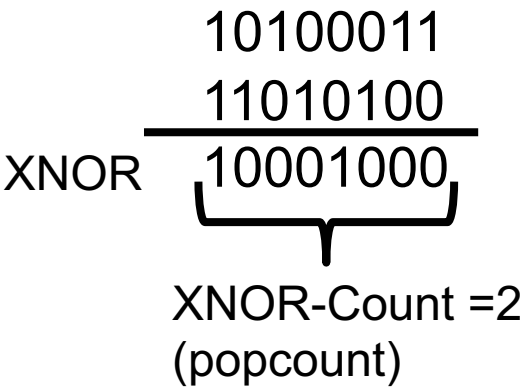
$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$

"Only binarize the weights during the forward and backward propagations, but not during the parameter update"

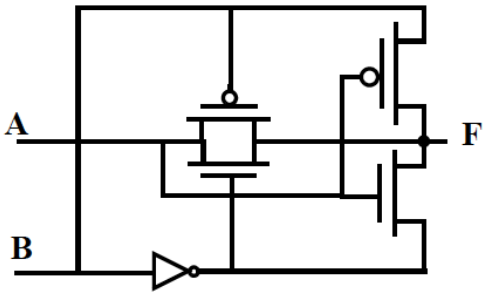
Keep full precision weights around during training (mixed precision)

Binary Nets (Weights & Activations)

- **Binarized Neural Networks (BNN)**
  - Weights {-1,1}, Activations {-1,1}
  - MAC → XNOR-Count
  - Accuracy loss: **29.8%** on AlexNet



Tiny XOR gate
(add inverter for XNOR)

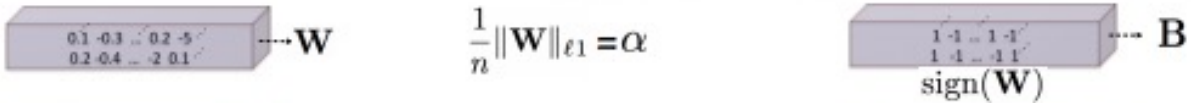


[**Courbariaux**, *arXiv* 2016]

Scale the Weights

- **Binary Weight Nets (BWN)**
  - Weights {-α, α} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - **Add scaling**
    - α determined by the l_1-norm of all weights in a filter
  - Accuracy loss: **0.8%** on AlexNet

Hardware needs to support both weight precisions

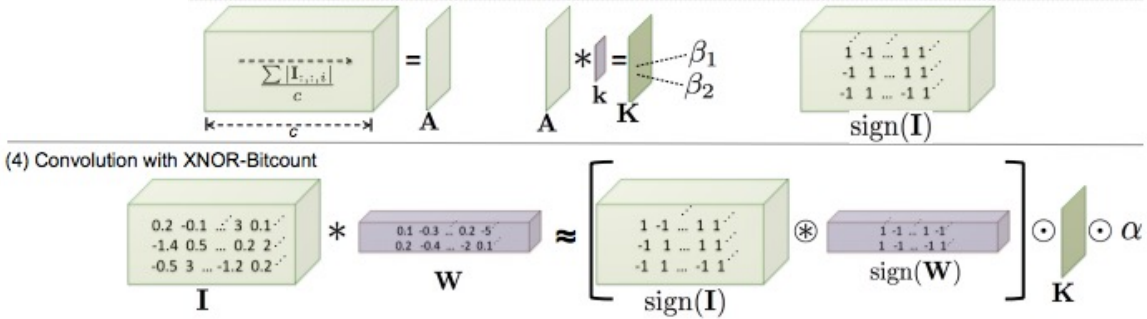


Scale factor (α) can change per filter

[**Rastegari**, BWN & XNOR-Net, *ECCV* 2016]

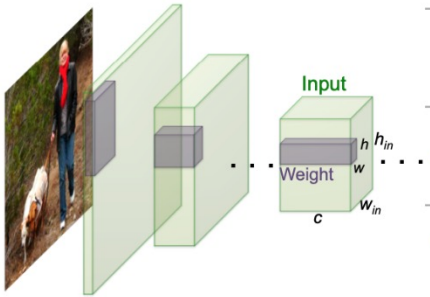
Scale the Weights and Activations

- **XNOR-Net**
  - Weights {-α, α}, Activations {-β_i, β_i}
    - except first & last layers are 32-bit float
  - **Add scaling**
    - β_i determined by the l_1-norm of all activations across channels *for given position i* of the input feature map
  - Accuracy loss: 11% on AlexNet



Scale factors (α, β_i) can change per filter or position in filter

XNOR-Net



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input **(XNOR-Net)**	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

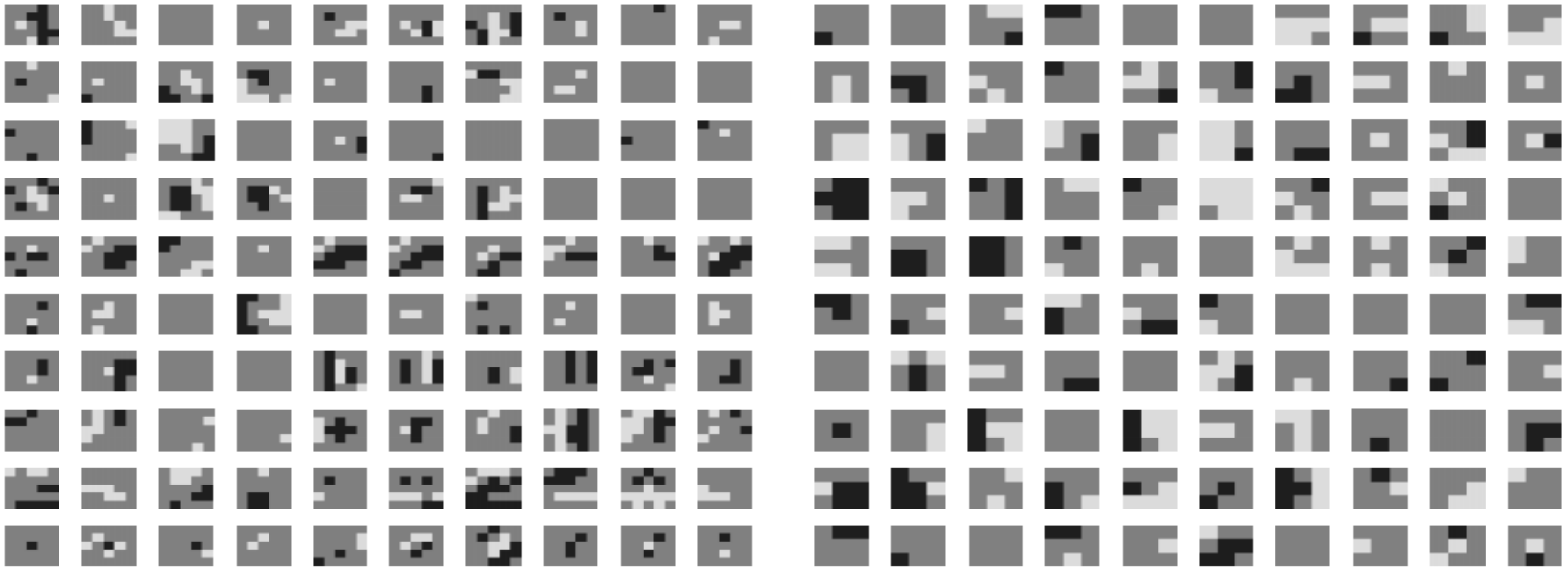
[**Rastegari**, BWN & XNOR-Net, *ECCV* 2016]

https://xnor.ai/

Ternary Nets

- **Allow for weights to be zero**
  - Increase sparsity, but also increase number of bits (2-bits)
- **Ternary Weight Nets (TWN)** [**Li**, Workshop @ *NeurIPS* 2016]
  - Weights {-w, 0, w} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - Accuracy loss: 3.7% on AlexNet
- **Trained Ternary Quantization (TTQ)** [**Zhu**, *ICLR* 2017]
  - Weights {-w_1, 0, w_2} → except first and last layers are 32-bit float
  - Activations: 32-bit float
  - Accuracy loss: 0.6% on AlexNet

Filter Kernels of TTQ

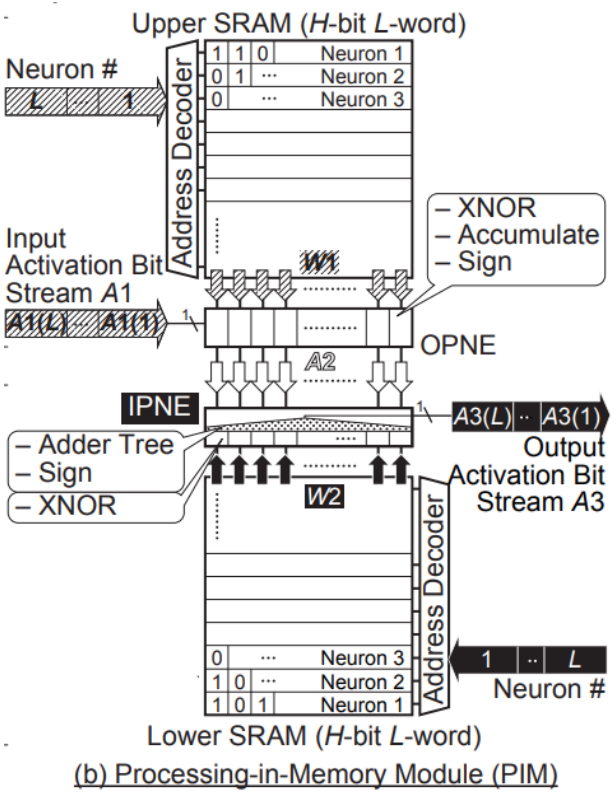


[**Zhu**, *ICLR* 2017]

Binary/Ternary Net Hardware

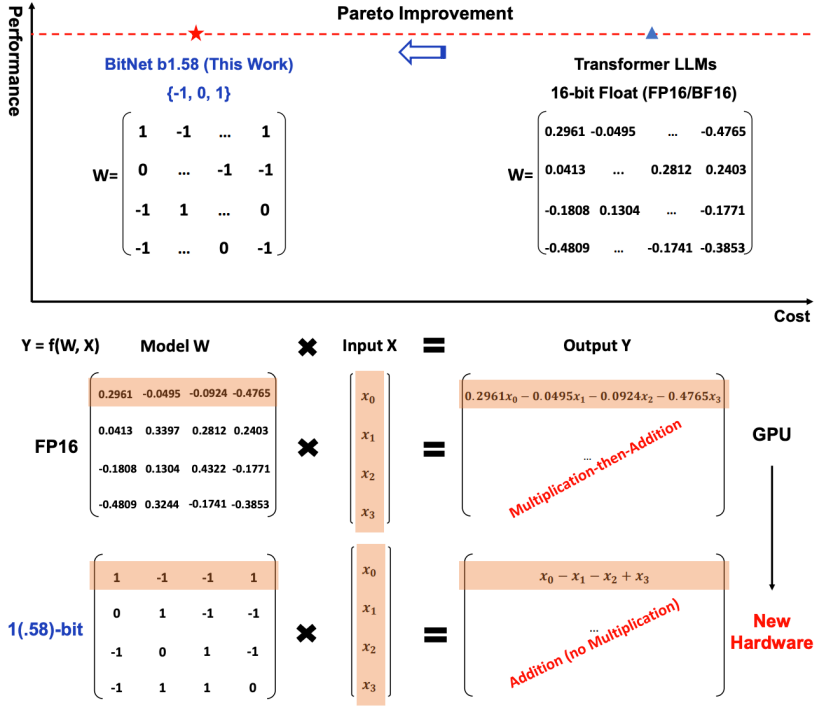
- Examples
  - **YodaNN** (binary weights)
  - **BRein** (binary weights and activations)
  - **TrueNorth** (ternary weights and binary activations)

These designs tend only support DNN models for digital classification ('MNIST') (except YodaNN)



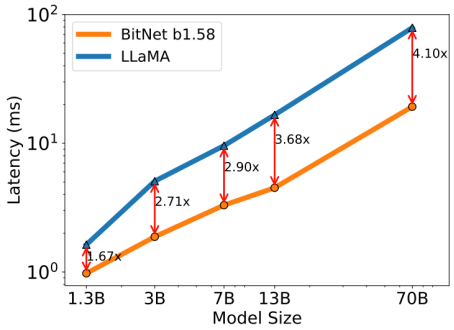
[BRein, *VLSI* 2017]

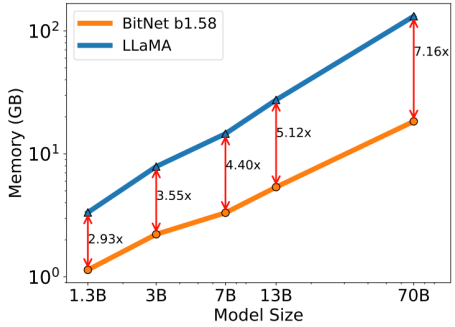
Binary and Ternary LLMs



BitNet = **1b weights**
Ternary BitNet = **~1.58b weights**
Both have 8-bit activations and trained from scratch

Models	Size	Memory (GB)↓	Latency (ms)↓	PPL↓
LLaMA LLM	700M	2.08 (1.00x)	1.18 (1.00x)	12.33
BitNet b1.58	700M	0.80 (2.60x)	0.96 (1.23x)	12.87
LLaMA LLM	1.3B	3.34 (1.00x)	1.62 (1.00x)	11.25
BitNet b1.58	1.3B	1.14 (2.93x)	0.97 (1.67x)	11.29
LLaMA LLM	3B	7.89 (1.00x)	5.07 (1.00x)	10.04
BitNet b1.58	3B	**2.22 (3.55x)**	**1.87 (2.71x)**	**9.91**
BitNet b1.58	3.9B	**2.38 (3.32x)**	**2.11 (2.40x)**	**9.62**





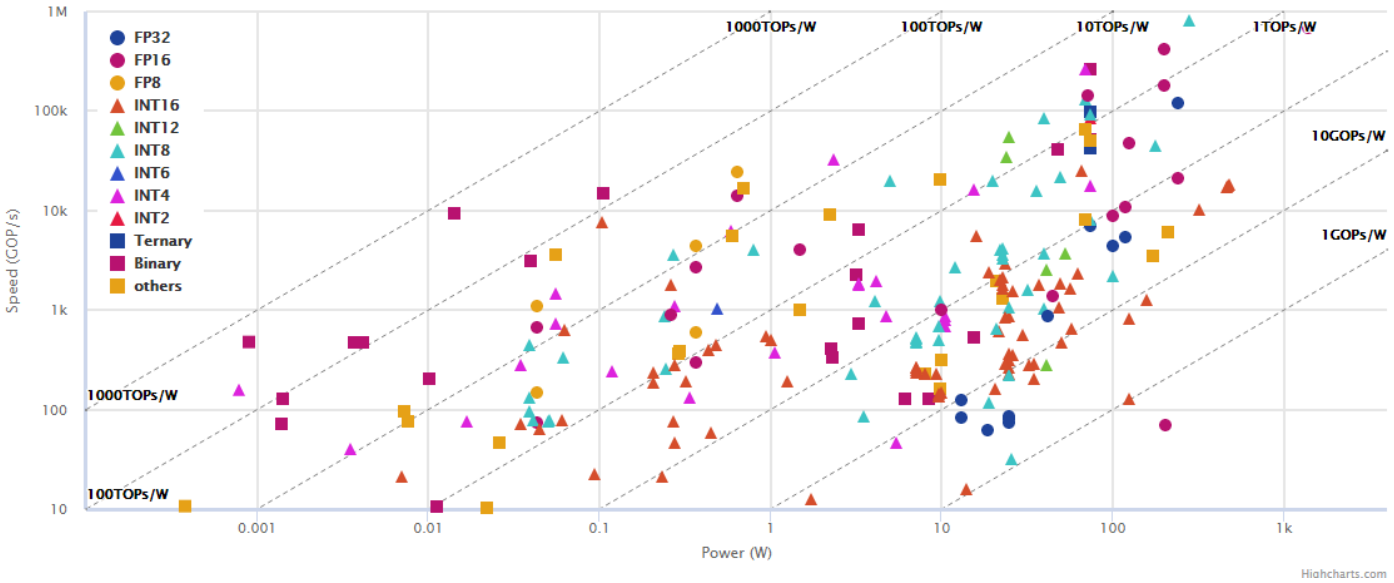
Source: https://arxiv.org/abs/2402.17764

Summary of Reduce Precision

Category	Method	Weights (# of bits)	Activations (# of bits)	Accuracy Loss vs. 32-bit float (%)
Dynamic Fixed Point	w/o fine-tuning	8	10	0.4
	w/ fine-tuning	8	8	0.6
Reduce weight	Ternary weights Networks (TWN)	2*	32	3.7
	Trained Ternary Quantization (TTQ)	2*	32	0.6
	Binary Connect (BC)	1	32	19.2
	Binary Weight Net (BWN)	1*	32	0.8
Reduce weight and activation	Binarized Neural Net (BNN)	1	1	29.8
	XNOR-Net	1*	1	11
Non-Uniform	LogNet	5(conv), 4(fc)	4	3.2
	Weight Sharing	8(conv), 4(fc)	16	0

* first and last layers are 32-bit float

Impact of Reduced Precision



Source: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator

Research on Reduced Precision for Training

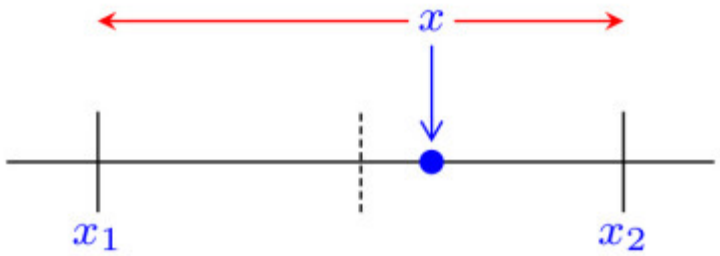
- Gradients have large dynamic ranges that vary across layers

- Hybrid 8-bit float [**Sun**, *NeurIPS* 2019]
  - HFP8 (forward: E=4, M=3, S=1, backward: E=5, M=2, S=1)

- 4-bit training [**Sun**, *NeurIPS* 2020]
  - Gradients use a radix-4 logarithmic format (FP4)
    - Also, FP8 for some layers
  - Per-layer trainable scale factor for gradient to utilize full range
  - Two-phase rounding (different quantization levels) to minimize quantization errors

Stochastic Rounding

Statistical information lost when rounding to nearest quantization level. Instead, use stochastic rounding where probability is proportional to distance from quantization level. Referred to as “unbiased rounding scheme”.

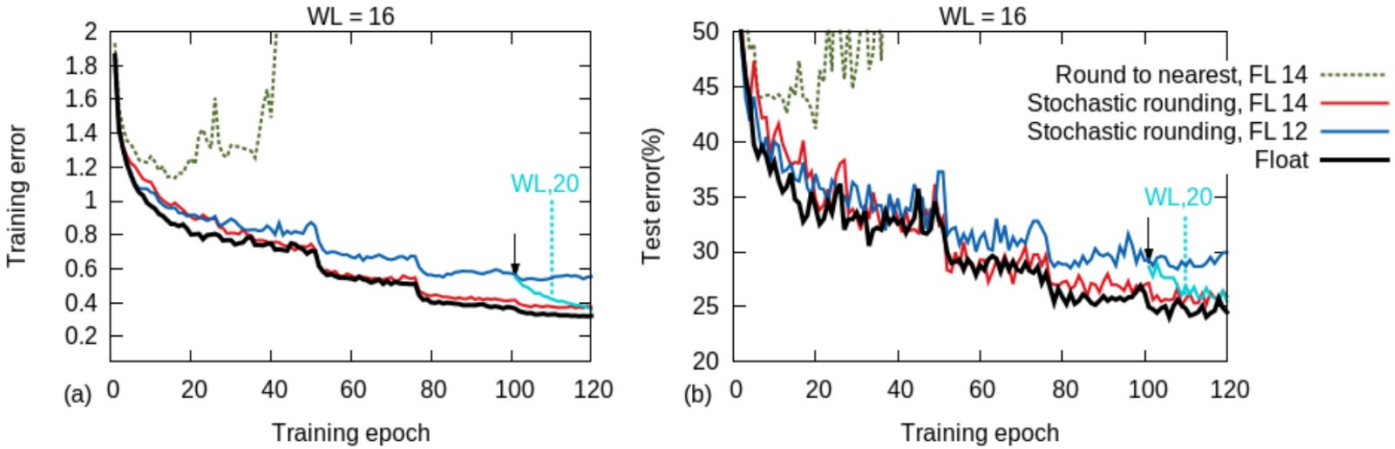


- Probability x rounds to x_1 = (**x**-x_1)/ (x_2-x_1)
- Probability x rounds to x_2 = (x_2-**x**)/ (x_2-x_1)

[**Barnes**, *Electronic Eng* 1951]

Stochastic Rounding

Impact on training on MNIST with fixed-point number of bit width (i.e., WL=word length) of 16 at different fractional length (FL)



[**Gupta**, *ICML* 2015]

Design Considerations for Reduced Precision

- Impact on accuracy
  - Must consider difficulty of dataset, task, and DNN model
    - e.g., Easy to reduce precision for an easy task (e.g., digit classification); does method work for a more difficult task?
  - Quantization-aware training vs. Post-training quantization
- Does hardware cost exceed benefits?
  - Need extra hardware to support variable precision
    - e.g., Additional shift-and-add logic and registers for variable precision
  - Granularity impacts hardware overhead as well as accuracy
    - e.g., More overhead to support (1b, 2b, 3b … 16b) than (2b, 4b, 8b, 16b)
- Evaluation
  - Use 8-bit for inference and 16-bit float for training for baseline
  - 32-bit float is a weak baseline

Interplay with Other Optimizations

- **DNN Model Shape**
  - **WRPN**: Wide Reduce Precision Network [**Mishra**, *ICLR* 2018]
    - Increasing width (# of channels) to recover accuracy from reduce precision (4-bits, 2-bits)

- **Dataflows**
  - **UNPU:** Unified neural processing unit [**Lee**, *JSSC* 2019]
    - Use input-stationary dataflow since weights are reduced precision

Summary

- Reducing precision is an effective way to reduce compute and storage costs
  - Widely exploited in industry already
- Fine tuning is critical for maintaining accuracy
  - Retraining needed for lower precision, especially binary nets
- Weight sharing reduces storage but not necessarily compute
- There are a **LOT** of publications in this space!

Recommended Reading

- Textbook Chapter 7
  - https://doi.org/10.1007/978-3-031-01766-7

- V. Camus et al., "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* October 2019
  - https://ieeexplore.ieee.org/abstract/document/8887521/