Exploring Low-Precision Formats in MAC Units for DNN Training

Silviu Filip
Inria Rennes
silviu.filip@inria.fr

joint work with Olivier Sentieys, Sami Ben Ali, Mariko Tatsumi, Guy Lemieux

Overview

Introduction

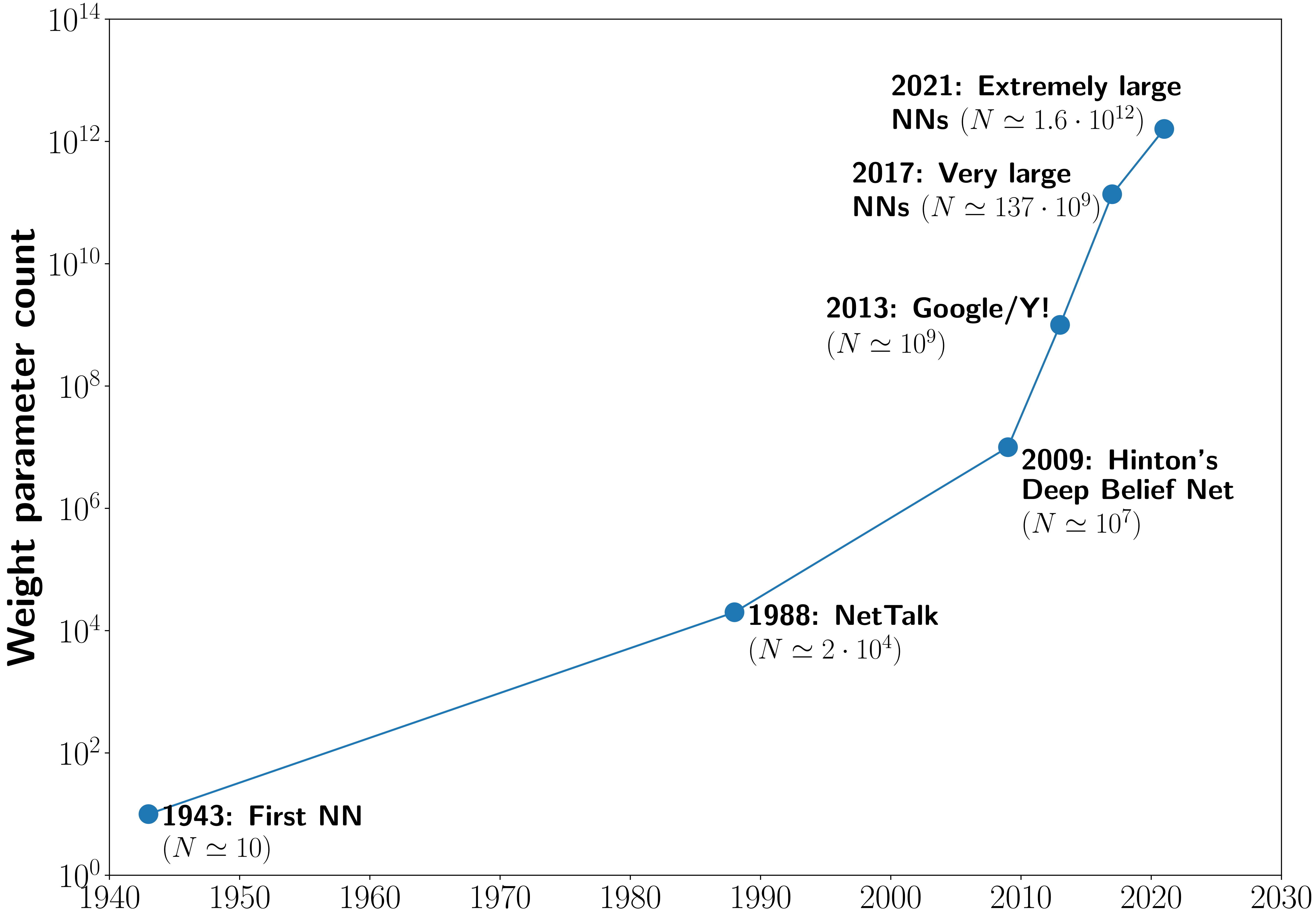
- Motivation: energy-efficient ML & the need for compression
- Quantization & low-precision computations for DNN training

Quantization for training acceleration

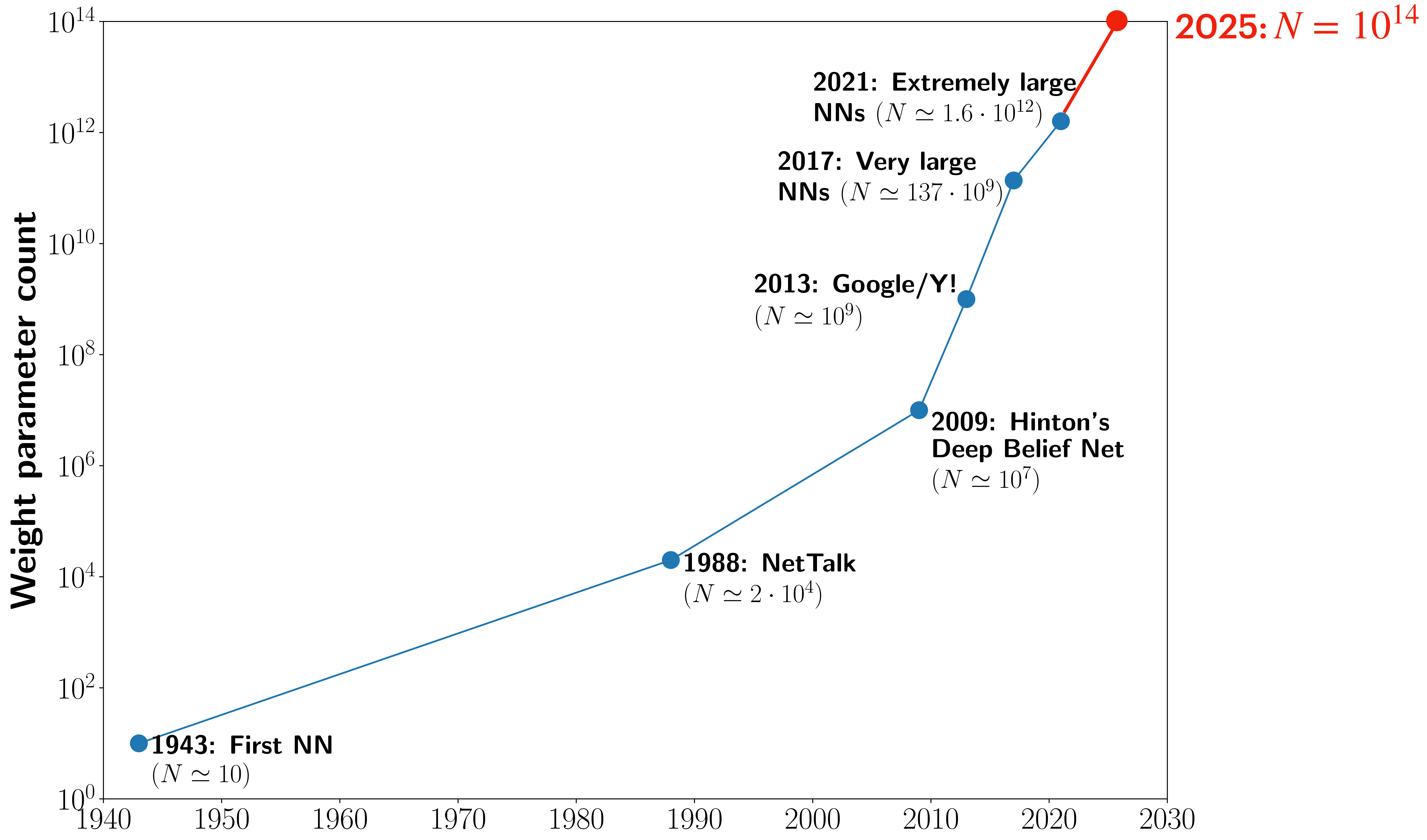
- **Custom precision simulation tools for DNN training acceleration**
- **Mixed precision MAC design space exploration for DNN training**

Summary & conclusions

Deep neural networks are growing fast

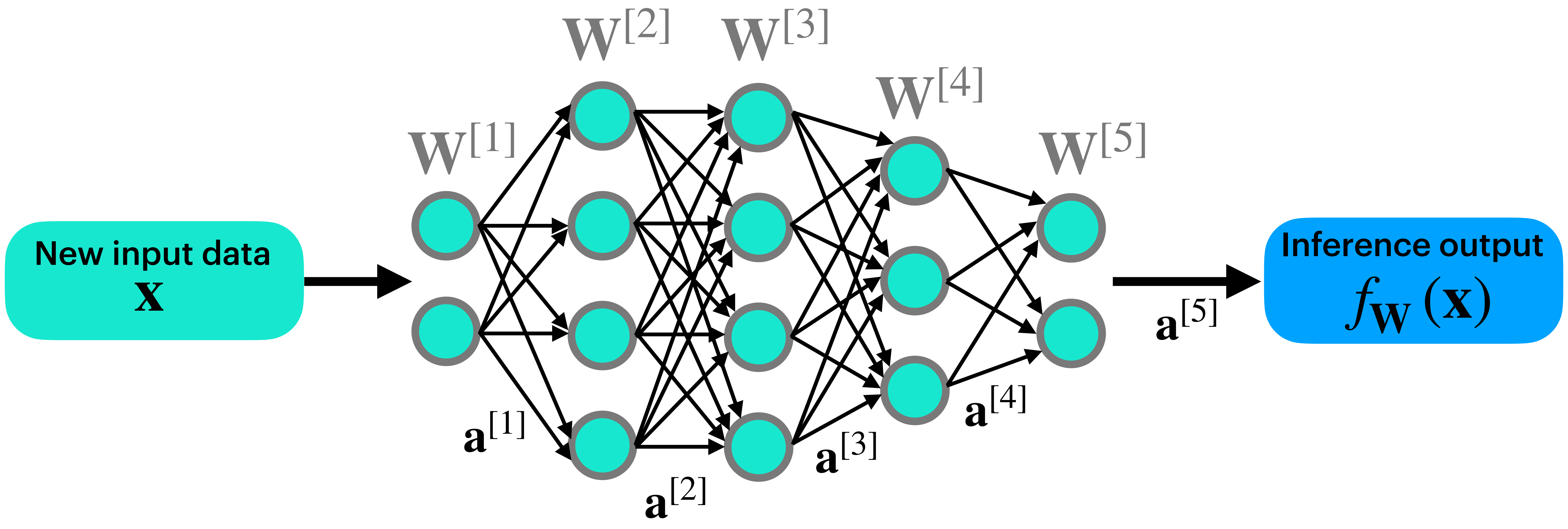


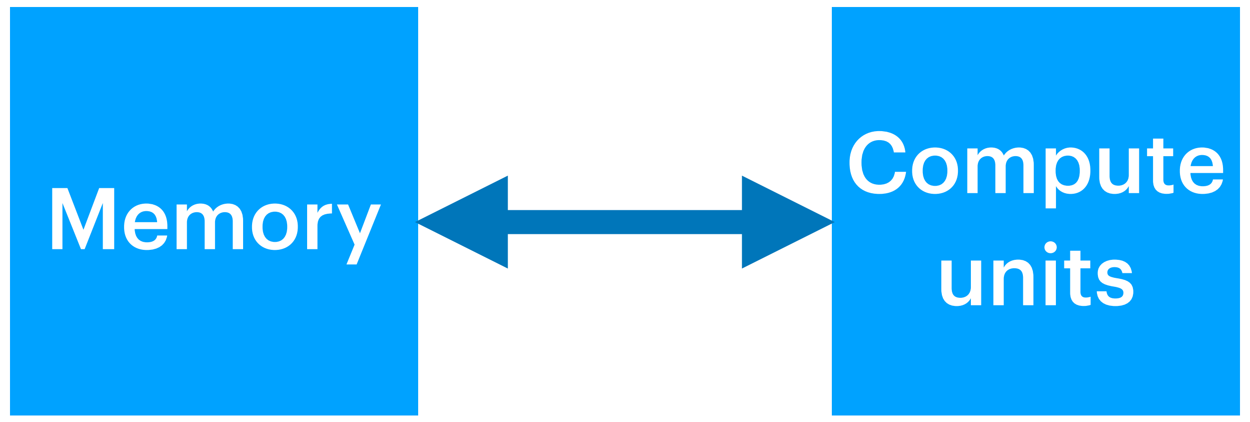
Deep neural networks are growing fast



The data movement bottleneck

Trained DNN model





Data movement

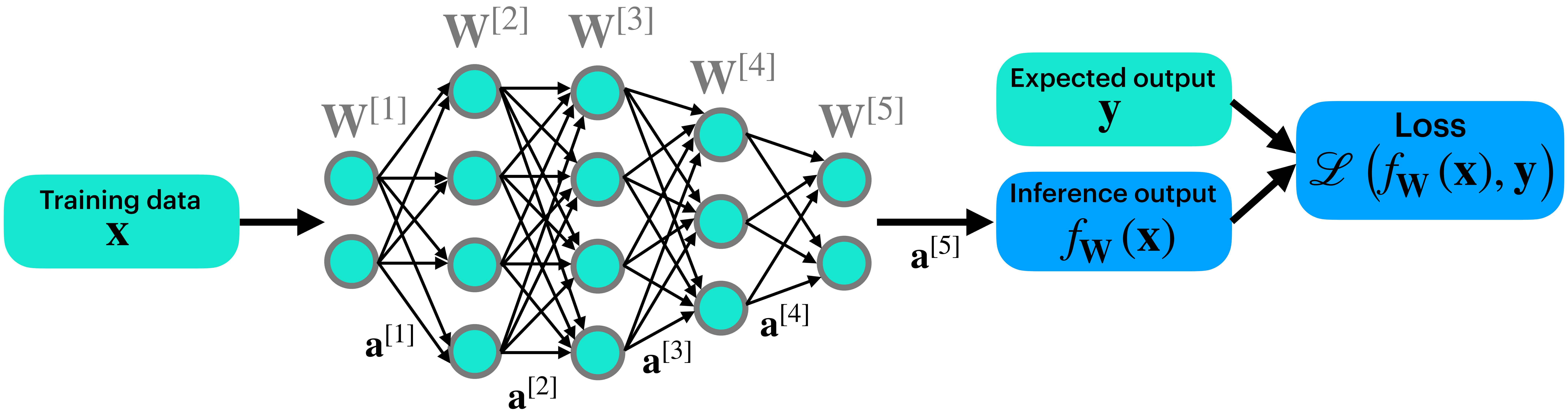
- move input data & model from memory to compute units
- send partial results back to memory

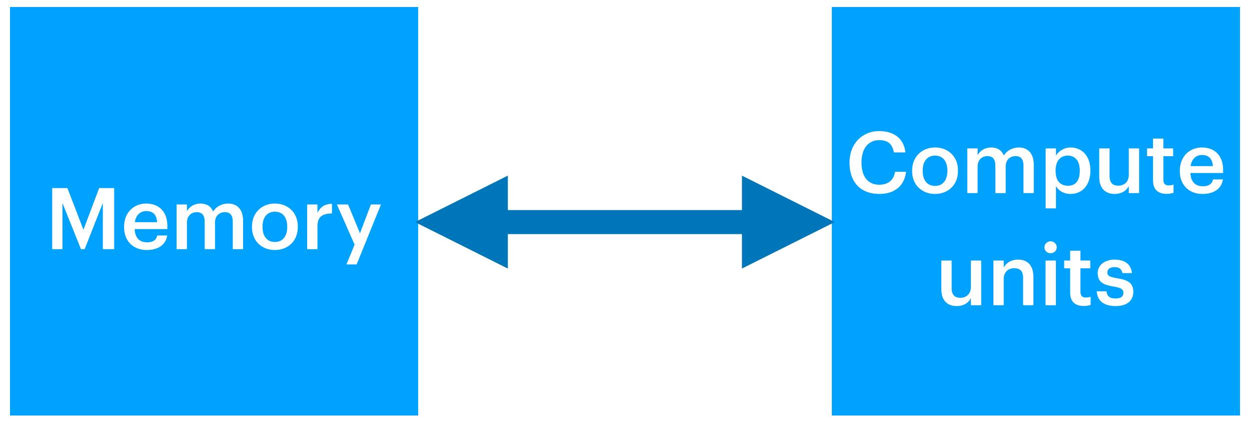
Computations

- vector/matrix manipulations
- done on CPU, GPU, DSP, or custom accelerators (e.g., FPGA, ASIC)

The data movement bottleneck

Training a DNN model





Data movement

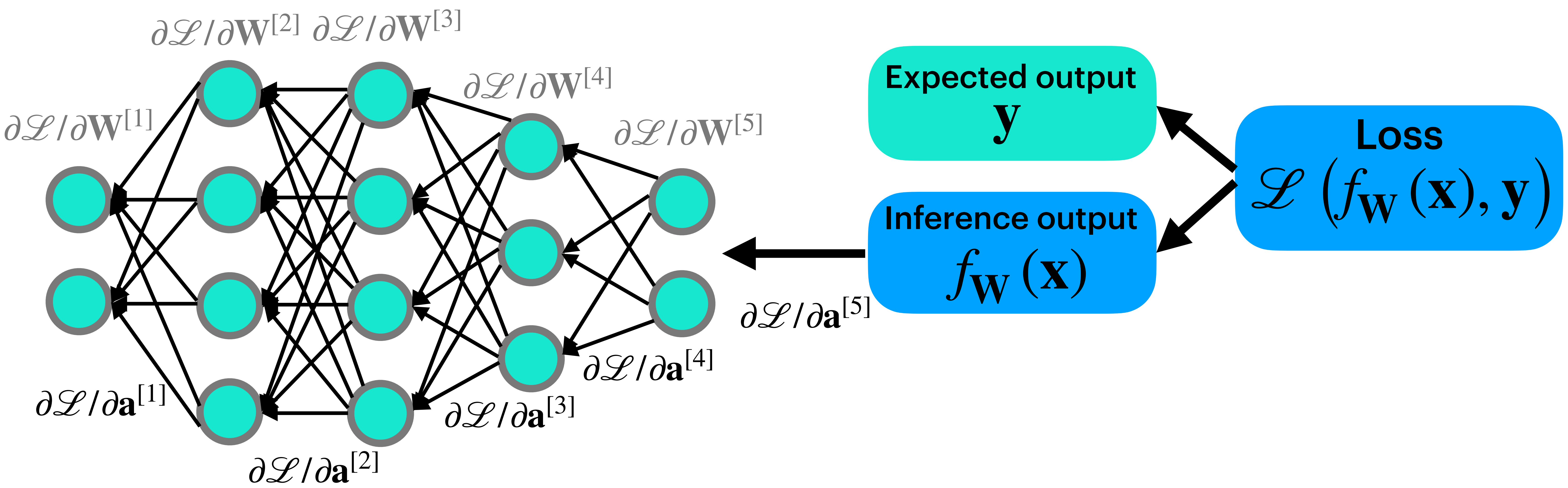
- move input data & model from memory to compute units
- send partial results back to memory

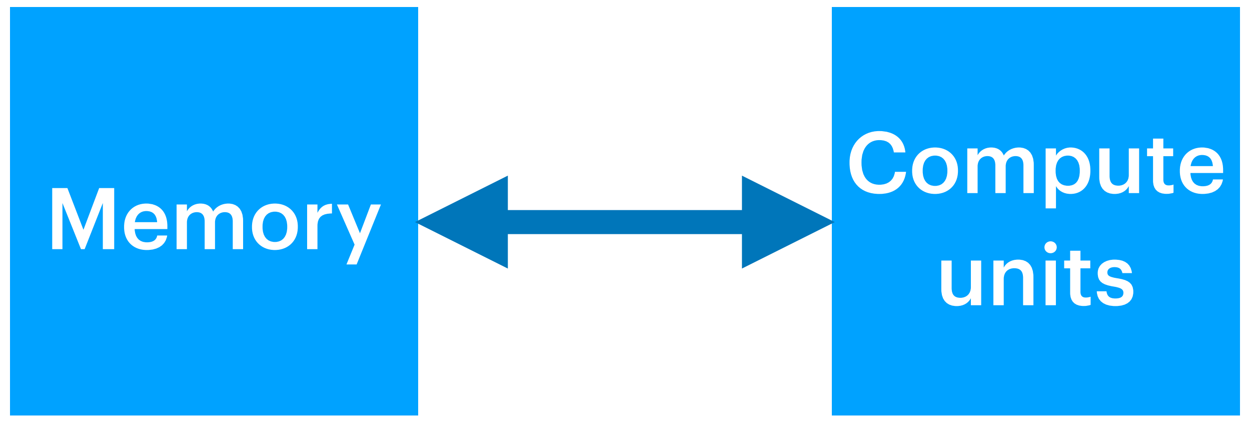
Computations

- vector/matrix manipulations
- done on CPU, GPU, DSP, or custom accelerators (e.g., FPGA, ASIC)

The data movement bottleneck

Training a DNN model





Computations

- vector/matrix manipulations
- done on CPU, GPU, DSP, or custom accelerators (e.g., FPGA, ASIC)

Data movement

- move input data & model from memory to compute units
- send partial results back to memory

What is DNN quantization?

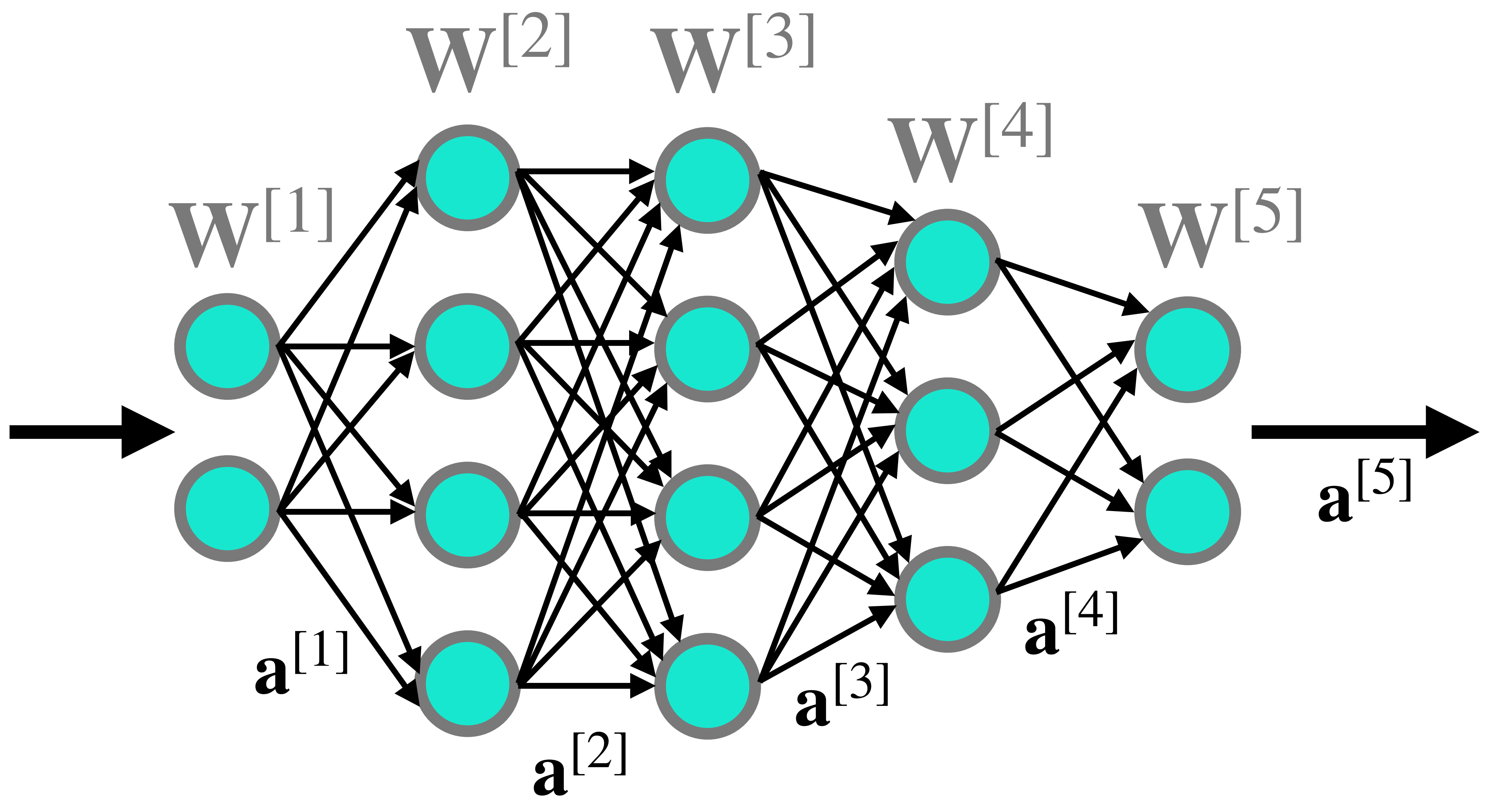
A visual quantization example:

- using fewer bits per pixel in an image

Image source: Here's why quantization matters for AI, *Jilei Hou*, 2019

What is DNN quantization?

During inference (i.e., for a trained network):



A visual quantization example:

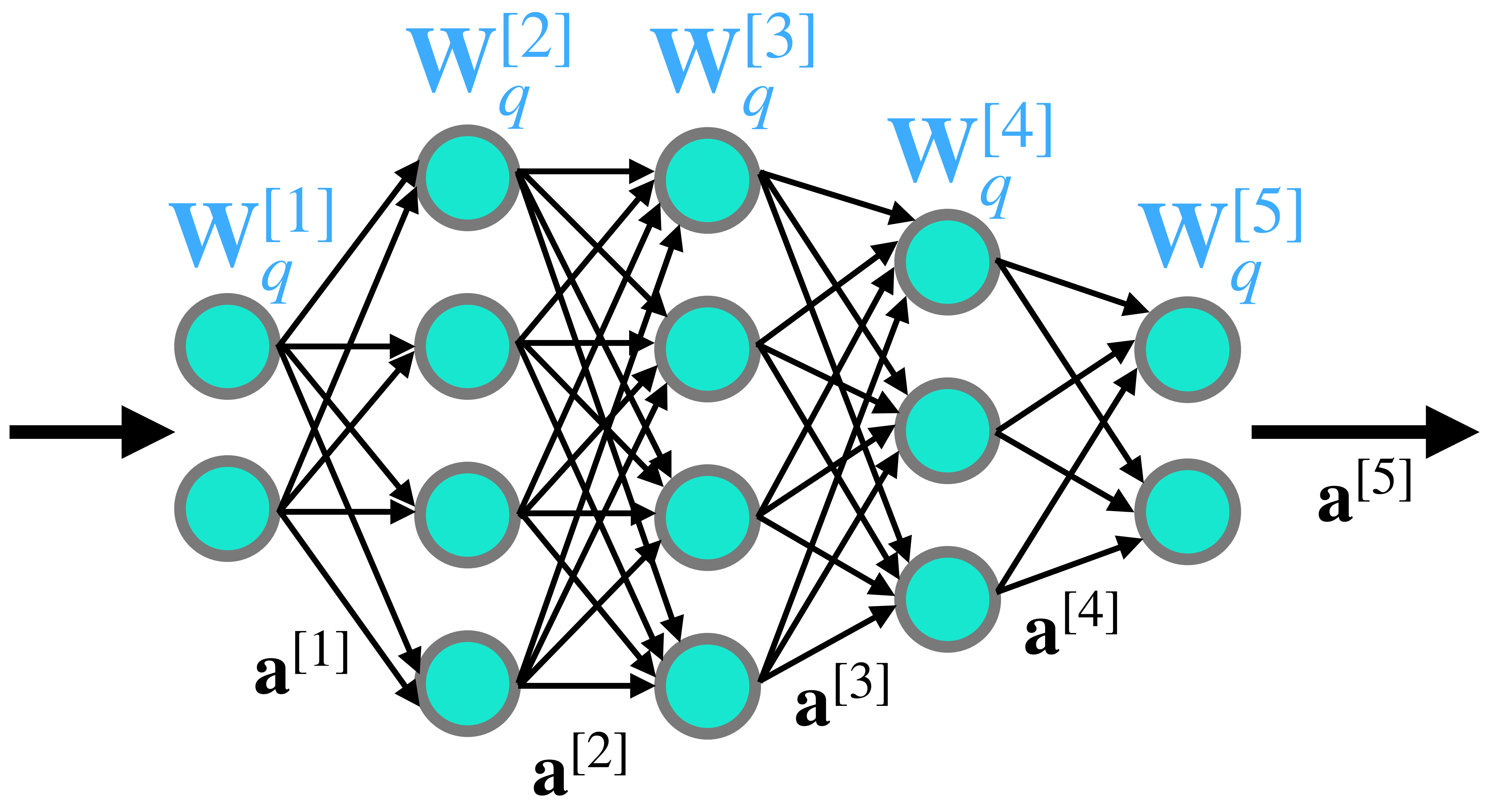
- using fewer bits per pixel in an image

Image source: Here's why quantization matters for AI, *Jilei Hou*, 2019

What is DNN quantization?

During inference (i.e., for a trained network):

- store network parameters in low precision



A visual quantization example:

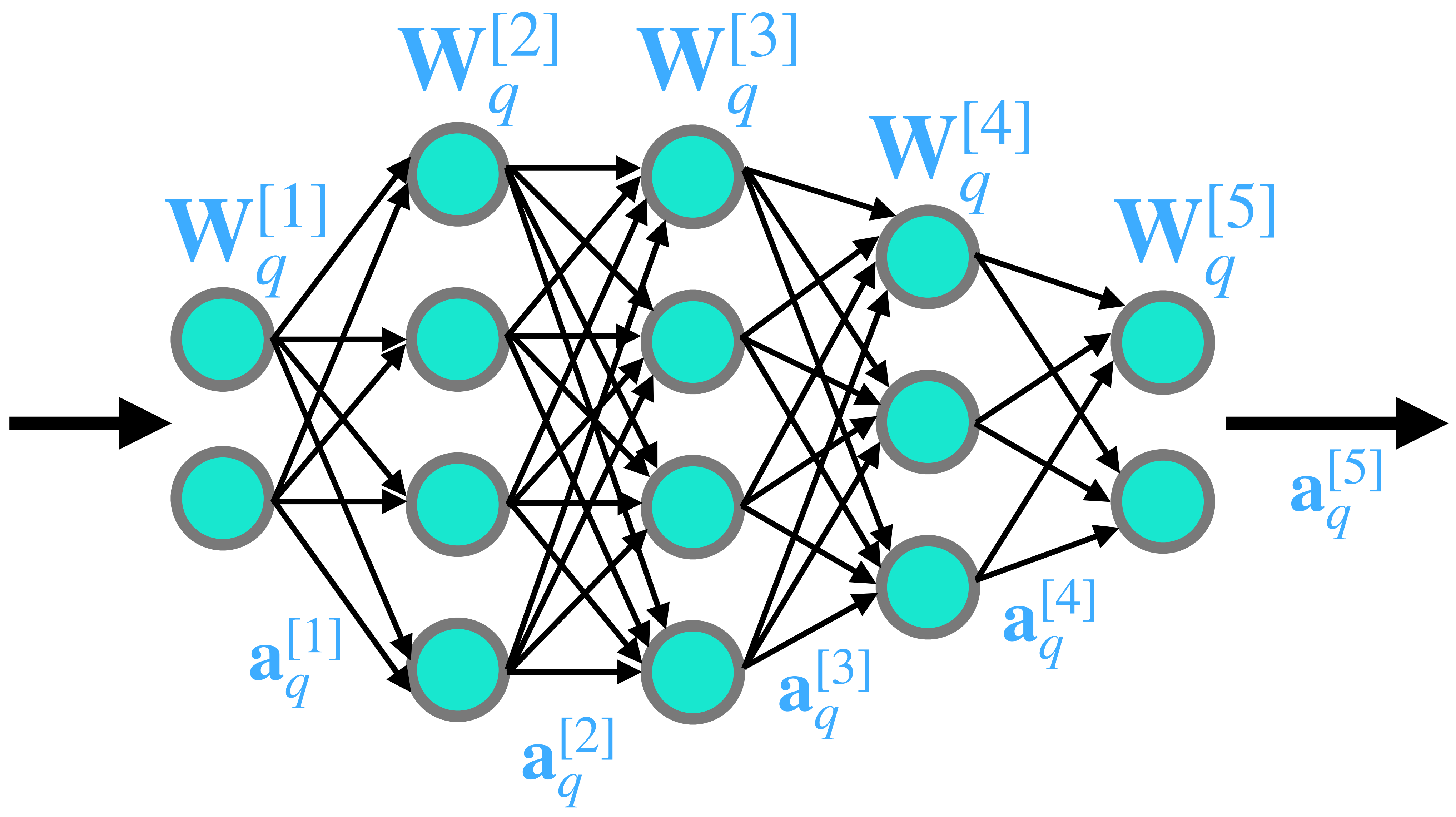
- using fewer bits per pixel in an image

Image source: Here's why quantization matters for AI, *Jilei Hou*, 2019

What is DNN quantization?

During inference (i.e., for a trained network):

- store network parameters in low precision
- store/compute intermediate signals in low precision



A visual quantization example:

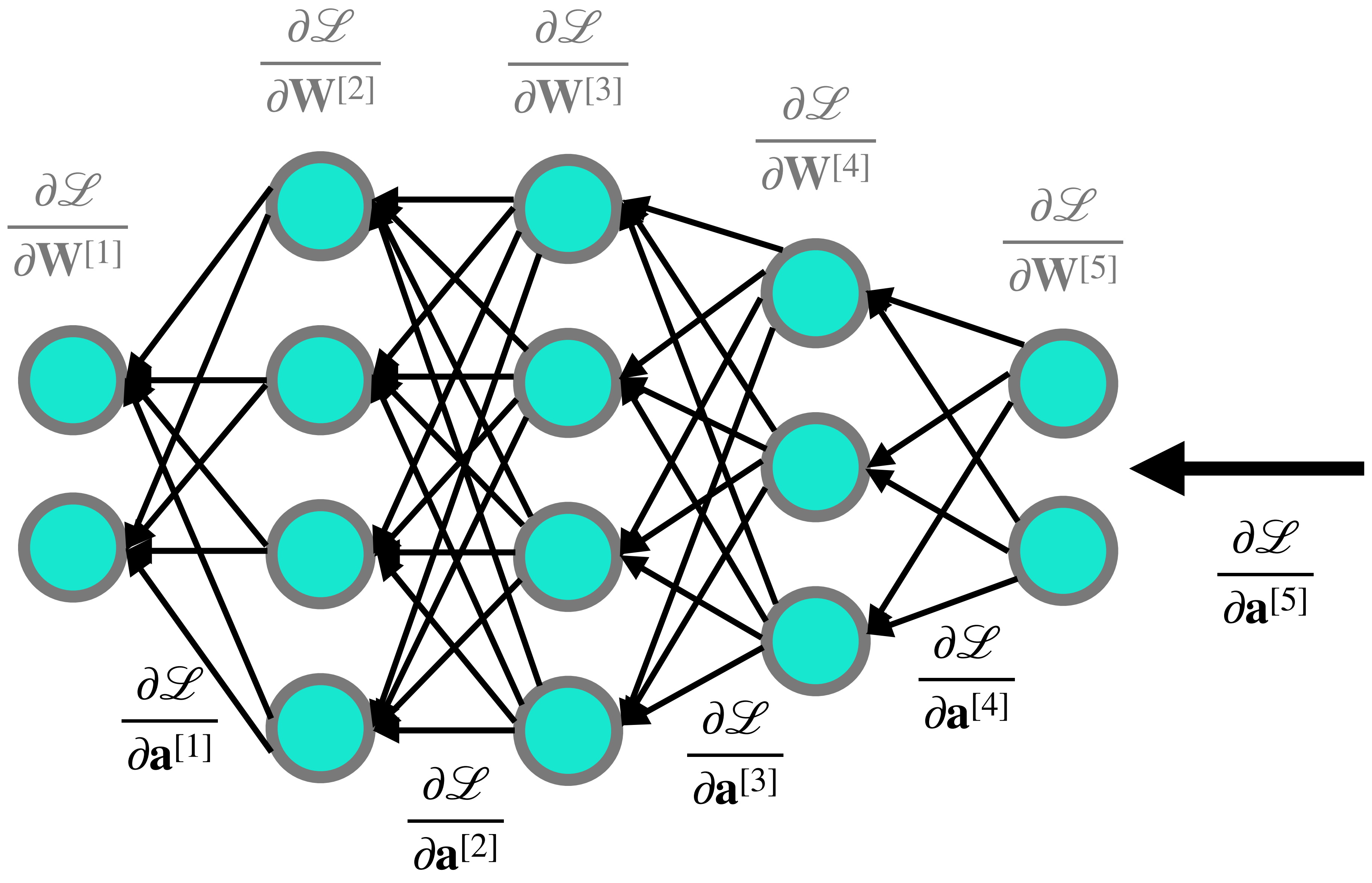
- using fewer bits per pixel in an image

Image source: Here's why quantization matters for AI, *Jilei Hou*, 2019

What is DNN quantization?

During inference (i.e., for a trained network):

- store network parameters in low precision
- store/compute intermediate signals in low precision



During training:

A visual quantization example:

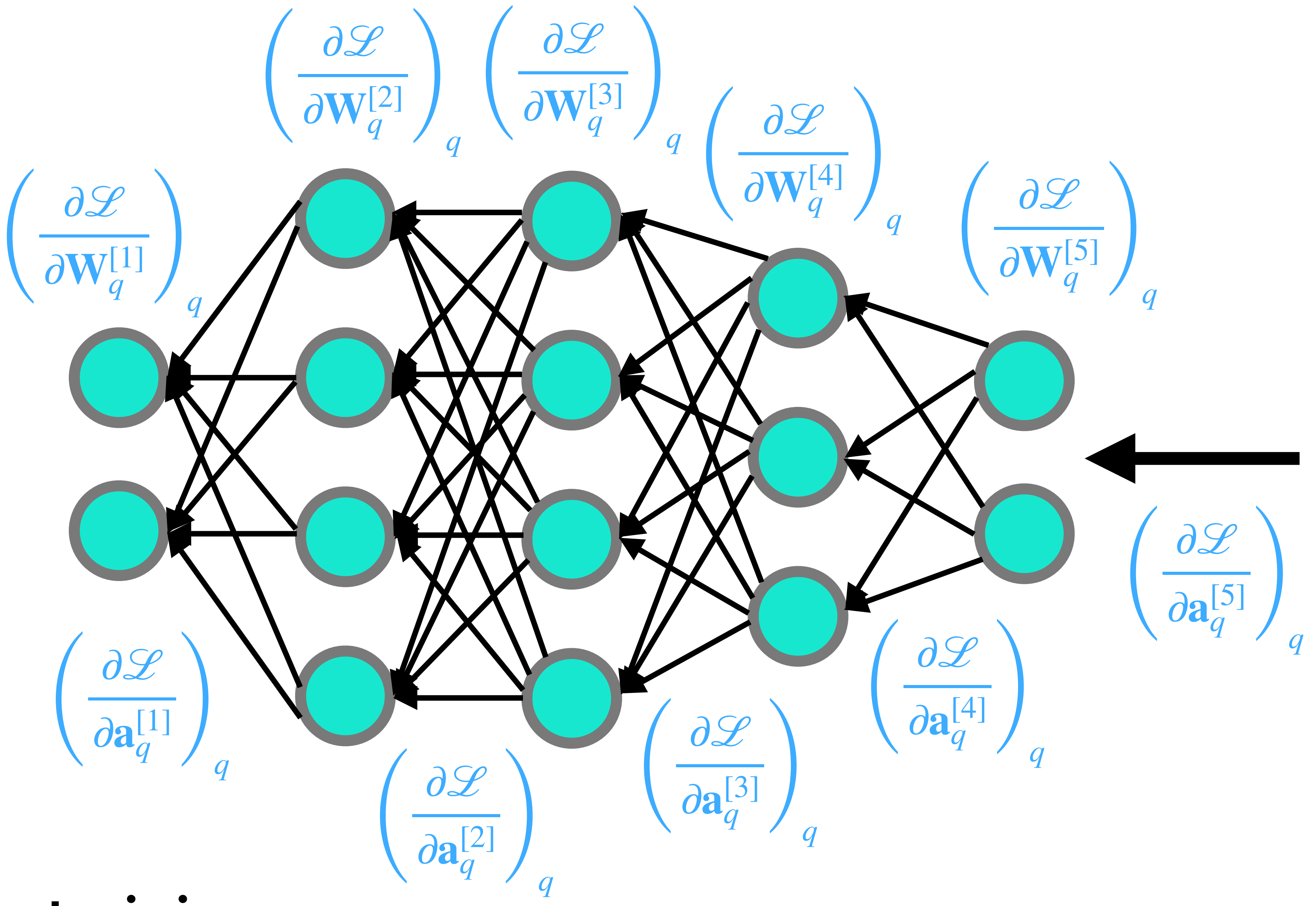
- using fewer bits per pixel in an image

Image source: Here's why quantization matters for AI, *Jilei Hou*, 2019

What is DNN quantization?

During inference (i.e., for a trained network):

- store network parameters in low precision
- store/compute intermediate signals in low precision



During training:

- store/compute back propagated gradients in low precision

A visual quantization example:

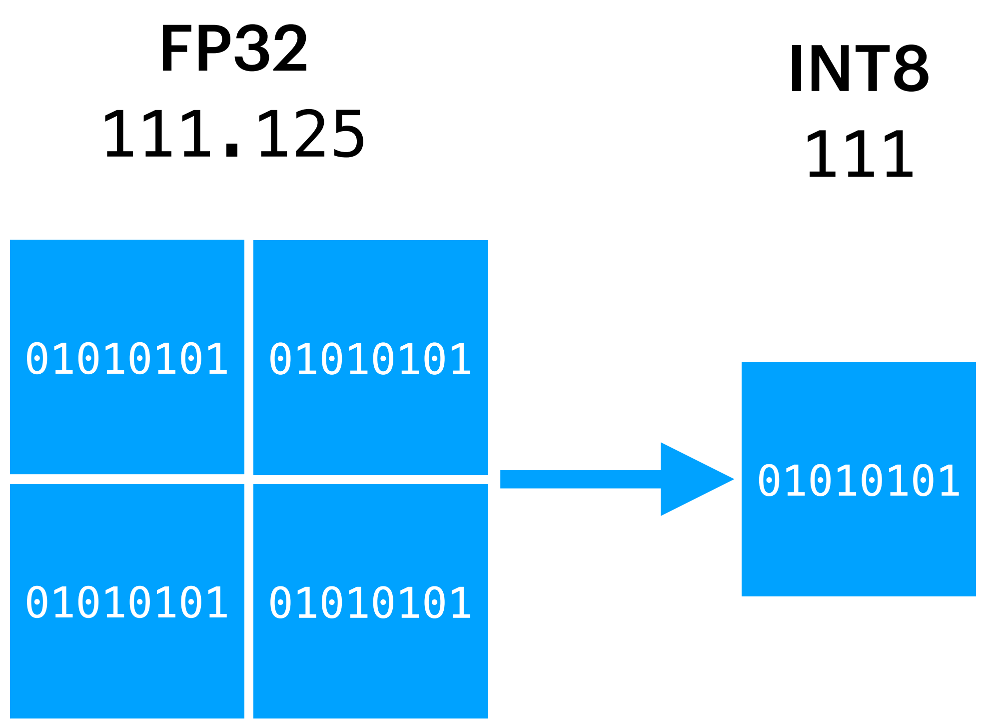
- using fewer bits per pixel in an image

Image source: Here's why quantization matters for AI, *Jilei Hou*, 2019

Quantization effects: the good

Memory usage

storage needed for weights and and activations is proportional to the bit width used



Power consumption

energy is significantly reduced for both computations and memory accesses

ADD energy (pJ)			
INT8	INT32	FP16	**FP32**
0.03	0.1	0.4	**0.9**
30x energy reduction			

MULT energy (pJ)			
INT8	INT32	FP16	**FP32**
0.2	3.1	1.1	**3.7**
18.5x energy reduction			

Memory access energy (pJ)	
Cache (64-bit)	
8KB	10
32KB	20
1MB	100
DRAM	1300-2600
Up to 4x energy reduction	

Latency

less memory access and simpler computations lead to faster runtimes and reduced latency

Silicon area

8-bit arithmetic and below requires less area than larger bit width FP compute units

MULT area (μm^2)			
INT8	INT32	FP16	**FP32**
282	3495	1640	**7700**
27x area reduction			

ADD area (μm^2)			
INT8	INT32	FP16	**FP32**
36	137	1360	**4184**
116x area reduction			

Sources: Mark Horowitz (Stanford), energy based on ASIC, area based on TSMC 45nm process
Wikimedia Commons

Why quantization for training?

➡quantization for inference acceleration is popular & widely studied in recent years

➡quantization for training acceleration is less studied, but still important

Why quantization for training?

➡quantization for inference acceleration is popular & widely studied in recent years

➡quantization for training acceleration is less studied, but still important

Why?

- SOTA models tend to get bigger & bigger, requiring more time & memory to train
- growing need & interest for edge/on-site learning

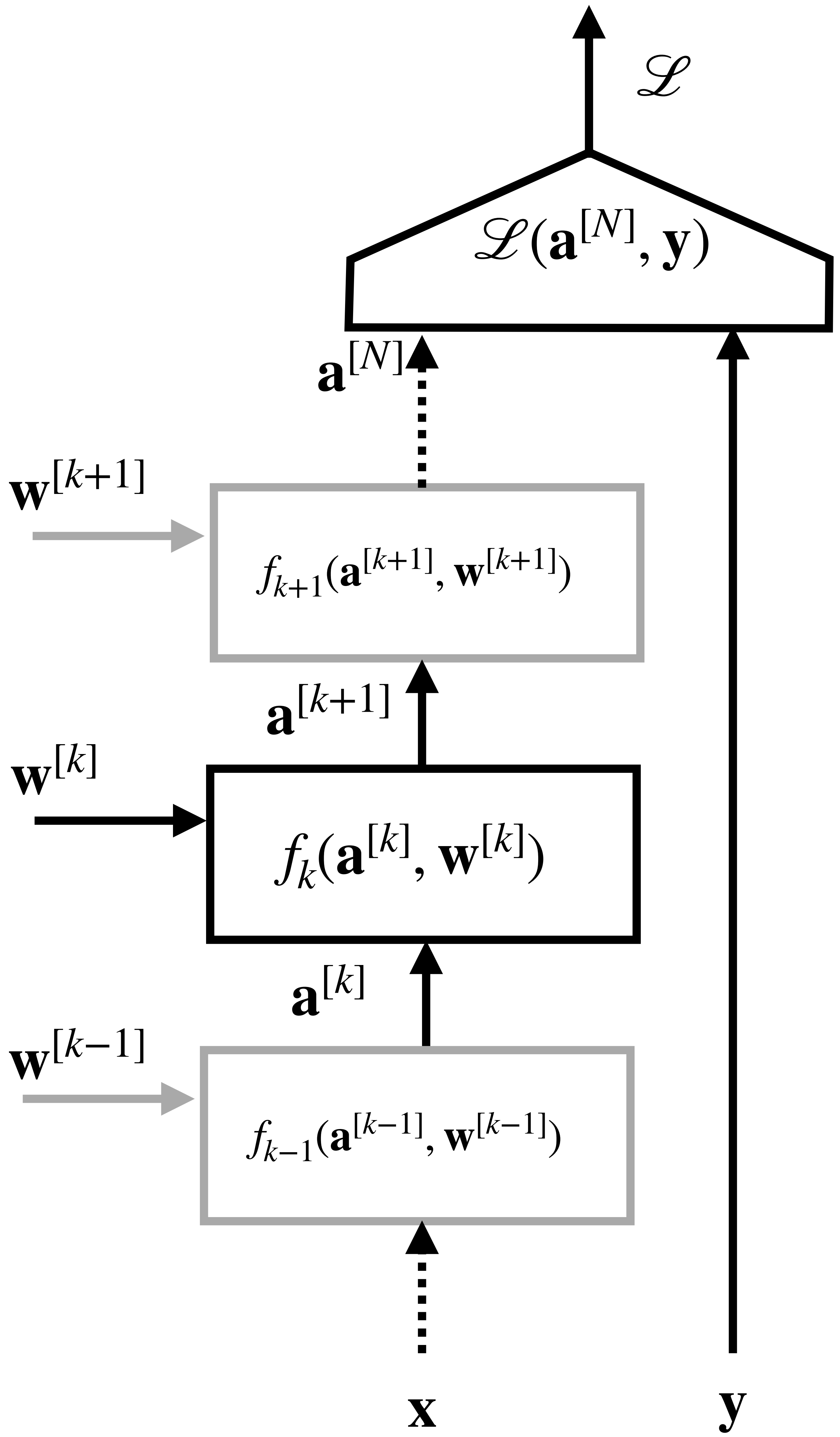
Estimated cost of training recent NLP models (adapted from [1])

Model	Hardware	Power (W)	Hours
$\text{Transformer}_{\text{base}}$	P100x8	1415.78	12
$\text{Transformer}_{\text{big}}$	P100x8	1515.43	84
ELMo	P100x3	517.66	336
$\text{BERT}_{\text{base}}$	V100x64	12041.51	79
$\text{BERT}_{\text{base}}$	TPUv2x64	N/A	96
NAS	P100x8	1515.43	274120
NAS	TPUv2x1	N/A	32623
GTP-2	TPUv2x32	N/A	168

[1] Energy and Policy Considerations for Deep Learning in NLP, *Strubell et al.*, arXiv:1906.02243, 2019

Why is training expensive?

➡during inference/forward path, we need to compute activations



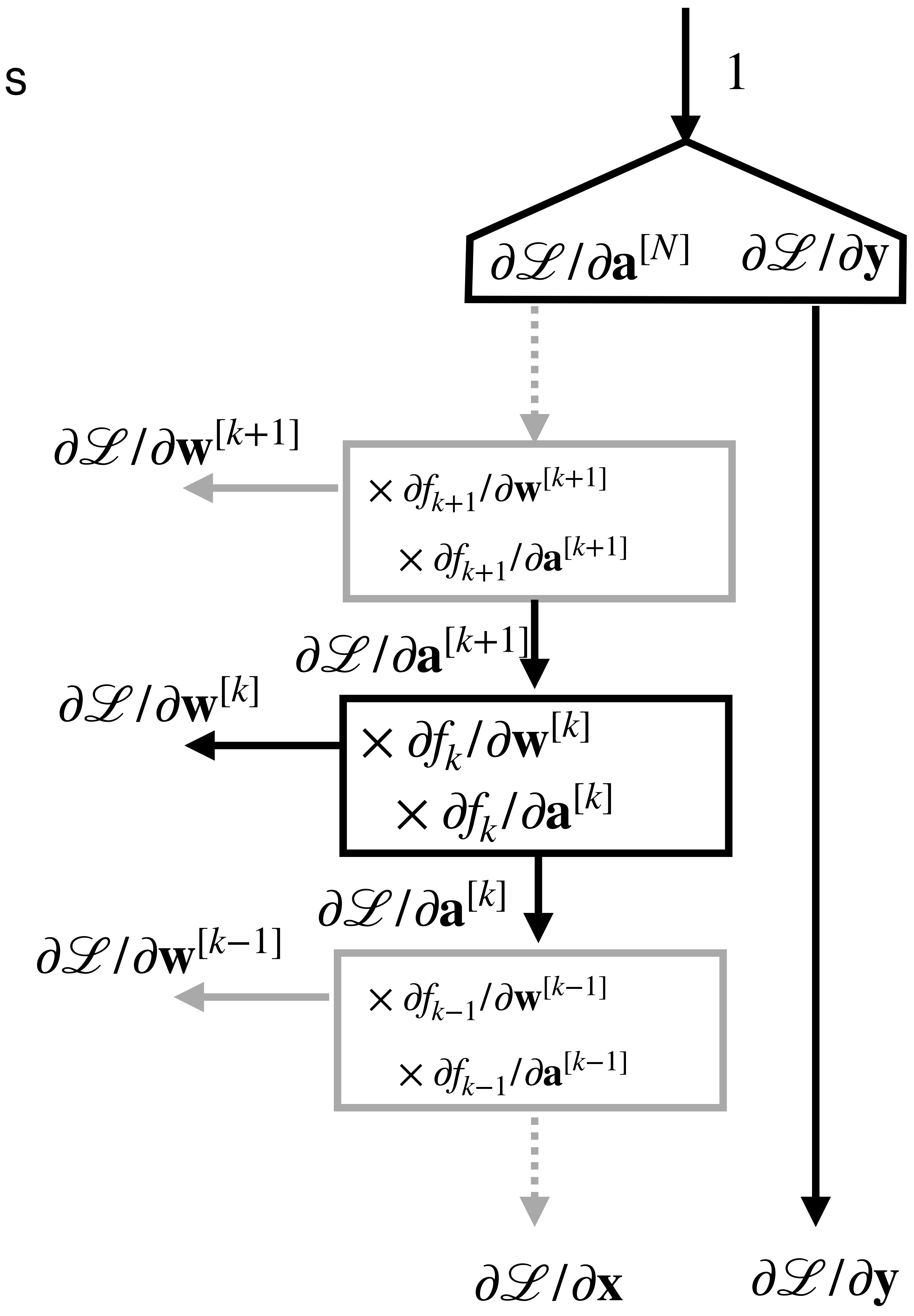
Why is training expensive?

➡during inference/forward path, we need to compute activations

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} - \alpha_t \frac{\partial \mathscr{L}}{\partial \mathbf{W}_{(t)}}$$

➡during training (backward path), we also need gradients:

- with respect to the activations (the $\mathbf{a}^{[k]}$ vectors)
- with respect to the parameters (the $\mathbf{w}^{[k]}$ vectors)



Why is training expensive?

➡during inference/forward path, we need to compute activations

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} - \alpha_t \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(t)}}$$

➡during training (backward path), we also need gradients:

- with respect to the activations (the $\mathbf{a}^{[k]}$ vectors)
- with respect to the parameters (the $\mathbf{w}^{[k]}$ vectors)

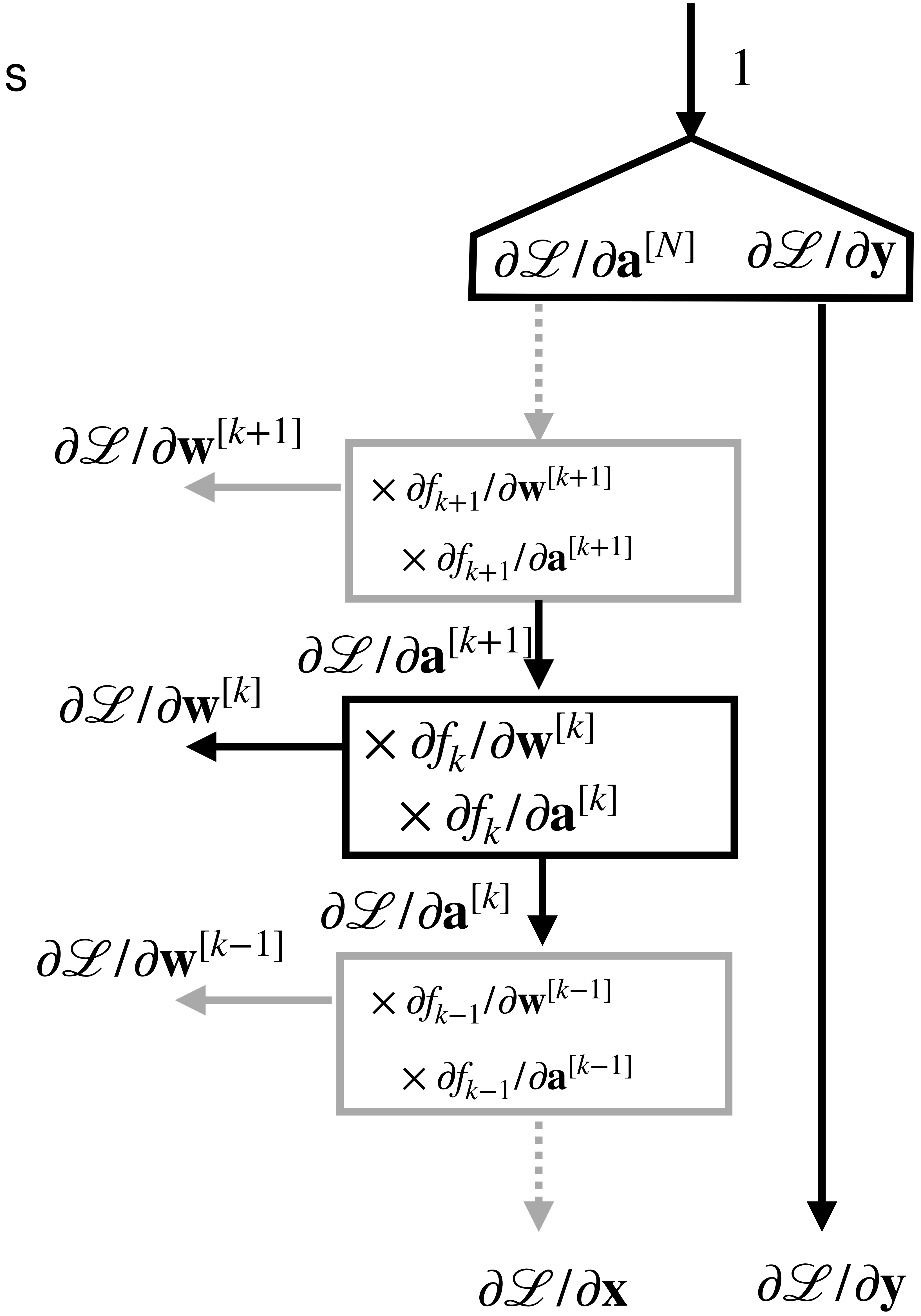
➡it is hard to reduce precision of operations during training

Why?

- vanishing & exploding gradients during back propagation
- small updates to parameters, i.e., $|w| \gg |\partial \mathcal{L} / \partial w|$

➡a (possibly) large dynamic range is needed

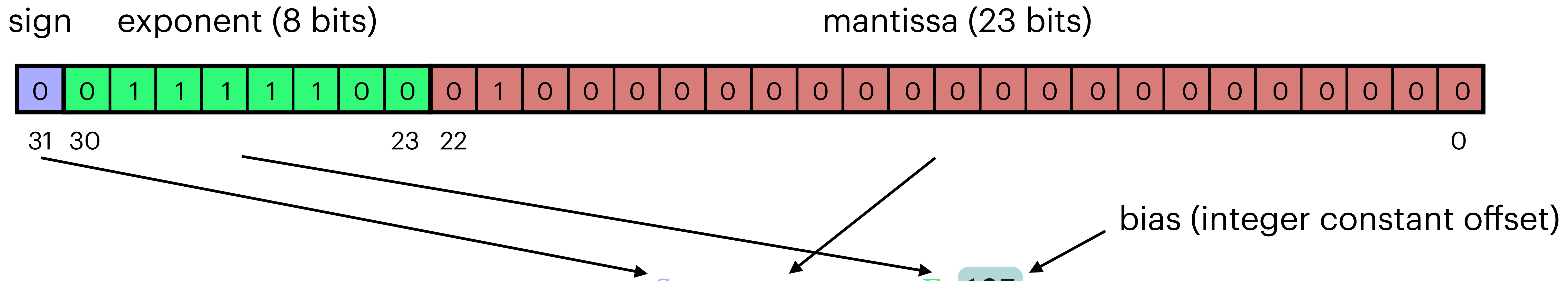
➡**use floating-point arithmetic**



Floating-point formats

➡the de facto family of formats for working with real numbers in the digital world

Example: The IEEE-754 `float32` format



$$x = (-1)^{S} \times 1.M_{(2)} \times 2^{E-\boxed{127}}$$

$$x = (-1)^{0} \times 1.01_{(2)} \times 2^{124-127} = 1.25 \times 2^{-3} = 0.15625$$

Floating-point formats

➡several formats are used in practice:

Format	Mantissa size	Exponent size	Bias	Range	Unit roundoff
fp128	112	15	16383	$10^{\pm 4932}$	1×10^{-34}
fp64	52	11	1023	$10^{\pm 308}$	1×10^{-16}
fp32	23	8	127	$10^{\pm 38}$	6×10^{-8}
fp16	10	5	15	$10^{\pm 5}$	5×10^{-4}
tfloat32 (tf32)	10	8	127	$10^{\pm 38}$	5×10^{-4}
bfloat16 (bf16)	7	8	127	$10^{\pm 38}$	4×10^{-3}
fp8	3 2	4 5	7 15	$10^{\pm 2}$ $10^{\pm 5}$	6×10^{-2} 1×10^{-1}

established IEEE-754 formats

emerging formats

➡FP32 is the workhorse format for training AI models

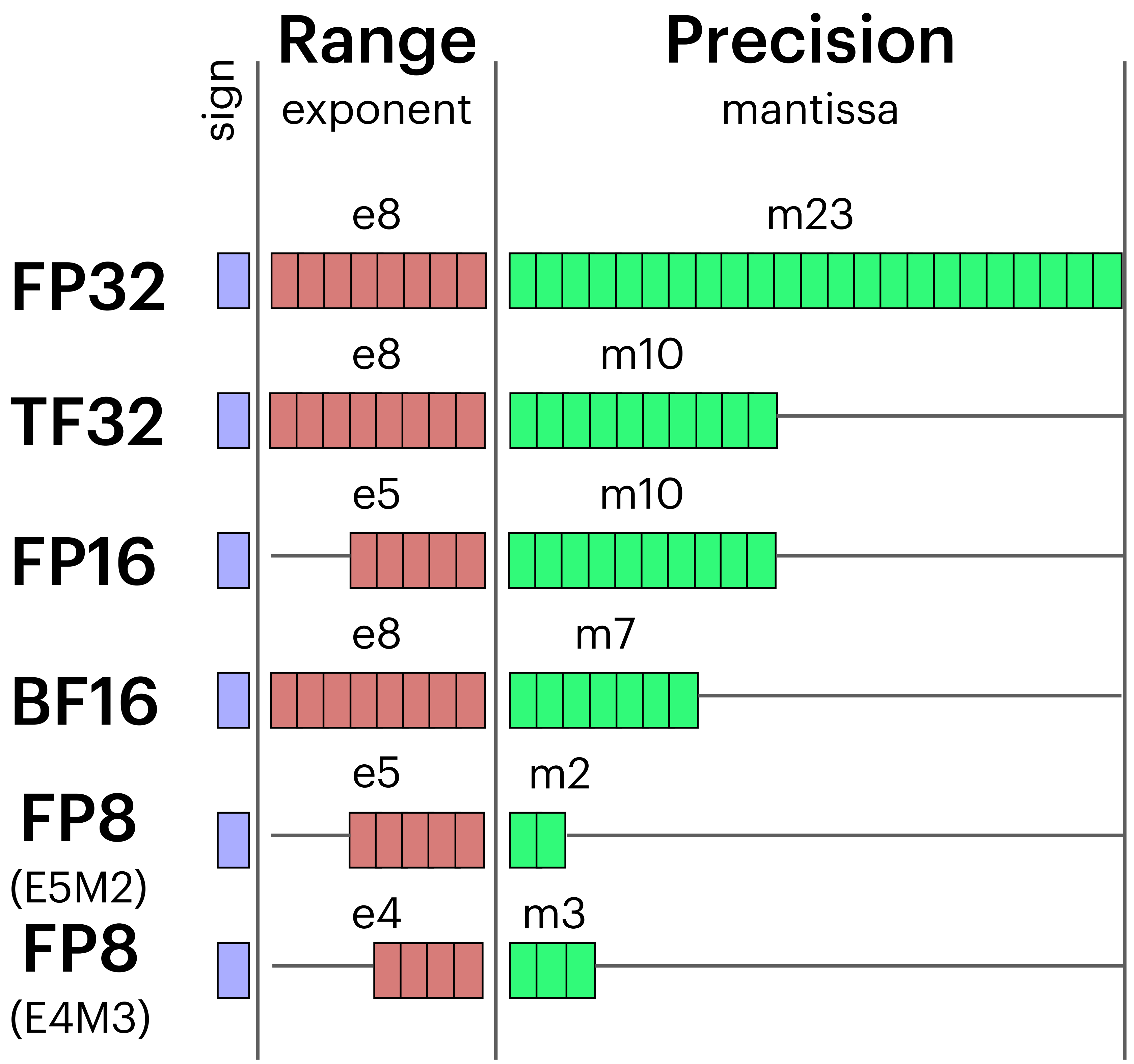
➡there are several emerging FP formats for AI acceleration

Floating-point formats

➡they offer various tradeoffs in terms of range, precision & performance

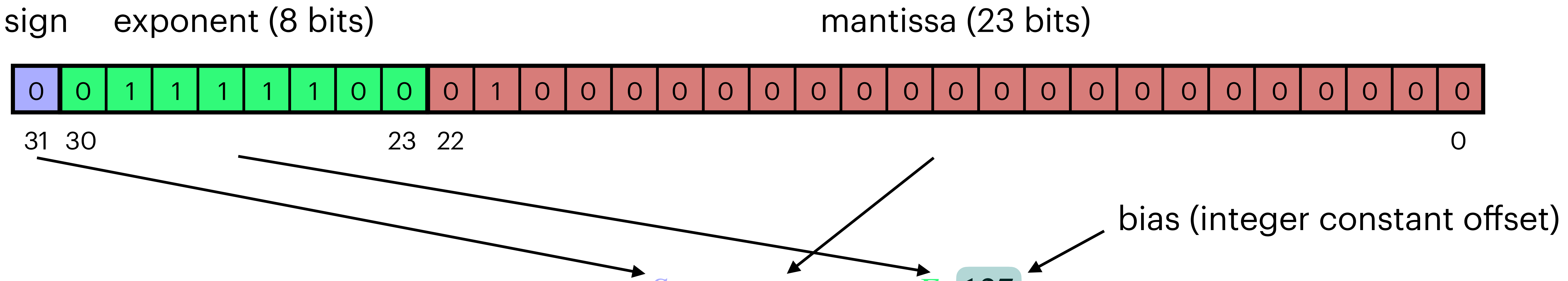
FP performance numbers for recent NVIDIA GPU architectures

Peak performance (TFLOPS)							
Device	Year	fp64	fp32	tfloat32	fp16	bfloat16	fp8
P100	2016	5	9	-	19	-	-
V100	2017-2019	8	16	-	125	-	-
A100	2020-2021	19	19	156	312	312	-
H100	2022	48	48	400	800	800	1600



Floating-point formats

When, where and how can we use smaller number formats during DNN training?



$$x = (-1)^{S} \times 1.M_{(2)} \times 2^{E-127}$$

$$x = (-1)^{0} \times 1.01_{(2)} \times 2^{124-127} = 1.25 \times 2^{-3} = 0.15625$$

➡exponent encoding is a offset-binary representation

- $E_{min} = 01_{(H)} - 7F_{(H)} = -126$
- $E_{max} = FE_{(H)} - 7F_{(H)} = 127$

E	$M = 0$	$M \neq 0$	Equation
$00_{(H)}$	± 0	subnormal value	$(-1)^{S} \times 0.M_{(2)} \times 2^{-126}$
$01_{(H)},\ldots,FE_{(H)}$	normal value		$(-1)^{S} \times 1.M_{(2)} \times 2^{E-127}$
$FF_{(H)}$	$\pm\infty$	NaN	

Floating-point formats

When, where and how can we use smaller number formats during DNN training?

sign exponent (8 bits) mantissa (23 bits)

0	0	1	1	1	1	1	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
31	30							23	22																						0

bias (integer constant offset)

$$x = (-1)^{S} \times 1.M_{(2)} \times 2^{E-\boxed{127}}$$

$$x = (-1)^{0} \times 1.01_{(2)} \times 2^{124-127} = 1.25 \times 2^{-3} = 0.15625$$

➡exponent encoding is a offset-binary representation

- E_{min} = 01(H) − 7F(H) = -126
- E_{max} = FE(H) − 7F(H) = 127

E	$M = 0$	$M \neq 0$	Equation
00(H)	± 0	subnormal value	$(-1)^{S} \times 0.M_{(2)} \times 2^{-126}$
01(H),…,FE(H)	normal value		$(-1)^{S} \times 1.M_{(2)} \times 2^{E-127}$
FF(H)	$\pm\infty$	NaN	

Training acceleration landscape

➡SOTA training acceleration methods are based on *mixed precision computing*

Idea: perform parameter updates in **high precision (HP)** + other ops in **low precision (LP)**

Training acceleration landscape

➡SOTA training acceleration methods are based on *mixed precision computing*

Idea: perform parameter updates in **high precision (HP)** + other ops in **low precision (LP)**

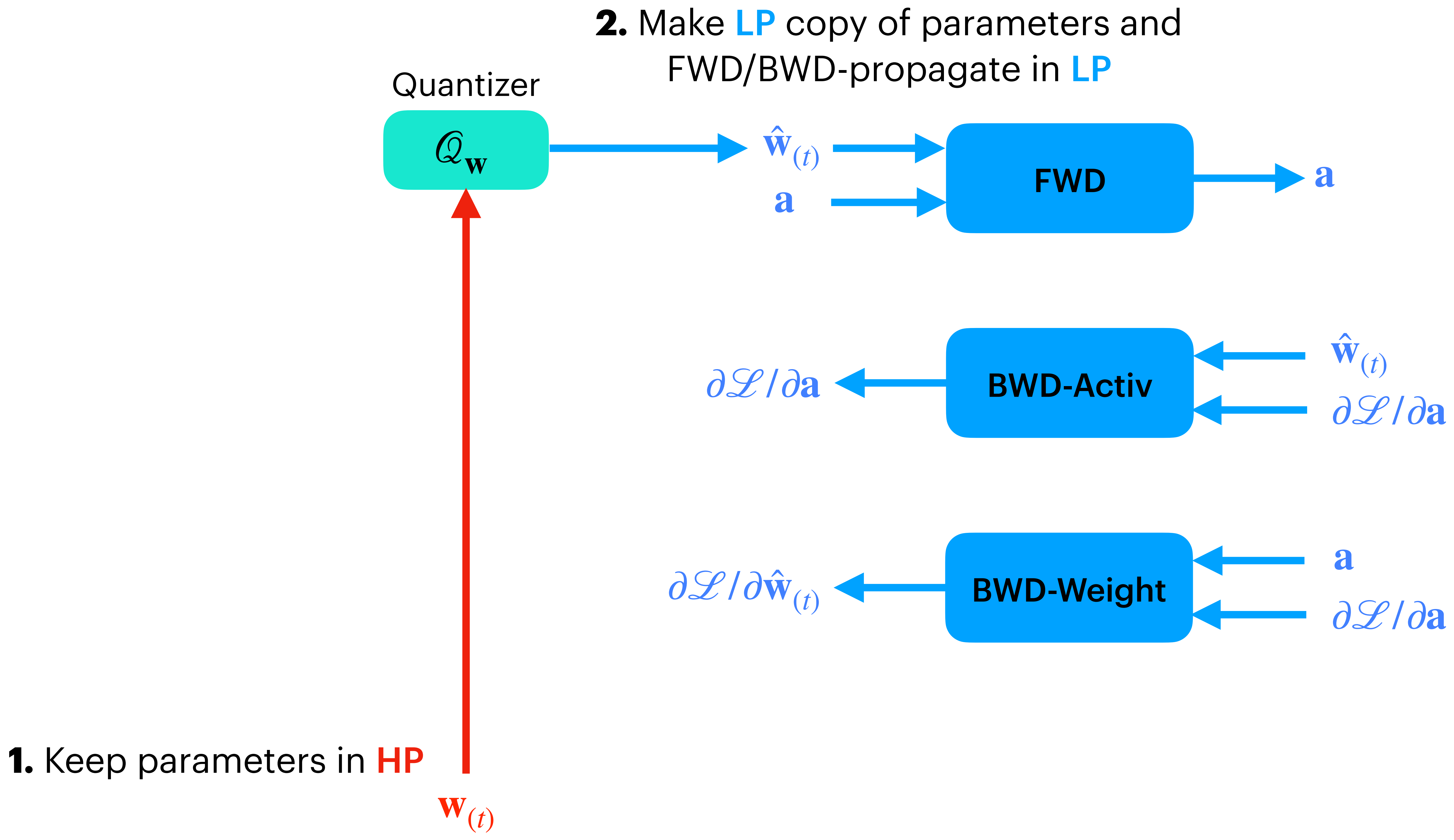
1. Keep parameters in **HP**

$\mathbf{w}_{(t)}$

Training acceleration landscape

➡SOTA training acceleration methods are based on *mixed precision computing*

Idea: perform parameter updates in high precision (HP) + other ops in low precision (LP)

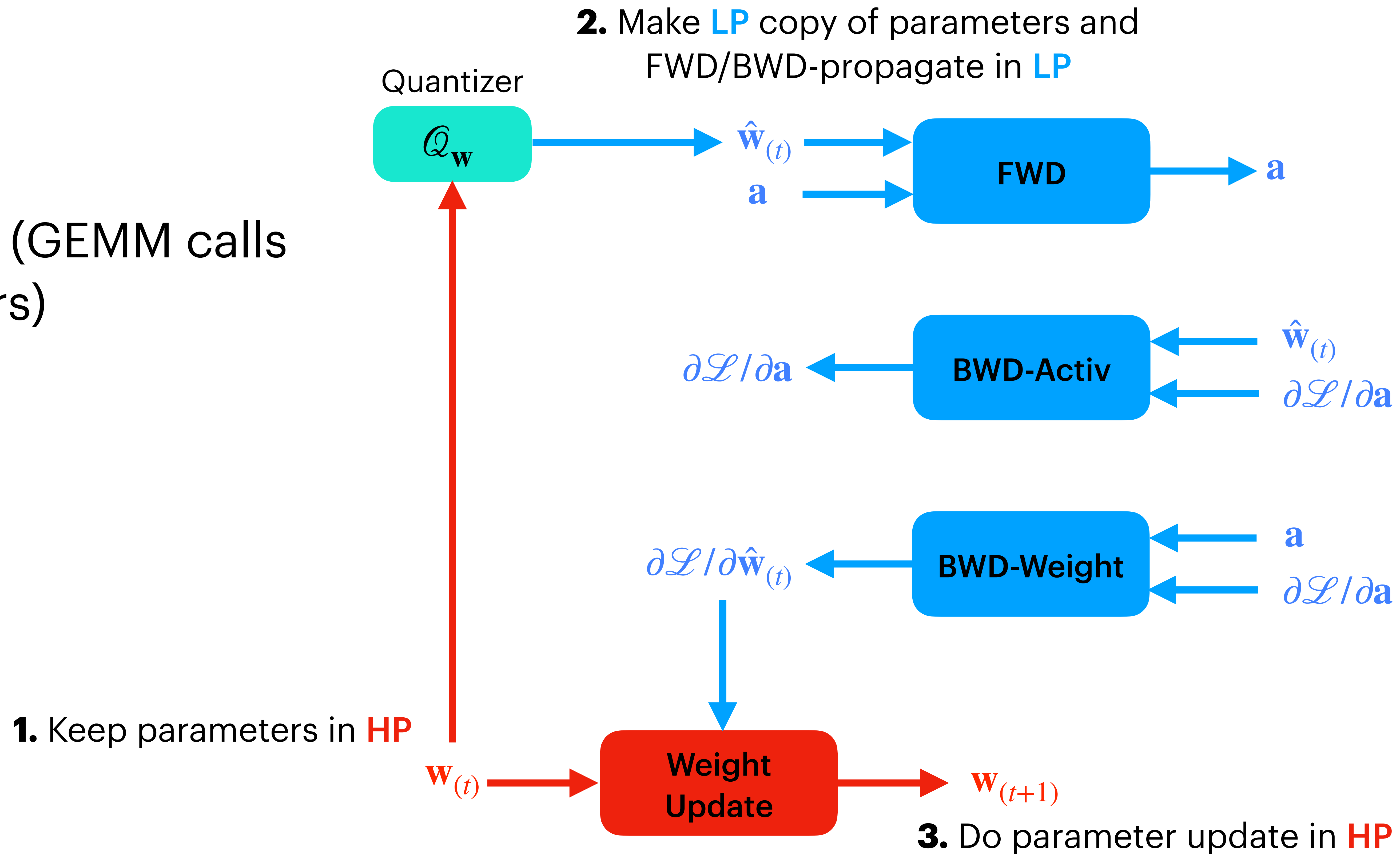


Training acceleration landscape

➡SOTA training acceleration methods are based on *mixed precision computing*

Idea: perform parameter updates in **high precision (HP)** + other ops in **low precision (LP)**

➡most compute happens in **FWD/BWD-part** (GEMM calls for fully connected and convolutional layers)



Training acceleration landscape

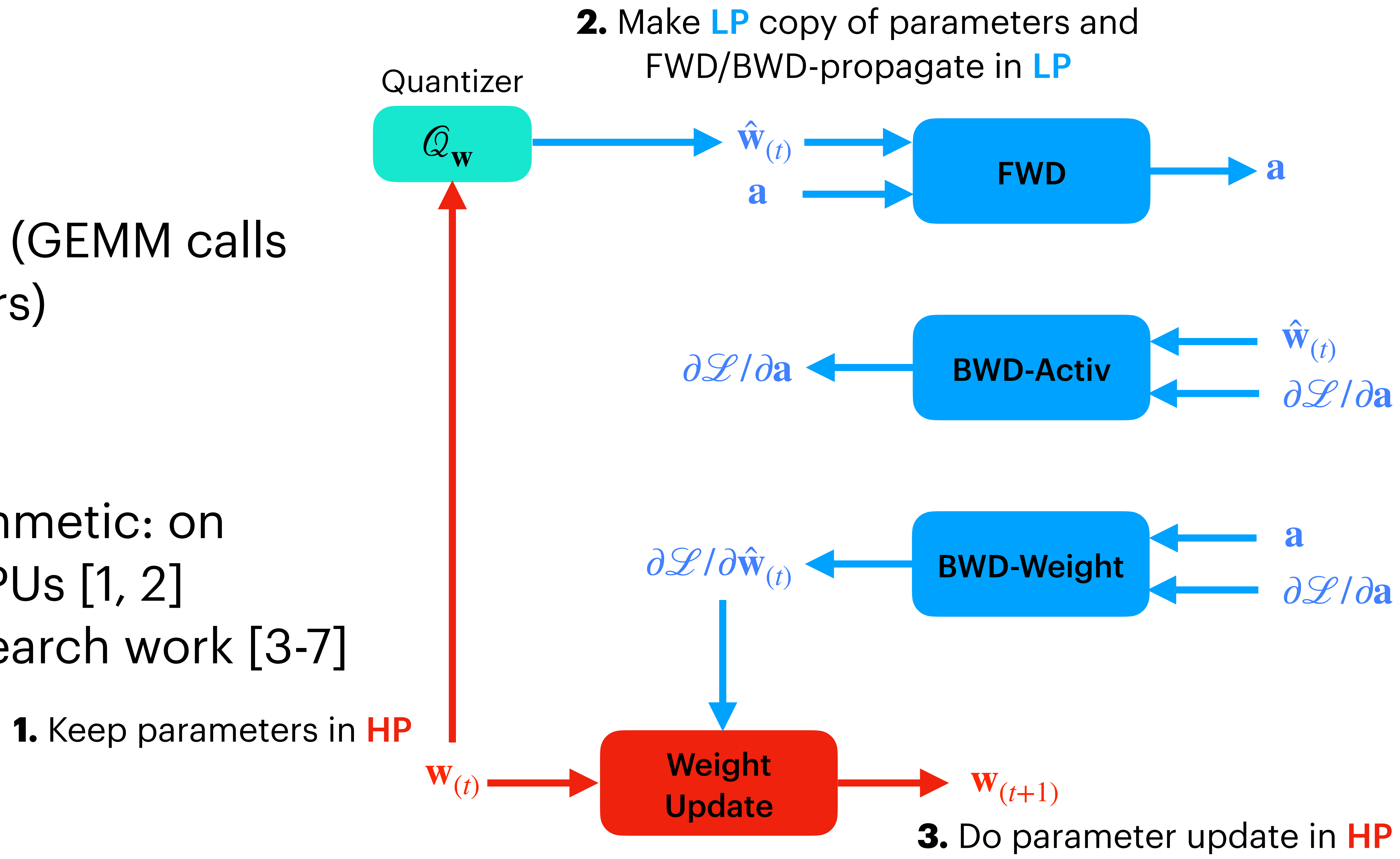
➡SOTA training acceleration methods are based on *mixed precision computing*

Idea: perform parameter updates in high precision (HP) + other ops in low precision (LP)

➡most compute happens in FWD/BWD-part (GEMM calls for fully connected and convolutional layers)

➡some notable examples:

- 32-bit (fp32) + 16-bit (fp16/bfloat16) arithmetic: on NVIDIA GPUs (NVIDIA AMP) & Google TPUs [1, 2]
- sub 16-bit & 8-bit training methods: research work [3-7]



[1] Mixed Precision Training, *Micikevicius et al.*, ICLR 2018
[2] A Study of bfloat16 for Deep Learning Training, *Kalamkar et al.*,
[3] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[4] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[5] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[6] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[7] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	GEMM Input x	BN Input	dw	da	Acc.	FP32	Proposed
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration.
Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	**GEMM Input x**	**BN Input**	**dw**	**da**	**Acc.**	**FP32**	**Proposed**
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration.
Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ use of a shared exponent bias/scaling factor at the tensor or block level or other similar tensor statistics

Why?

- shifts dynamic range at runtime, following the distribution of the data (with a small overhead)

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	**GEMM Input x**	**BN Input**	**dw**	**da**	**Acc.**	**FP32**	**Proposed**
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration. Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ use of a shared exponent bias/scaling factor at the tensor or block level or other similar tensor statistics

Why?

- shifts dynamic range at runtime, following the distribution of the data (with a small overhead)

➡ scale the loss function before back propagation + rescale gradients before parameter update

Why?

- shifts gradients in a representable range when using low precision (i.e., to avoid under/overflows)

$$\mathscr{L} \rightarrow \mathscr{L}_{\text{scaled}} = \boxed{2^s} \cdot \mathscr{L}$$

$$\partial \mathscr{L} / \partial \mathbf{w} = \boxed{2^{-s}} \cdot \partial \mathscr{L}_{\text{scaled}} / \partial \mathbf{w}$$

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	GEMM Input x	BN Input	dw	da	Acc.	FP32	Proposed
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration. Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

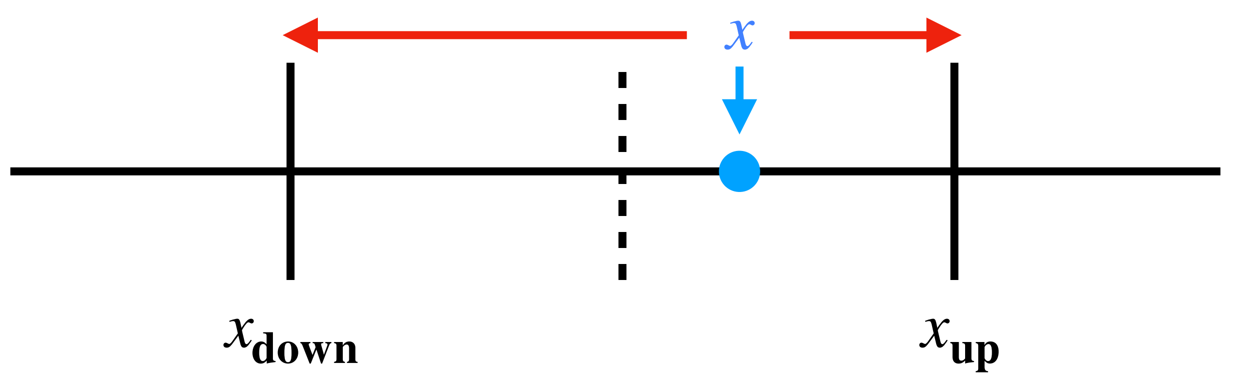
Some notable ideas:

➡ rounding used in the quantizer: stochastic [1] & hysteresis [6]

Why?

- stochastic rounding can recapture information that is discarded when bits are rounded off

$$p_{\text{down}} = \frac{x - x_{\text{down}}}{x_{\text{up}} - x_{\text{down}}} \qquad p_{\text{up}} = \frac{x_{\text{up}} - x}{x_{\text{up}} - x_{\text{down}}}$$



[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	**GEMM Input x**	**BN Input**	**dw**	**da**	**Acc.**	**FP32**	**Proposed**
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration. Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ rounding used in the quantizer: stochastic [1] & hysteresis [6]

Why?

- stochastic rounding can recapture information that is discarded when bits are rounded off
- hysteresis rounding seems to smooth fluctuations in param. updates & stabilizes training

$$\mathcal{Q}^{(t)}\left(w^{(t)}\right) = \begin{cases} \lfloor w^{(t)} \rfloor & \text{if } w^t > \mathcal{Q}^{(t-1)}\left(w^{(t-1)}\right) \\ \lceil w^{(t)} \rceil & \text{if } w^t \leqslant \mathcal{Q}^{(t-1)}\left(w^{(t-1)}\right) \end{cases}$$

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	**GEMM Input x**	**BN Input**	**dw**	**da**	**Acc.**	**FP32**	**Proposed**
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration.
Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ rounding used in the quantizer:
stochastic [1] & hysteresis [6]

Why?

- stochastic rounding can recapture information that is discarded when bits are rounded off
- hysteresis rounding seems to smooth fluctuations in param. updates & stabilizes training

➡ *smart* accumulator design (algorithmic/architectural) to optimize accuracy at low precision

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	GEMM Input x	BN Input	dw	da	Acc.	FP32	Proposed
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration. Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ rounding used in the quantizer: stochastic [1] & hysteresis [6]

Why?

- stochastic rounding can recapture information that is discarded when bits are rounded off
- hysteresis rounding seems to smooth fluctuations in param. updates & stabilizes training

➡ *smart* accumulator design (algorithmic/architectural) to optimize accuracy at low precision

Limitations:

➡ many approaches use coarse-grained simulation results

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	GEMM Input x	BN Input	dw	da	Acc.	FP32	Proposed
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration. Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ rounding used in the quantizer: stochastic [1] & hysteresis [6]

Why?

- stochastic rounding can recapture information that is discarded when bits are rounded off
- hysteresis rounding seems to smooth fluctuations in param. updates & stabilizes training

➡ *smart* accumulator design (algorithmic/architectural) to optimize accuracy at low precision

Limitations:

➡ many approaches use coarse-grained simulation results

➡ HW synthesis results are not that common (yet!)

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

An overview of recent results in MP training

Quantization Scheme	Formats ((Exponent, Mantissa) / Width)						Top-1 Accuracy	
	w	GEMM Input x	BN Input	dw	da	Acc.	FP32	Proposed
SWALP [1]	8	8	N/A	8	8	32	70.3	65.8
S2FP8 [3]	(5,2)/(8,23)	(5,2)	N/A	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [2]	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [4]	(2,5)	(2,5)	31	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [5]	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [6]	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Overview/Comparison of data formats used in recent research on mixed precision training acceleration. Results are ImageNet accuracy (%) using ResNet18 (adapted from [6]).

Some notable ideas:

➡ rounding used in the quantizer: stochastic [1] & hysteresis [6]

Why?

- stochastic rounding can recapture information that is discarded when bits are rounded off
- hysteresis rounding seems to smooth fluctuations in param. updates & stabilizes training

➡ *smart* accumulator design (algorithmic/architectural) to optimize accuracy at low precision

Limitations:

➡ many approaches use coarse-grained simulation results

➡ HW synthesis results are not that common (yet!)

➡ accumulator results are usually in high precision

[1] SWALP: Stochastic Weight Averaging in Low Precision, *Yang et al.*, ICML 2019
[2] Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, *Sun et al.*, NeurIPS 2019
[3] Shifted and Squeezed 8-bit Floating Point Format for Low-Precision Training of Deep Neural Networks, *Cambier et al.*,ICLR, 2020
[4] A Block Minifloat Representation for Training Deep Neural Networks, *Fox et al.*, ICLR 2020
[5] A Neural Network Training Processor with 8-Bit Shared Exponent Bias Floating Point and Multiple-Way Fused Multiply-Add Trees, *Park et al.*, IEEE 2021
[6] Towards Efficient Low-Precision Training: Data Format Optimization and Hysteresis Quantization, *Lee et al.*, ICLR 2022

Simulation support for MP training

Features	QPyTorch[1]	TensorQuant [2]	FASE [3]	MPTorch [4]	Archimedes-MPO [4, 5]
Fast	++	+	+	+	+
Accurate	—	+	+	+	+
Seamless	—	—	+	—	—
Dynamic Libraries	—	—	+	—	—
Independent	—	—	+	—	—
Platforms	CPU/GPU	CPU/GPU	CPU	CPU/GPU/FPGA	CPU/GPU/FPGA

[1] QPyTorch: A Low-Precision Arithmetic Simulation Framework, *Zhang et al.*, arXiv:1910.04540, 2019
[2] TensorQuant — A Simulation Toolbox for Deep Neural Network Quantization, *Loroch et al.*, arXiv:1710.05758, 2017
[3] FASE: A Fast, Accurate and Seamless Emulator for Custom Numerical Formats, *Osorio et al.*, ISPASS 2022
[4] MPTorch and MPArchimedes: Open Source Frameworks to Explore Custom Mixed-Precision Operations for DNN Training on Edge Devices, *Tatsumi et al.*, ROAD4NN 2021
[5] Mixing Low-Precision Formats in Multiply-Accumulate (MAC) Units for DNN Training, *Tatsumi et al.*, FPT 2022

Simulation support for MP training

Features	QPyTorch[1]	TensorQuant [2]	FASE [3]	**MPTorch [4]**	**Archimedes-MPO [4, 5]**
Fast	++	+	+	+	+
Accurate	—	+	+	+	+
Seamless	—	—	+	—	—
Dynamic Libraries	—	—	+	—	—
Independent	—	—	+	—	—
Platforms	CPU/GPU	CPU/GPU	CPU	CPU/GPU/*FPGA*	CPU/GPU/FPGA

MPTorch repository: https://github.com/mptorch/mptorch

[1] QPyTorch: A Low-Precision Arithmetic Simulation Framework, *Zhang et al.*, arXiv:1910.04540, 2019
[2] TensorQuant — A Simulation Toolbox for Deep Neural Network Quantization, *Loroch et al.*, arXiv:1710.05758, 2017
[3] FASE: A Fast, Accurate and Seamless Emulator for Custom Numerical Formats, *Osorio et al.*, ISPASS 2022
[4] MPTorch and MPArchimedes: Open Source Frameworks to Explore Custom Mixed-Precision Operations for DNN Training on Edge Devices, Tatsumi et al., ROAD4NN 2021
[5] Mixing Low-Precision Formats in Multiply-Accumulate (MAC) Units for DNN Training, Tatsumi et al., FPT 2022

Archimedes-MPO & MPTorch goals

➡vehicles for producing:

- mixed/low precision DNN training accelerator hardware prototypes
- explore novel algorithms for mixed precision DNN training

Work in progress

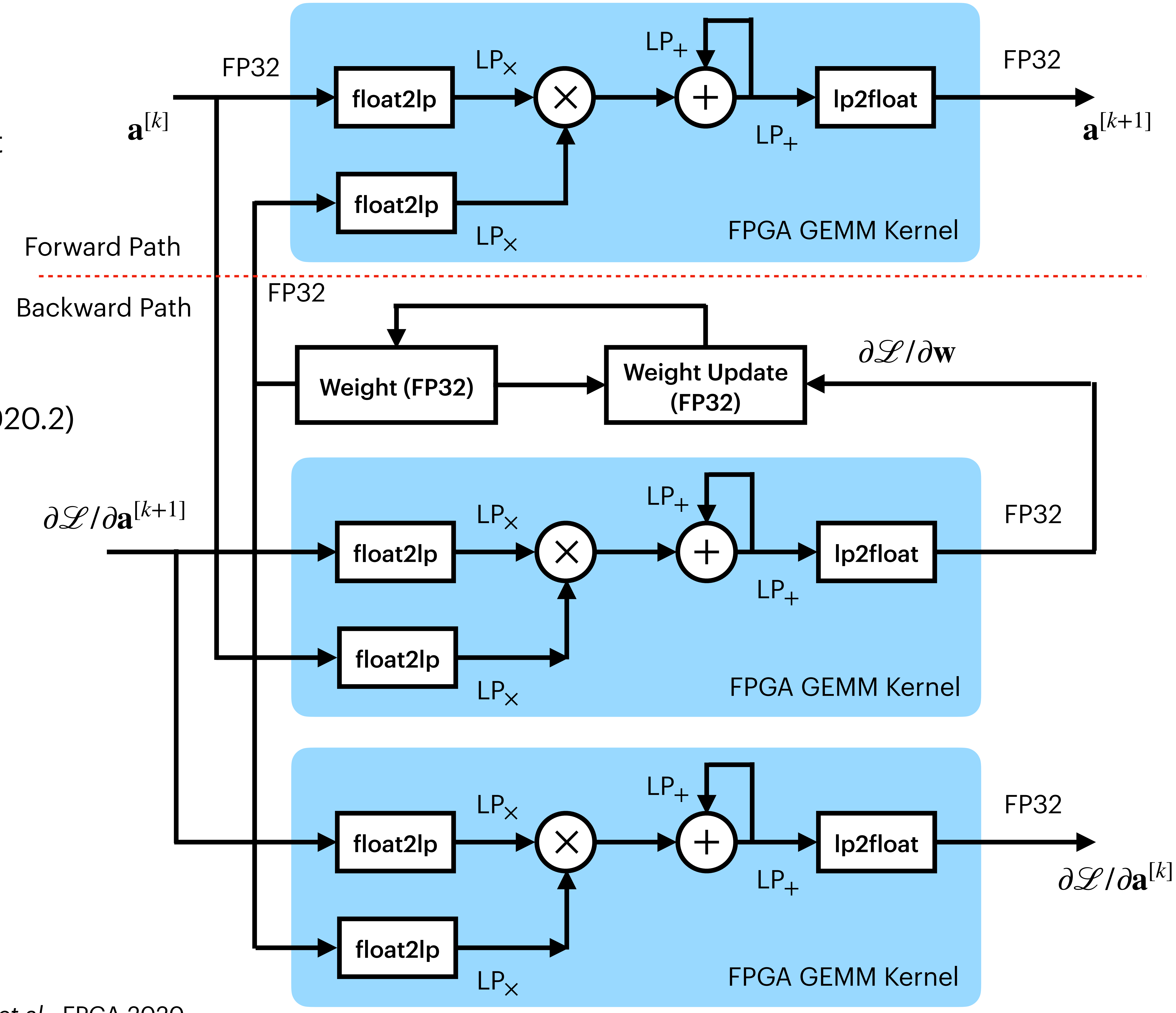
Starting topic: explore multiply-accumulate (MAC) unit design space

Archimedes-MPO Overview

➡ extends TinyDNN [1] C++ deep learning library:
- support for custom precision fixed-point and floating-point
- GPU & FPGA versions with custom GEMM kernels

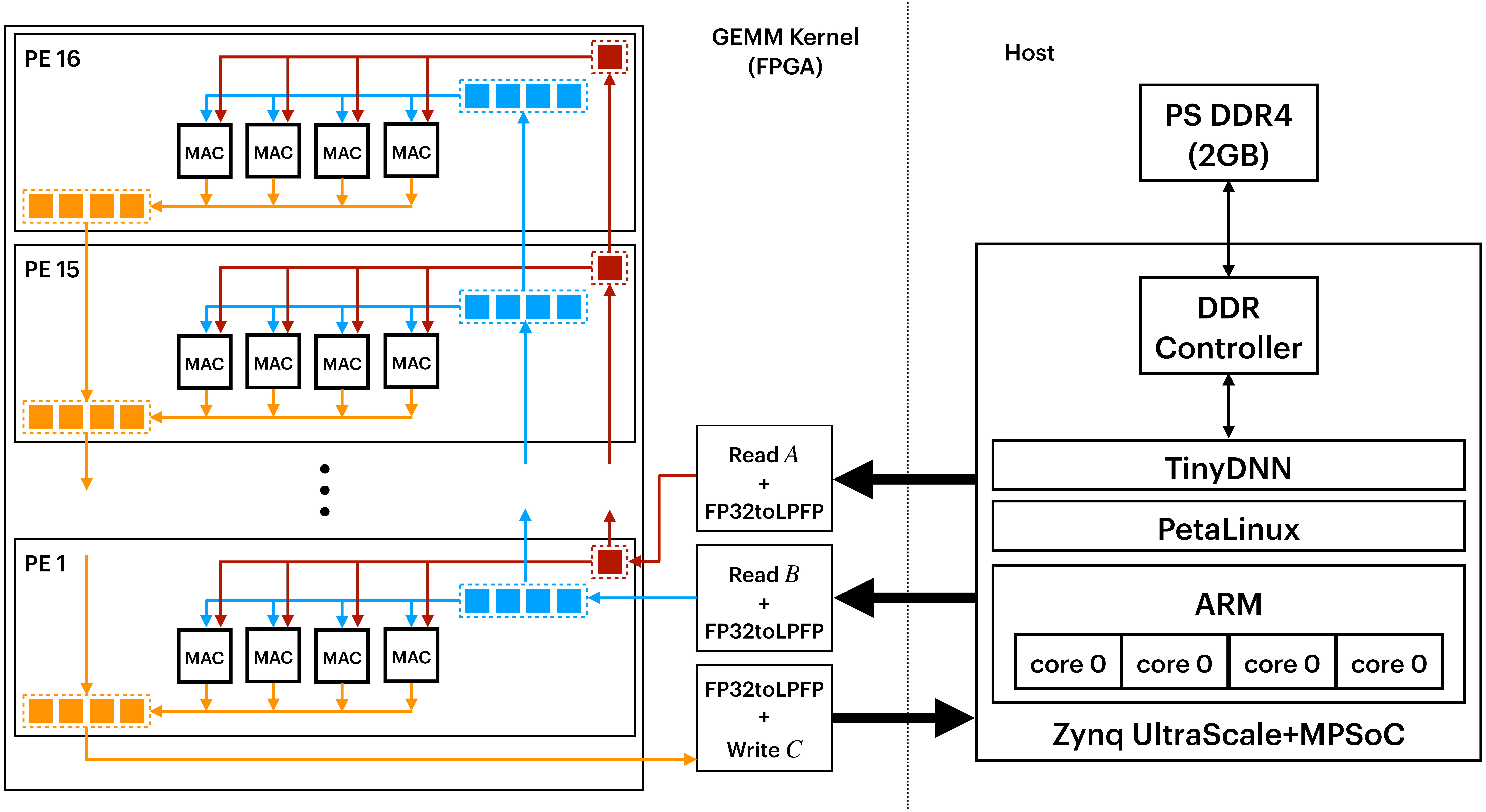
➡ GEMM kernel on FPGA:
- adds custom precision support to prior work [2]:
  - data type converter (FP32 ↔ LP)
  - custom multiplier and adder (MAC) in HLS (Vitis HLS 2020.2)
- parametrizable architecture:
  - currently using 16 × 4 systolic array (@ 280MHz)
- one HW kernel is synthesized
- Xilinx ZCU104 development board

➡ GEMM kernel on GPU:
- bit-accurate with the FPGA version
- more convenient to deploy & test



[1] https://github.com/tiny-dnn/tiny-dnn
[2] Flexible Communication Avoiding Matrix Multiplication on FPGA with HLS, *de Fine Licht et al.*, FPGA 2020

Archimedes-MPO FPGA Block Diagram



MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Multiplier

➡floating-point:

- limit input mantissa size to 3 bits → use LUTs for multiplying operand mantissas
- basic configuration (**`CFG–1`**):
  - support for NaNs/$\pm\infty$
  - round to nearest, subnormals

➡fixed-point:

- integer multiplier with output rounded to input data type
- uses DSP blocks because required fixed-point formats are wider

I/O precision	LUTs	DSPs
FP32 (no DSP)	987	0
FP32	374	2
FP16/bfloat16	195/180	1
E6M3 (CFG–1)	115	0
E5M3 (CFG–1)	86	0
E4M3 (CFG–1)	78	0
Q16.16	279	4
Q8.8	106	1
Q7.7	93	1
Q6.6	81	1

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Multiplier

➡floating-point:

- limit input mantissa size to 3 bits → use LUTs for multiplying operand mantissas
- basic configuration (**`CFG–1`**):
  - support for NaNs/±∞
  - round to nearest, subnormals

➡fixed-point:

- integer multiplier with output rounded to input data type
- uses DSP blocks because required fixed-point formats are wider

I/O precision	LUTs	DSPs
FP32 (no DSP)	987	0
FP32	374	2
FP16/bfloat16	195/180	1
E6M3 (CFG–1)	115	0
E5M3 (CFG–1)	86	0
E4M3 (CFG–1)	78	0
Q16.16	279	4
Q8.8	106	1
Q7.7	93	1
Q6.6	81	1

➡better resource usage for small floating-point vs fixed-point in training accuracy results (later)

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Multiplier

➡decrease resource use by ***gradually*** removing ancillary support:

`CFG–2:` subnormal output removal

- information loss + LUT reduction

`CFG–3:` output rounding removal

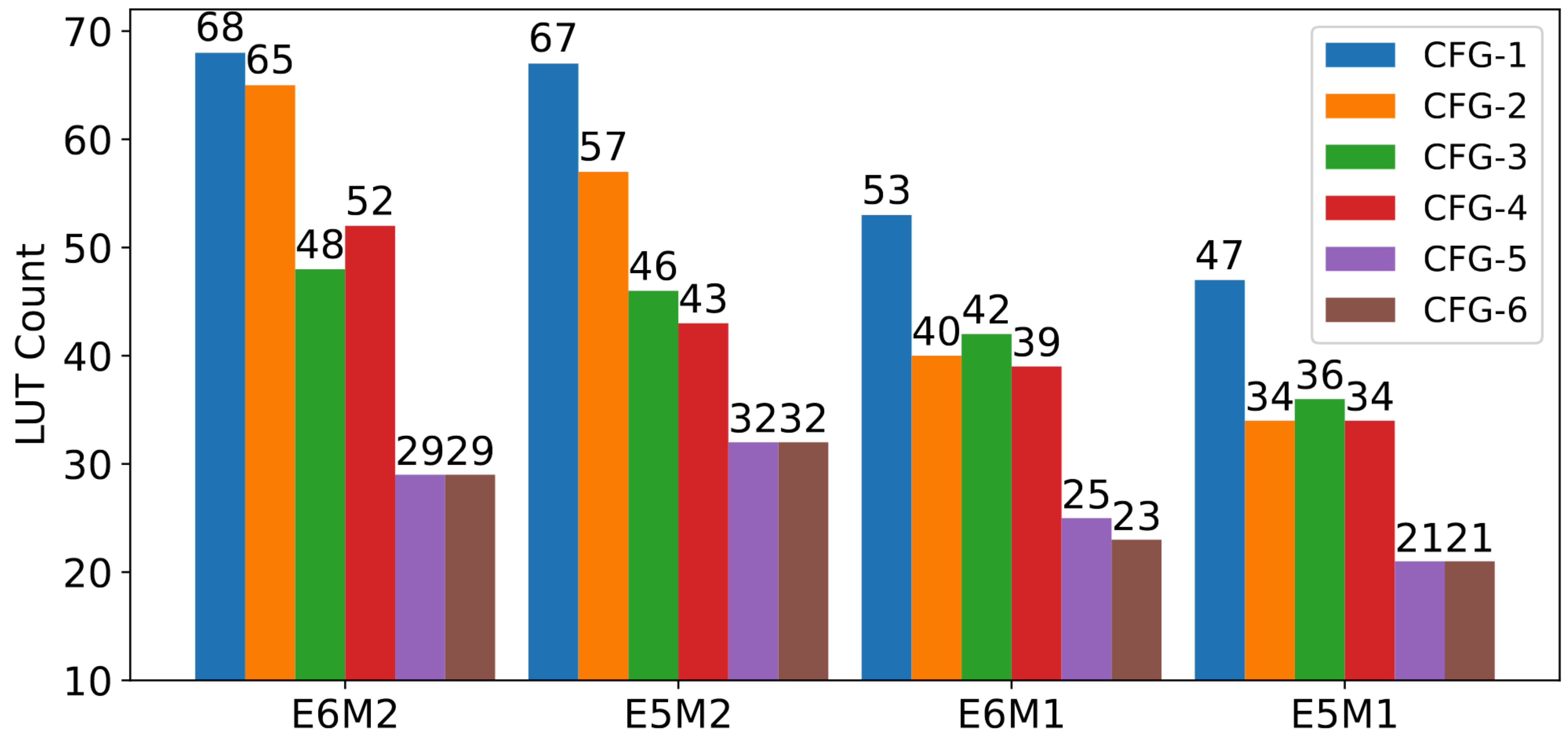
- restores information + output length increases

`CFG–4:` NaN encoding removal

- NaN values become normal values
- remapping $\pm\infty$ to all 1 mantissa

`CFG–5 & CFG–6:` alternative subnormal inputs

- CFG–5 treats subnormals as normal values
- CFG–6 truncates all subnormals to zero



Area of floating-point multiplier variants

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Multiplier

➡decrease resource use by ***gradually*** removing ancillary support:

`CFG-2:` subnormal output removal

- information loss + LUT reduction

`CFG-3:` output rounding removal

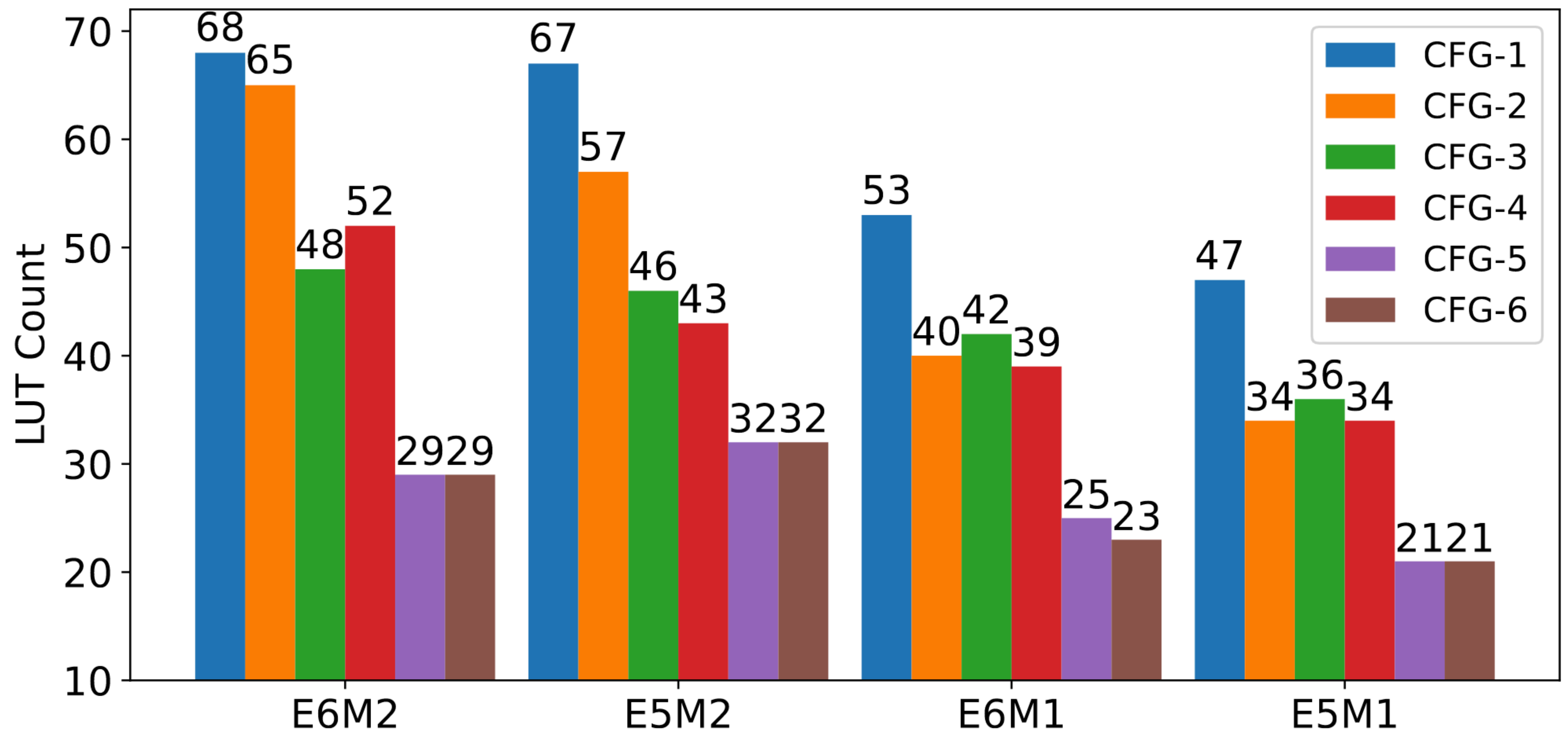
- restores information + output length increases

`CFG-4:` NaN encoding removal

- NaN values become normal values
- remapping $\pm\infty$ to all 1 mantissa

`CFG-5 & CFG-6:` alternative subnormal inputs

- CFG-5 treats subnormals as normal values
- CFG-6 truncates all subnormals to zero



Area of floating-point multiplier variants

➡over 50% area reduction going from CFG-1 to CFG-5/CFG-6

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Multiplier

➡decrease resource use by ***gradually*** removing ancillary support:

`CFG-2:` subnormal output removal

- information loss + LUT reduction

`CFG-3:` output rounding removal

- restores information + output length increases

`CFG-4:` NaN encoding removal

- NaN values become normal values
- remapping $\pm\infty$ to all 1 mantissa

`CFG-5 & CFG-6:` alternative subnormal inputs

- CFG–5 treats subnormals as normal values
- CFG–6 truncates all subnormals to zero

	NaN/∞		**Subnormal**		
mantissa	conventional `CFG1 to CFG3`	custom `CFG4 to CFG6`	conventional `CFG1 to CFG4`	custom `CFG5`	truncate `CFG6`
b00	∞	65,536	0	0	0
b01	NaN	81,920	1.53E-5	3.81E-5	0
b10	NaN	98,304	3.05E-5	4.58E-5	0
b11	NaN	∞	4.58E-5	5.34E-5	0

Alternative encoding schemes for E5M2

➡over 50% area reduction going from CFG–1 to CFG–5/CFG–6

➡prefer CFG–5 due to increased representation range

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Accumulator

➡look at low-precision floating-point and fixed-point designs:

- fixed-point: saturation logic
- floating-point: subnormals, swapping, operand shifting, extra bits

FP-mult input (CFG5)	FP-mult output	Accumulator LUTs	Accumulator DSPs
FP32	FP32	189	2
E6M3	E7M7	255	0
E6M2	E7M5	185	0
E6M1	E7M3	187	0
E5M3	E6M7	242	0
E5M2	E6M5	187	0
E5M1	E6M3	165	0
-	Q16.16	89	0
-	Q8.13	55	0
-	Q8.8	43	0
-	Q7.7	35	0
-	Q6.6	32	0

Area of accumulator

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Accumulator

➡look at low-precision floating-point and fixed-point designs:

- fixed-point: saturation logic
- floating-point: subnormals, swapping, operand shifting, extra bits

➡fixed-point designs more efficient
(NO I/O alignment shifters or rounding)

FP-mult input (CFG5)	FP-mult output	Accumulator LUTs	Accumulator DSPs
FP32	FP32	189	2
E6M3	E7M7	255	0
E6M2	E7M5	185	0
E6M1	E7M3	187	0
E5M3	E6M7	242	0
E5M2	E6M5	187	0
E5M1	E6M3	165	0
-	Q16.16	89	0
-	Q8.13	55	0
-	Q8.8	43	0
-	Q7.7	35	0
-	Q6.6	32	0

Area of accumulator

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Accumulator

➡look at low-precision floating-point and fixed-point designs:

- fixed-point: saturation logic
- floating-point: subnormals, swapping, operand shifting, extra bits

➡fixed-point designs more efficient
(NO I/O alignment shifters or rounding)

Full MAC unit

➡floating-point multiplier + fixed-point acc. ?

FP-mult input (`CFG5`)	FP-mult output	Accumulator LUTs	Accumulator DSPs
FP32	FP32	189	2
E6M3	E7M7	255	0
E6M2	E7M5	185	0
E6M1	E7M3	187	0
E5M3	E6M7	242	0
E5M2	E6M5	187	0
E5M1	E6M3	165	0
-	Q16.16	89	0
-	Q8.13	55	0
-	Q8.8	43	0
-	Q7.7	35	0
-	Q6.6	32	0

Area of accumulator

MAC Design Space Exploration

➡start by looking at the multiplier and accumulator separately

Accumulator

➡look at low-precision floating-point and fixed-point designs:

- fixed-point: saturation logic
- floating-point: subnormals, swapping, operand shifting, extra bits

➡fixed-point designs more efficient
(NO I/O alignment shifters or rounding)

Full MAC unit

➡floating-point multiplier + fixed-point acc. ?

- requires float-to-fixed converters (data shifters)
- type conversion cannot be ignored

FP-mult input (`CFG5`)	FP-mult output	Accumulator LUTs	Accumulator DSPs	Converter (to Q8.13) LUTs
FP32	FP32	189	2	-
E6M3	E7M7	255	0	116
E6M2	E7M5	185	0	103
E6M1	E7M3	187	0	72
E5M3	E6M7	242	0	97
E5M2	E6M5	187	0	81
E5M1	E6M3	165	0	67
-	Q16.16	89	0	-
-	Q8.13	55	0	-
-	Q8.8	43	0	-
-	Q7.7	35	0	-
-	Q6.6	32	0	-

Area of accumulator and data converter

Training Results

Experimental setting

➡image classification tasks using:

- ResNet-20 [1] & VGG16 [2] CNN architectures with CIFAR-10 dataset
- ResNet-50 [1] CNN on subset of the ImageNet dataset (ImageWoof)

➡optimizer (SGD + momentum) and hyperparam. & preprocessing based on the original papers

➡use adaptive loss scaling [3]

[1] Deep Residual Learning for Image Recognition, *He et al.,* CVPR 2016
[2] Very Deep Convolutional Neural Networks for Large-Scale Image Recognition, *Simonyan et al.,* ICLR 2015
[3] Mixed Precision Training, *Micikevicius et al.,* ICLR 2018

Training Results

Experimental setting

➡image classification tasks using:

- ResNet-20 [1] & VGG16 [2] CNN architectures with CIFAR-10 dataset
- ResNet-50 [1] CNN on subset of the ImageNet dataset (ImageWoof)

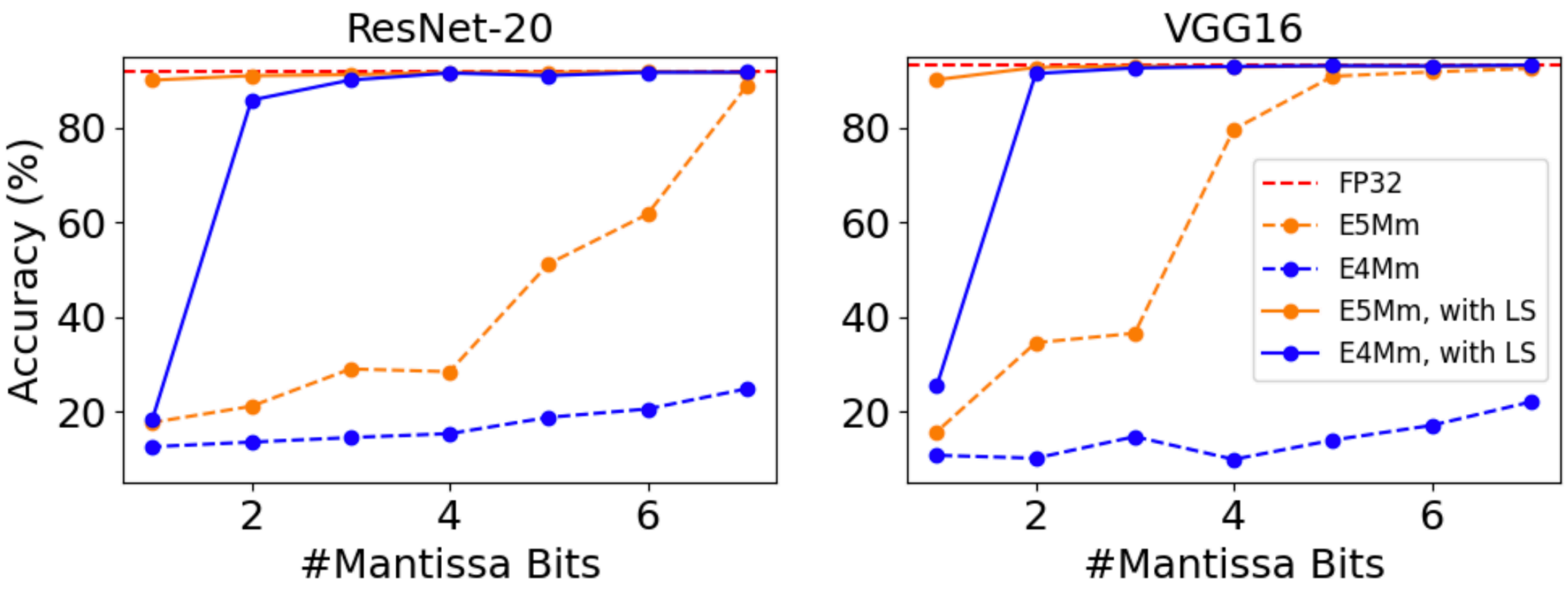
➡optimizer (SGD + momentum) and hyperparam. & preprocessing based on the original papers

➡use **adaptive loss scaling** [3]

[1] Deep Residual Learning for Image Recognition, *He et al.,* CVPR 2016
[2] Very Deep Convolutional Neural Networks for Large-Scale Image Recognition, *Simonyan et al.,* ICLR 2015
[3] Mixed Precision Training, *Micikevicius et al.,* ICLR 2018

Training Results

Impact of loss scaling



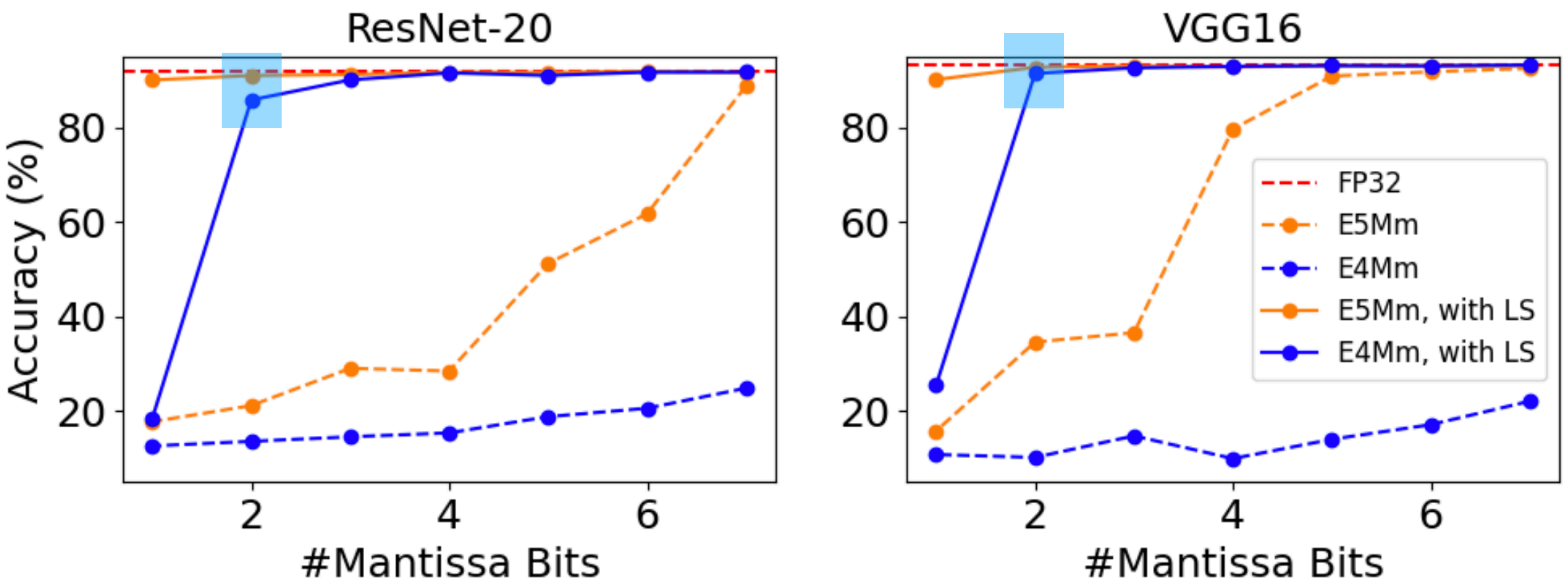
Loss scaling impact on test accuracy when using E4 and E5 multipliers

➡loss scaling important to keep gradients in representable range when using small formats

➡similar trends when varying the format/precision in the accumulators

Training Results

Impact of loss scaling



Loss scaling impact on test accuracy when using E4 and E5 multipliers

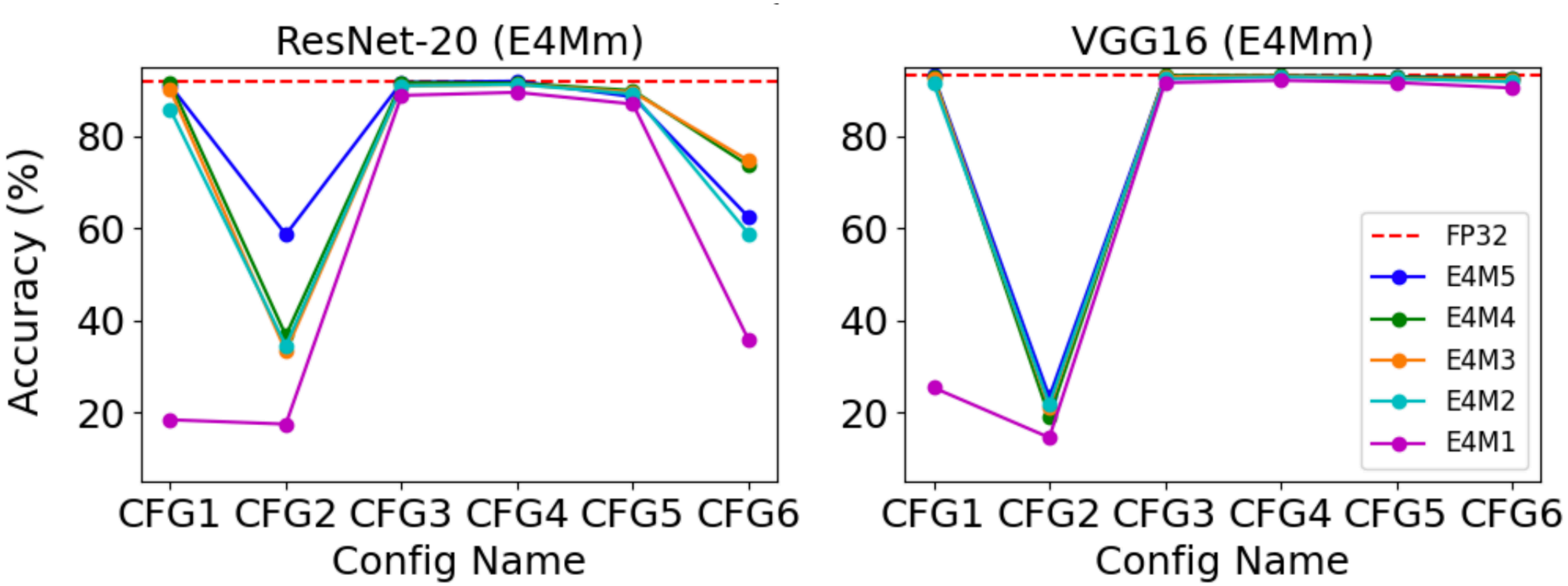
➡loss scaling important to keep gradients in representable range when using small formats

➡similar trends when varying the format/precision in the accumulators

➡E4M2 looks like a good place to start for these examples

Training Results

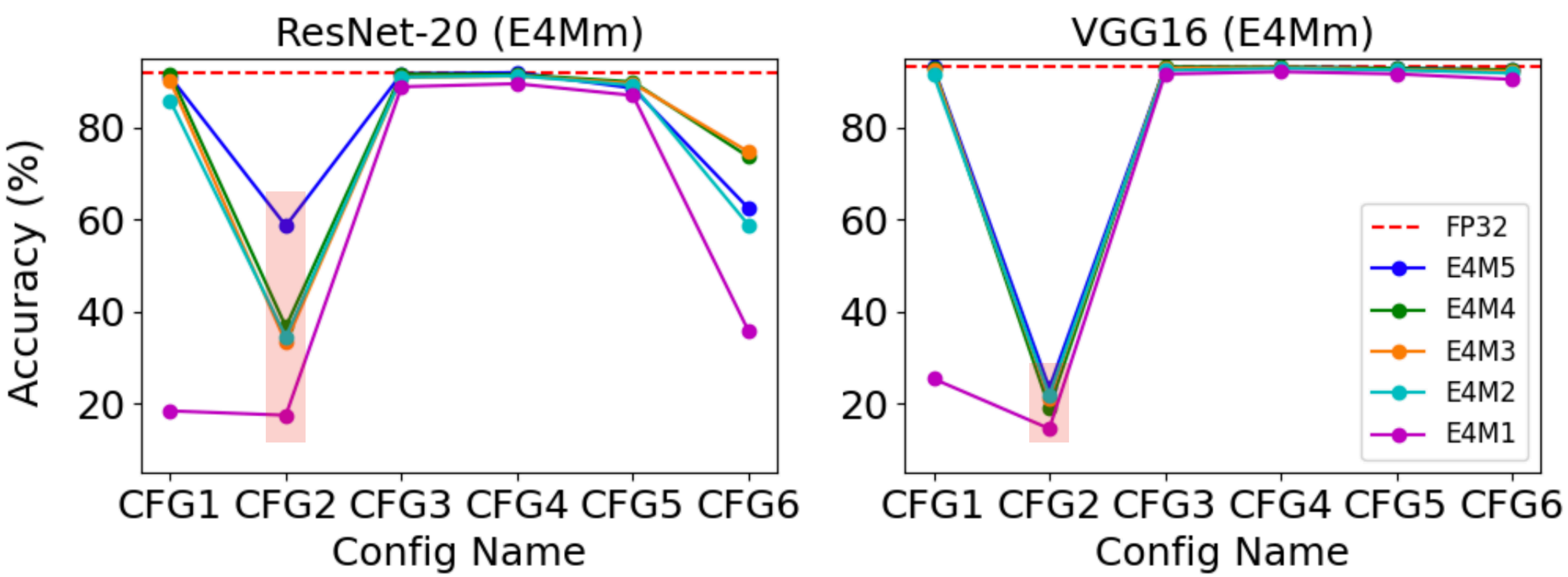
Multiplier variants



Multiplier variant impact on test accuracy

Training Results

Multiplier variants

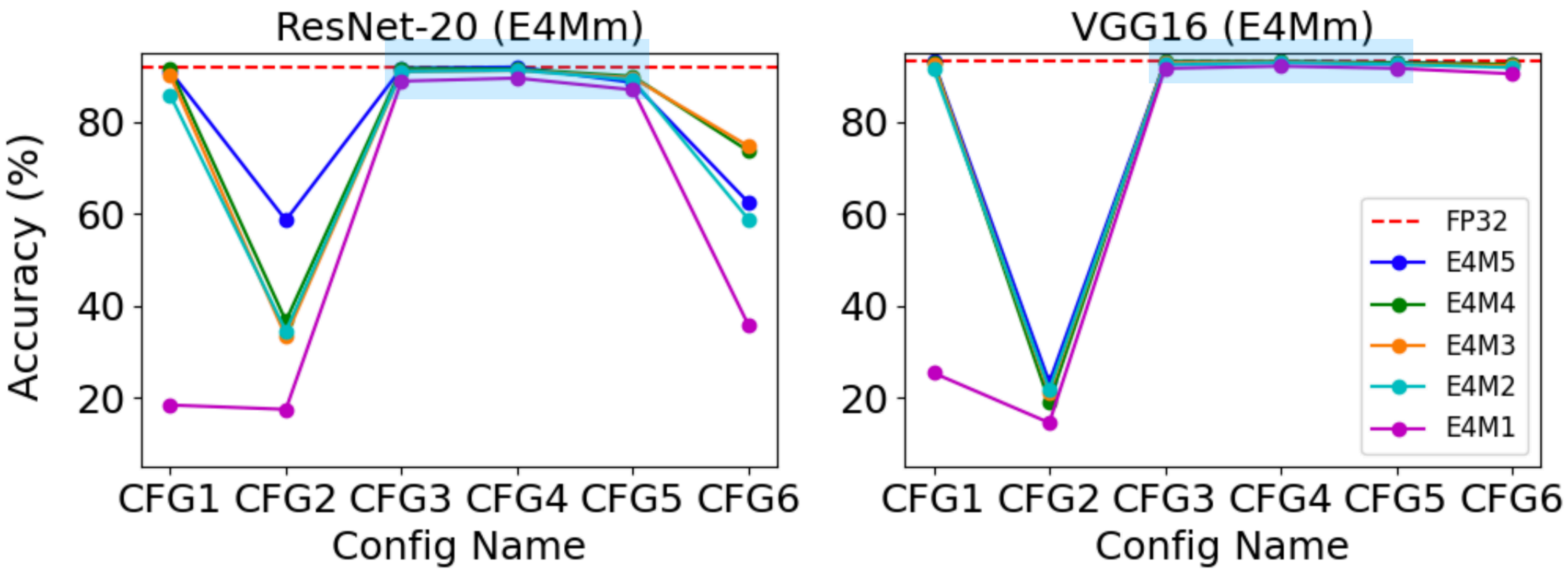


Multiplier variant impact on test accuracy

➡removing subnormal output support (CFG–2) hurts accuracy significantly

Training Results

Multiplier variants



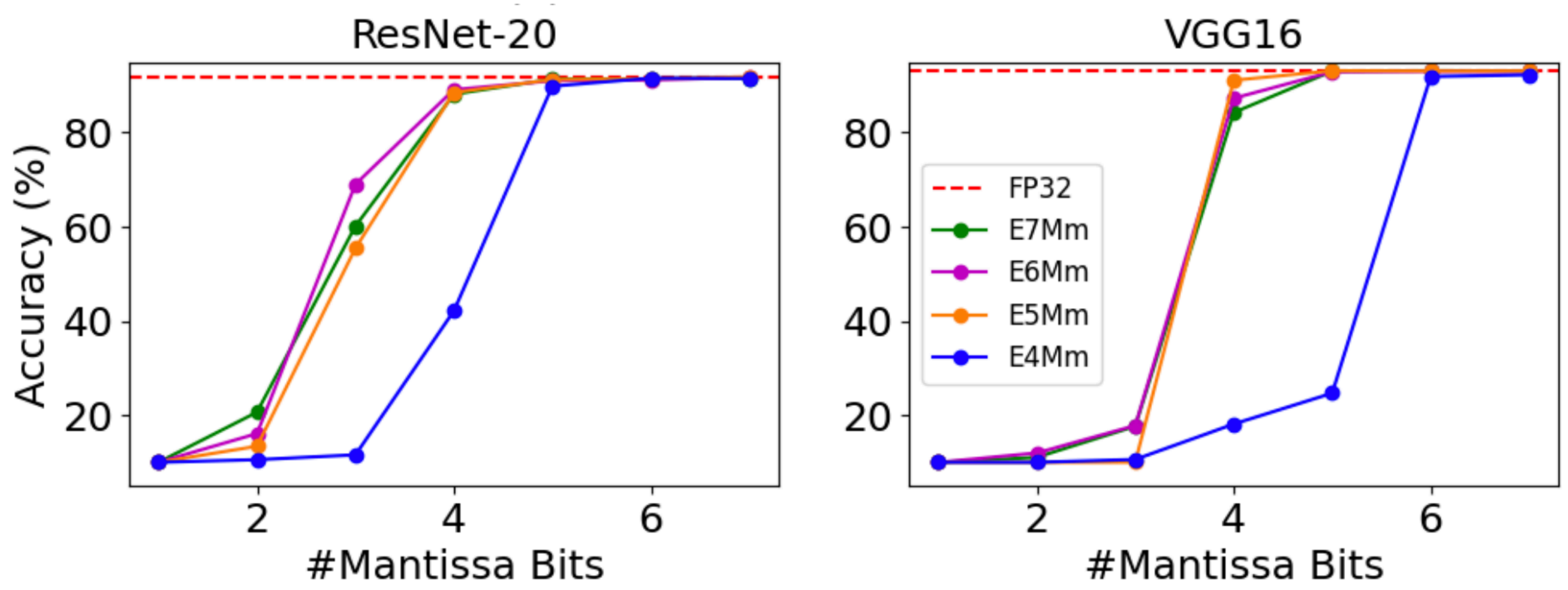
Multiplier variant impact on test accuracy

➡removing subnormal output support (CFG–2) hurts accuracy significantly

➡output rounding removal (CFG–3), NaN encoding removal (CFG–4), alternative subnormal inputs (CFG–5) restores accuracy

Training Results

Accumulator variants: floating-point

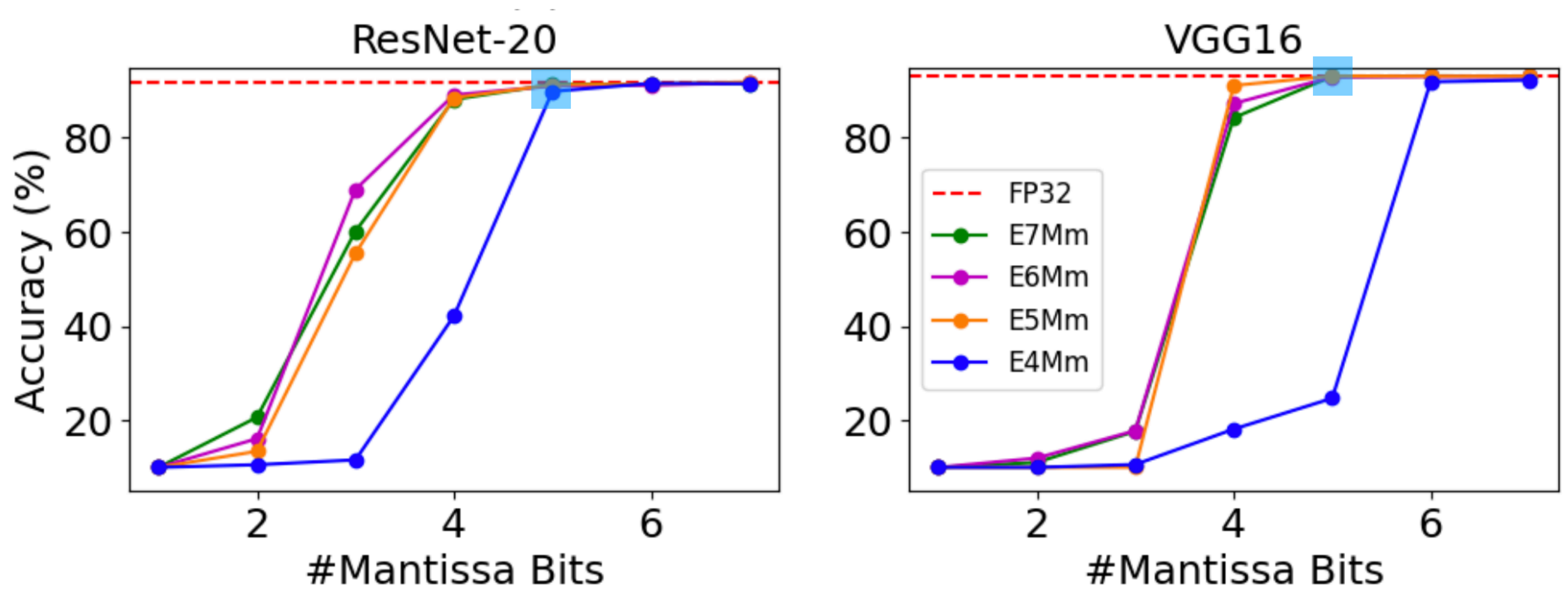


Floating-Point Accumulator impact on test accuracy

➡accuracy more sensitive to exponent width than mantissa width (even with loss scaling)

Training Results

Accumulator variants: floating-point

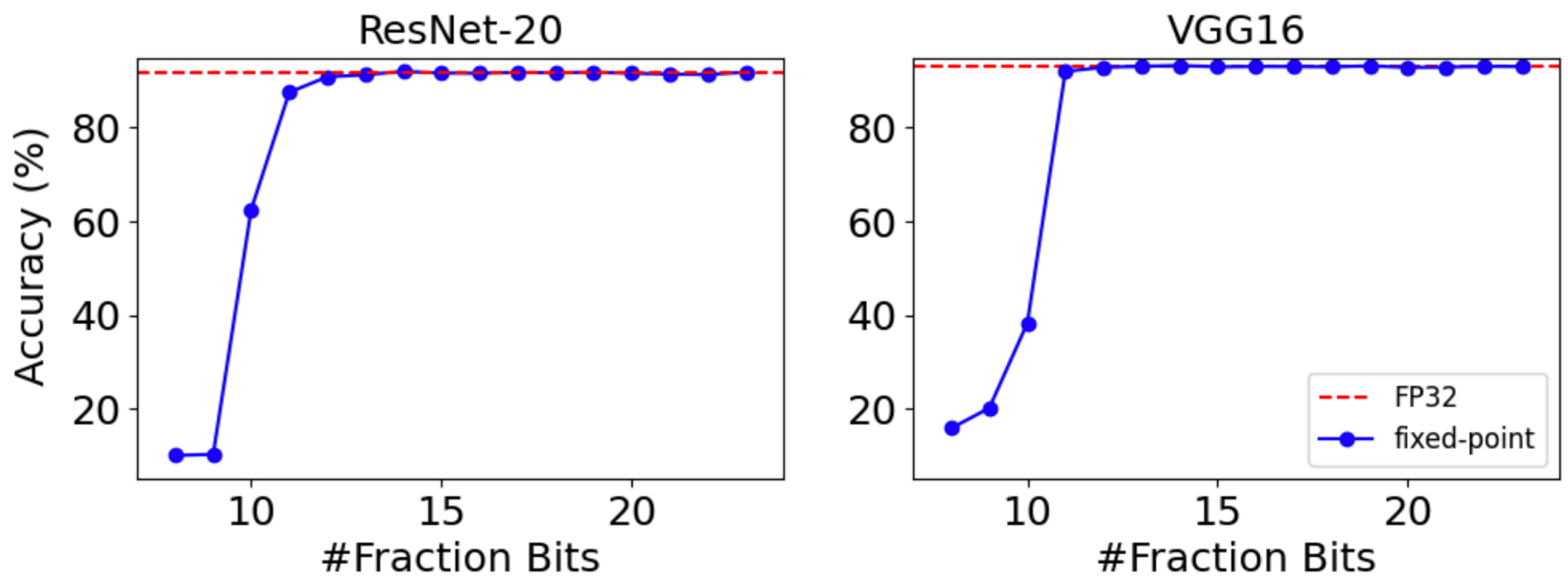


Floating-Point Accumulator impact on test accuracy

➡accuracy more sensitive to exponent width than mantissa width (even with loss scaling)
➡E5M5 seems like a good choice
➡investigating accumulation strategies might help

Training Results

Accumulator variants: fixed-point

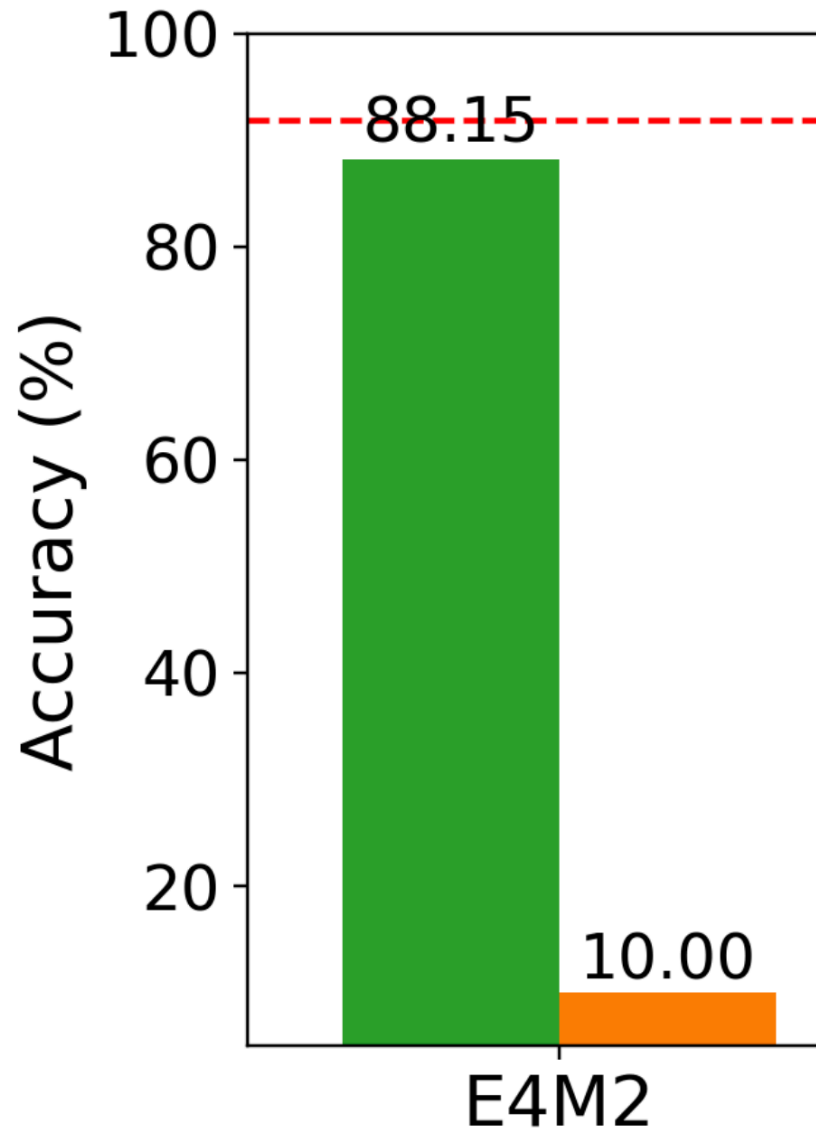


Fixed-Point Accumulator (Q8.f) impact on test accuracy

➡larger format needed: Q8.12

Training Results

Full MAC configuration

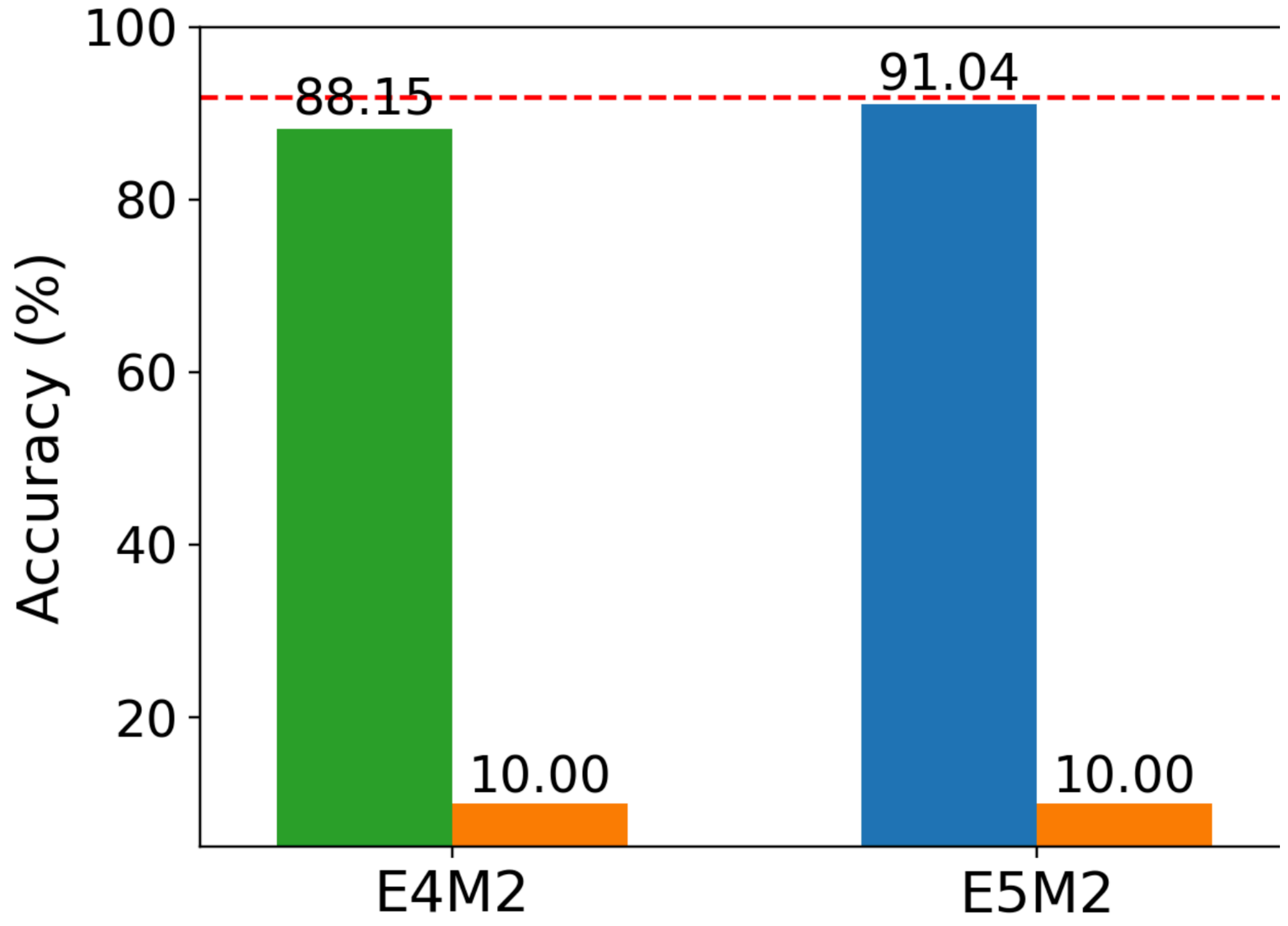


MAC configurations impact on test accuracy (ResNet20 + CIFAR-10)

➡start with E4M2 (CFG–5) multiplier + E5M5/Q8.12 (green/orange) accumulator

Training Results

Full MAC configuration



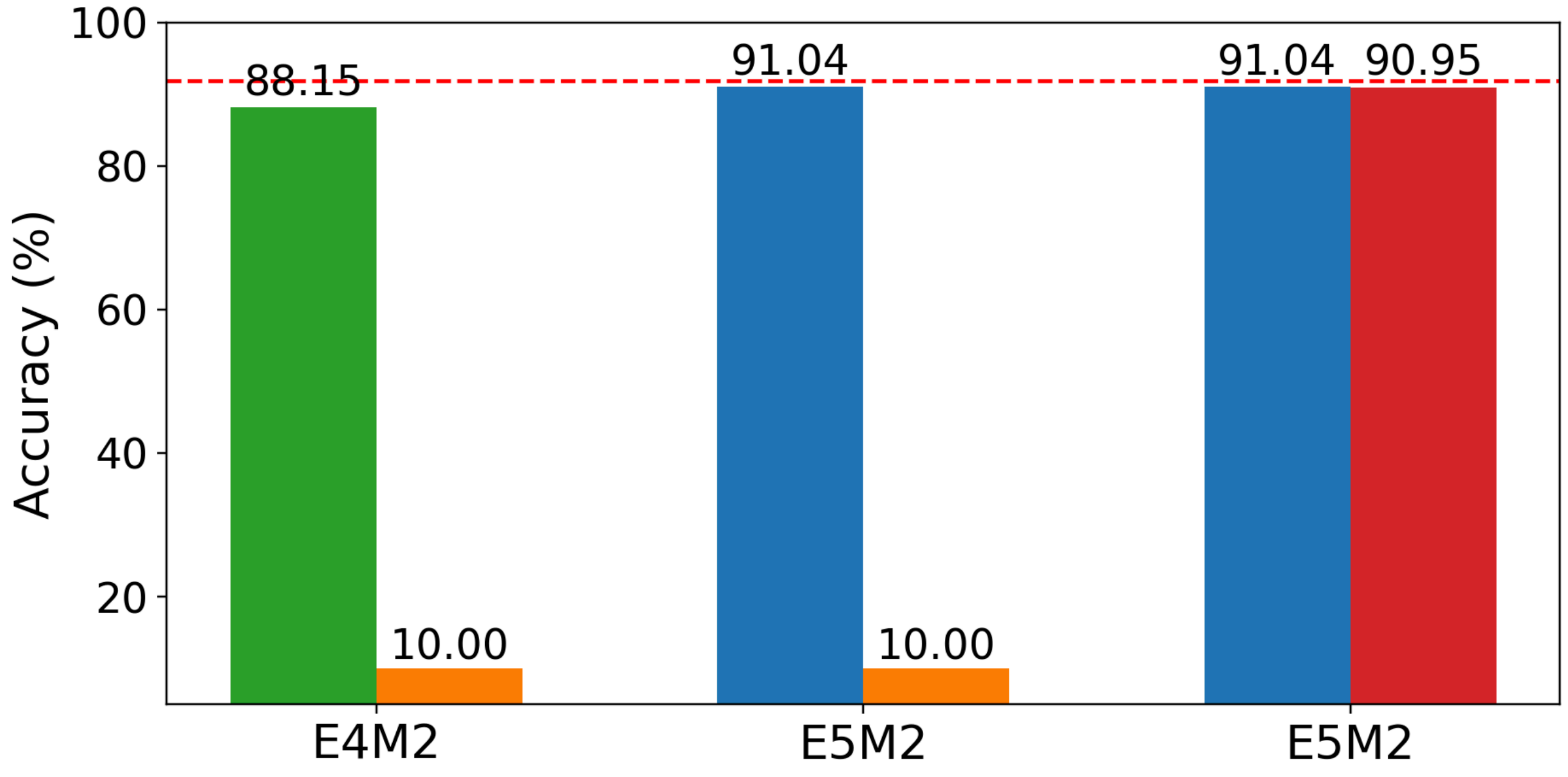
MAC configurations impact on test accuracy (ResNet20 + CIFAR-10)

➡start with E4M2 (`CFG–5`) multiplier + E5M5/Q8.12 (green/orange) accumulator

➡increasing floating-point multiplier input format to E5M2 (`CFG–5`) (blue) restores accuracy in all-FP MAC, but not for fixed-point accumulator

Training Results

Full MAC configuration



MAC configurations impact on test accuracy (ResNet20 + CIFAR-10)

➡start with E4M2 (CFG–5) multiplier + E5M5/Q8.12 (green/orange) accumulator

➡increasing floating-point multiplier input format to E5M2 (CFG–5) (blue) restores accuracy in all-FP MAC, but not for fixed-point accumulator

➡going to Q8.13 (red) accumulator restores mixed float/fixed MAC accuracy

Training Results

Full MAC configuration: system-level area and test accuracy

configurations	LUTs	DSPs	ResNet-20/ CIFAR-10 Acc. (%)
FP32	58,420	320	91.85
E5M2(`CFG5`) + E6M5	42,640	0	91.04
E5M2(`CFG5`) + Q8.13	43,471	0	90.95

➡25% LUT count reduction + no DSPs compared to a FP32 design

➡higher system-level LUT count for fixed point configuration:

- downstream interconnect + buffering logic for wider accum. output

➡further investigation needed on accumulation techniques at MAC and system level (e.g. [1])

[1] An FPGA-Specific Approach to Floating-Point Accumulation and Sum-Of-Products, *de Dinechin et al.*, FPT 2008

Training Results

Full MAC configuration: system-level area and test accuracy

configurations	LUTs	DSPs	ResNet-20/ CIFAR-10 Acc. (%)	ResNet-50/ Imagewoof Acc. (%)
FP32	58,420	320	91.85	57.57
E5M2(`CFG5`) + E6M5	42,640	0	91.04	59.79
E5M2(`CFG5`) + Q8.13	43,471	0	90.95	10.92

➡25% LUT count reduction + no DSPs compared to a FP32 design

➡higher system-level LUT count for fixed point configuration:

- downstream interconnect + buffering logic for wider accum. output

➡further investigation needed on accumulation techniques at MAC and system level (e.g. [1])

➡fixed-point accumulator more sensitive to DNN model size

[1] An FPGA-Specific Approach to Floating-Point Accumulation and Sum-Of-Products, *de Dinechin et al.*, FPT 2008

Training Results

Full MAC configuration: system-level area and test accuracy

configurations	LUTs	DSPs	ResNet-20/ CIFAR-10 Acc. (%)	ResNet-50/ Imagewoof Acc. (%)
FP32	58,420	320	91.85	57.57
E5M2(CFG5) + E6M5	42,640	0	91.04	59.79
E5M2(CFG5) + Q8.13	43,471	0	90.95	10.92
E5M2(CFG5) + Q11.13	44,876	0	n/a.	59.38

➡25% LUT count reduction + no DSPs compared to a FP32 design

➡higher system-level LUT count for fixed point configuration:

- downstream interconnect + buffering logic for wider accum. output

➡further investigation needed on accumulation techniques at MAC and system level (e.g. [1])

➡fixed-point accumulator more sensitive to DNN model size

[1] An FPGA-Specific Approach to Floating-Point Accumulation and Sum-Of-Products, *de Dinechin et al.*, FPT 2008

Summary

➡Archimedes-MPO & mptorch: study **resource-accuracy tradeoffs** with custom arithmetic during DNN training

➡narrow floating-point formats seem safe in GEMM multipliers

➡save multiplier area by modifying exceptional value support [1, 2]

➡fixed-point accumulators are interesting (e.g. small area), but can require significant extra logic

[1] FP8 Formats for Deep Learning, *Micikevicius et al.*, arXiv:2209.02915, 2022
[2] 8-bit Numerical Formats for Deep Neural Networks, *Noune et al.*, arXiv:2206.02915, 2022

Summary

➡Archimedes-MPO & mptorch: study **resource-accuracy tradeoffs** with custom arithmetic during DNN training

➡narrow floating-point formats seem safe in GEMM multipliers

➡save multiplier area by modifying exceptional value support [1, 2]

➡fixed-point accumulators are interesting (e.g. small area), but can require significant extra logic

Limitations & ongoing/future work

- small number of models and datasets
- explore/compare with other data formats besides floating-point & fixed-point
- accumulation architecture exploration at the MAC and system level
- arithmetic aspects of different training algorithms
- error analysis-guided choice of number formats during training
- assess resource-accuracy impact of other training operations:
  - parameter updates
  - other layer types (e.g. normalization)
  - activation function evaluation

[1] FP8 Formats for Deep Learning, *Micikevicius et al.*, arXiv:2209.02915, 2022
[2] 8-bit Numerical Formats for Deep Neural Networks, *Noune et al.*, arXiv:2206.02915, 2022

Summary

➡Archimedes-MPO & mptorch: study **resource-accuracy tradeoffs** with custom arithmetic during DNN training

➡narrow floating-point formats seem safe in GEMM multipliers

➡save multiplier area by modifying exceptional value support [1, 2]

➡fixed-point accumulators are interesting (e.g. small area), but can require significant extra logic

Limitations & ongoing/future work

- small number of models and datasets
- explore/compare with other data formats besides floating-point & fixed-point
- accumulation architecture exploration at the MAC and system level
- arithmetic aspects of different training algorithms
- error analysis-guided choice of number formats during training
- assess resource-accuracy impact of other training operations:
  - parameter updates
  - other layer types (e.g. normalization)
  - activation function evaluation

Thank You! Questions?

[1] FP8 Formats for Deep Learning, *Micikevicius et al.*, arXiv:2209.02915, 2022
[2] 8-bit Numerical Formats for Deep Neural Networks, *Noune et al.*, arXiv:2206.02915, 2022