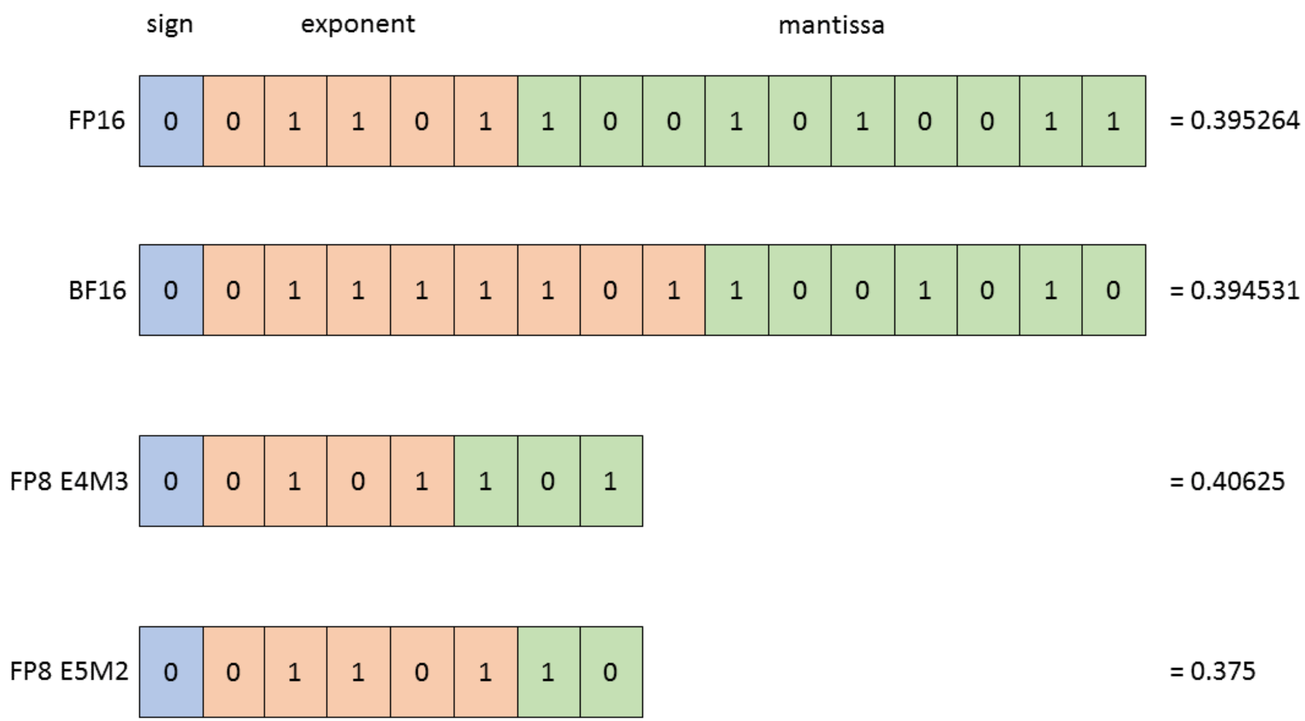
Low-precision training

EECE695D: Efficient ML Systems

Spring 2025

Low-precision training

- **Idea.** Exploit low-precision for training
  - Faster computation → More throughput, less energy
  - Reduced memory bandwidth → Bigger batch



Key challenge

- The archenemy is the **limited dynamic range**

- Limited precision leads to **over-/underflows**
  - FP8 (E4M3) value covers maximum 448 and minimum 2^{-6}
  - Can amplify the gradient noise
  - Imprecise updates accumulate over time

- Further affects **gradient, optimizer states, BN/LN statistics** (quantize what?)

This week

- We take a very brief look at some notable examples:
  - Early IBM work (2015)
  - Binary networks (2015-2016)
    - BNN, XNOR, DoReFa, eventually to BitNets
  - FP16/FP8 training (2018-)

Gupta et al., (2015)

Background

- FP32 is a modern standard; old works used various precisions for DL:
  - Iwata et al. (1989) uses FP24
  - Hammerstrom (1990) uses 8-16bits in fixed point
  - (...)
  - Chen et al. (2014) observes that at least 32bit fixed point is needed for their supercomputer

- **This work.** Train using **16bit fixed-point** ≈ FP32 (MNIST, CIFAR-10)
  - + Hardware prototyping

Iwata et al., "An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors," IJCNN 1989
Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," IJCNN 1990
Chen et al., "DaDianNao: A Machine-Learning Supercomputer," MICRO 2014

Method

- To compute $\mathbf{a}^\top \mathbf{b}$ for two k-bit fixed-point vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$:
  - **Step 1.** Compute the MAC

$$\mathbf{z} = \sum_{i=1}^{d} \mathbf{a}_i \mathbf{b}_i$$

    - $\mathbf{z}$ requires bitwidth at most $2k + 1 + \log_2 d$
  - **Step 2.** Convert the sum to k-bit
    - Round-to-nearest
    - Stochastic rounding
- Gradient in DL is matmul, thus no special consideration needed

Suyog et al., "Deep Learning with Limited Numerical Precision," ICML 2015

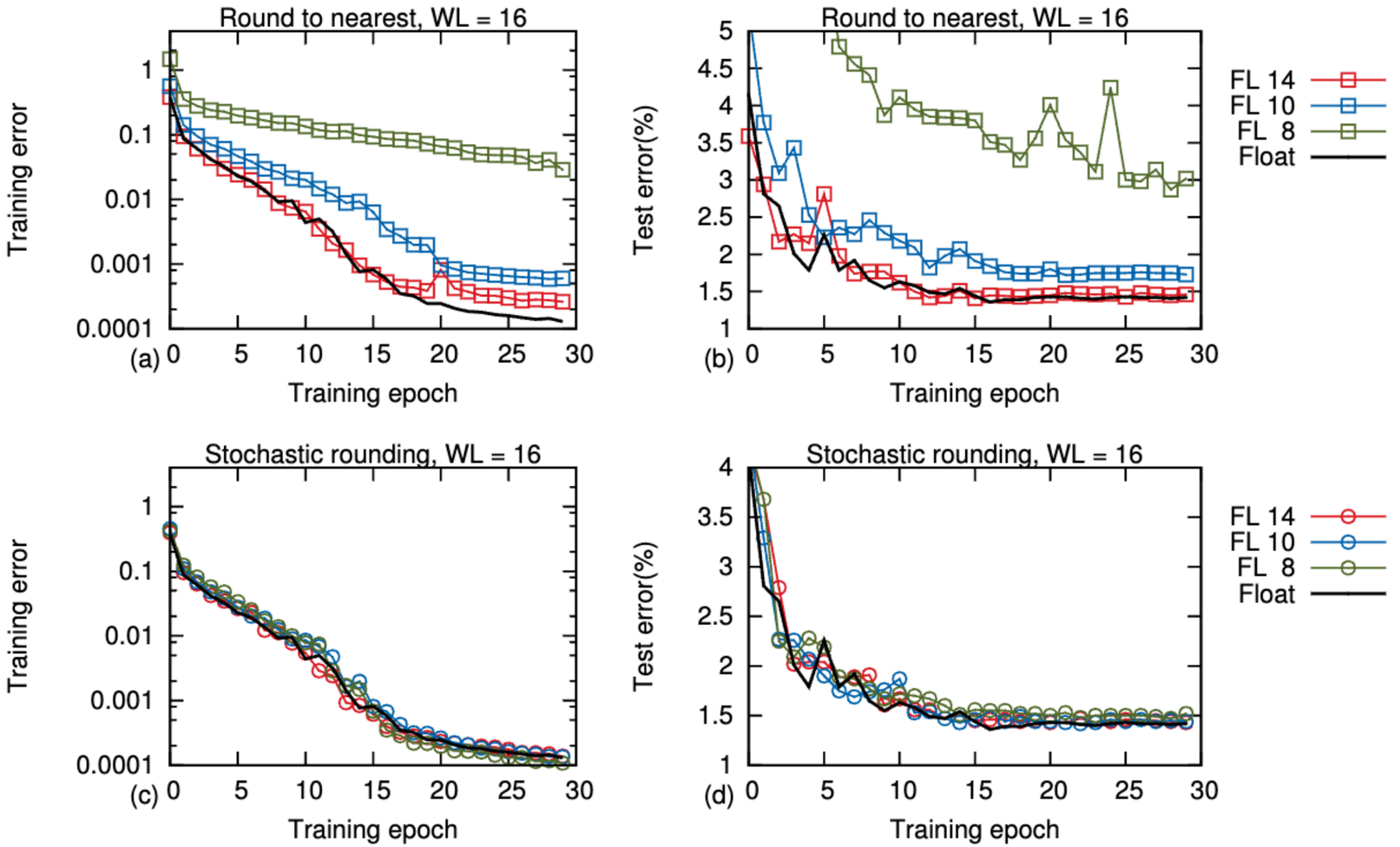
Method

- Quantized many things:
  - Weights
  - Biases
  - Activations
  - Back-propagated error
  - Weight update
  - Bias update

Suyog et al., “Deep Learning with Limited Numerical Precision,” ICML 2015

Observation

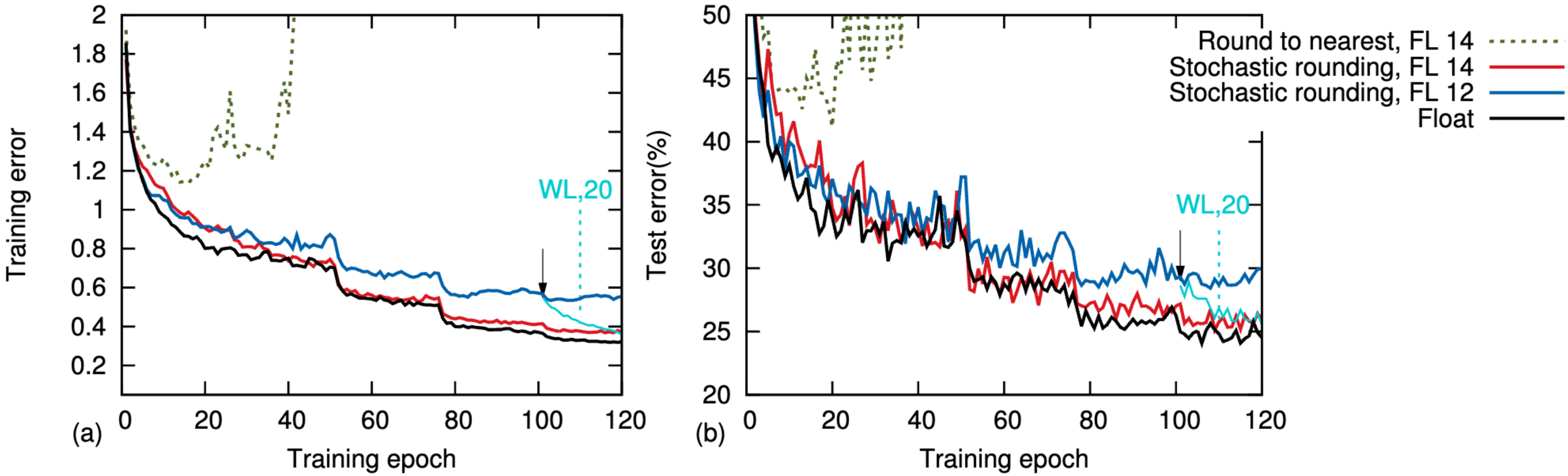
- Stochastic rounding is quite essential (MNIST+MLP example)



Suyog et al., "Deep Learning with Limited Numerical Precision," ICML 2015

Observation

- Up to 16bit fixed-point, for CIFAR-10 training
  - Assign more bits for the integer bits than fraction
  - Late high-precision training helps



Suyog et al., "Deep Learning with Limited Numerical Precision," ICML 2015

Binary nets

Background

- Another team at Montréal worked on early shapings of low-precision training, with special focus on **binary networks**
  - “Training deep neural networks with low precision multiplications”
    ICLR workshop 2015
  - “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”
    NeurIPS 2015
  - “Binarized Neural Networks”
    NeurIPS 2016
- Loses scaling
  - BatchNorm layers can handle these, presumably

Method

- Binarize the weight during forward, and keep high-precision weights intact
  - **Forward.** Compute output with binarize(w)
    - Stochastic rounding; applies hard sigmoid
  - **Backward.** Compute gradients in full-precision
  - **Update.** Update full-precision weight w
    - Straight-through estimator
    - Apply clipping to [-1,+1]; prevents w from growing endlessly

- Binarization noise is deemed independent and zero-mean
  - cancels out eventually

Courbariaux et al., "BinaryConnect: Training Deep Neural Networks with binary weights during propagations" NeurIPS 2015

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}

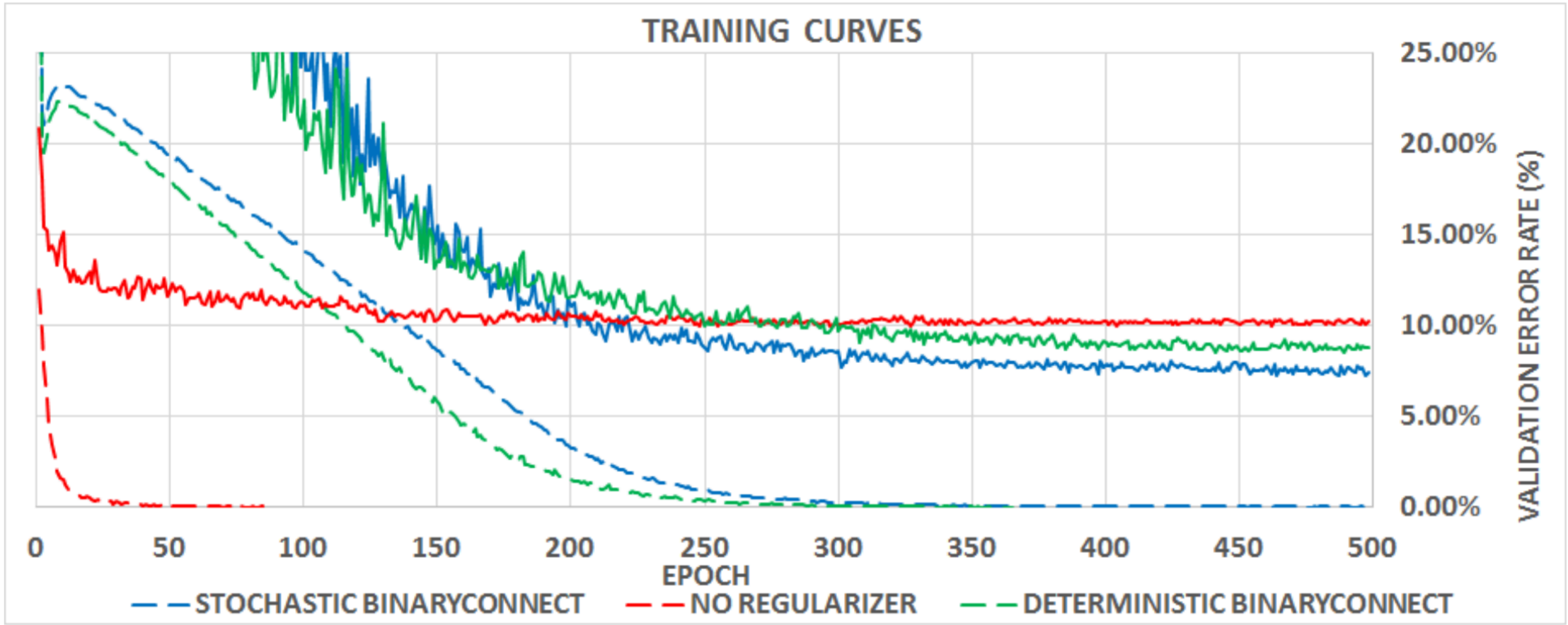
$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

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

Courbariaux et al., "BinaryConnect: Training Deep Neural Networks with binary weights during propagations" NeurIPS 2015

Observations

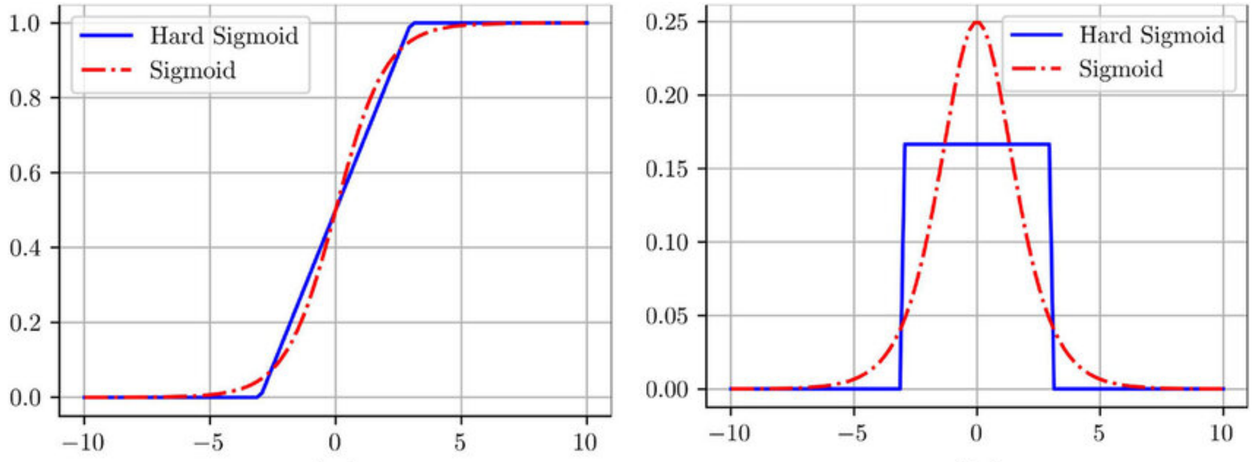
- Requires much longer training than full-precision
- Interestingly, advantages in generalization



Courbariaux et al., "BinaryConnect: Training Deep Neural Networks with binary weights during propagations" NeurIPS 2015

2016 work

- Binarizes both **W&A**
  - Keeps both W&A in full-precision as well
- Uses hard-sigmoid as activation function, with additional binarization
  - Gradients for activations outside [-1,+1] is set to zero
    - Empirically works well, rather than being well-justified
      (in fact, later works remove this; e.g., XNOR-Net, DoReFA-Net)



Hubara et al., “Binarized neural networks” NeurIPS 2016

Limitation

- Keeps track of **full-precision W&A**
  - Much memory cost
- Requires **full-precision gradient**
  - Weak computation advantage during backward

XNOR-Net

- Introduces **scaling factor** to the forward binary operation
  - This change force us to update weights greater than 1, as well.
  - **BNN.**

$$\textbf{Forward: } r_o = \text{sign}(r_i)$$

$$\textbf{Backward: } \frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}\mathbb{I}_{|r_i|\leq 1}.$$

  - **XNOR-Net.**

$$\textbf{Forward: } r_o = \text{sign}(r_i) \times \mathbf{E}_F(|r_i|)$$

$$\textbf{Backward: } \frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}.$$

(expectation denotes averaging w.r.t. output channel)

Rastegari et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks" ECCV 2016

DoReFa-Net

- Generalizes the idea to **binarizing gradients** as well

- Unlike activations, gradients are **unbounded** (typically larger than activations)
  - Thus perform quantization with normalization factors

$$f_\gamma^k(\mathrm{d}r) = 2\max_0(|\mathrm{d}r|)\left[\mathrm{quantize}_k[\frac{\mathrm{d}r}{2\max_0(|\mathrm{d}r|)} + \frac{1}{2} + N(k)] - \frac{1}{2})\right]$$

  - Here, $N(k)$ is a uniform noise which compensates for any bias due to gradient quantization

Zhou et al., "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients" arXiv 2016

Further readings

- This line of work has been followed-up by BitNet, in the context of LLMs
- https://arxiv.org/abs/2310.11453

FP16/FP8 training

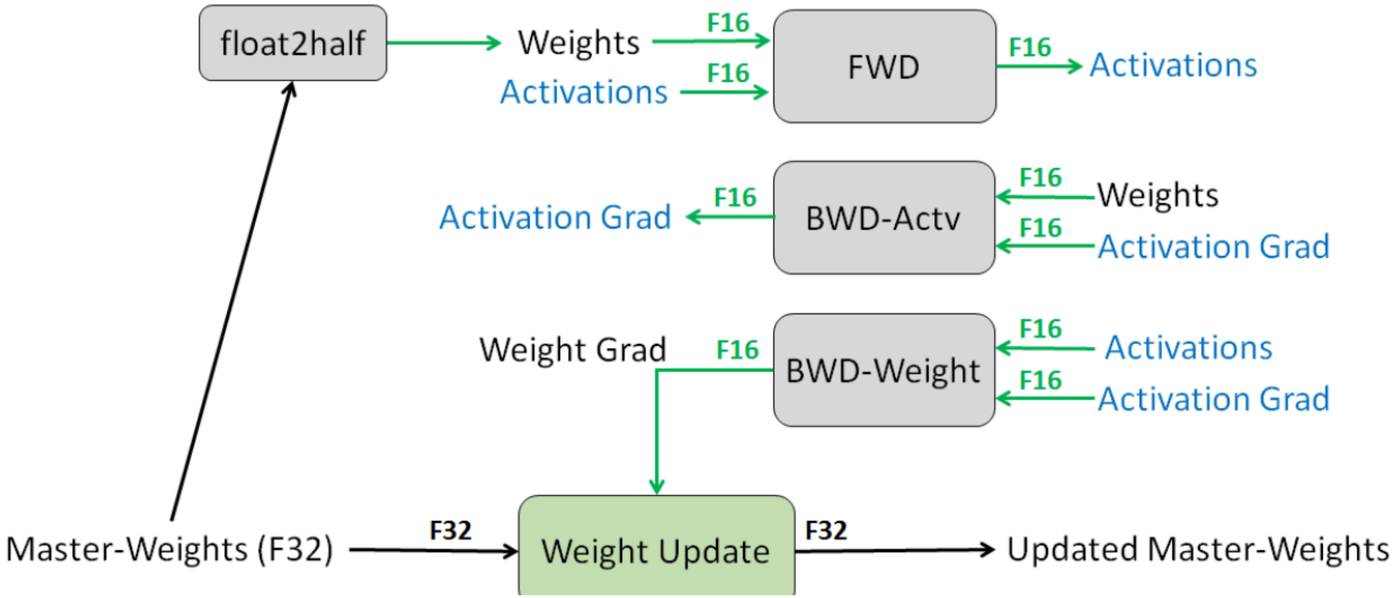
Background

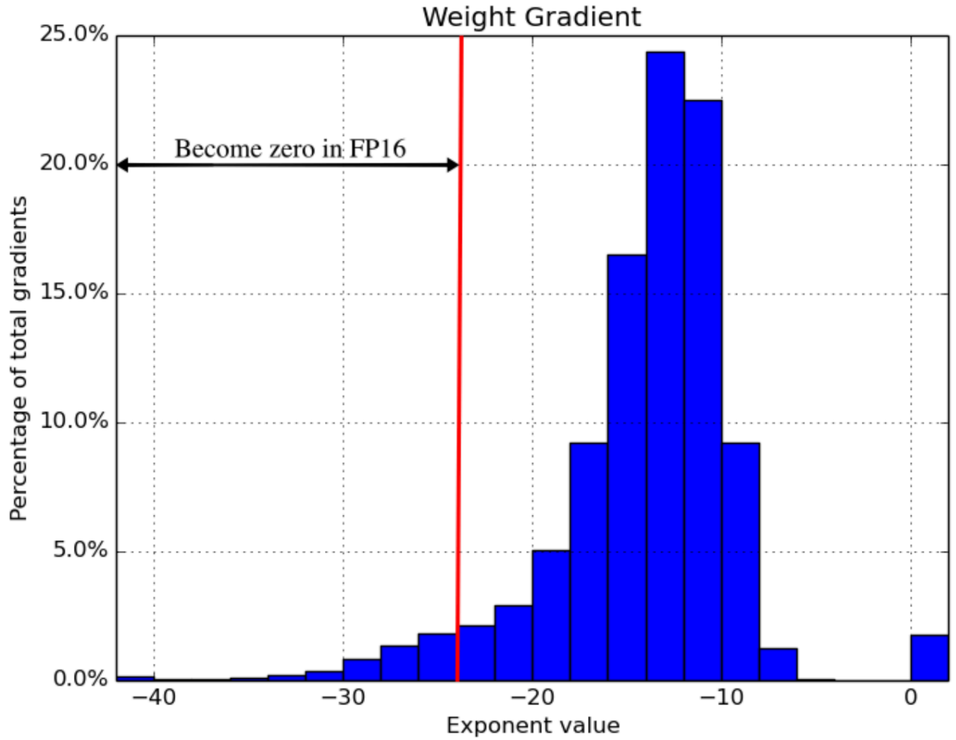
- Concerns regarding “will fixed-point scale up?”
    - ImageNet-scale experiments
    - Applications to language modeling

- **Solution.** Use floating point?

FP16 Training (NVIDIA ver.)

- NVIDIA & Baidu worked on extending the idea to FP16
  - Weights are also kept in FP32, but not others
  - Reason. Weight updates will be too small, when multiplied with LR

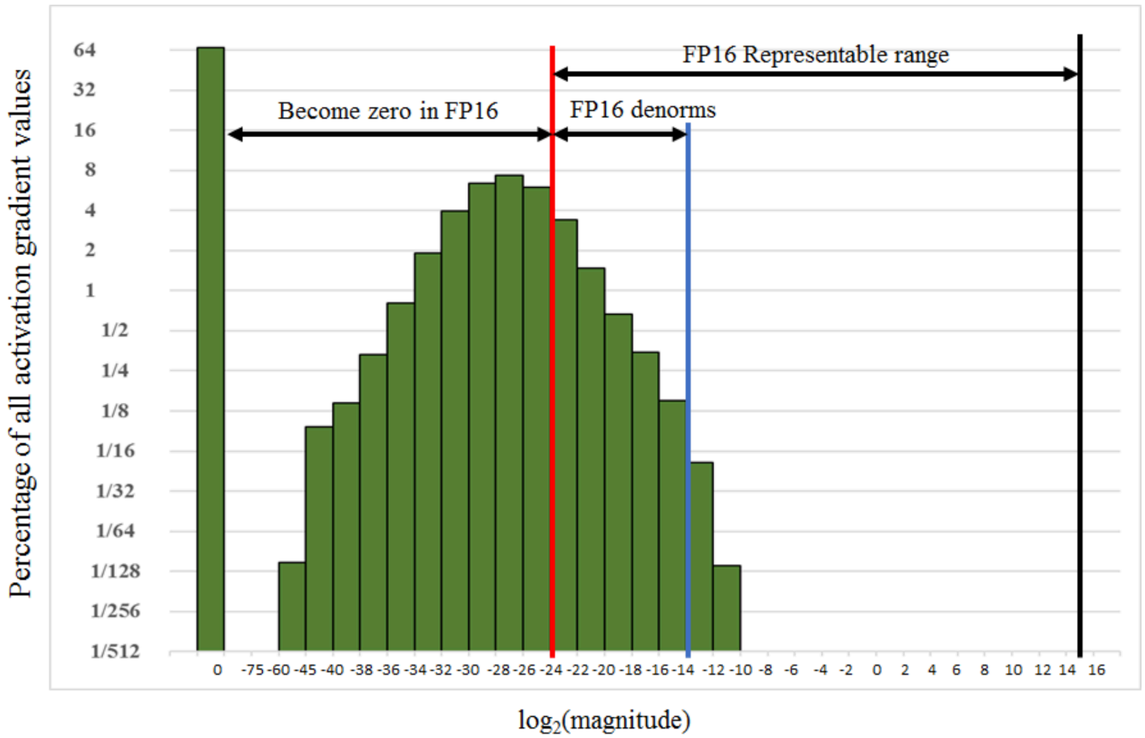




Micikevicius et al., "Mixed Precision Training" ICLR 2018

FP16 Training (NVIDIA ver.)

- Also, we cannot use the same dynamic range for weights and gradients
  - Many gradients have very small values
  - **<u>Solution.</u>** Scale up the loss (x8), so that the gradient become larger



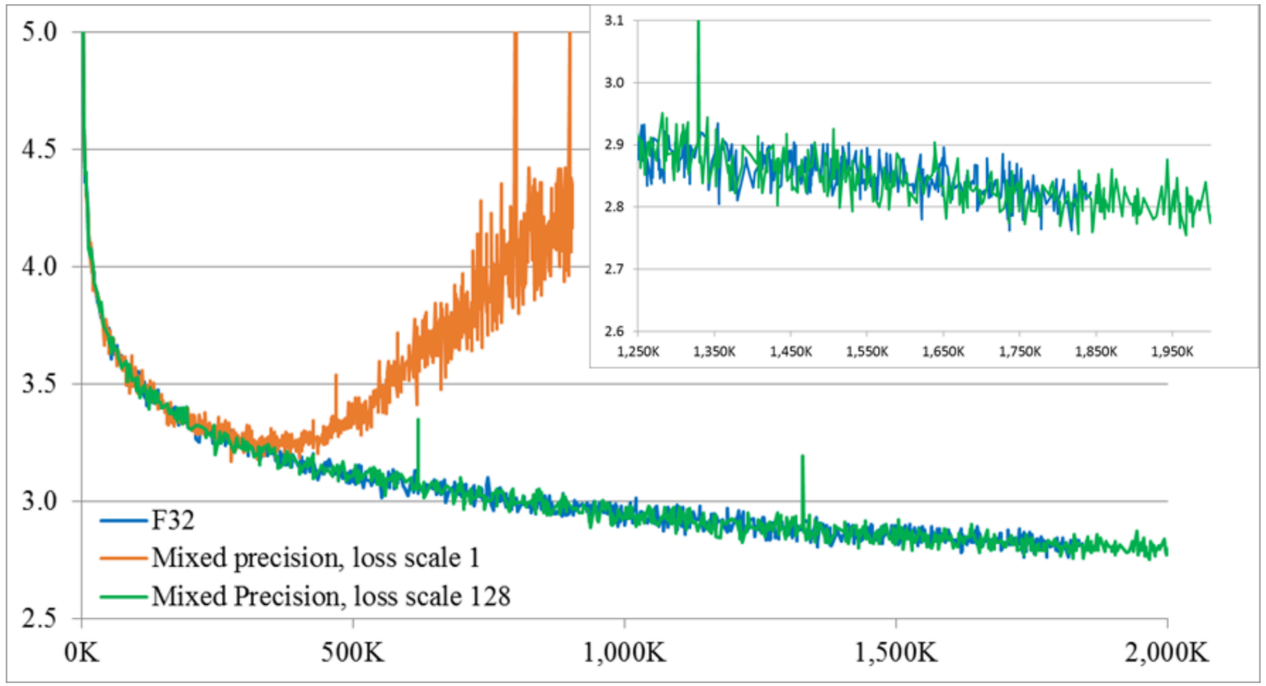


Figure 5: bigLSTM training perplexity

Micikevicius et al., "Mixed Precision Training" ICLR 2018

Further readings

- Intel concurrently studied INT16 training (ICLR 2018):
    - https://arxiv.org/abs/1802.00930

FP8 Training (IBM ver.)

- IBM introduced FP8 format (E5M2)
  - Accumulation in FP16 (E6M9); still a "master copy" of weights
  - AXPY (vector addition; $y = \alpha x + y$) done in FP16

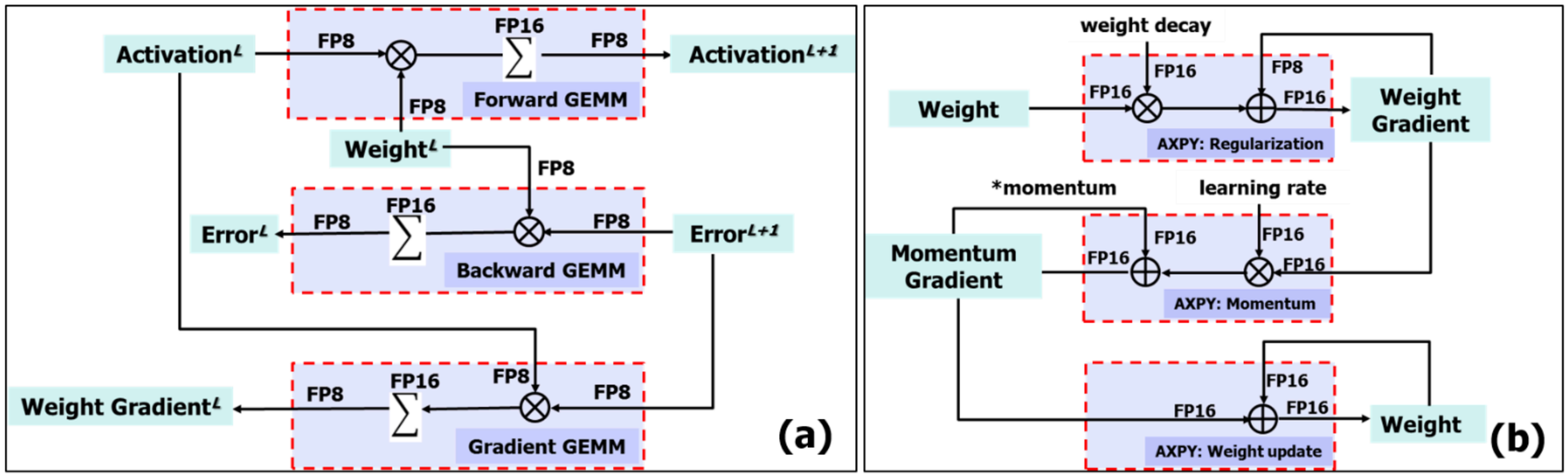


Figure 2: A diagram showing the precision settings for (a) three GEMM functions during forward and backward passes, and (b) three AXPY operations during a standard SGD weight update process.

Wang et al., "Training Deep Neural Networks with 8-bit Floating Point Numbers" NeurIPS 2018

FP8 Training (IBM ver.)

- **Observation.** "Swamping" happens a lot in DL
  - Losing information when adding large + small FP values
  - Especially problematic if:
    - Adding a lot of non-zero mean values (as the sum gradually grows)
    - Some of the elements are extremely large

$$
\begin{array}{rccccccccccc}
 & 0 & . & d_1 & d_2 & \cdots & d_{m-p} & d_{m-p+1} & \cdots & d_{t-1} & d_t \\
+ & 0 & . & 0 & 0 & \cdots & 0 & e_1 & e_2 & \cdots & e_{t-(m-p)} \\
\hline
 & 0 & . & d_1 & d_2 & \cdots & d_{m-p} & f_{m-p+1} & \cdots & f_{t-1} & f_t
\end{array}
$$

Wang et al., "Training Deep Neural Networks with 8-bit Floating Point Numbers" NeurIPS 2018

FP8 Training (IBM ver.)

- **Solution. Chunk-based accumulation**
  - Generate partial sums to avoid scale mismatch
    - DON'T:
      - a1 = a1 + a2
      - a1 = a1 + a3
      - a1 = a1 + a4
    - DO:
      - a1 = a1 + a2
      - a3 = a3 + a4
      - a1 = a1 + a3

Wang et al., "Training Deep Neural Networks with 8-bit Floating Point Numbers" NeurIPS 2018

FP8 Training (IBM ver.)

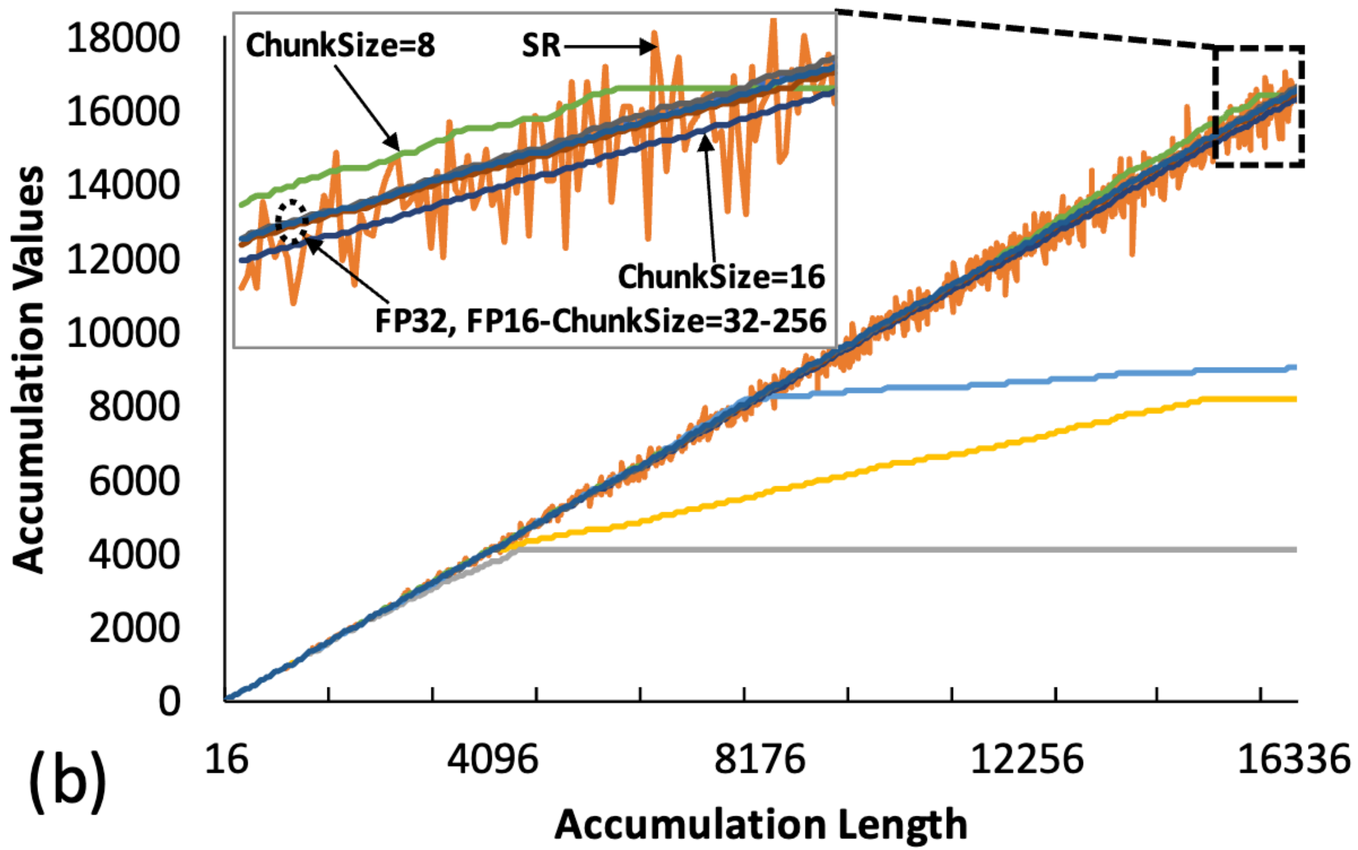
- **Solution. Stochastic rounding**
  - Rounded-off values (least significant bits) give chance of rounding up, instead of simply being truncated
  - For floating-point, rounding the value $x = s \cdot 2^e \cdot (1 + m)$ is done by:

$$\text{Round}(x) = \begin{cases} s \cdot 2^e \cdot (1 + \lfloor m \rfloor + \epsilon) & \text{w.p.} \quad \frac{m - \lfloor m \rfloor}{\epsilon} \\ s \cdot 2^e \cdot (1 + \lfloor m \rfloor) & \text{w.p.} \quad 1 - \frac{m - \lfloor m \rfloor}{\epsilon} \end{cases}$$

Wang et al., “Training Deep Neural Networks with 8-bit Floating Point Numbers” NeurIPS 2018

FP8 Training (IBM ver.)

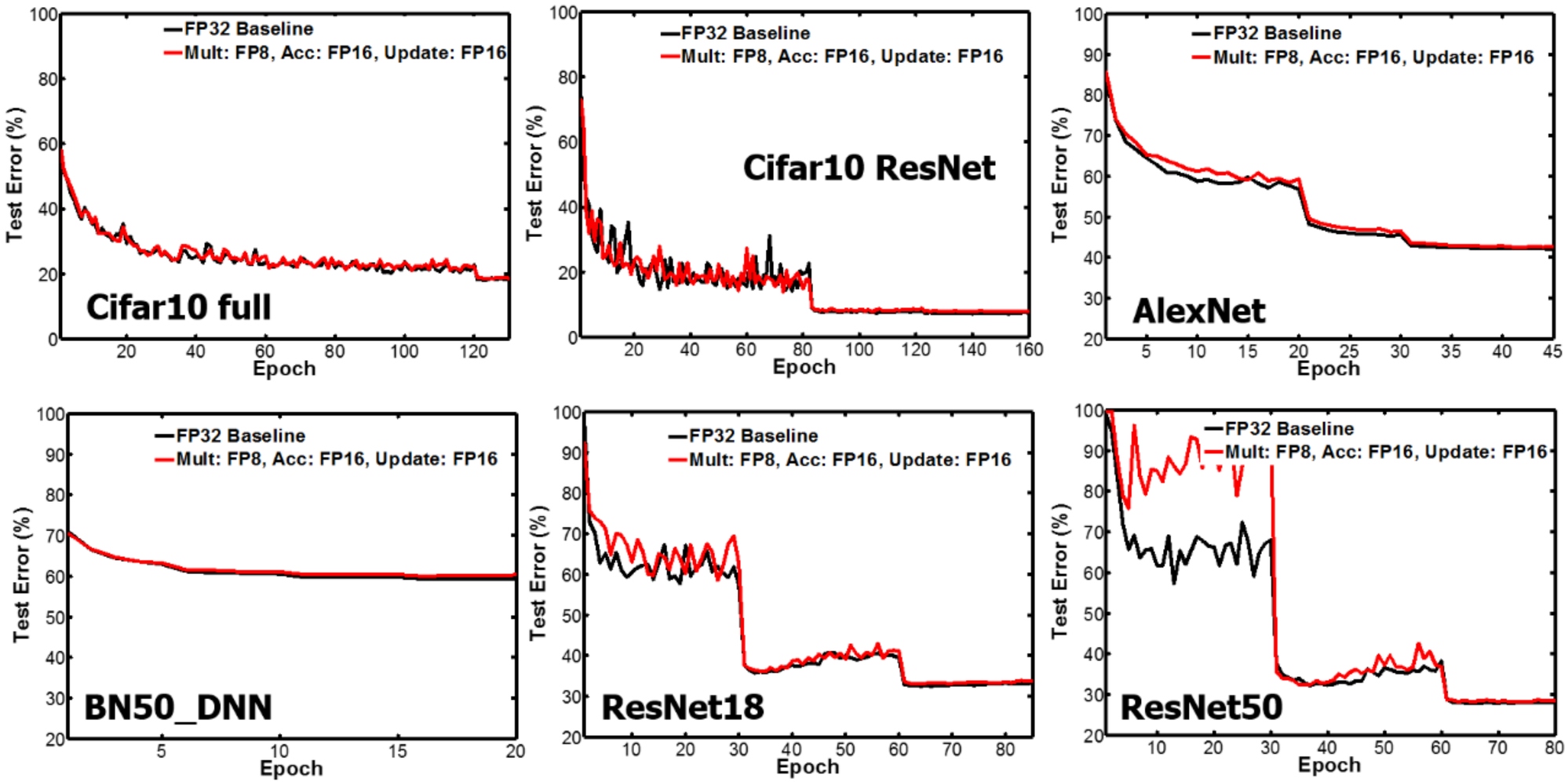
- Use Chunking for GEMM, and SR for AXPY
  - AXPY is done over data batches; no hierarchical aggregation doable



Wang et al., "Training Deep Neural Networks with 8-bit Floating Point Numbers" NeurIPS 2018

FP8 Training (IBM ver.)

- **Other details.** No quantization for 1st and the last layer



Wang et al., "Training Deep Neural Networks with 8-bit Floating Point Numbers" NeurIPS 2018

Follow-ups

- Later work from IBM uses **E4M3 for forward** and **E5M2 for backward**
  - Works better on MobileNetV2 and Transformers https://papers.nips.cc/paper_files/paper/2019/hash/65fc9fb4897a89789352e211ca2d398f-Abstract.html
  - Also much discussions on BatchNorm
- Similar findings by Graphcore
  - https://arxiv.org/abs/2206.02915
  - See also 2023 work: https://openreview.net/forum?id=nErbvDkucY
    - More discussions on loss scaling — per-tensor scaling

Sun et al., "Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks" NeurIPS 2019
Noune et al, "8-bit Numerical Formats for Deep Neural Networks," arXiv 2022
Perez et al., "Training and inference of large language models using 8-bit floating point," WANT workshop @ NeurIPS 2023

Follow-ups

- NVIDIA, ARM, and Intel proposes an interchange format of E4M3 and E5M2:
  - https://arxiv.org/abs/2209.05433
  - Conducts extensive empirical study, showing that FP8 is stable enough

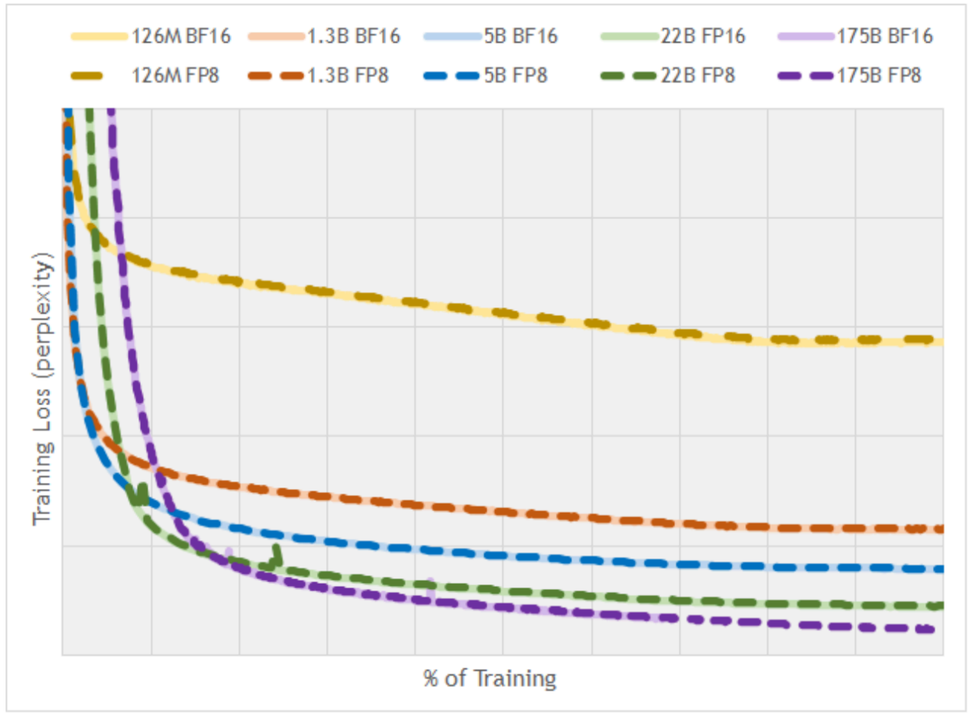
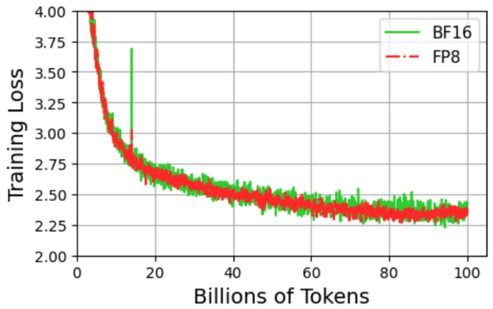


Figure 1: Training loss (perplexity) curves for various GPT-3 models. x-axis is normalized number of iterations.

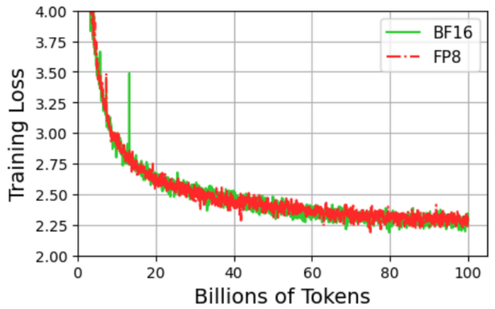
Micikevicius et al, "FP8 formats for deep learning," arXiv 2022

Follow-ups

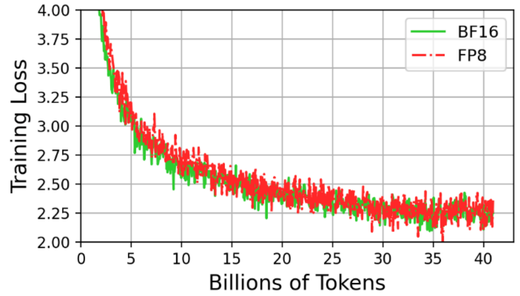
- Microsoft applies FP8 on all pipeline: including parallelism & optimizer
  - https://arxiv.org/abs/2310.18313
    - Precision decoupling: Reduced precision for insensitive weights
    - Automatic scaling: Per-tensor scaling of loss



(a) GPT-7B



(b) GPT-13B



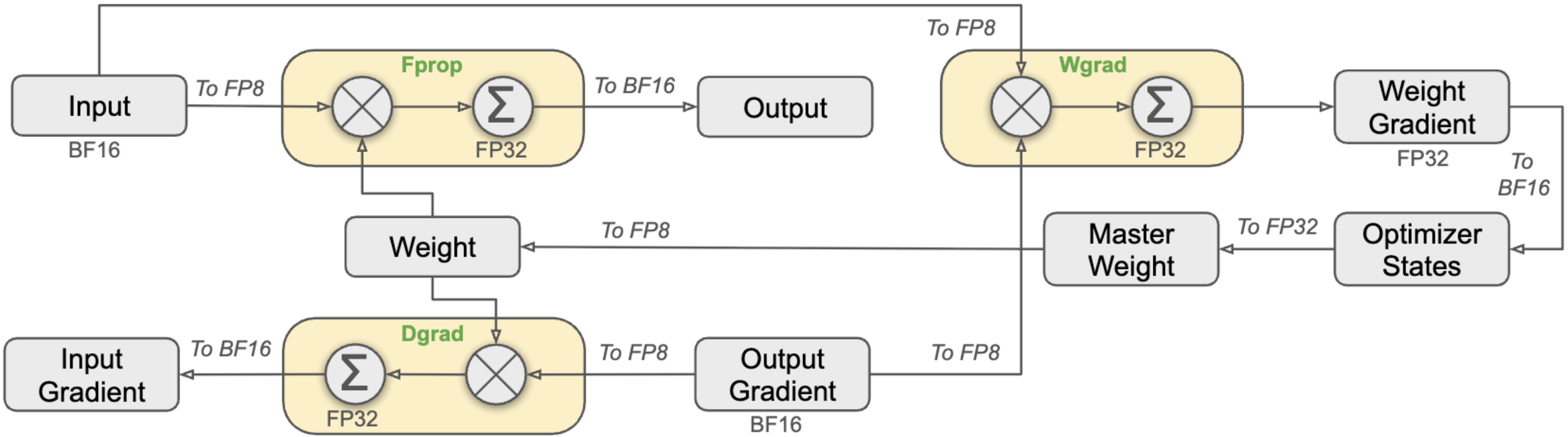
(c) GPT-175B

	HS	Lambada	BoolQ	PIQA	COPA	Winogrande	Arc-C	Arc-E	ObQA	Avg
GPT-7B model zero-shot performance										
BF16	61.3	61.4	61.2	75.0	79.0	58.5	32.9	59.7	36.4	58.4
FP8	60.0	61.8	62.0	74.2	78.0	59.8	32.9	58.7	34.6	58.0
GPT-13B model zero-shot performance										
BF16	64.8	64.9	63.4	75.9	82.0	61.0	35.2	61.5	40.6	61.0
FP8	64.1	63.4	63.9	76.2	81.0	61.6	34.9	61.3	36.8	60.4

Peng et al., "FP8-LM: Training FP8 Large Language Models," arXiv 2023

Follow-ups

- DeepSeek-V3 applies E4M3 for all ops
  - Combine with fine-grained quantization (and many other details) https://arxiv.org/abs/2412.19437



DeepSeek-AI, "DeepSeek-V3 Technical Report," arXiv 2024

Takeaways

- Both families are seeing much progress nowadays
    - **PTQ.** Any lessons to learn from SOTA techniques?
    - **QAT.** Similar ideas applicable for QAT?

- We'll see more in the student presentations...

That’s it for today 🙌