



CS 498: Machine Learning System

Spring 2026

Minja Zhang

The Grainger College of Engineering

Model Parallelism:

- Tensor parallelism: communication-heavy
- Pipeline parallelism: tuning microbatch sizes to minimize bubbles
- Data parallelism: cannot run if layer (model) size exceeds GPU capacity

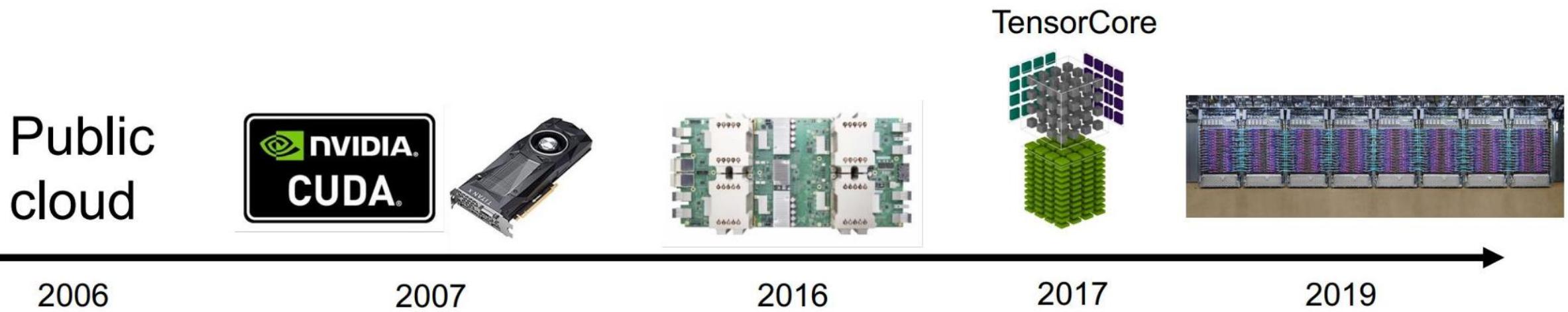
Mixed Precision Training

Learning Objectives

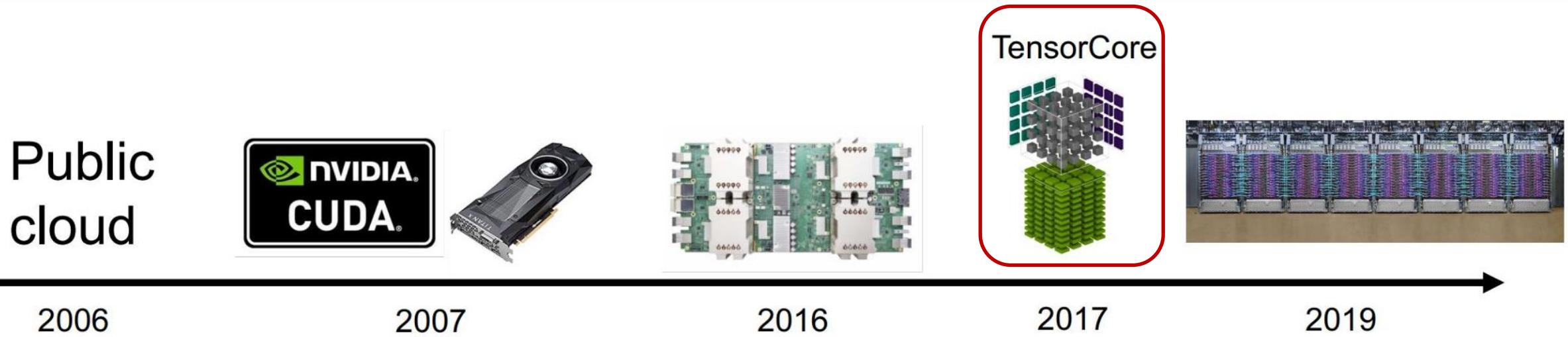
- Explain why mixed precision improves training speed using modern hardware
- Describe the key techniques that make mixed precision stable

- **Mixed Precision Hardware**
- **What is Mixed Precision Training?**
- **Considerations for Mixed Precision**
- **Mixed Precision Software**

Mixed Precision Hardware



Compute scaling

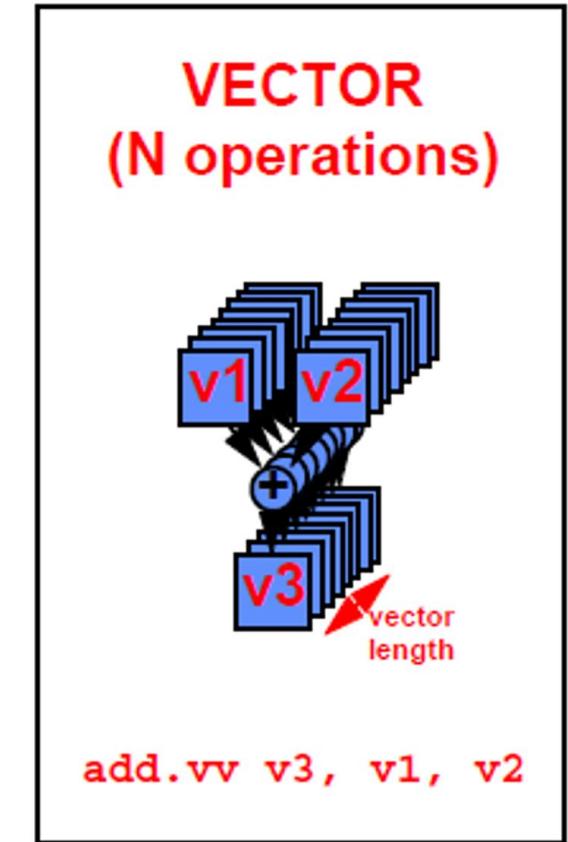
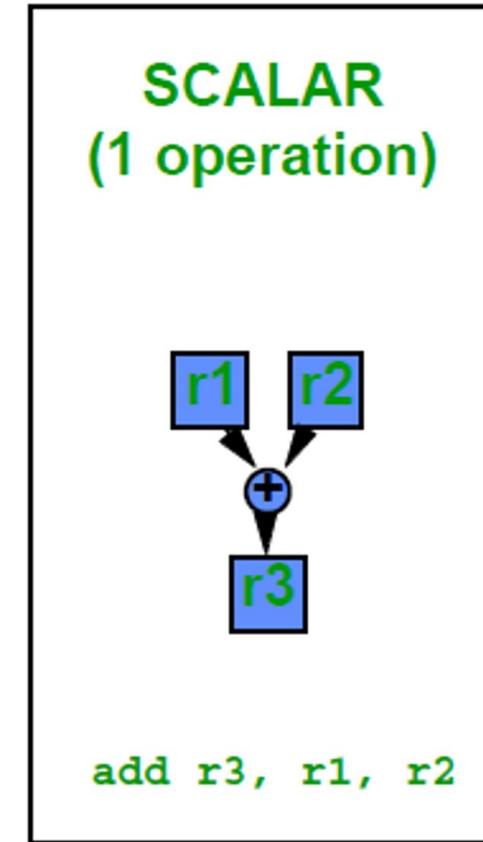


Compute scaling

Vector Processing Unit



- A processing unit that operates on an entire vector in one instruction (SIMD)
- The operand to the instructions are vectors instead of a scalar
- Work done in parallel in one instruction cycle
- Accelerated speed



Tensor Cores: Matrix Multiplication Units



VOLTA ARCHITECTURE



Tensor Cores: Matrix Multiplication Units



- **Tensor cores are:**
 - Special hardware execution units
 - Execute matrix multiply operations in parallel
 - Fused multiply-add on small matrices in one instruction cycle
 - Built to accelerate deep learning



VOLTA ARCHITECTURE

Tensor Cores: Matrix Multiplication Units

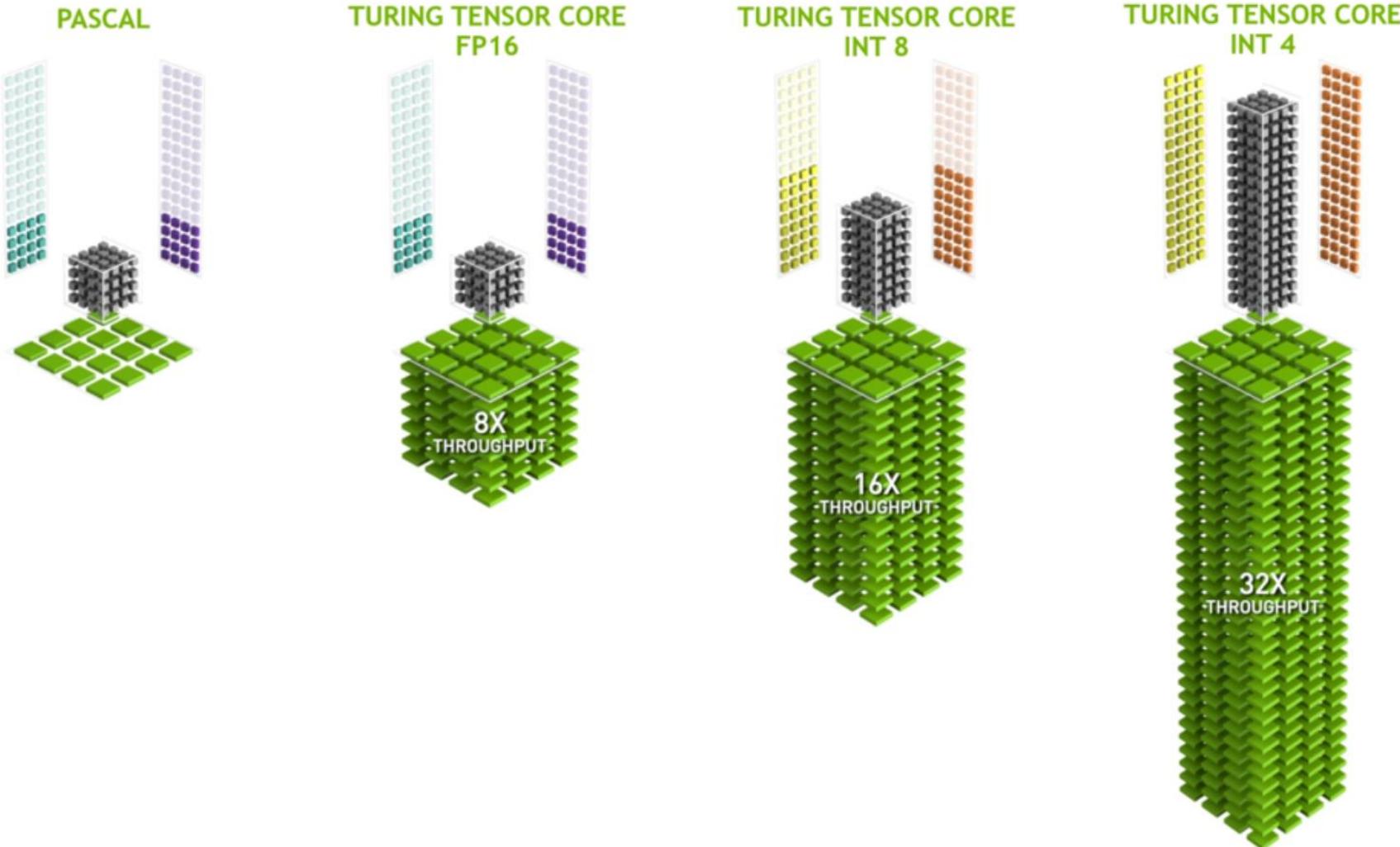


- **Tensor cores are:**
 - Special hardware execution units
 - Execute matrix multiply operations in parallel
 - Fused multiply-add on small matrices in one instruction cycle
 - Built to accelerate deep learning
- **Different flavors**
 - (V100) Volta Tensor Cores FP16
 - (T4) Turing Tensor Cores FP16/INT8/INT4
 - ...

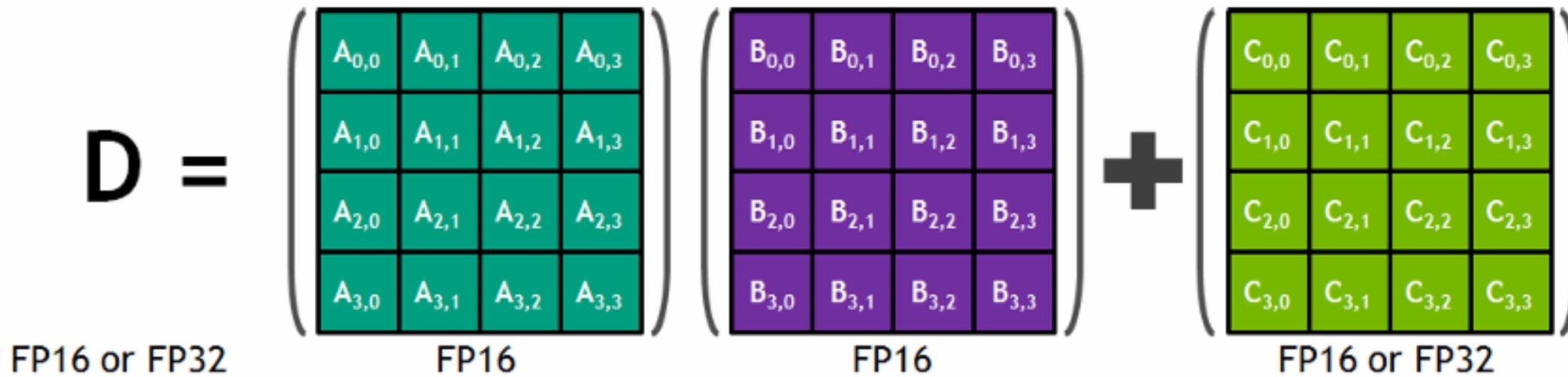


VOLTA ARCHITECTURE

Tensor Cores: Different Flavors

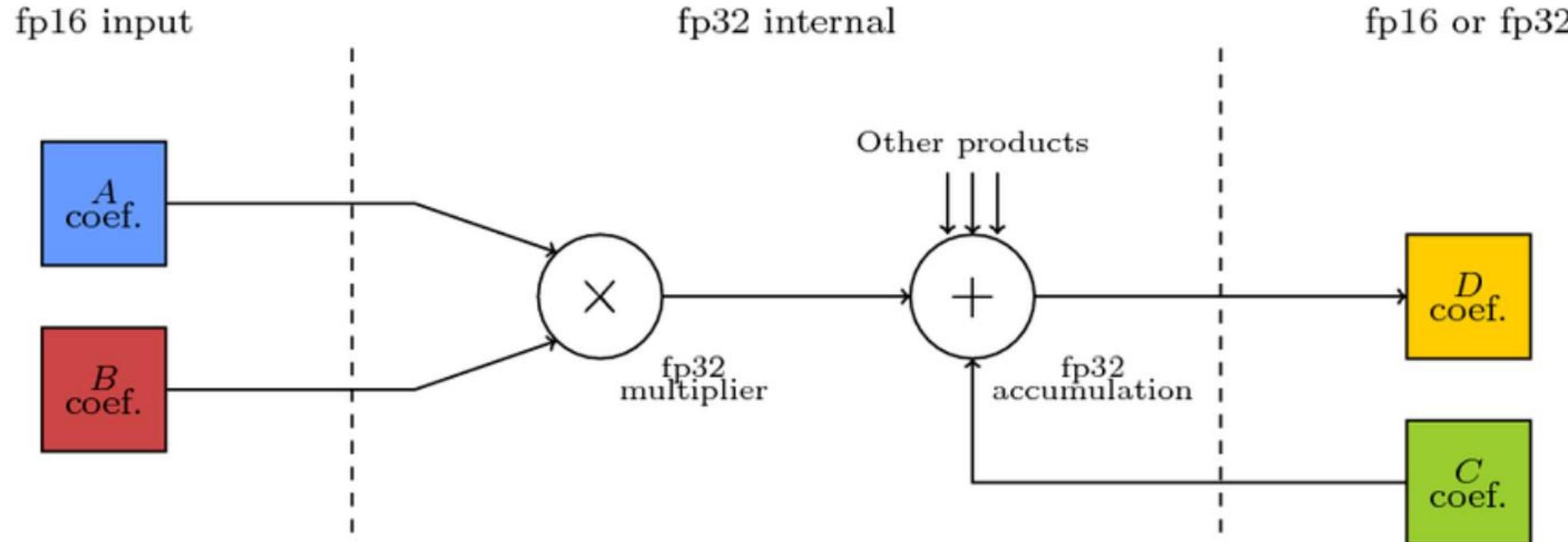


Tensor Cores: Mixed Precision Matrix Math on 4x4 Matrices



$$D = AB + C$$

Internals of Tensor Cores (Volta)



- Accelerate matrix multiplications and convolutions
- Tensor core optimized libraries: cuDNN, cuBLAS, CUTLASS

<https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/>

What is Mixed Precision Training?

What is Mixed Precision Training? In a Nutshell



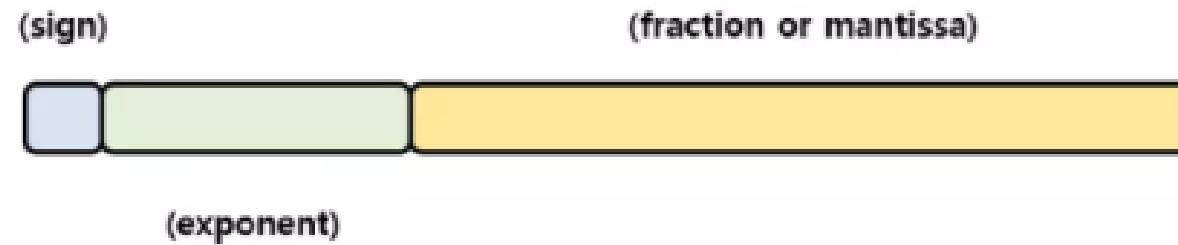
- Idea that you can train deep neural networks in multiple precisions:
 - Make precision decisions per layer or operation
 - Full precision (Fp32) where needed to *maintain task-specific accuracy*
 - Reduced precision (Fp16) everywhere else for speed and scale

What is Mixed Precision Training? In a Nutshell



- Idea that you can train deep neural networks in multiple precisions:
 - Make precision decisions per layer or operation
 - Full precision (Fp32) where needed to *maintain task-specific accuracy*
 - Reduced precision (Fp16) everywhere else for speed and scale
- By using *multiple* precisions, we can have the best of both worlds: **speed and accuracy**
- Goal: accelerate deep neural network training with mixed precision under the constraints of **matching accuracy of full precision training and no changes to how model is trained**

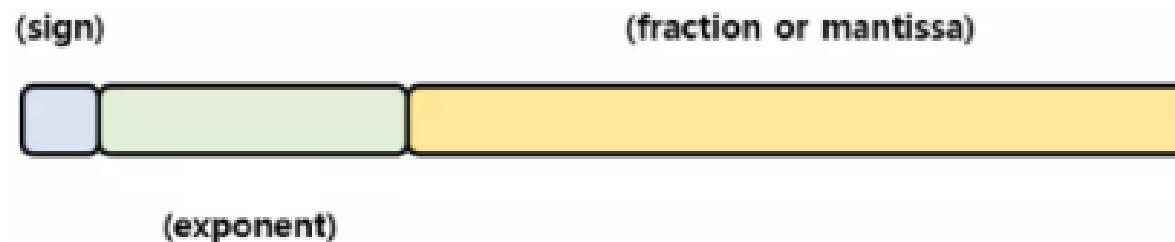
IEEE 754 Floating Point Representation Recap



IEEE 754 Floating Point Representation Recap



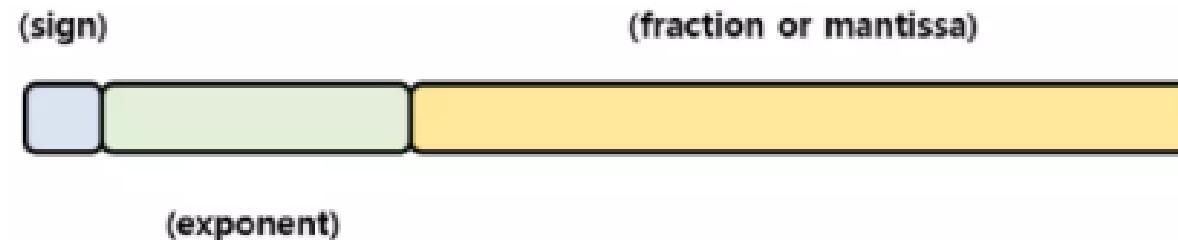
- Number (e.g., 2.45) can be represented by ?



IEEE 754 Floating Point Representation Recap



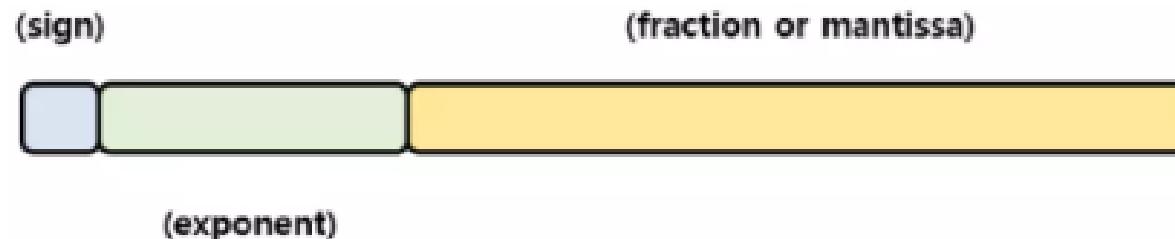
- Number (e.g., 2.45) can be represented by $(-1)^S * (1.M) * 2^{(E - Bias)}$



IEEE 754 Floating Point Representation Recap



- Number can be represented by $(-1)^S * (1.M) * 2^{(E - Bias)}$



Half Precision (Floating Point 16)	1 bit	5 bit	10 bit
Single Precision (Floating Point 32)	1 bit	8 bit	23 bit
Double Precision (Floating Point 64)	1 bit	11 bit	52 bit
Quadruple Precision (Floating Point 128)	1 bit	15 bit	113 bit

IEEE 754 Floating Point Representation Recap



- Number can be represented by $(-1)^S * (1.M) * 2^{(E - Bias)}$



Half Precision (Floating Point 16)	1 bit	5 bit	10 bit
Single Precision (Floating Point 32)	1 bit	8 bit	23 bit
Double Precision (Floating Point 64)	1 bit	11 bit	52 bit
Quadruple Precision (Floating Point 128)	1 bit	15 bit	113 bit

IEEE 754 Floating Point Representation Recap

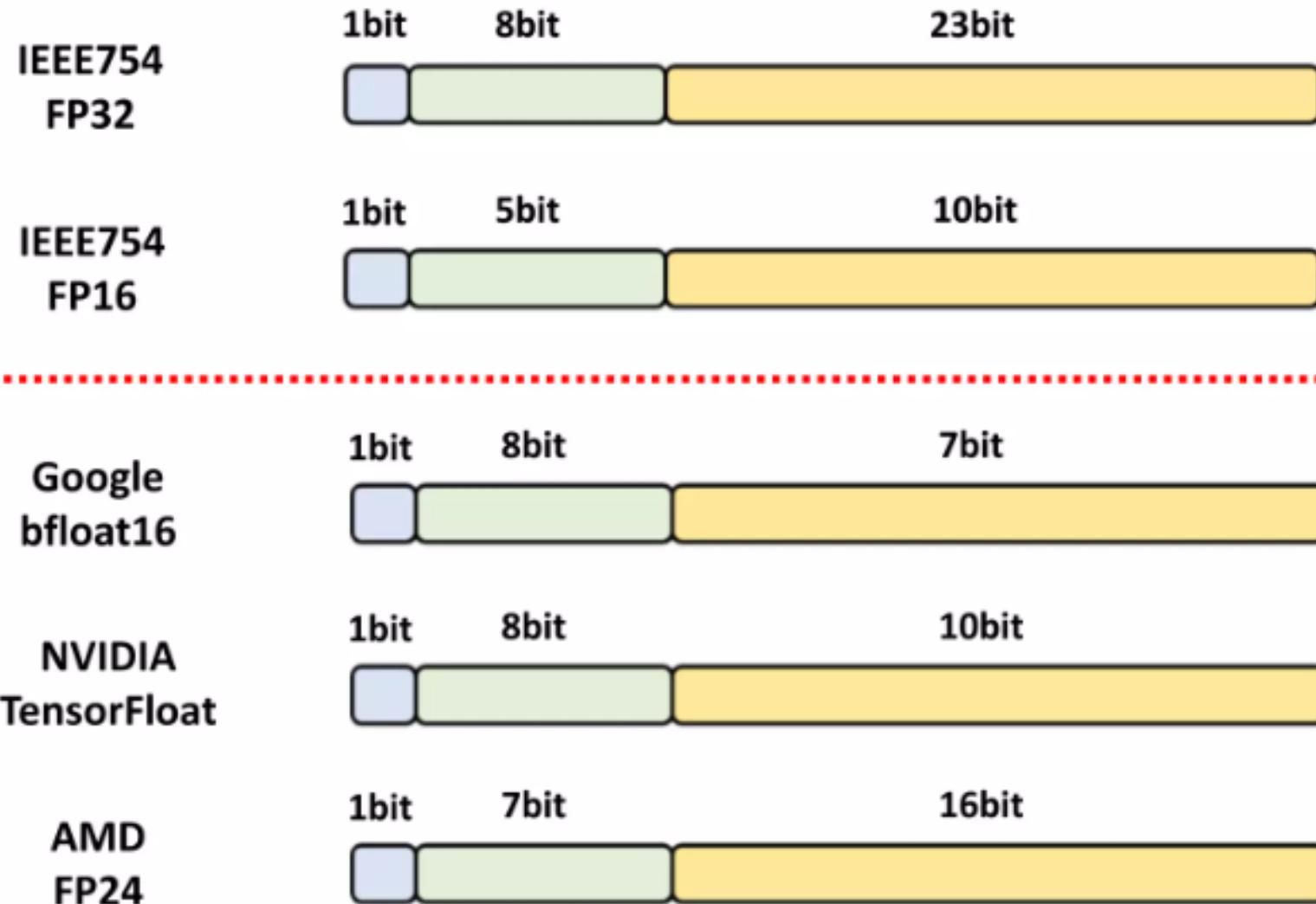


- Number can be represented by $(-1)^S * (1.M) * 2^{(E - Bias)}$



Half Precision (Floating Point 16)	1 bit	5 bit	10 bit
Single Precision (Floating Point 32)	1 bit	8 bit	23 bit
Double Precision (Floating Point 64)	1 bit	11 bit	52 bit
Quadruple Precision (Floating Point 128)	1 bit	15 bit	113 bit

New Floating-Point Format



Mixed Precision Training Benefits



- **Accelerates speed**
 - Tensor cores are 8x faster than FP32

- **Accelerates speed**
 - Tensor cores are 8x faster than FP32
- **Reduces memory bandwidth pressure**
 - FP16 halves memory traffic compared to FP32

- **Accelerates speed**
 - Tensor cores are 8x faster than FP32
- **Reduces memory bandwidth pressure**
 - FP16 halves memory traffic compared to FP32
- **Reduces memory consumption**
 - FP16 halves the size of activation and gradient tensors
 - Enables larger models, mini batches or inputs

Why Not Use Pure FP16?



- Models cannot always converge properly in pure FP16
 - FP16 has a narrower dynamic range than FP32
 - May cause underflow/overflow issues and other arithmetic issues

- Models cannot always converge properly in pure FP16
 - FP16 has a narrower dynamic range than FP32
 - May cause underflow/overflow issues and other arithmetic issues
- **Weight updates**
 - Optimizer takes very **small increments** when search narrows into a solution
 - Late updates often cannot be represented in FP16, but can be crucial for accuracy

- Models cannot always converge properly in pure FP16
 - FP16 has a narrower dynamic range than FP32
 - May cause underflow/overflow issues and other arithmetic issues
- **Weight updates**
 - Optimizer takes very **small increments** when search narrows into a solution
 - Late updates often cannot be represented in FP16, but can be crucial for accuracy
- **Reductions**
 - Large sums of values, e.g., in linear layers and convolutions, can be **too big** for FP16
 - Adding small values to a large sum can lead to **rounding errors**

Considerations for Mixed Precision

- Goal #1: Make mixed precision training general purpose, not only for limited class of applications
- Goal #2: With no changes to hyperparameters or the model architecture

Three parts:

PRECISION OF OPS

Decide which operations to compute in FP16 and FP32.

MASTER WEIGHTS

Keep an FP32 copy of the model weights.

LOSS SCALING

Scale the loss value to retain small gradients.



Matrix Multiplication

linear, matmul, bmm, conv

8x performance boost from Tensor Cores

Matrix Multiplication

linear, matmul, bmm, conv

8x performance boost from Tensor Cores

Pointwise

relu, sigmoid, tanh, exp, log

Reductions

batch norm, layer norm, sum, softmax

Loss Functions

cross entropy, l2 loss, weight decay

Matrix Multiplication

linear, matmul, bmm, conv

8x performance boost from Tensor Cores

Pointwise

relu, sigmoid, tanh, exp, log

Reductions

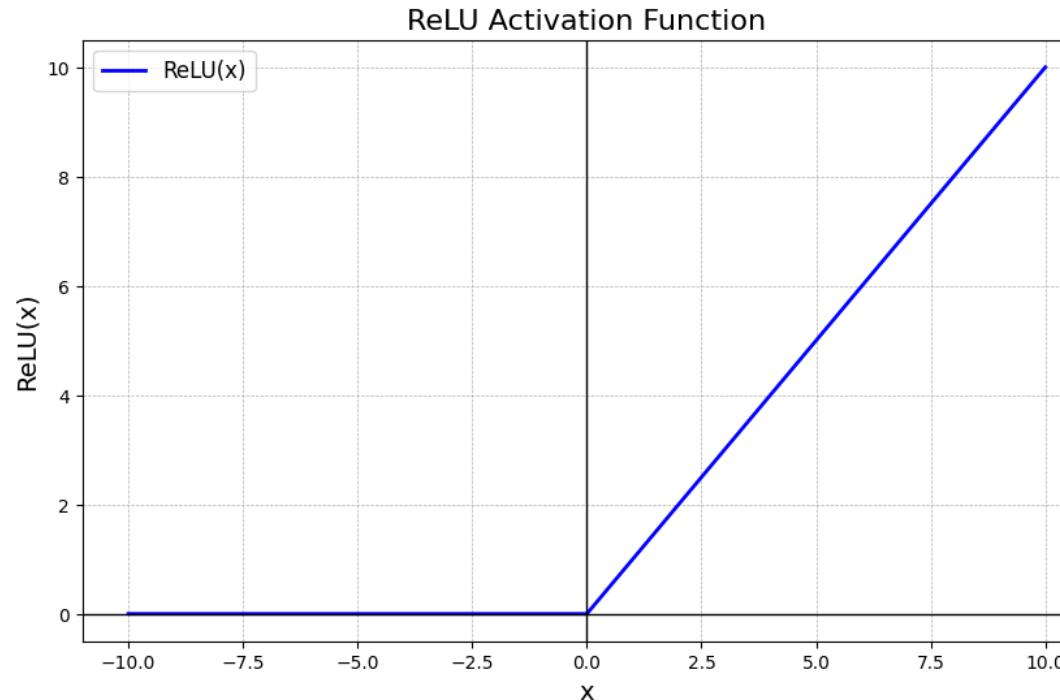
batch norm, layer norm, sum, softmax

Loss Functions

cross entropy, l2 loss, weight decay

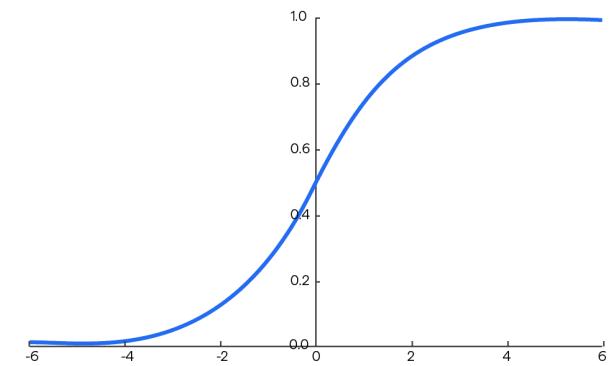
still get some speedup (e.g. 2x memory savings), but without sacrificing accuracy

- Operations that can use FP16
 - Matrix multiplications
 - Most element-wise operations (e.g., relu, tanh, add, sub, mul)



- Operations that can use FP16
 - Matrix multiplications
 - Most element-wise operations (e.g., relu, tanh, add, sub, mul)
- Operations that need FP32 mantissa
 - Reduction operations (e.g., softmax, layernorm)

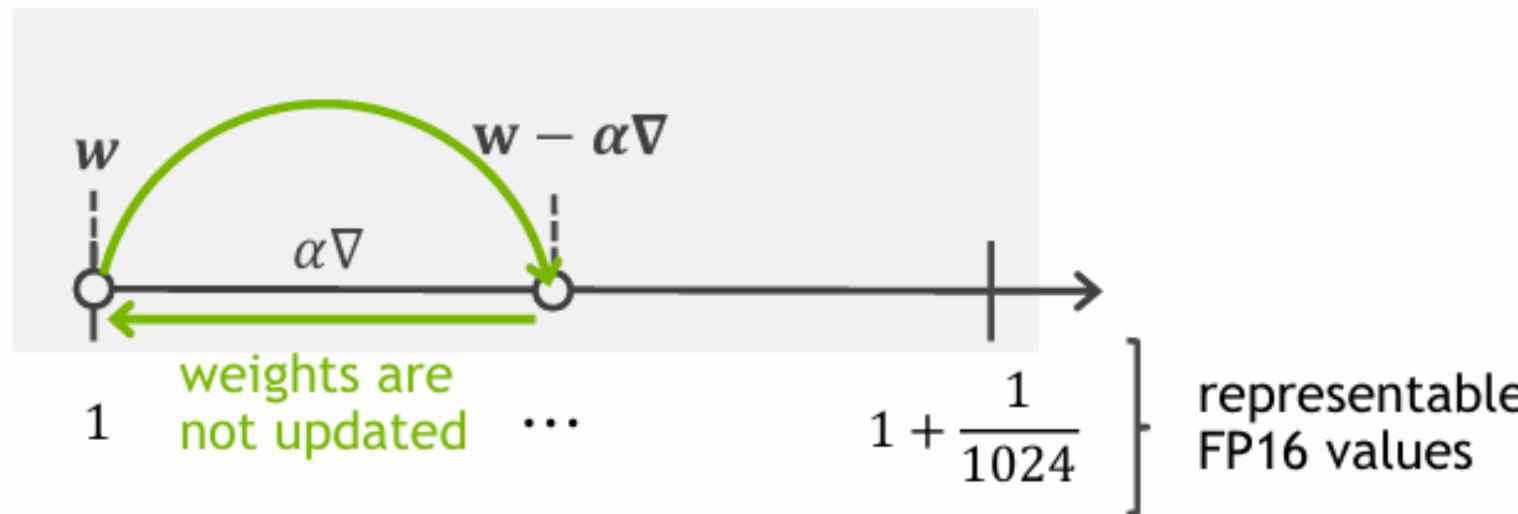
Softmax Function



- Operations that can use FP16
 - Matrix multiplications
 - Most element-wise operations (e.g., relu, tanh, add, sub, mul)
- Operations that need FP32 mantissa
 - Reduction operations (e.g., softmax, layernorm)
- Operations that need FP32 range
 - Element-wise operations where $|f(x)| \gg |x|$, e.g., exp, log, pow
 - Loss functions

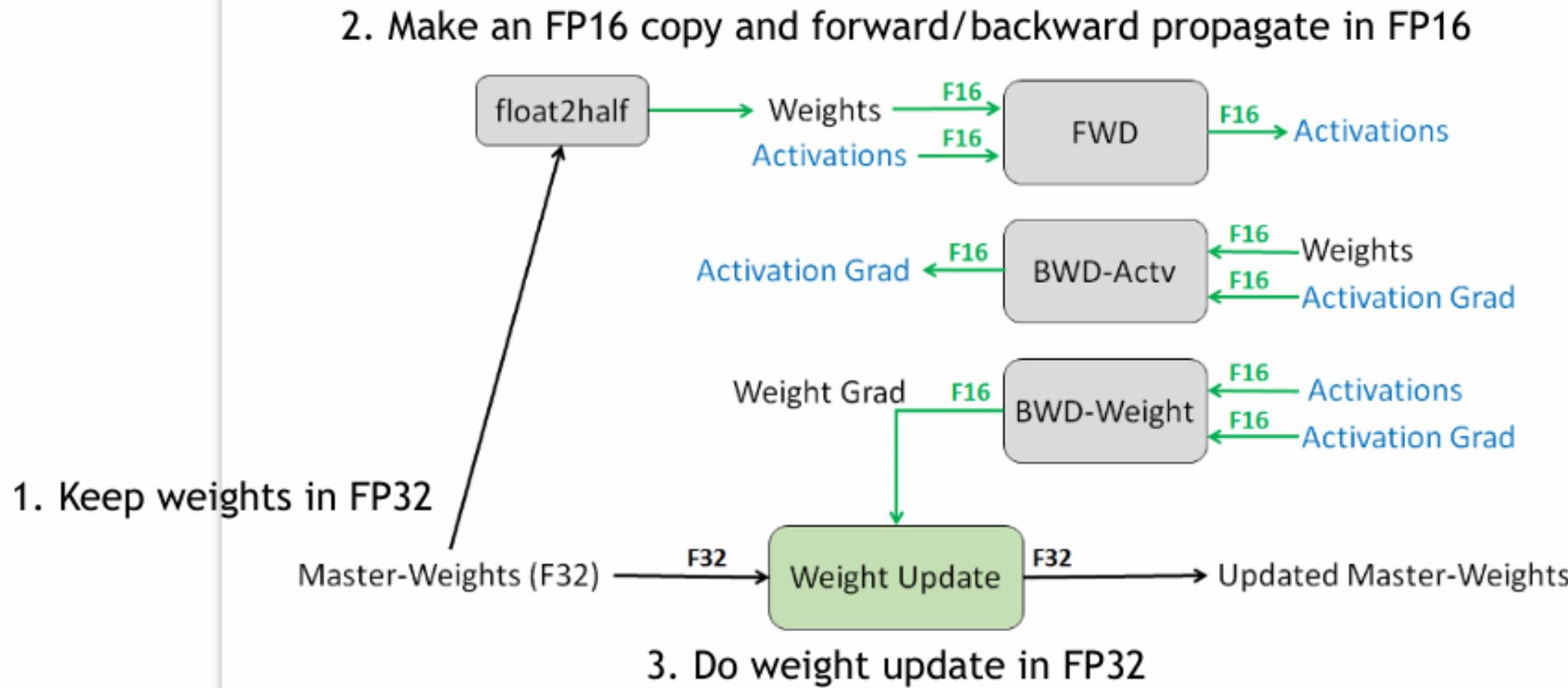
- **Problem:** in late stages of training, weight updates become too small for addition in FP16
- **Consequence:** weight update gets clipped to zero when $w \gg \alpha \nabla$

- **Problem:** in late stages of training, weight updates become too small for addition in FP16
- **Consequence:** weight update gets clipped to zero when $w \gg \alpha \nabla$



Example:
 $w \times 0.01$ leads to $\sim 2^7$ ratio,
 $w \times 0.001$ leads to $\sim 2^{10}$ ratio

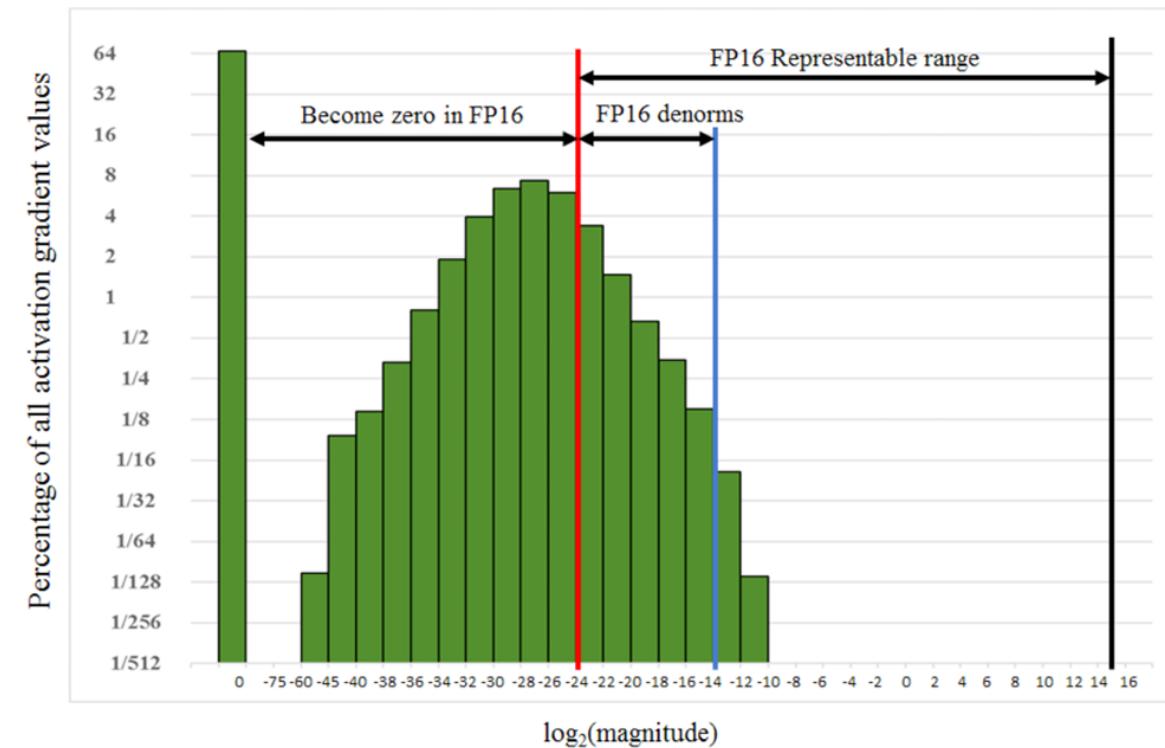
- **Conservative solution:** keep master copy of weights in FP32 so small updates can accumulate



Loss Scaling: Put All Tensors in FP16 Range



- Range representable in FP16: ~ 40 powers of 2

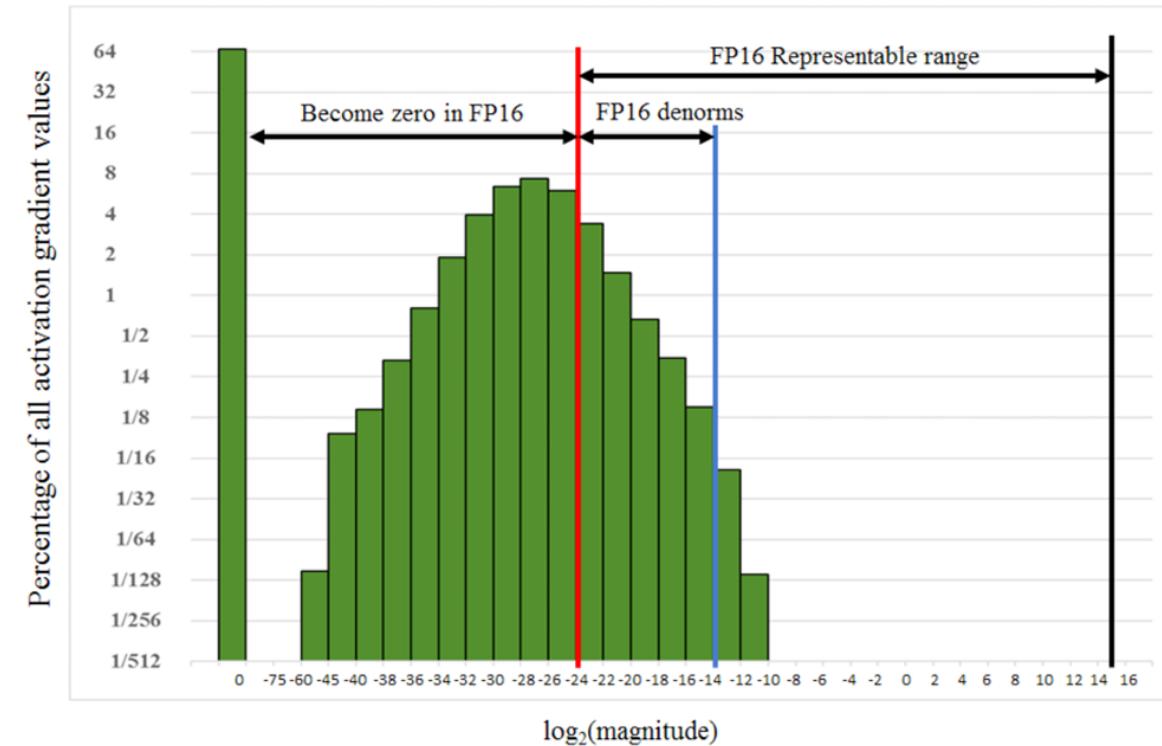


Histogram of activation gradient values

Loss Scaling: Put All Tensors in FP16 Range



- Range representable in FP16: ~ 40 powers of 2
- Gradients are small:
 - If cast to Fp16, some lost to zero
 - Much of fp16 representable ranges was left unused



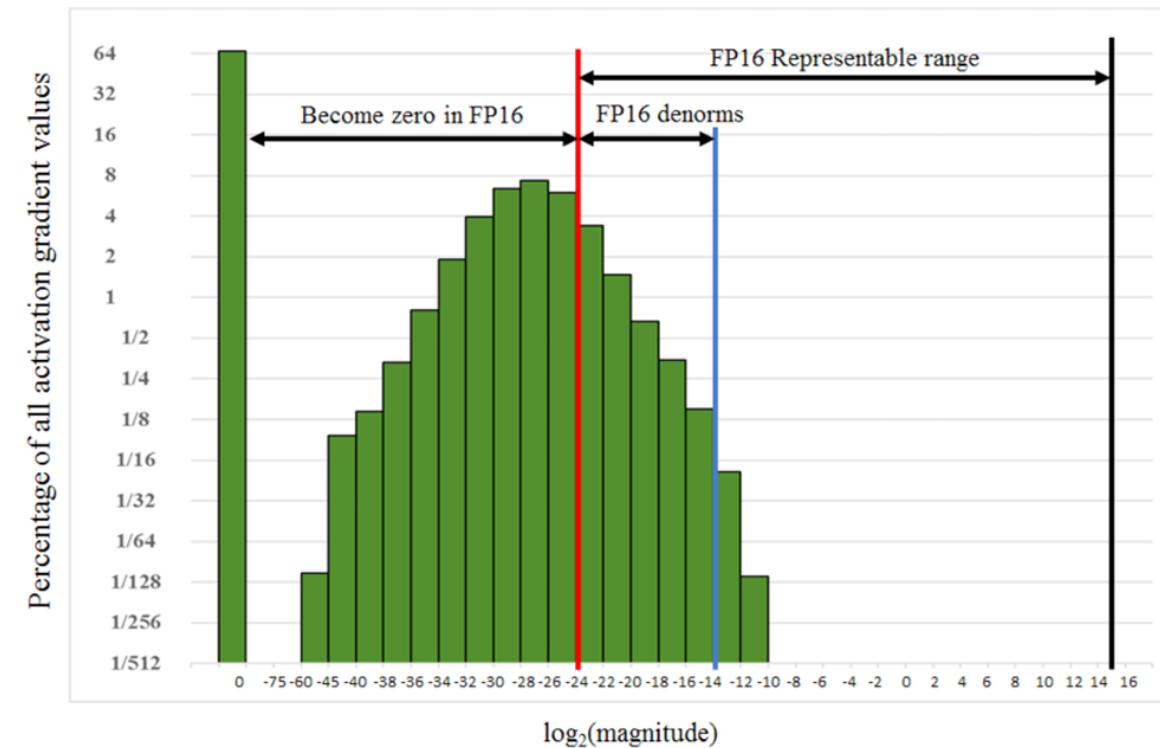
Histogram of activation gradient values

Loss Scaling: Put All Tensors in FP16 Range



- Range representable in FP16: ~ 40 powers of 2
- Gradients are small:
 - If cast to Fp16, some lost to zero
 - Much of fp16 representable ranges was left unused

Question: How can we make better use of FP16 representable range to capture gradients?

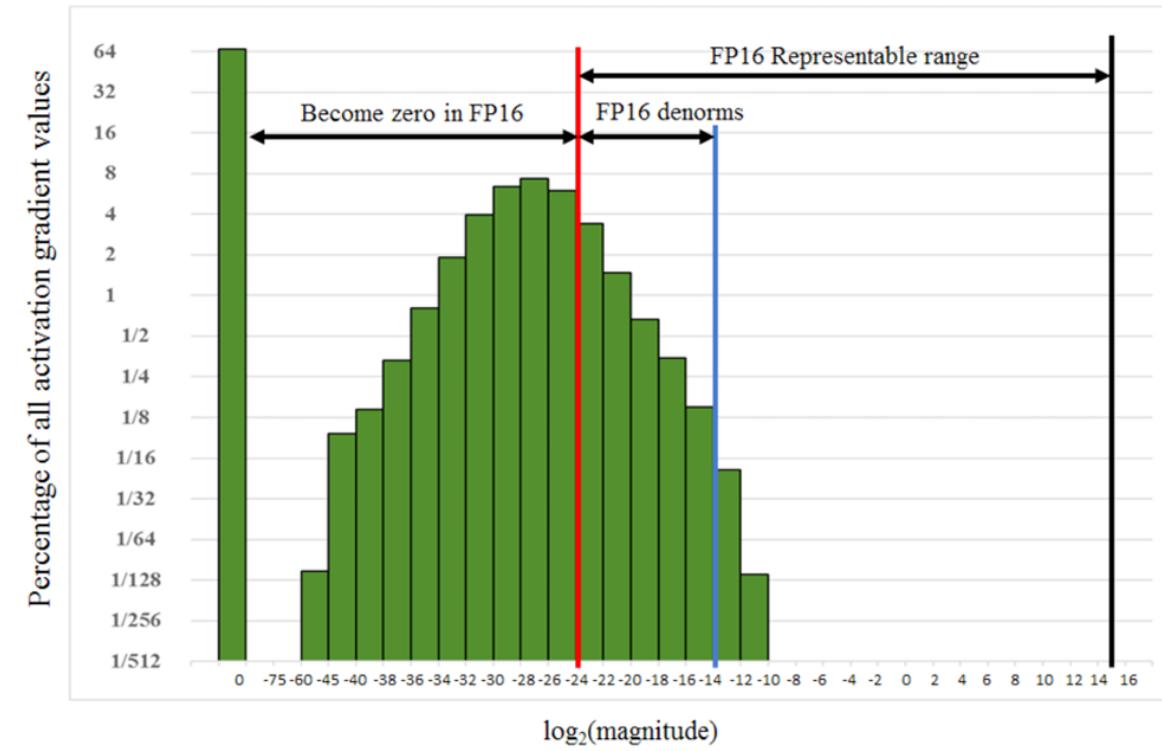


Histogram of activation gradient values

Loss Scaling: Put All Tensors in FP16 Range

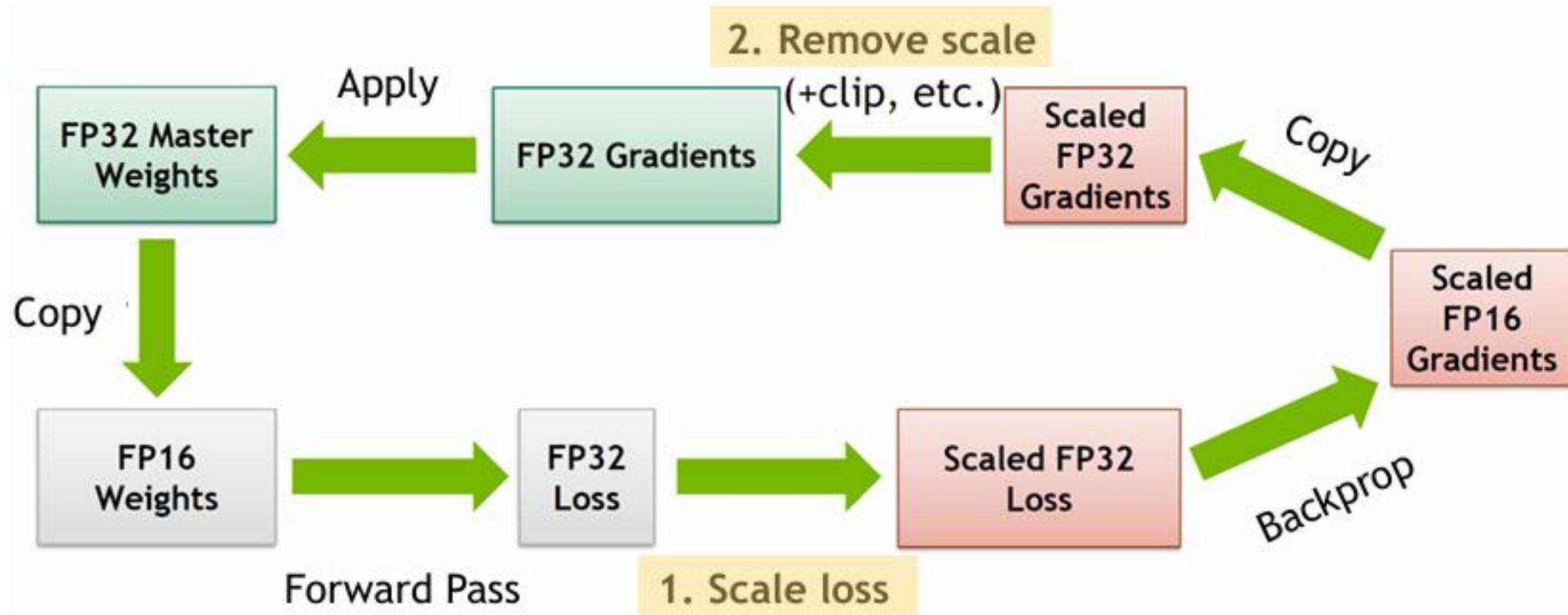


- Range representable in FP16: ~ 40 powers of 2
- Gradients are small:
 - If cast to Fp16, some lost to zero
 - Much of fp16 representable ranges was left unused
- Solution:
 - **Shift** small gradients values to fp16 range
 - Scaling gradients during backpropagation to prevent underflow
 - Gradients are scaled before backpropagation begins and rescaled before updating weights

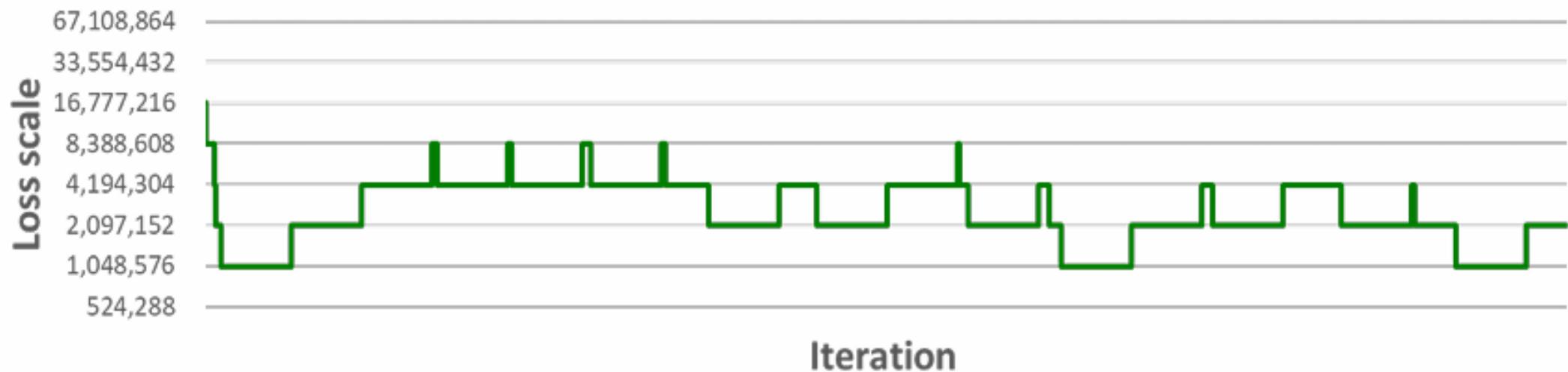


Histogram of activation gradient values

Loss Scaling: Algorithm



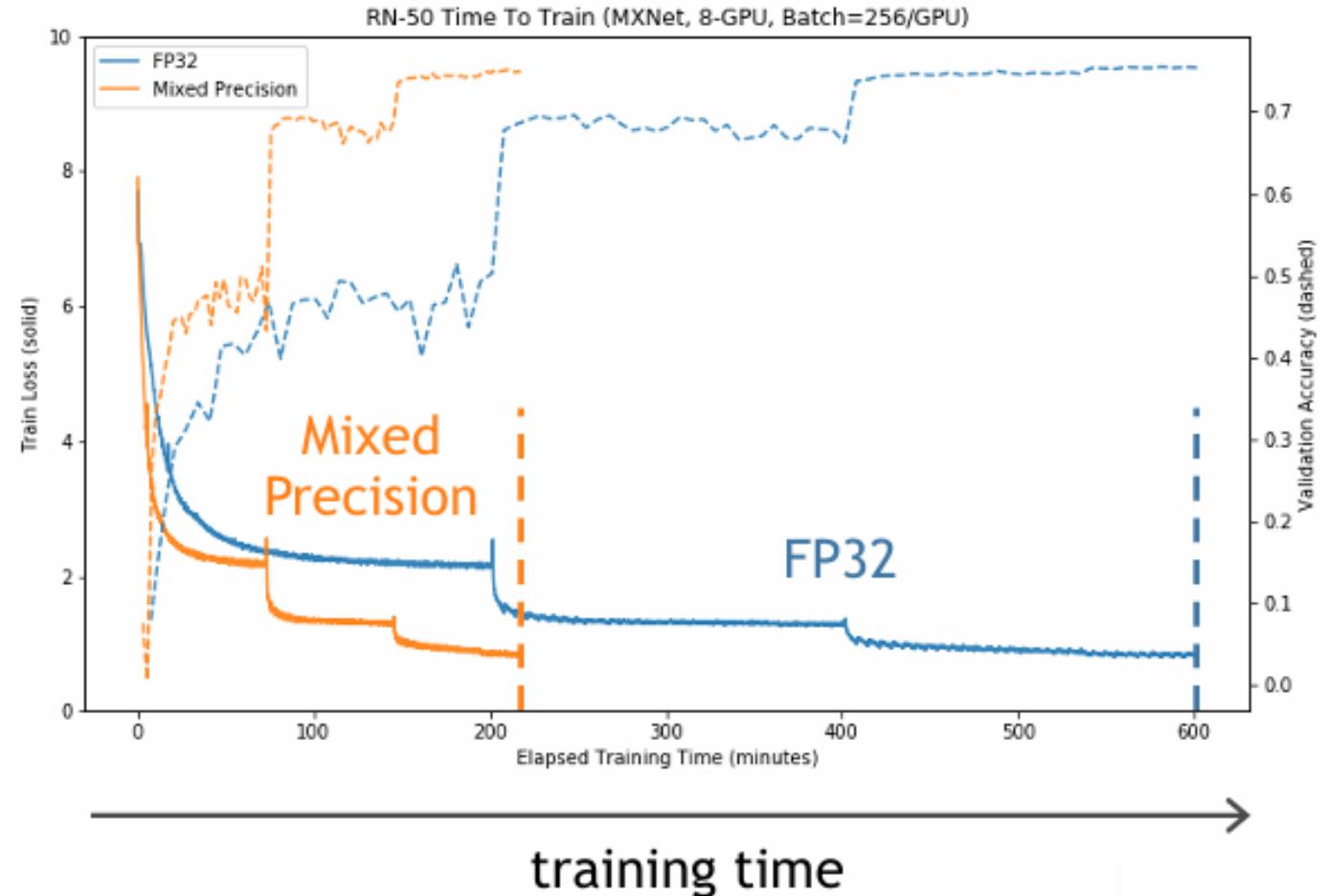
- 1. Start with very large scale factor (e.g., 2^{24})
- 2. If gradients overflows (with Inf or a Nan): decrease the scale by 2 and skip the update
- 3. If no overflows have occurred for some time: increase the scale by 2 (e.g., 2000 iterations)



Mixed Precision Training Example



- ResNet-50 training for ImageNet classification
 - 8 GPUs on DGX-1
- Comparing to FP32 training
 - **~3x speedup**
 - **Equal accuracy**
 - No hyperparameters changed



Mixed Precision is Generalizable



- Works across a wide range of tasks, problem domains, deep neural network architectures

Image Classification
AlexNet
DenseNet
Inception
MobileNet
NASNet
ResNet
ResNeXt
ShuffleNet
SqueezeNet
VGG
Xception

Detection / Segmentation
DeepLab
Faster R-CNN
Mask R-CNN
SSD
NVIDIA Automotive
RetinaNet
UNET

Recommendation
DeepRecommender
NCF

Generative Models (Images)
DLSS
GauGAN
Partial Image Inpainting
Progress GAN
Pix2Pix

Speech
Deep Speech 2
Jasper
Tacotron
Wave2vec
WaveNet
WaveGlow

Language Modeling
BERT
BigLSTM
Gated Convolutions
mLSTM
RoBERTa
Transformer XL

Translation
Convolutional Seq2Seq
Dynamic Convolutions
GNMT (RNN)
Levenshtein Transformer
Transformer (Self-Attention)

Mixed Precision Trains Faster: Drastically Reduces Training Time



Task	Model	Speedup
Image Classification	ResNet-50	3.6x
	DenseNet 201	2.2x*
	Xception	2.1x*
Detection / Segmentation	SSD	2.5x**
	Mask R-CNN	1.5x
	RetinaNet	2.0x

* 2x batch size

** Larger batch size

Weeks to days

Days to hours

Hours to minutes

Task	Model	Speedup
Translation	GNMT	2.3x
	Transformer	2.9x
	Convolutional Seq2Seq	4.9x**
Speech	Deep Speech 2	2.5x*
	Wav2letter	4.5x**
	WaveGlow	3.0x*
Language Modeling	mLSTM	1.9x
	BERT	4.0x**
		3.3x

Mixed Precision Software

- Available for major DL frameworks

PyTorch | DeepSpeed | Megatron | ...

- Works with all types of optimizers
 - SGD, Adam, etc
- Works with multiple models, optimizers, and losses

- Traditionally a lot of work (recap on methodology)
- Model conversion
 - Switch everything to run on FP16 values
 - Cast to FP32 for loss functions, normalization, and element-wise ops that need full precision
- Master weights
 - Keep FP32 copy of model parameters
 - Make FP16 copies during forward/backward passes
- Loss scaling
 - Scale the loss value, unscale the gradients in FP32
 - Check gradients at each iteration to adjust loss scale and skip on overflow

- Automates everything from the previous slides
- Framework software to run mixed precision automatically
- Details vary by framework, but core idea is the same
- Automatic Mixed Precision (AMP) does two things:

AUTOMATIC LOSS SCALING

Wraps the optimizer in order to:

- scale loss value & unscale gradients
- adjust scale & skip on gradient overflow

AUTOMATIC CASTING

Wraps model and operations in order to:

- cast data to FP16
- switch *everything* to run on FP16
- keep certain operations in FP32
- keep master copy of weights in FP32

- Make type decisions for each operation a prior (static graph) or at runtime (eager execution)
- Define conservative set of rules to replace “by-hand” mixed precision

Divide the universe of operations into three kinds:

WHITELIST

FP16 enables Tensor Cores.

e.g. linear, bmm, convs

Rule: always run in FP16,
cast if necessary.

BLACKLIST

FP32 is needed for accuracy.

e.g. loss, exp, sum, softmax

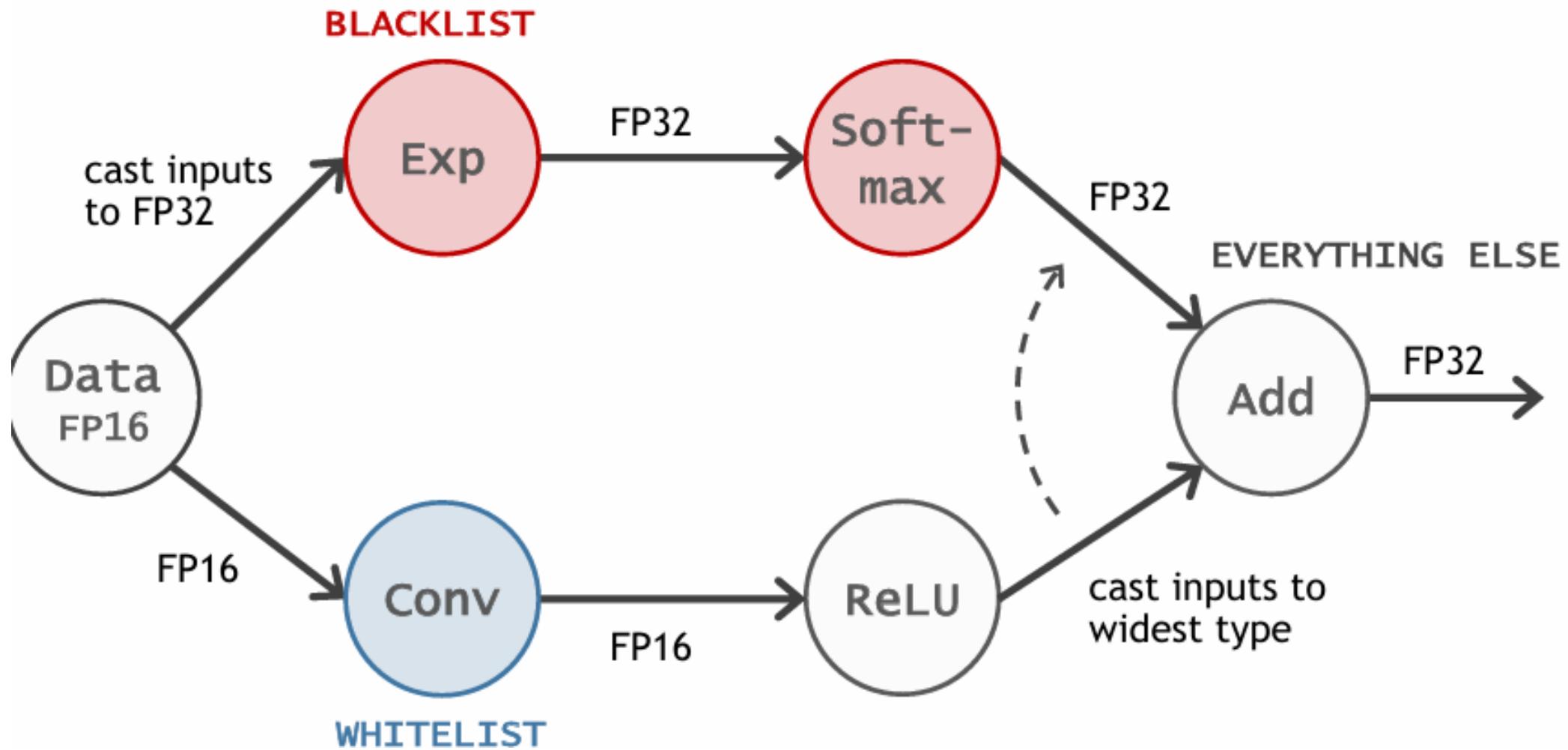
Rule: always run in FP32,
cast if necessary.

EVERYTHING ELSE

Can run in FP16, but only
if inputs already in FP16.

e.g. relu, add, maxpool

Rule: run in existing input
type.



To control the operations being casted

```
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
```

O0	O1	O2
FP32 Training Leave everything in FP32.	Mixed Precision Training FP16 whitelist and FP32 blacklist ops.	FP16 Training FP16 model/data with FP32 batchnorm.

“This paper shows that reduced mixed precision and large batch training can speedup training by nearly **5x** on a single 8-GPU machine with careful tuning and implementation.”

Scaling Neural Machine Translation, Facebook

“We train with mixed precision floating point arithmetic on DGX-1 machines [...] using 1024 V100 GPUs for approximately **one day**.”

RoBERTa, Facebook

“leverages mixed precision training [...] **largest transformer based language model ever trained** at 24x the size of BERT and 5.6x the size of GPT-2.”

MegatronLM, NVIDIA

- **Mixed precision improves efficiency but may hurt quality**
 - FP16 vs. INT8
- **Automatic Scaling at runtime may help quality**
 - E.g., scaling the gradient to prevent underflow

Course Project (Reproducibility Challenge)



4-credit undergraduate students, 3-credit graduate students

Some suggestions below, but it can be any machine learning system related papers that you are interested in

SonicMoE: Accelerating MoE with IO and Tile-aware Optimizations	https://github.com/Dao-AI-Lab/sonic-moe
Scaling Speculative Decoding with Lookahead Reasoning	https://github.com/hao-ai-lab/LookaheadReasoning
Taming the Long Tail: Efficient Reinforcement Learning for Language Models via Adaptive Speculative Decoding	https://github.com/mit-han-lab/fastrl
Fast-dLLM: Training-free Acceleration of Diffusion LLM by Enabling KV Cache and Parallel Decoding	https://github.com/NVlabs/Fast-dLLM
Kimi Linear: An Expressive, Efficient Attention Architecture	https://github.com/fla-org/flash-linear-attention
Mem0: The Memory Layer for Personalized AI	https://github.com/mem0ai/mem0
...	...

4-credit graduate students

- Benchmark and analyze important DL workloads to understand their performance gap and identify important angles to optimize their performance.
- Apply and evaluate how existing solutions work in the context of emerging AI/DL workloads.
- Design and implement new algorithms that are both theoretically and practically efficient.
- Design and implement system optimizations, e.g., parallelism, cache-locality, IO-efficiency, to improve the compute/memory/communication efficiency of AI/DL workloads.
- Offer customized optimization for critical DL workloads where latency is extremely tight.
- Build library/tool/framework to improve the efficiency of a class of problems.
- Integrate important optimizations into existing frameworks (e.g., DeepSpeed), providing fast and agile inference.
- Combine system optimization with modeling optimizations.
- Combine and leverage hardware resources (e.g., GPU/CPU, on-device memory/DRAM/NVMe/SSD) in a principled way.

Questions?