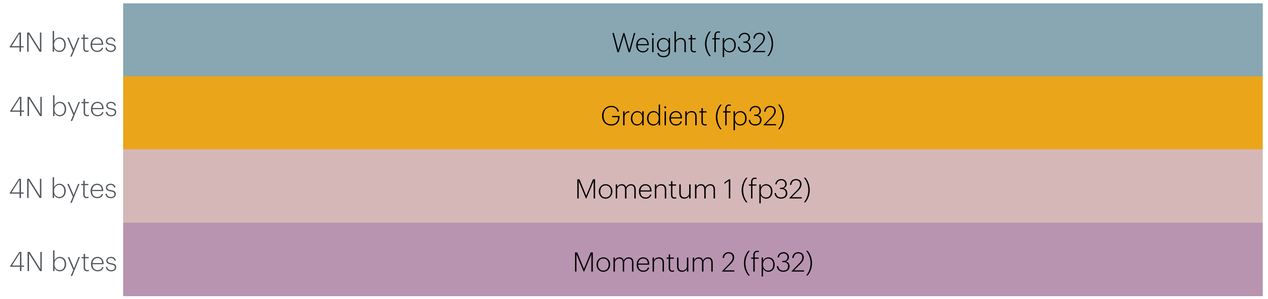
Mixed Precision Training

Philipp Krähenbühl, UT Austin

Training large models

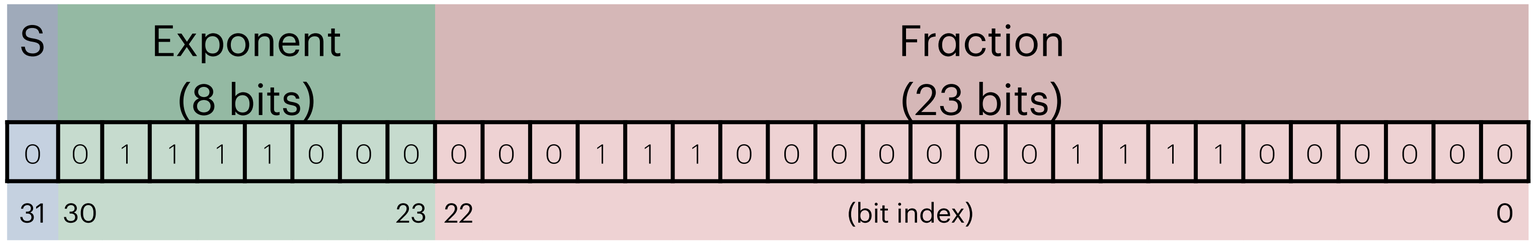
Memory requirements

- Without optimization
  - Model parameters: N
  - Weights: N floats
  - Gradients: N floats
  - Momentum: N floats
  - 2nd momentum (ADAM): N floats
- 16N bytes without counting activations



float32

float32	
Sign	1 bit
Exponent	8 bit
Mantissa	23 bit
Precision (relative)	1E-07
Max value	3E+38
Min value (normal)	1.7E-38



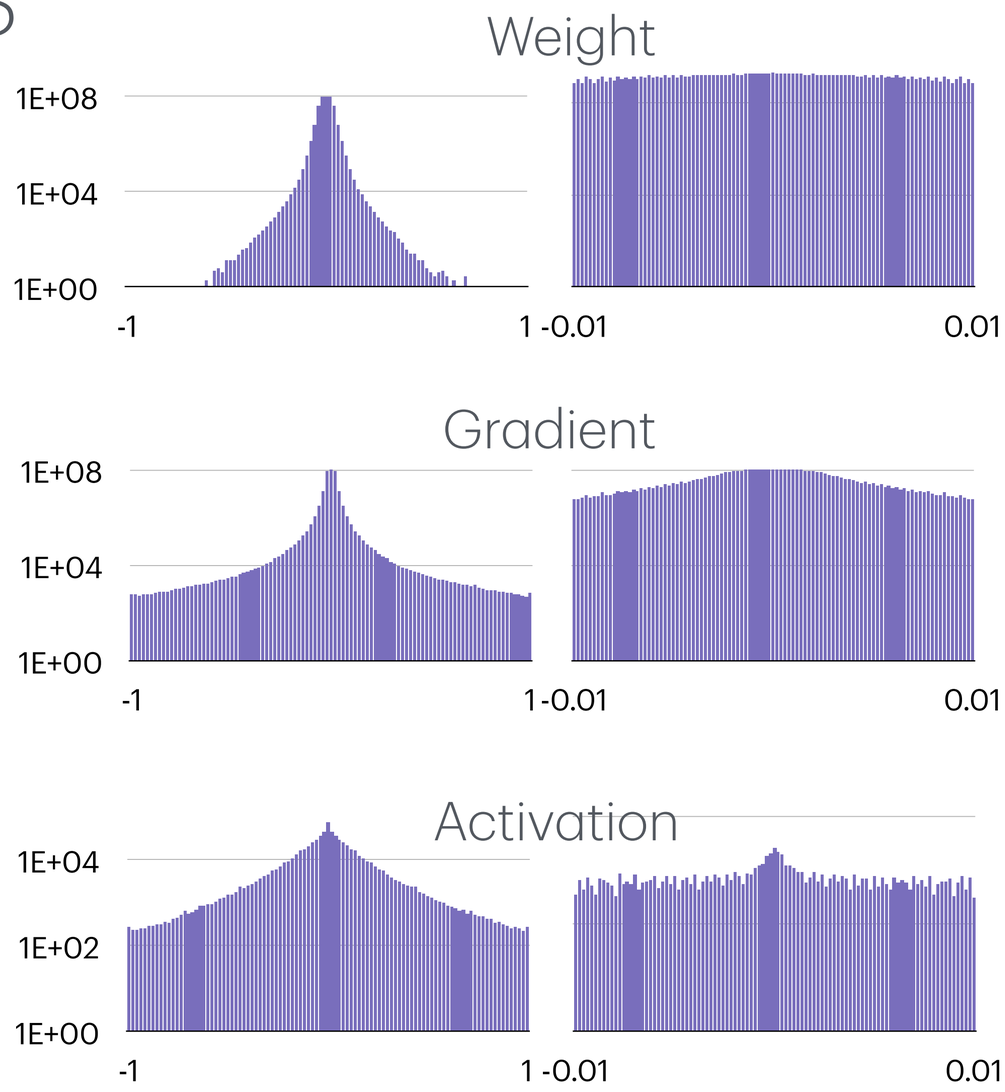
$$\text{Precision (relative)} = \frac{x_2 - x_1}{x_1}$$

where x_2 is smallest value $x_2 > x_1$

float32 vs deep networks

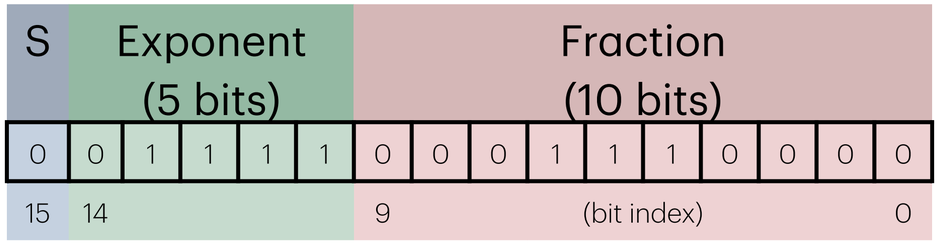
float32	
Sign	1 bit
Exponent	8 bit
Mantissa	23 bit
Precision (relative)	1E-07
Max value	3E+38
Min value (normal)	1.7E-38

Llama 3.1 8b	
Largest weight	3.21
Smallest weight	-0.83
Largest gradient	5.09
Smallest gradient	-4.41
Largest activation	227.42
Smallest activation	-206.84



Float16

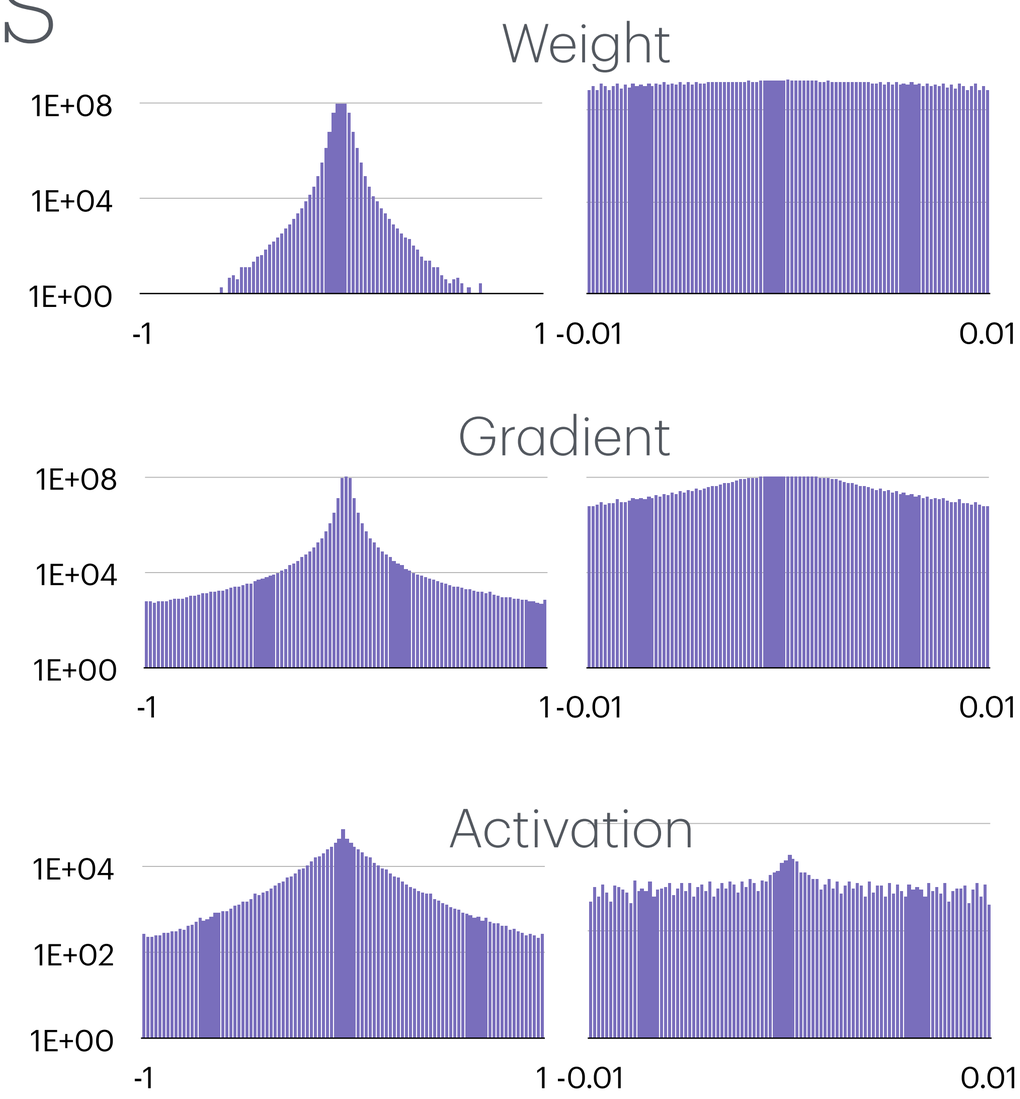
Float16	
Sign	1 bit
Exponent	5 bit
Mantissa	10 bit
Precision (relative)	1E-04
Max value	65504
Min value (normal)	6E-05



Float16 vs deep networks

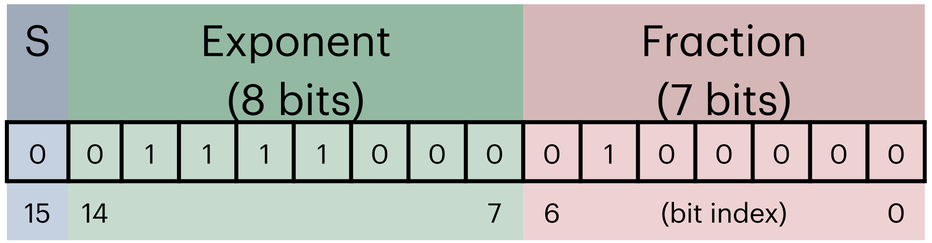
Float16	
Sign	1 bit
Exponent	5 bit
Mantissa	10 bit
Precision (relative)	1E-04
Max value	65504
Min value (normal)	6E-05

Llama 3.1 8b	
Largest weight	3.21
Smallest weight	-0.83
Largest gradient	5.09
Smallest gradient	-4.41
Largest activation	227.42
Smallest activation	-206.84



Bfloat16

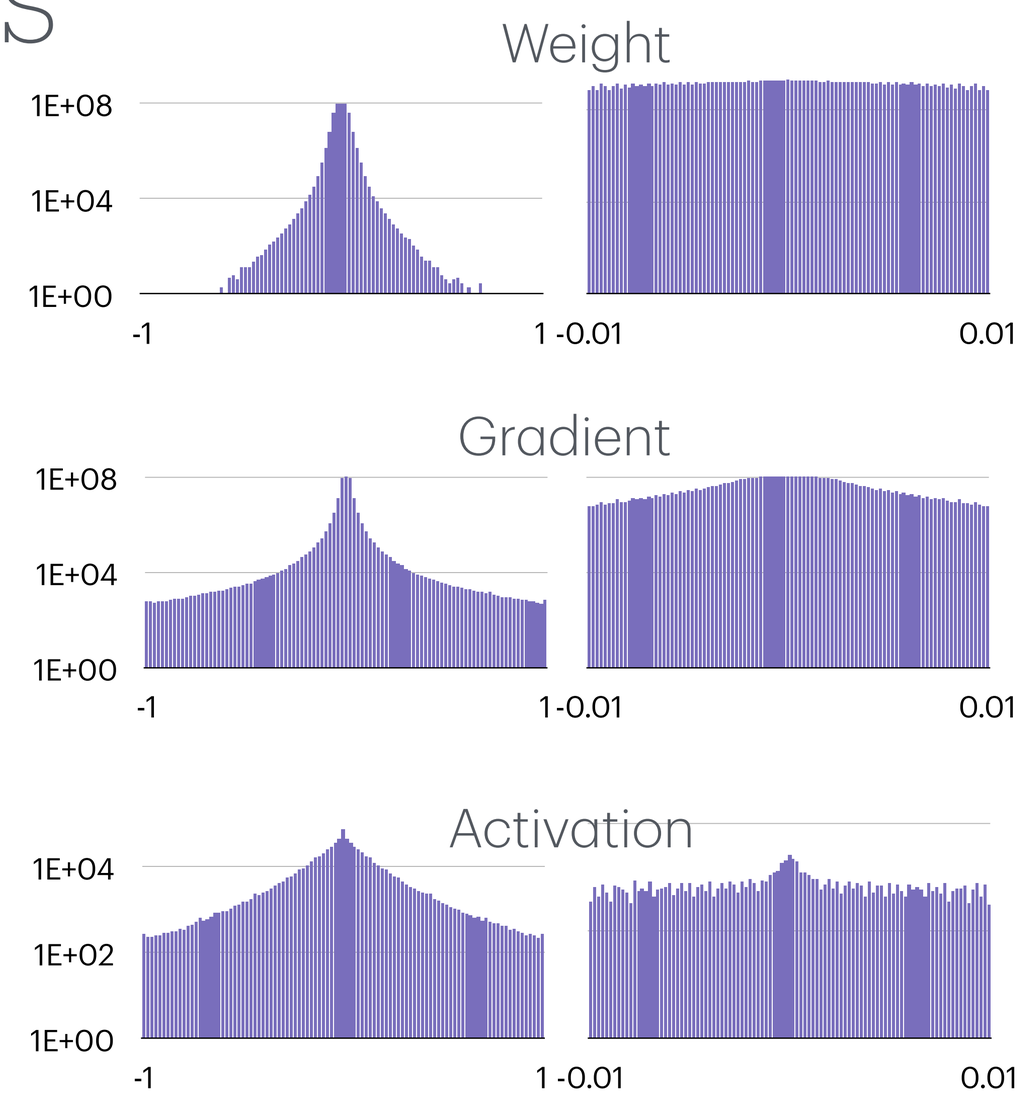
Bfloat16	
Sign	1 bit
Exponent	8 bit
Mantissa	7 bit
Precision (relative)	7.8E-03
Max value	3E+38
Min value (normal)	1.7E-38



Float16 vs deep networks

Bfloat16	
Sign	1 bit
Exponent	8 bit
Mantissa	7 bit
Precision (relative)	7.8E-03
Max value	3E+38
Min value (normal)	1.7E-38

Llama 3.1 8b	
Largest weight	3.21
Smallest weight	-0.83
Largest gradient	5.09
Smallest gradient	-4.41
Largest activation	227.42
Smallest activation	-206.84



Float32 vs (b)float16

- (B)float16 uses **half** the memory
- (B)float16 can be more than twice as fast
  - Why more than twice?

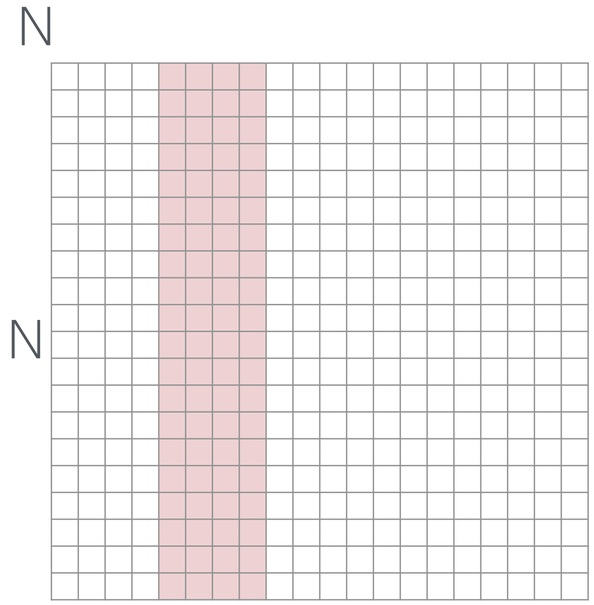
	V100 SXM	A100 SXM	H100 SXM
TF32 FLOPs Tensor Core	N/A	156T	494.7T
FP16 FLOPs Tensor Core	125T	312T	989.4T

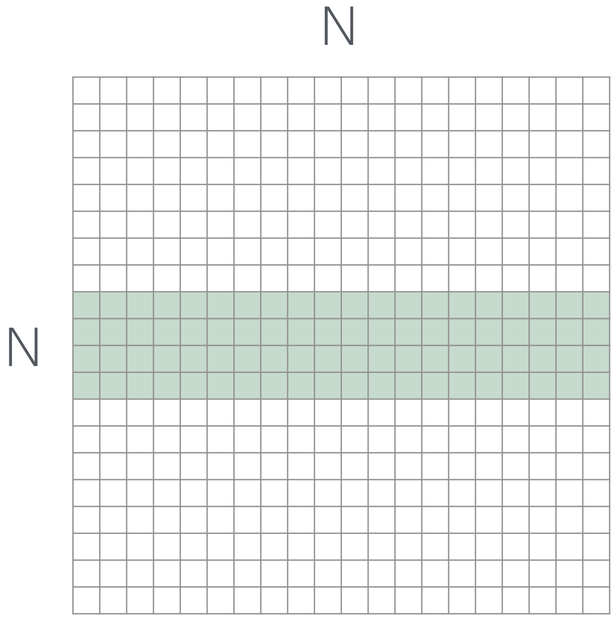
[1] NVIDIA, V100 Datasheet, 2020
[2] NVIDIA, A100 Datasheet, 2021
[3] NVIDIA, H100 Datasheet, 2023

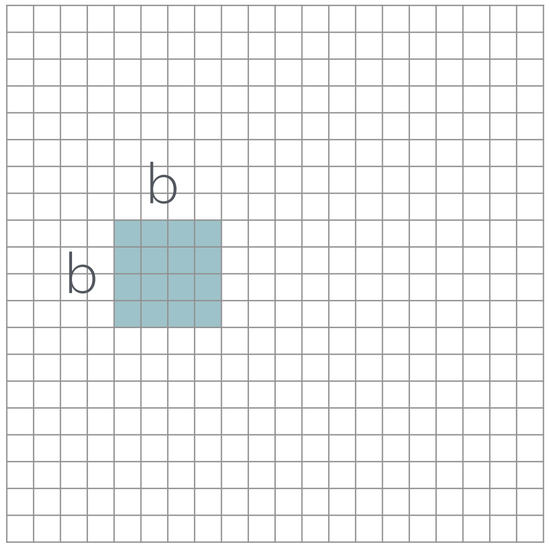
Computation in bfloat16

Case study: matmul

- $C = AB$
  - $C_{ij} = \sum_k A_{ik} B_{kj}$
- What is the bottleneck in modern GPUs?
  - Memory read and writes







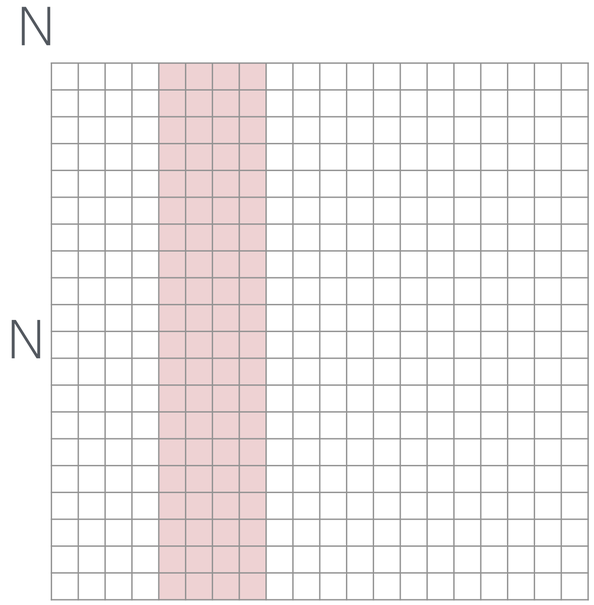
Computation in bfloat16

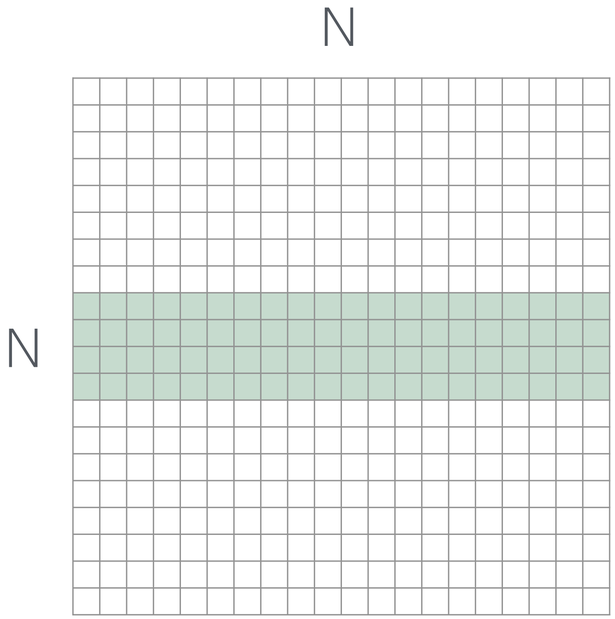
Case study: matmul

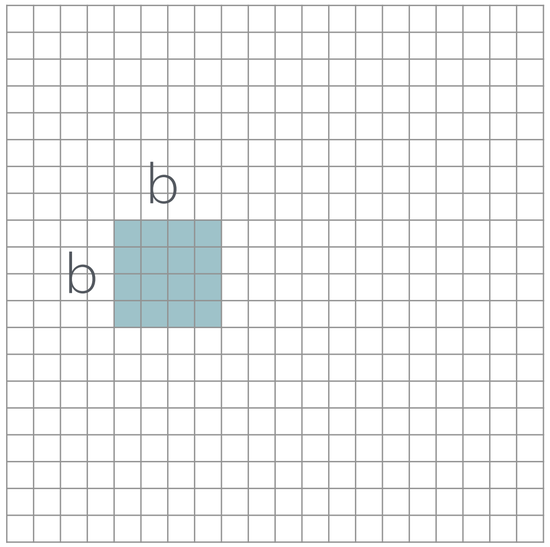
- Naive implementation

```
C_ij = 0

for k = 1 … N:

    C_ij += A_ik B_kj
```

- Number of memory reads $2N^3$







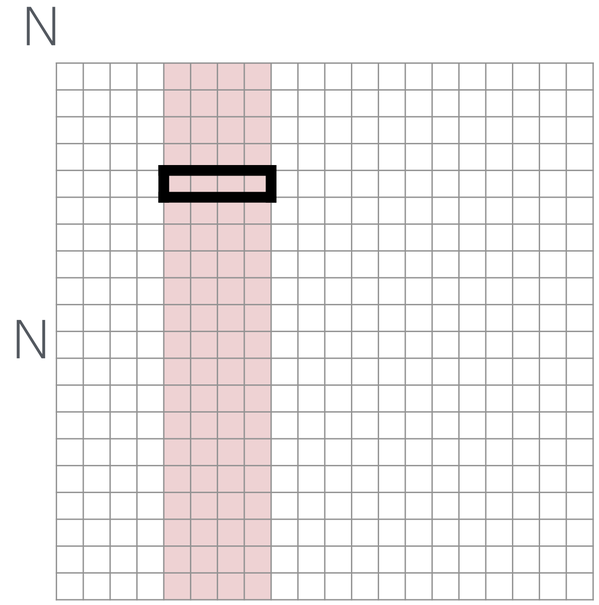
Computation in bfloat16

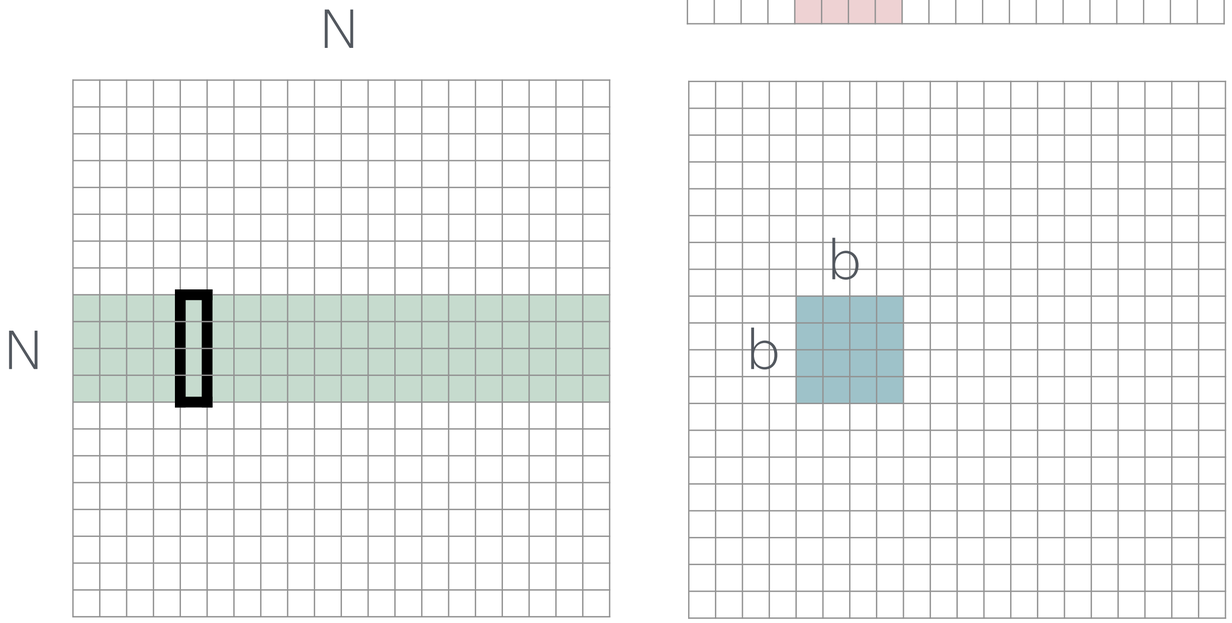
Case study: matmul

- Faster matmul

```
# for each bxb block
C_ij = 0
for k = 1 … N:
    # load all A_:k B_:j in
    # block into shared memory
    C_ij += A_ik B_kj
```

- Number of memory reads $\frac{2N^3}{b}$

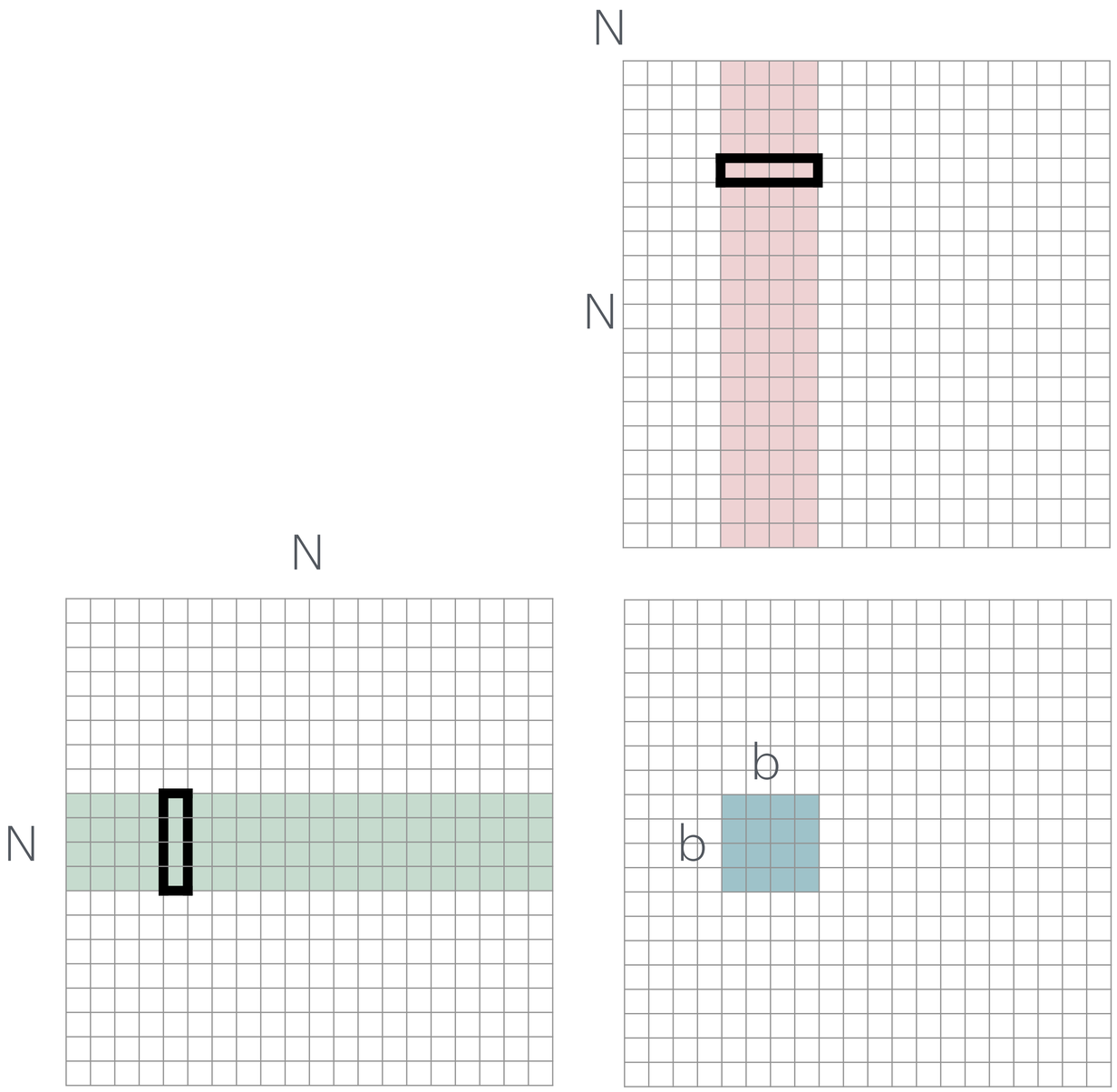




Why is bfloat16 faster

Case study: matmul

- Each read is half size
  - $2\times$ faster
- Block size b^2 can be $2\times$ larger
  - $\sqrt{2}\times$ faster
- In total $2.82\times$ faster



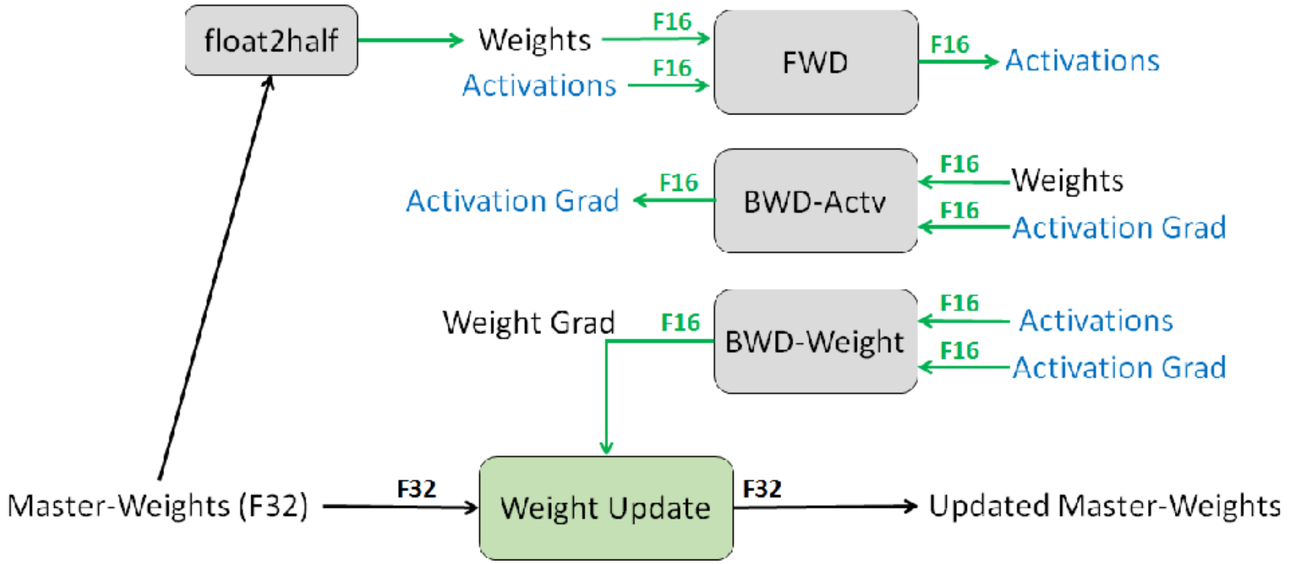
Float32 vs (b)float16

- (B)float16 uses **half** the memory
- (B)float16 can be more than twice as fast
  - Most operations are memory bound
  - Some operations have super-linear memory access patterns

	V100 SXM	A100 SXM	H100 SXM
TF32 FLOPs Tensor Core	N/A	156T	494.7T
FP16 FLOPs Tensor Core	125T	312T	989.4T

Training with BF16

- Original paper
  - Weights in fp32
  - Forward/backward in fp16/bf16
- Variants (llama3.1)
  - Weights in bf16
  - Gradient accumulation in bf16, but can be unstable

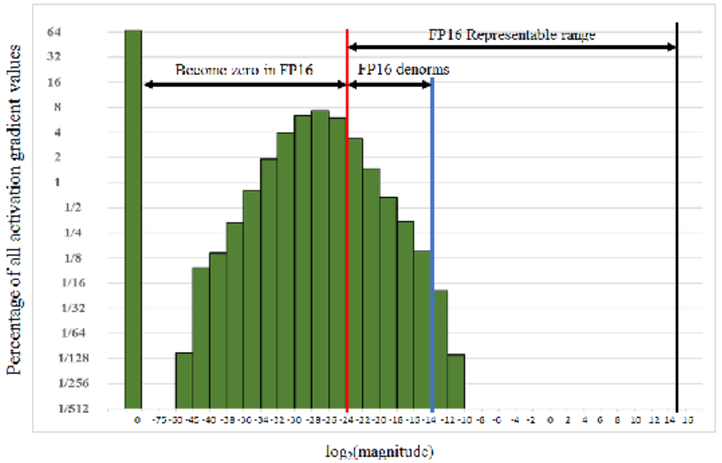



```
for epoch in range(...):
    for input, target in zip(data, targets):
        with torch.autocast(device_type=device, dtype=torch.bfloat16):
            output = net(input)
            loss = loss_fn(output, target)

        opt.zero_grad()
        loss.backward()
        opt.step()
```

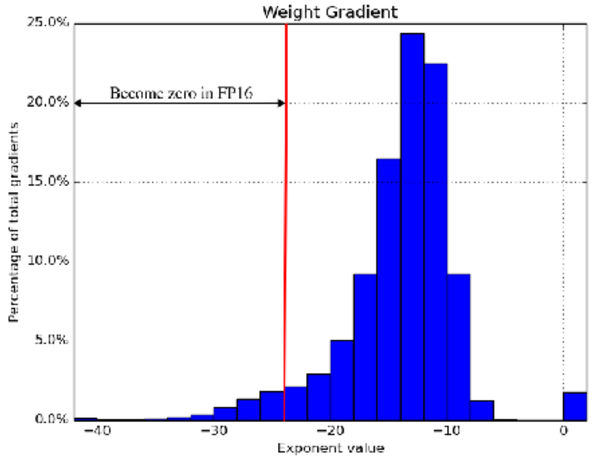

[1] Micikevicius, et al. Mixed Precision Training. 2017
[2] Llama Team, The Llama 3 Herd of Models. 2024

Training with BF16

- Issue: Loss of precision (underflows)
- Solution: Gradient Scaling
  - Multiple loss by a large value (i.e. 2^{16})
  - If overflow (inf or NaN)
    - ignore gradient (set to 0)
    - Lower gradient scale



Multibox SSD network



Mandarin speech recognition

```
scaler = torch.cuda.amp.GradScaler()

for epoch in range(...):
    for input, target in zip(data, targets):
        with torch.autocast(device_type=device, dtype=torch.bfloat16):
            output = net(input)
            loss = loss_fn(output, target)

        opt.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(opt)
        scaler.update()
```

[1] Micikevicius, et al. Mixed Precision Training. 2017

Training with BF16

- Issue: Loss of precision (underflows) in normalizations
- Solution: Keep normalization in float32
  - Use `torch.autocast`
  - Do not use `model.to(torch.float16)`

```
scaler = torch.cuda.amp.GradScaler()

for epoch in range(...):
    for input, target in zip(data, targets):
        with torch.autocast(device_type=device, dtype=torch.bfloat16):
            output = net(input)
            loss = loss_fn(output, target)

        opt.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(opt)
        scaler.update()
```

[1] Micikevicius, et al. Mixed Precision Training. 2017

Adam in (b)float16

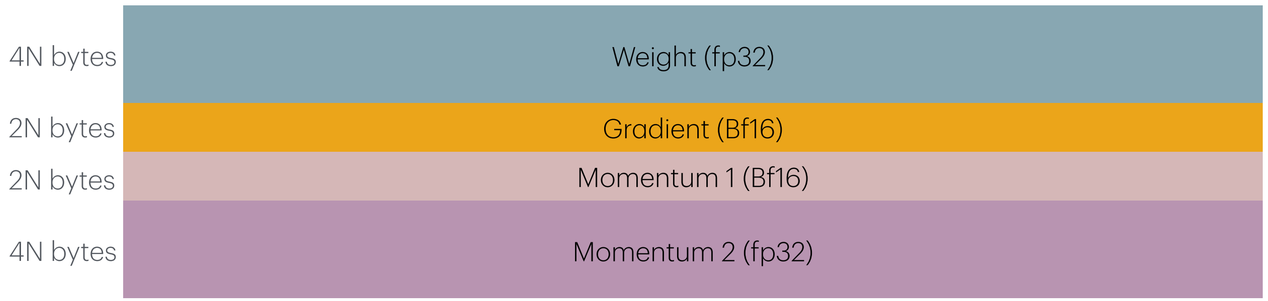
- First momentum: possible
- Second momentum: Unstable, Not recommended [1]
- Specialized very low-bit Adam implementations exist [2]

[1] Zhai, et al. Scaling Vision Transformers. 2021
[2] Tang, et al. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed, 2021

Training large models

Memory requirements

- Mixed precision
  - Model parameters: N
  - Weights: N floats
  - Gradients: N bfloat16
  - Momentum: N bfloat16
  - 2nd momentum (ADAM): N floats
- 12N bytes without counting activations



⚡ ⚡ Training is faster ⚡ ⚡

References

- [1] Micikevicius, et al. Mixed Precision Training. 2017 (link)
- [2] Llama Team, The Llama 3 Herd of Models. 2024 (link)
- [3] Zhai, et al. Scaling Vision Transformers. 2021 (link)
- [4] Tang, et al. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed, 2021 (link)