

Introduction to Large Language Models

Mixed Precision Training

Kun Yuan

Peking University

Full precision training and low precision training

- Large language model is difficult to train
 - take massive resource to **compute**
 - take massive resource to **store**
- Full precision training (e.g. FP32)
 - used in training most DNNs; very precise
 - takes a lot of computations and memories
- Low precision training (e.g. FP16)
 - able to train larger models due to computational and memory efficiency
 - not precise enough; overflow and underflow occur occasionally

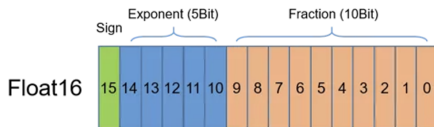
Full precision training and low precision training

- Large language model is difficult to train
 - take massive resource to **compute**
 - take massive resource to **store**
- Full precision training (e.g. FP32)
 - used in training most DNNs; very precise
 - takes a lot of computations and memories
- Low precision training (e.g. FP16)
 - able to train larger models due to computational and memory efficiency
 - not precise enough; overflow and underflow occur occasionally

Full precision training and low precision training

- Large language model is difficult to train
 - take massive resource to **compute**
 - take massive resource to **store**
- Full precision training (e.g. FP32)
 - used in training most DNNs; very precise
 - takes a lot of computations and memories
- Low precision training (e.g. FP16)
 - able to train larger models due to computational and memory efficiency
 - not precise enough; overflow and underflow occur occasionally

Float 16 (FP16)



- **Sign:** 1 bit; 0 for positive and 1 for negative
- **Exponent:** 5 bits; range: 00001(1)-11110(30); value range: $2^{-14} \sim 2^{15}$

Example: 00111(7) $\longrightarrow 2^{7-15} = 2^{-8}$

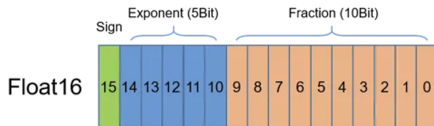
where -15 is the offset

- **Fraction:** 10 bits;

Example: 100100000 $\longrightarrow 1.100100000$
 $\longrightarrow 1 + 576/1024 = 1.5625$

where binary 100100000 translates into decimal 576

Float 16 (FP16)



- **Sign:** 1 bit; 0 for positive and 1 for negative
- **Exponent:** 5 bits; range: 00001(1)-11110(30); value range: $2^{-14} \sim 2^{15}$

Example: 00111(7) $\longrightarrow 2^{7-15} = 2^{-8}$

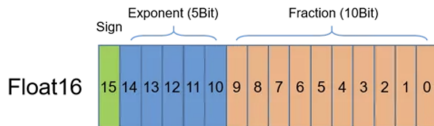
where -15 is the offset

- **Fraction:** 10 bits;

Example: 100100000 $\longrightarrow 1.100100000$
 $\longrightarrow 1 + 576/1024 = 1.5625$

where binary 100100000 translates into decimal 576

Float 16 (FP16)



- **Sign:** 1 bit; 0 for positive and 1 for negative
- **Exponent:** 5 bits; range: 00001(1)-11110(30); value range: $2^{-14} \sim 2^{15}$

Example: 00111(7) $\longrightarrow 2^{7-15} = 2^{-8}$

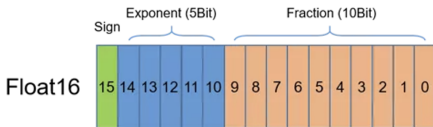
where -15 is the offset

- **Fraction:** 10 bits;

Example: 1001000000 $\longrightarrow 1.1001000000$
 $\longrightarrow 1 + 576/1024 = 1.5625$

where binary 1001000000 translates into decimal 576

Float 16 (FP16)



- **Translation law**

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-15} \times \left(1 + \frac{\text{fraction}}{1024}\right)$$

- **Largest positive number:**

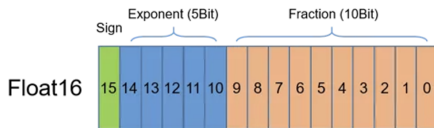
$$(-1)^0 \times 2^{15} \times \left(1 + \frac{1023}{1024}\right) = 65504$$

The range of FP16 is $[-65504, +65504]$.

- **Smallest positive number:**

$$(-1)^0 \times 2^{-14} \times \left(0 + \frac{1}{1024}\right) \approx 2^{-24}$$

Float 16 (FP16)



- **Translation law**

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-15} \times \left(1 + \frac{\text{fraction}}{1024}\right)$$

- **Largest positive number:**

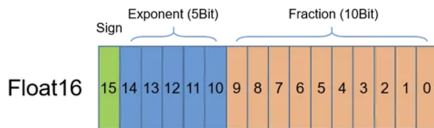
$$(-1)^0 \times 2^{15} \times \left(1 + \frac{1023}{1024}\right) = 65504$$

The range of FP16 is $[-65504, +65504]$.

- **Smallest positive number:**

$$(-1)^0 \times 2^{-14} \times \left(0 + \frac{1}{1024}\right) \approx 2^{-24}$$

Float 16 (FP16)



- **Translation law**

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-15} \times \left(1 + \frac{\text{fraction}}{1024}\right)$$

- **Largest positive number:**

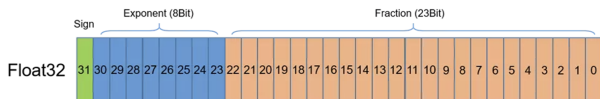
$$(-1)^0 \times 2^{15} \times \left(1 + \frac{1023}{1024}\right) = 65504$$

The range of FP16 is $[-65504, +65504]$.

- **Smallest positive number:**

$$(-1)^0 \times 2^{-14} \times \left(0 + \frac{1}{1024}\right) \approx 2^{-24}$$

Float 32 (FP32)

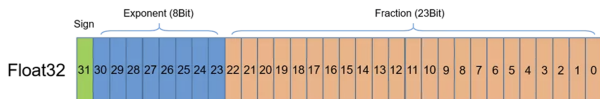


- **Translation law**

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times \left(1 + \frac{\text{fraction}}{2^{23}}\right)$$

- **Range:** $[-3.40282 \times 10^{38}, +3.40282 \times 10^{38}]$
- **Smallest positive number:** 1.17549×10^{-38}
- FP 32 is much more powerful than FP 16; but takes too much memory

Float 32 (FP32)



- **Translation law**

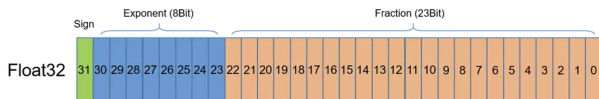
$$(-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times \left(1 + \frac{\text{fraction}}{2^{23}}\right)$$

- **Range:** $[-3.40282 \times 10^{38}, +3.40282 \times 10^{38}]$

- **Smallest positive number:** 1.17549×10^{-38}

- FP 32 is much more powerful than FP 16; but takes too much memory

Float 32 (FP32)



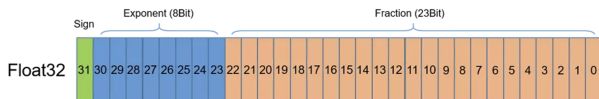
- **Translation law**

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times \left(1 + \frac{\text{fraction}}{2^{23}}\right)$$

- **Range:** $[-3.40282 \times 10^{38}, +3.40282 \times 10^{38}]$
- **Smallest positive number:** 1.17549×10^{-38}

- FP 32 is much more powerful than FP 16; but takes too much memory

Float 32 (FP32)



- **Translation law**

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times \left(1 + \frac{\text{fraction}}{2^{23}}\right)$$

- **Range:** $[-3.40282 \times 10^{38}, +3.40282 \times 10^{38}]$
- **Smallest positive number:** 1.17549×10^{-38}
- FP 32 is much more powerful than FP 16; but takes too much memory

Mixed precision training

- When both FP32 and FP16 are used in training, we get **Mixed precision training** (Micikevicius et al., 2017)
- Save memory and computations without hurting performance
- Three key techniques:
 - FP32 weight copies
 - Loss scaling
 - Arithmetic precision

Mixed precision training

- When both FP32 and FP16 are used in training, we get **Mixed precision training** (Micikevicius et al., 2017)
- Save memory and computations without hurting performance
- Three key techniques:
 - FP32 weight copies
 - Loss scaling
 - Arithmetic precision

Mixed precision training

- When both FP32 and FP16 are used in training, we get **Mixed precision training** (Micikevicius et al., 2017)
- Save memory and computations without hurting performance
- Three key techniques:
 - FP32 weight copies
 - Loss scaling
 - Arithmetic precision

FP32 weight copies

- An FP32 weight copy is maintained and updated with gradient

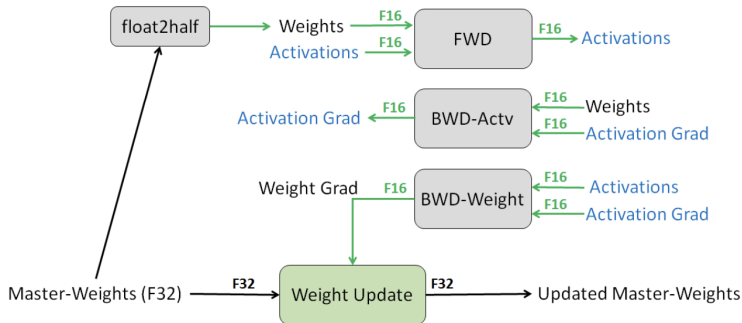


Figure 1: Mixed precision training iteration for a layer.

FP32 weight copies

- Reason I: **maintain small values in the weight update**
- Weight update = learning rate \times gradient; typically very small in late phase
- Values less than $2^{-24} \approx 5.96 \times 10^{-8}$ become 0 when using FP16
- About 5% values are less than 2^{-24}

FP32 weight copies

- Reason I: **maintain small values in the weight update**
- Weight update = learning rate \times gradient; typically very small in late phase
- Values less than $2^{-24} \approx 5.96 \times 10^{-8}$ become 0 when using FP16
- About 5% values are less than 2^{-24}

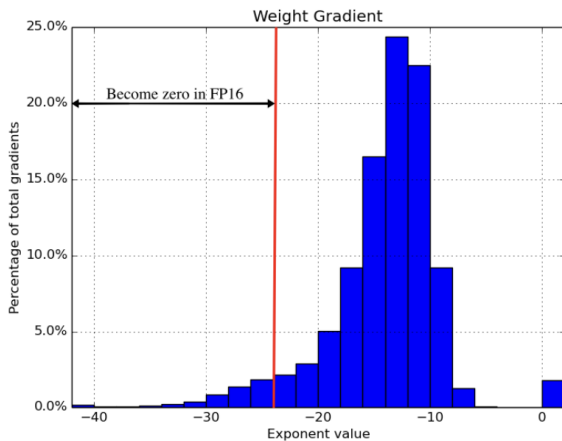
FP32 weight copies

- Reason I: **maintain small values in the weight update**
- Weight update = learning rate \times gradient; typically very small in late phase
- Values less than $2^{-24} \approx 5.96 \times 10^{-8}$ become 0 when using FP16
- About 5% values are less than 2^{-24}

FP32 weight copies

- Reason I: **maintain small values in the weight update**
- Weight update = learning rate \times gradient; typically very small in late phase
- Values less than $2^{-24} \approx 5.96 \times 10^{-8}$ become 0 when using FP16
- About 5% values are less than 2^{-24}

FP32 weight copies



FP32 weight copies

- Reason II: **big value-to-update ratio**

¹This figure is from wikipedia

FP32 weight copies

- Reason II: **big value-to-update ratio**
- The resolution in each period is shown as follows¹

Min	Max	interval
0	2^{-13}	2^{-24}
2^{-13}	2^{-12}	2^{-23}
2^{-12}	2^{-11}	2^{-22}
2^{-11}	2^{-10}	2^{-21}
2^{-10}	2^{-9}	2^{-20}
2^{-9}	2^{-8}	2^{-19}
2^{-8}	2^{-7}	2^{-18}
2^{-7}	2^{-6}	2^{-17}
2^{-6}	2^{-5}	2^{-16}
2^{-5}	2^{-4}	2^{-15}
2^{-4}	$\frac{1}{8}$	2^{-14}

$\frac{1}{8}$	$\frac{1}{4}$	2^{-13}
$\frac{1}{4}$	$\frac{1}{2}$	2^{-12}
$\frac{1}{2}$	1	2^{-11}
1	2	2^{-10}
2	4	2^{-9}
4	8	2^{-8}
8	16	2^{-7}
16	32	2^{-6}
32	64	2^{-5}
64	128	2^{-4}
128	256	$\frac{1}{8}$
256	512	$\frac{1}{4}$

512	1024	$\frac{1}{2}$
1024	2048	1
2048	4096	2
4096	8192	4
8192	16384	8
16384	32768	16
32768	65519	32
65519	∞	∞

¹This figure is from wikipedia

FP32 weight copies

- If the value-to-update ratio is bigger than $2^{11} = 2048$, it holds that

$$\text{value} + \text{update} = \text{value}$$

- The update has no influence on the value
- For reasons I and II, we maintain FP32 copies for both the weight and weight decay

FP32 weight copies

- If the value-to-update ratio is bigger than $2^{11} = 2048$, it holds that

$$\text{value} + \text{update} = \text{value}$$

- The update has no influence on the value
- For reasons I and II, we maintain FP32 copies for both the weight and weight decay

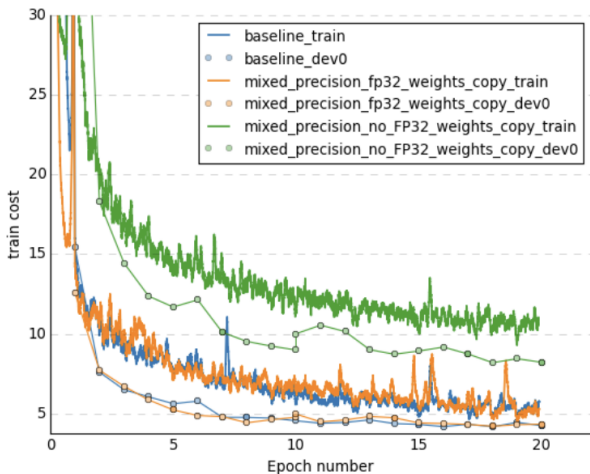
FP32 weight copies

- If the value-to-update ratio is bigger than $2^{11} = 2048$, it holds that

$$\text{value} + \text{update} = \text{value}$$

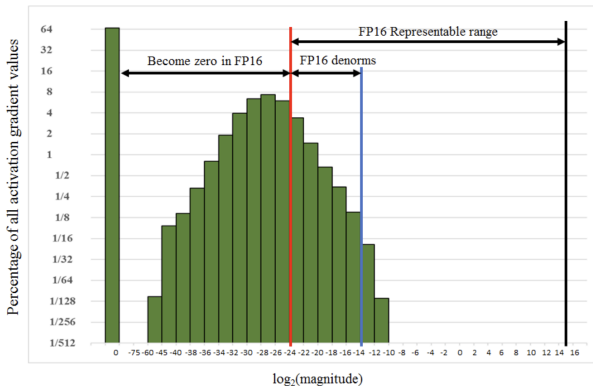
- The update has no influence on the value
- For reasons I and II, we maintain FP32 copies for both the weight and weight decay

FP32 weight copies



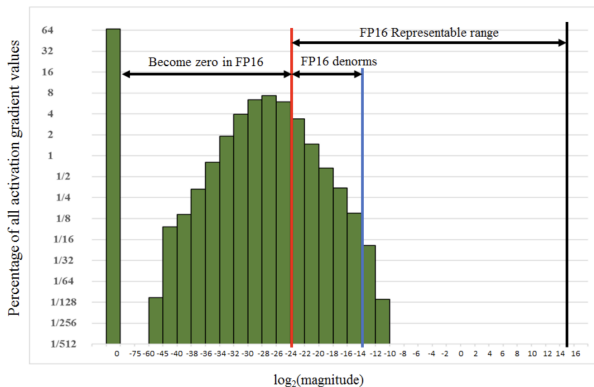
Loss scaling

- FP16 representation range $[2^{-24}, 2^{15}]$
- The activation gradient is typically very small; most values are smaller than 2^{-24}



Loss scaling

- FP16 representation range $[2^{-24}, 2^{15}]$
- The activation gradient is typically very small; most values are smaller than 2^{-24}



Loss scaling

- Scale-up the loss value before the back-propagation
- Unscale the gradient after back-propagation but before the update

$$g = \frac{\partial L}{\partial x} = \frac{1}{c} \frac{\partial (c \cdot L)}{\partial x}$$

- Effectively shift the graient value to the FP representation range
- Tricky to choose the scale-up coefficient
- $c = 8$ typically works

Loss scaling

- Scale-up the loss value before the back-propagation
- Unscale the gradient after back-propagation but before the update

$$g = \frac{\partial L}{\partial x} = \frac{1}{c} \frac{\partial (c \cdot L)}{\partial x}$$

- Effectively shift the gradient value to the FP representation range
- Tricky to choose the scale-up coefficient
- $c = 8$ typically works

Loss scaling

- Scale-up the loss value before the back-propagation
- Unscale the gradient after back-propagation but before the update

$$g = \frac{\partial L}{\partial x} = \frac{1}{c} \frac{\partial (c \cdot L)}{\partial x}$$

- Effectively shift the graient value to the FP representation range
- Tricky to choose the scale-up coefficient
- $c = 8$ typically works

Loss scaling

- Scale-up the loss value before the back-propagation
- Unscale the gradient after back-propagation but before the update

$$g = \frac{\partial L}{\partial x} = \frac{1}{c} \frac{\partial (c \cdot L)}{\partial x}$$

- Effectively shift the gradient value to the FP representation range
- Tricky to choose the scale-up coefficient
- $c = 8$ typically works

Loss scaling

- Scale-up the loss value before the back-propagation
- Unscale the gradient after back-propagation but before the update

$$g = \frac{\partial L}{\partial x} = \frac{1}{c} \frac{\partial (c \cdot L)}{\partial x}$$

- Effectively shift the gradient value to the FP representation range
- Tricky to choose the scale-up coefficient
- $c = 8$ typically works

Loss scaling

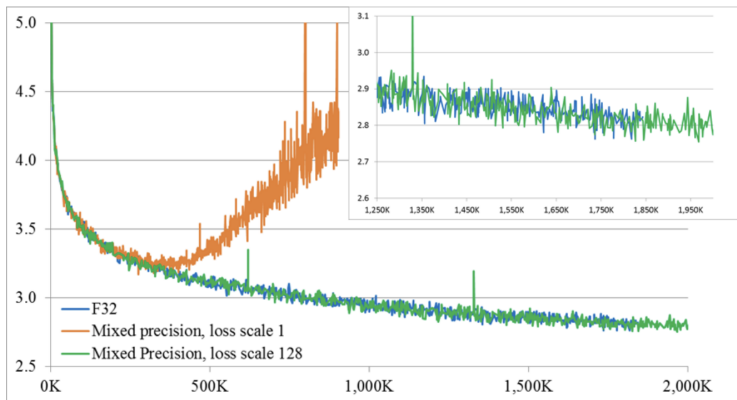


Figure 5: bigLSTM training perplexity

Arithmetic precision

- Not as important as the above two techniques
- Three key computation steps: vector dot-products; reductions; point-wise operations
- It is suggested in (Micikevicius et al., 2017) that vector dot-products and reductions are read and written in FP16 but carried out in FP32
- Point-wise operations can be carried in FP16

Arithmetic precision

- Not as important as the above two techniques
- Three key computation steps: vector dot-products; reductions; point-wise operations
- It is suggested in (Micikevicius et al., 2017) that vector dot-products and reductions are read and written in FP16 but carried out in FP32
- Point-wise operations can be carried in FP16

Arithmetic precision

- Not as important as the above two techniques
- Three key computation steps: vector dot-products; reductions; point-wise operations
- It is suggested in (Micikevicius et al., 2017) that vector dot-products and reductions are read and written in FP16 but carried out in FP32
- Point-wise operations can be carried in FP16

Arithmetic precision

- Not as important as the above two techniques
- Three key computation steps: vector dot-products; reductions; point-wise operations
- It is suggested in (Micikevicius et al., 2017) that vector dot-products and reductions are read and written in FP16 but carried out in FP32
- Point-wise operations can be carried in FP16

Table 1: ILSVRC12 classification top-1 accuracy.

Model	Baseline	Mixed Precision	Reference
AlexNet	56.77%	56.93%	(Krizhevsky et al., 2012)
VGG-D	65.40%	65.43%	(Simonyan and Zisserman, 2014)
GoogLeNet (Inception v1)	68.33%	68.43%	(Szegedy et al., 2015)
Inception v2	70.03%	70.02%	(Ioffe and Szegedy, 2015)
Inception v3	73.85%	74.13%	(Szegedy et al., 2016)
Resnet50	75.92%	76.04%	(He et al., 2016b)

Table 2: Detection network average mean precision.

Model	Baseline	MP without loss-scale	MP with loss-scale
Faster R-CNN	69.1%	68.6%	69.7%
Multibox SSD	76.9%	diverges	77.1%

AMP FOR PYTORCH

As simple as two lines of code

Wrap the model and optimizer

```
model, optimizer = amp.initialize(model, optimizer)
```

Apply automatic loss scaling and backpropagate with scaled loss

```
with amp.scaled_loss(loss, optimizer) as scaled_loss:  
    scaled_loss.backward()
```

²<https://nvlabs.github.io/iccv2019-mixed-precision-tutorial/>

Nvidia AMP³: An example

```
import torch
import amp
model = ...
optimizer = ...
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
for data, label in data_iter:
    out = model(data)
    loss = criterion(out, label)
    optimizer.zero_grad()
    with amp.scaled_loss(loss, optimizer) as scaled_loss:
        scaled_loss.backward()
optimizer.step()
```

allows AMP to perform automatic casting

replaces
loss.backward()

³<https://nvlabs.github.io/iccv2019-mixed-precision-tutorial/>

References I

- P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, “Mixed precision training,” *arXiv preprint arXiv:1710.03740*, 2017.
- T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, “8-bit optimizers via block-wise quantization,” *arXiv preprint arXiv:2110.02861*, 2021.
- J. Liu, C. Zhang *et al.*, “Distributed learning systems with first-order methods,” *Foundations and Trends® in Databases*, vol. 9, no. 1, pp. 1–100, 2020.