Optimization for Deep Learning

Lecture 11: Mixed Precision Training

Kun Yuan

Peking University

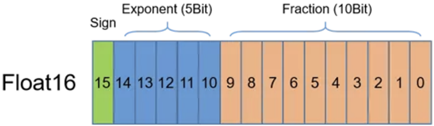
Main contents in this lecture

- Mixed precision training
- Mixed precision Adam
- Theory behind compression
- Error Feedback

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 DNN; 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: $2^{-14} \sim 2^{15}$

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

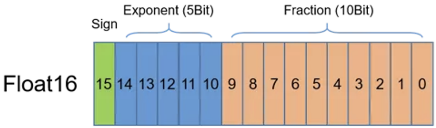
where -15 is the offset

- **Fraction**: 10 bits;

$$\begin{aligned} \text{Example:} \quad 1001000000 \quad &\longrightarrow \quad 1.1001000000 \\ &\longrightarrow \quad 1 + 576/1024 = 1.5625 \end{aligned}$$

where binary 1001000000 translates into decimal 576

Float 16 (FP16)



- **Translation law**

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

- **Largest positive number**:

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

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

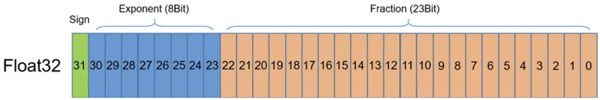
- **Smallest positive number**:

$$(-1)^0 \times 2^{-14} \times (1 + \frac{1}{1024}) \approx 6.1 \times 10^{-5}$$

Float 16 (FP16)

Binary	Hex	Value	Notes
0 00000 0000000000	0000	0	
0 00000 0000000001	0001	$2^{-14} \times (0 + \frac{1}{1024}) \approx 0.000000059604645$	smallest positive subnormal number
0 00000 1111111111	03ff	$2^{-14} \times (0 + \frac{1023}{1024}) \approx 0.000060975552$	largest subnormal number
0 00001 0000000000	0400	$2^{-14} \times (1 + \frac{0}{1024}) \approx 0.00006103515625$	smallest positive normal number
0 01101 0101010101	3555	$2^{-2} \times (1 + \frac{341}{1024}) \approx 0.33325195$	nearest value to 1/3
0 01110 1111111111	3bff	$2^{-1} \times (1 + \frac{1023}{1024}) \approx 0.99951172$	largest number less than one
0 01111 0000000000	3c00	$2^{0} \times (1 + \frac{0}{1024}) = 1$	one
0 01111 0000000001	3c01	$2^{0} \times (1 + \frac{1}{1024}) \approx 1.00097656$	smallest number larger than one
0 11110 1111111111	7bff	$2^{15} \times (1 + \frac{1023}{1024}) = 65504$	largest normal number
0 11111 0000000000	7c00	∞	infinity

Float 32 (FP32)



- **Translation law**

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

- **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 precisioin

FP32 weight copies

- An FP32 weight copy is maintained and updated with gradient

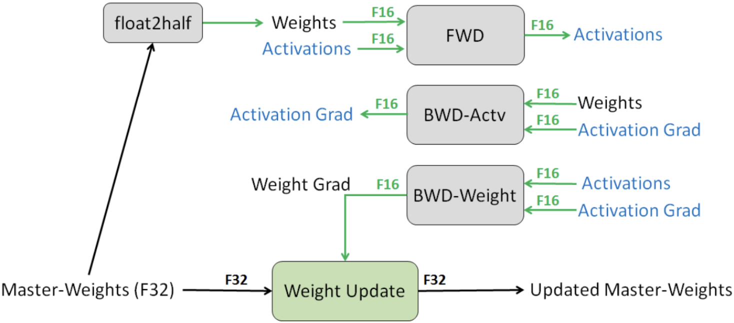
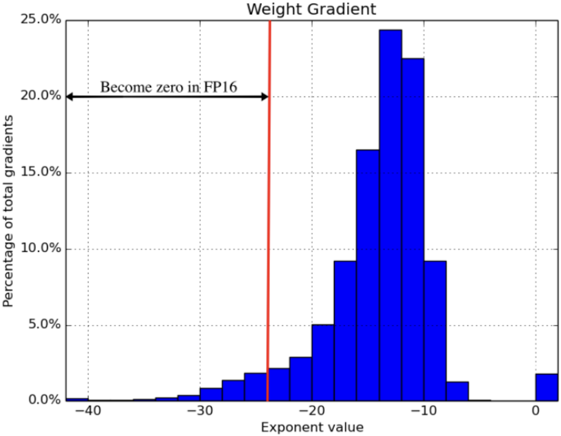


Figure 1: Mixed precision training iteration for a layer.

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



FP32 weight copies

- Reason II: **big value-to-update ratio**
- The resolution in each period is shown as follows[1]

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	∞	∞

[1] This figure is from wikipedia

FP32 weight copies

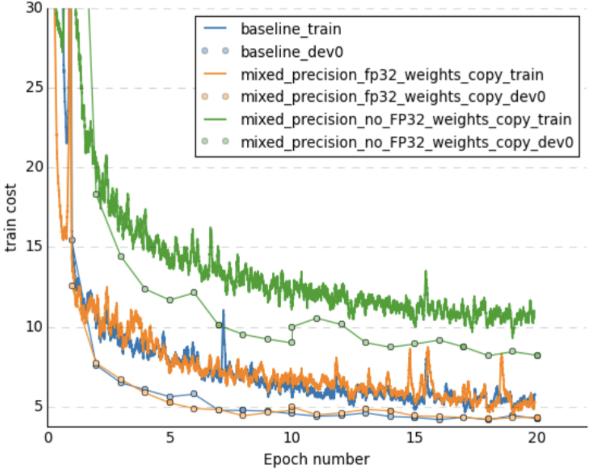
- If the value-to-update ratio is bigger than $2^{11} = 2048$, it holds that

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

- The update has no influence on the value

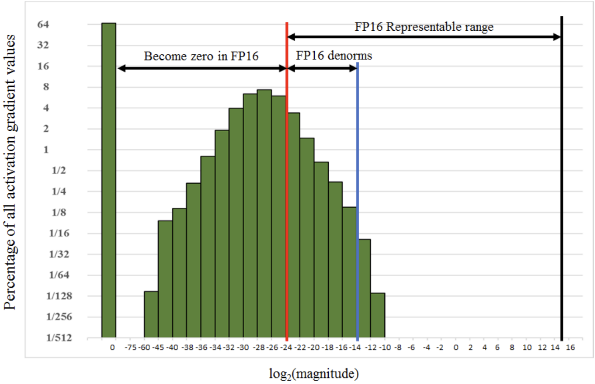
- For reasons I and II, we maintain FP32 copies for the weight

FP32 weight copies



Loss scaling

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



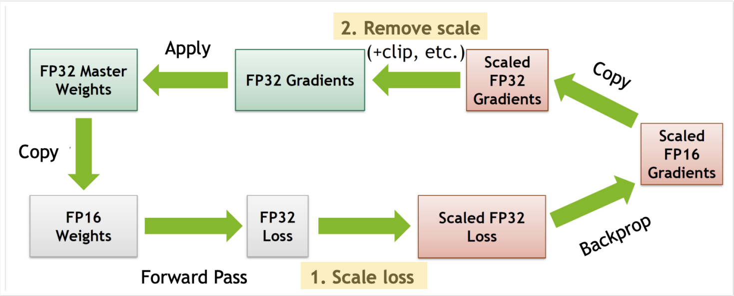
Loss scaling

- 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



Loss scaling

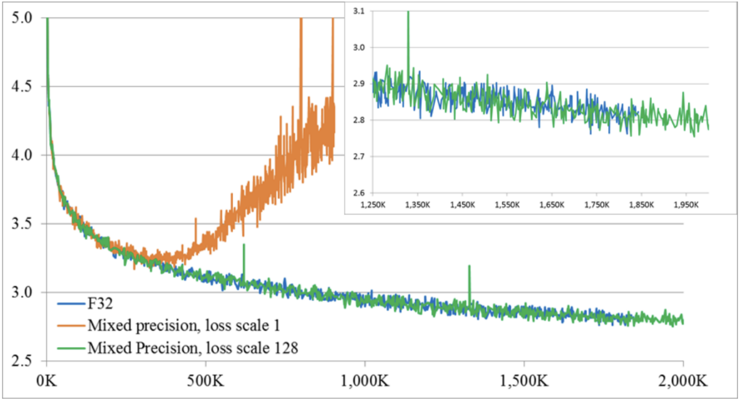


Figure 5: bigLSTM training perplexity

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

Numerical studies

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)

Numerical studies

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%

Nvidia AMP[2]

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 AMP[3]: 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/

Mixed-precision Adam

- Adam is widely-used in LLM
- Adam states use $33\% \sim 75\%$ memories
- For example, Adam states use 11GB for GPT2 and 41GB for T5
- It is urgent to reduce the memory footprint caused by Adam states

8-bit Adam optimizer

- Recall the Adam optimizer

$$
\begin{aligned}
g_k &= \nabla F(x_k; \xi_k) \\
m_k &= \beta_1 m_{k-1} + (1-\beta_1) g_k \\
s_k &= \beta_2 s_{k-1} + (1-\beta_2) g_k \odot g_k \\
x_{k+1} &= x_k - \frac{\gamma}{\sqrt{s_k} + \epsilon} \odot m_k
\end{aligned}
$$

- 8-bit only supports $2^8 = 256$ values; much less than FP16 and FP32

- (Dettmers et al., 2021) develops 8-bit Adam optimizer with block-wise quantization and dynamic quantization

Background: A uniform law for quantization

- Consider a mapping of a k-bit integer to a real element in D

$$Q^{\mathrm{map}}(i) : [0, 2^k - 1] \to D$$

- FP32 maps $0, \cdots, 2^{32} - 1$ to domain $[-3.4 \times 10^{38}, +3.4 \times 10^{38}]$

- We let i be the binary index and q_i be the real value, we have $Q^{\mathrm{map}}(i) = q_i$

- Example: $Q(2^{31} + 131072) = 2.03125$

Background: A uniform law for quantization

- Procedure to perform a general quantization from one data type to another
- **Step 1**: Normalize the input tensor $\mathbf{T}/N$ to fall into the range of D
- **Step 2**: For each $\mathbf{T}/N$, find the closes value q_i in D
- **Step 3**: Store the index corresponding q_i as $\mathbf{T}^Q$
- $\mathbf{T}^Q$ is a quantized binary representation of $\mathbf{T}$
- To dequantize, we have $\mathbf{T}^{DQ} = Q^{\mathrm{map}}(\mathbf{T}^Q) \times N$

Background: A uniform law for quantization (example)

- Suppose we have a 8-bit strategy Q to map $0, \cdots, 2^8 - 1$ to $[-1, 1]$

$$Q^{\text{map}}(i) : [0, 2^8 - 1] \rightarrow [-1, 1]$$

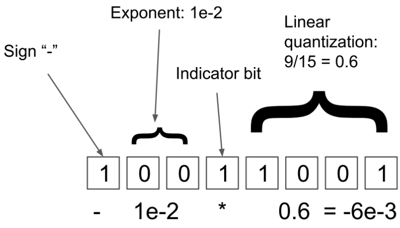
  Now we quantize a input FP32 tensor $\mathbf{T}$ to 8 bit

- **Step 1**: Let $N = \max\{|\mathbf{T}|\}$

- **Step 2**: Find the closest value via binary search

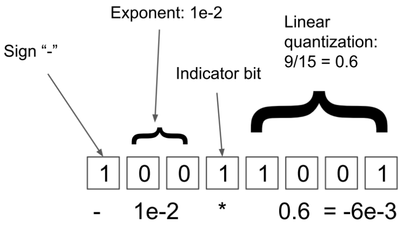
$$T_i^Q = \underset{j \in \{0, \cdots, 255\}}{\arg\min} |Q^{\text{map}}(j) - \frac{T_j}{N}|$$

Dynamic tree quantization



- The first bit is for **sign**
- The subsequent zero bits are for the magnitude of the exponent
- The first "1" bit indicates all following values are for linear quantization

Dynamic tree quantization



- Range: $[-1.0, +1.0]$
- Small values can have a large exponent 10^{-7}; Precision as high as $1/63$
- Better absolute and relative quantization error than linear quantization

Block-wise quantization

- 8-bit Adam optimizer uses dynamic quantization with value range $[-1, 1]$
- Given a long tensor $\mathbf{T}$, we need to compute $N = \max\{\mathbf{T}\}$
- Compute $N = \max\{\mathbf{T}\}$ requires multiple GPU synchronization
- There exits a few outliers that are very big.
    - In a tensor with 1 million elements, less than 3% of elements of the tensor will be in the range $[3, +\infty)$
    - If we normalize with $N = \max\{|\mathbf{T}|\}$, most quantization blocks are unused

Block-wise quantization

- We treat $\mathbf{T}$ as an one-dimensional vector with n elements
- Trunk the long vector into blocks of size B; result in n/B blocks
- Quantize the element block-wise. Let $N_b = \max\{|\mathbf{T}_b|\}$ for $b = 1, \cdots, \frac{n}{B}$

$$T_i^Q = \underset{j \in \{0, \cdots, 255\}}{\arg\min} |Q^{\text{map}}(j) - \frac{T_{bj}}{N_b}|\Big|_{0<j<B}$$

- Overcome all drawbacks of quantization as-a-whole

Block-wise quantization

- We treat $\mathbf{T}$ as an one-dimensional vector with n elements
- Trunk the long vector into blocks of size B; result in n/B blocks
- Quantize the element block-wise. Let $N_b = \max\{|\mathbf{T}_b|\}$ for $b = 1, \cdots, \frac{n}{B}$

$$T_i^Q = \underset{j \in \{0, \cdots, 255\}}{\arg\min} |Q^{\text{map}}(j) - \frac{T_{bj}}{N_b}|\Big|_{0<j<B}$$

- Overcome all drawbacks of quantization as-a-whole

8-bit optimizer procedure

- Quantize and store Adam states with 8 bit
- When updating and using state, dequantize it to FP32, update the weight, and quantize back to 8 bit
- 8-bit to 32-bit conversion element-by-element in registers; no additional temporary memory

Performance for common benchmarks

Optimizer	Task	Data	Model	Metric†	Time	Mem saved
32-bit AdamW	GLUE	Multiple	RoBERTa-Large	88.9	–	Reference
32-bit AdamW	GLUE	Multiple	RoBERTa-Large	88.6	17h	0.0 GB
32-bit Adafactor	GLUE	Multiple	RoBERTa-Large	**88.7**	24h	1.3 GB
8-bit AdamW	GLUE	Multiple	RoBERTa-Large	**88.7**	**15h**	**2.0 GB**
32-bit Momentum	CLS	ImageNet-1k	ResNet-50	77.1	–	Reference
32-bit Momentum	CLS	ImageNet-1k	ResNet-50	77.1	118h	0.0 GB
8-bit Momentum	CLS	ImageNet-1k	ResNet-50	**77.2**	**116 h**	**0.1 GB**
32-bit Adam	MT	WMT'14+16	Transformer	29.3	–	Reference
32-bit Adam	MT	WMT'14+16	Transformer	29.0	126h	0.0 GB
32-bit Adafactor	MT	WMT'14+16	Transformer	29.0	127h	0.3 GB
8-bit Adam	MT	WMT'14+16	Transformer	**29.1**	**115h**	**1.1 GB**
32-bit Momentum	MoCo v2	ImageNet-1k	ResNet-50	67.5	–	Reference
32-bit Momentum	MoCo v2	ImageNet-1k	ResNet-50	67.3	30 days	0.0 GB
8-bit Momentum	MoCo v2	ImageNet-1k	ResNet-50	**67.4**	**28 days**	**0.1 GB**
32-bit Adam	LM	Multiple	Transformer-1.5B	9.0	308 days	0.0 GB
32-bit Adafactor	LM	Multiple	Transformer-1.5B	**8.9**	316 days	5.6 GB
8-bit Adam	LM	Multiple	Transformer-1.5B	9.0	**297 days**	**8.5 GB**
32-bit Adam	LM	Multiple	GPT3-Medium	10.62	795 days	0.0 GB
32-bit Adafactor	LM	Multiple	GPT3-Medium	10.68	816 days	1.5 GB
8-bit Adam	LM	Multiple	GPT3-Medium	**10.62**	**761 days**	**1.7 GB**
32-bit Adam	Masked-LM	Multiple	RoBERTa-Base	3.49	101 days	0.0 GB
32-bit Adafactor	Masked-LM	Multiple	RoBERTa-Base	3.59	112 days	0.7 GB
8-bit Adam	Masked-LM	Multiple	RoBERTa-Base	**3.48**	**94 days**	**1.1 GB**

†**Metric**: GLUE=Mean Accuracy/Correlation. CLS/MoCo = Accuracy. MT=BLEU. LM=Perplexity.

Enable larger models

	Largest finetunable Model (parameters)	
GPU size in GB	32-bit Adam	8-bit Adam
6	RoBERTa-base (110M)	RoBERTa-large (355M)
11	MT5-small (300M)	MT5-base (580M)
24	MT5-base (580M)	MT5-large (1.2B)
24	GPT-2-medium (762M)	GPT-2-large (1.5B)

Open question

- Fine-tuning nowadays afford 4-bit quantization
- Training still halts at 8-bit quantization; no 4-bit quantization strategy works

SGD with mixed-precision

- Consider the stochastic optimization problem:

$$\min_{x \in \mathbb{R}^d} \quad f(x) = \mathbb{E}_{\xi \sim \mathcal{D}}[F(x; \xi)]$$

- SGD with mixed-precision training can be approximated by

$$g_k = \nabla F(x_k)$$
$$x_{k+1} = x_k - \gamma Q(g_k)$$

  where operator $Q(\cdot)$ quantizes g_k with fewer bits

- Quantization operator $Q(\cdot)$ influences convergence of the above algorithm

SGD with mixed-precision

We assume the following property (Liu et al., 2020)

Assumption 1

The (probably randomized) quantization operator $Q(\cdot)$ satisfies

$$\mathbb{E}[Q(g)] = g, \quad \forall g$$
$$\mathbb{E}[\|Q(g) - g\|^2] \leq \zeta^2, \quad \forall g$$

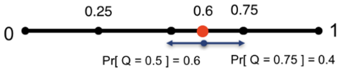
This assumption implies $\mathbb{E}[Q(g_k)] = \nabla f(x_k)$ and

$$\begin{aligned}
\mathbb{E}\|Q(g_k) - \nabla f(x_k)\|^2 &= \mathbb{E}\|Q(g_k) - g_k + g_k - \nabla f(x_k)\|^2 \\
&\leq \mathbb{E}\|Q(g_k) - g_k\|^2 + \mathbb{E}\|g_k - \nabla f(x_k)\|^2 \\
&\leq \sigma^2 + \zeta^2
\end{aligned}$$

Therefore, SGD with mixied-precision converges at rate $\mathcal{O}([\sigma + \zeta]/\sqrt{K})$

SGD with mixed-precision

- Some quantization strategy satisfies the unbiasedness



E[Q] = 0.6 × 0.5 + 0.4 * 0.75 = 0.6

Unbiased: $$\mathbb{E}[Q(x)] = \frac{Q_+(x) - x}{Q_+(x) - Q_-(x)} \cdot Q_-(x) + \frac{x - Q_-(x)}{Q_+(x) - Q_-(x)} \cdot Q_+(x) = x$$

- But most strategies are not

- SGD with mixed-precision cannot work with biased quantization (in theory)

Error compensation

- To be compatible with arbitrary quantization, we consider a new algorithm

$$
\begin{aligned}
g_k &= \nabla F(x_k; \xi_k) + \delta_{k-1} \\
\delta_k &= g_k - Q(g_k) \\
x_{k+1} &= x_k - \gamma Q(g_k)
\end{aligned}
$$

  where δ_k is the quantization error and will be added back to the vector to be quantized

- Converge with biased quantization; will discuss it in future lectures

- However, error compensation introduces a new state δ, which may offset the quantization saving; not commonly used in mixed-precision training but is very useful to save communication in distributed learning

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.