Technology behind LLM Training

报告人：刘益

报告时间：2023.06.08

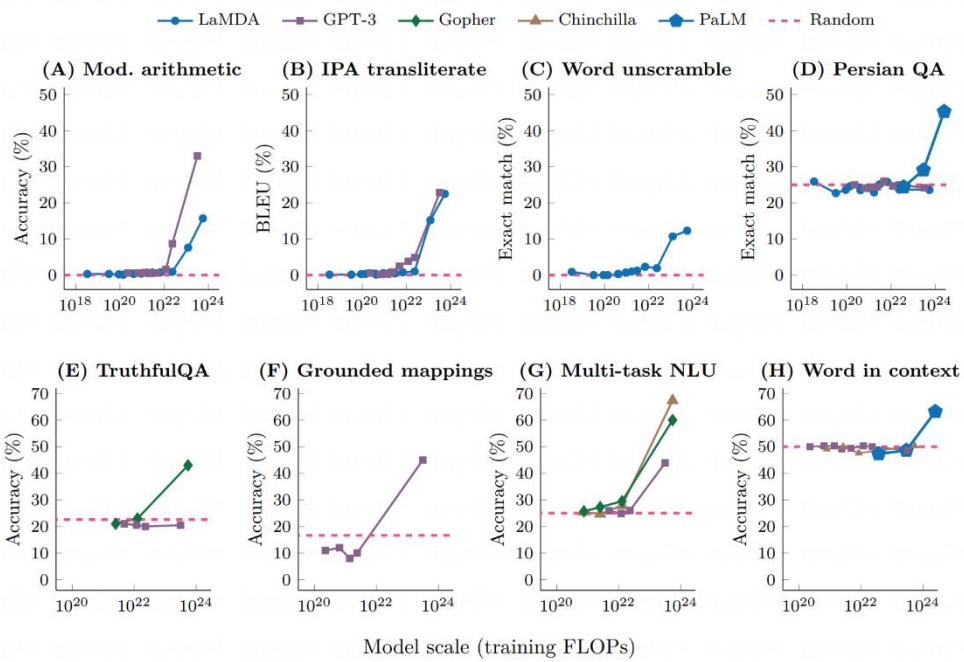
中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

目录

- 介绍
  - 大语言模型 (Large Language Models，LLMs)
  - 训练大模型的挑战
- 分布式训练 (Distributed Training)
- 内存优化技术 (Memory-Saving Technologies)
- 参数高效微调方法 (Parameter-Efficient Fine-Tuning, PEFT)
- 国内外大模型训练进展

1.1 大语言模型

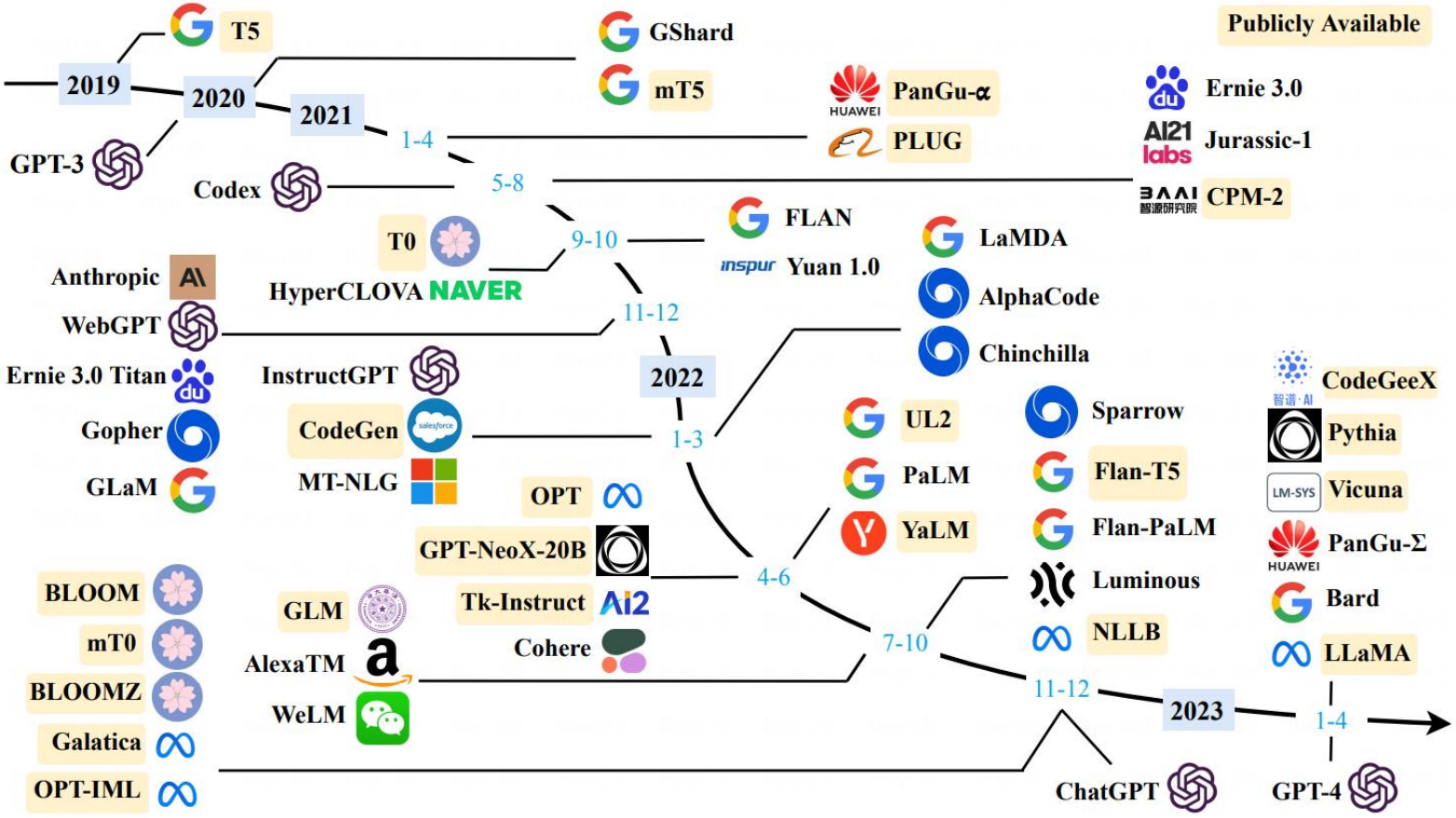
- 大语言模型通常指参数规模在十亿以上，Decoder-only 结构的自回归模型
- 模型参数规模达到一定尺度（百亿）后，出现涌现能力 (emerge abilities)



- 大模型有更好的Zero-Shot性能、长上下文理解、复杂逻辑推理等能力

1.2 大语言模型的发展

- T5-large (3B) → GPT3 (175B) → GPT4 (800B?)
- 参数规模的不断增大为模型训练带来存储、计算、通信等多方面的挑战



1.3.1 训练大模型的挑战：内存瓶颈

- 问：使用混合精度训练一个1750亿（175B)参数，Adam优化器的模型需要多大的显存?

- 答：每一个参数对应fp16模型参数、 fp16模型梯度和Adam状态（fp32的模型参数备份，fp32的一阶动量和fp32的二阶动量），fp16双字节，fp32四字节

  所需内存 = $175 * 10^{10} * (2 + 2 + 4 + 4 + 4) / 1024^3$ = 2600GB

- 大模型训练的内存瓶颈主要来自**优化器状态 (75%)**
- 即使只考虑模型参数和优化器状态的存储，忽略计算时产生的中间激活值，所需的内存容量也**远超单个GPU的显存 (80G A100/H100)**

1.3.2 训练大模型的挑战：计算效率

- 使用单张V100做GPT-3的预训练，需要90年。。

Table 2. Estimated training time on one NVIDIA V100 GPU.

Name	#Param	Dataset Size	Time
GPT [1]	110M	4GB	3 days
BERT [2]	340M	16GB	50 days
GPT-2 [3]	1.5B	40GB	200 days
GPT-3 [6]	175B	560GB	90 years

- 使用多机训练则存在通信成本和内存冗余等问题，导致计算效率随模型参数增加**非线性地快速增加**

目录

- 介绍
- 分布式训练 (Distributed Training)
  - 数据并行 (Data Parallelism, DP)
  - 流水线并行 (Pipeline Parallelism, PP)
  - 张量并行 (Tensor Parallelism, TP)
- 内存优化技术 (Memory-Saving Technologies)
- 参数高效微调方法 (Parameter-Efficient Fine-Tuning, PEFT)
- 国内外大模型训练进展

2.1.1 数据并行（DP）

- 将大规模的数据切分到多个模型实例上进行处理，加快训练速度
- 流程：
  - 把一份数据X
  - 每块GPU做一
  - AllReduce：
    orker）的梯度，再下发给Worke

- 缺陷：
  - 存储开销大，每块GPU上都存了一份完整的模型，造成冗余
  - 通讯开销大，Server需要和每一个Worker进行梯度传输，从而承担了系统所有的通讯压力 (一核有难，八核围观。。)

2.1.2 分布式数据并行（DDP）

- 针对DP中通讯负载不均的问题，将Server上的通讯压力均衡转到各个Worker上
- Ring-AllReduce
  - Reduce-Scatter：逐步交换彼此的梯度并融合，最后每个 GPU 都会包含完整融合梯度的一部分
  - All-Gather：逐步交换彼此不完整的融合梯度，最后所有 GPU 都会得到完整的融合梯度
- 通讯量

假设模型参数W的大小为 Φ ，GPU个数为 N 。则梯度大小也为 Φ ，每个梯度块的大小为 $\frac{\Phi}{N}$

对单卡GPU来说（只算其send通讯量）：

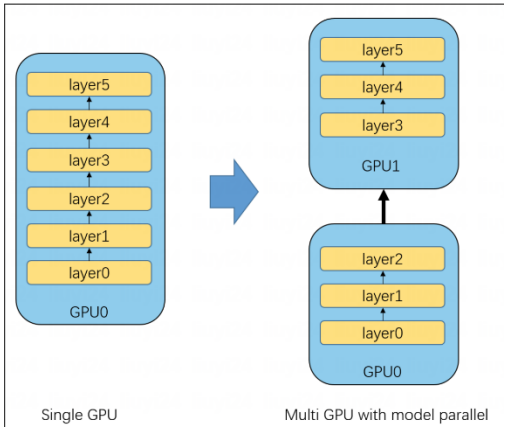
- Reduce-Scatter阶段，通讯量为 $(N-1)\frac{\Phi}{N}$
- All-Gather阶段，通讯量为 $(N-1)\frac{\Phi}{N}$

单卡总通讯量为 $2(N-1)\frac{\Phi}{N}$ ，随着N的增大，可以近似为 2Φ 。全卡总通讯量为 $2N\Phi$

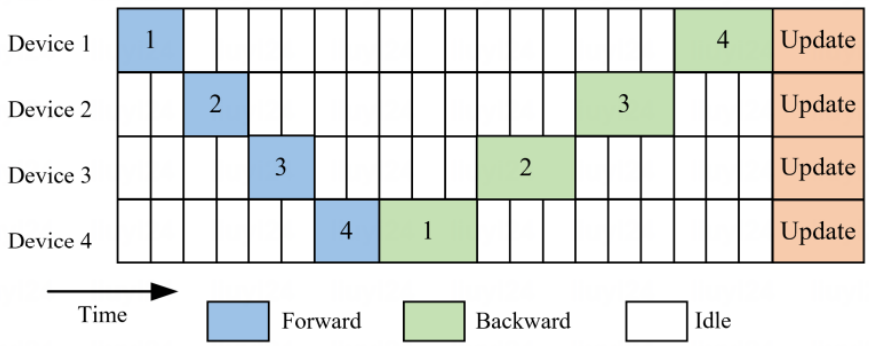
而对前文的DP来说，它的Server承载的通讯量是 $N\Phi$ ，Workers为 $N\Phi$ ，全卡总通讯量依然为 $2N\Phi$ 。**虽然通讯量相同，但搬运相同数据量的时间却不一定相同。**DDP把通讯量均衡负载到了每一时刻的每个Worker上，而DP仅让Server做勤劳的搬运工。当越来越多的GPU分布在距离较远的机器上时，DP的通讯时间是会增加的。

2.2.1 模型并行（MP）

- 当单个GPU装不下大模型时，一个直接的解决办法是，把模型隔成不同的层，每一层都放到一块GPU上

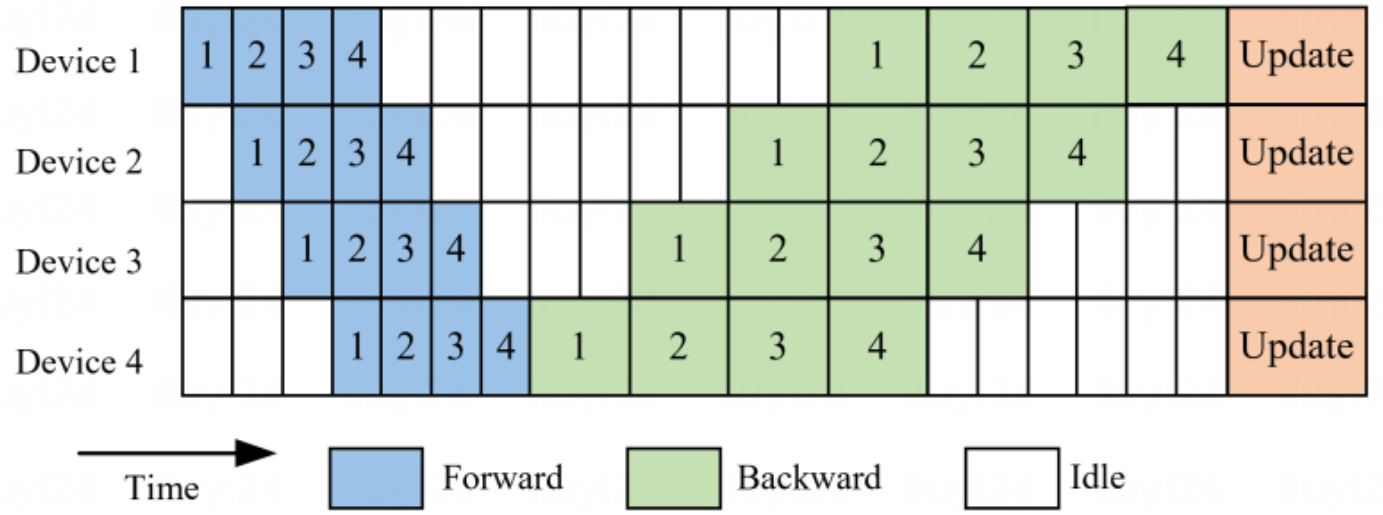


- 每个GPU维护若干层的参数，**串行**地进行前向和后向计算



2.2.2 流水线并行（PP）

- 在模型并行的基础上，进一步引入数据并行的办法，把原先一个batch数据再划分成若干个micro-batch，以流水线的形式计算
- 每个micro-batch独立的计算前后向传播，然后将梯度累加，就能得到整个batch的梯度

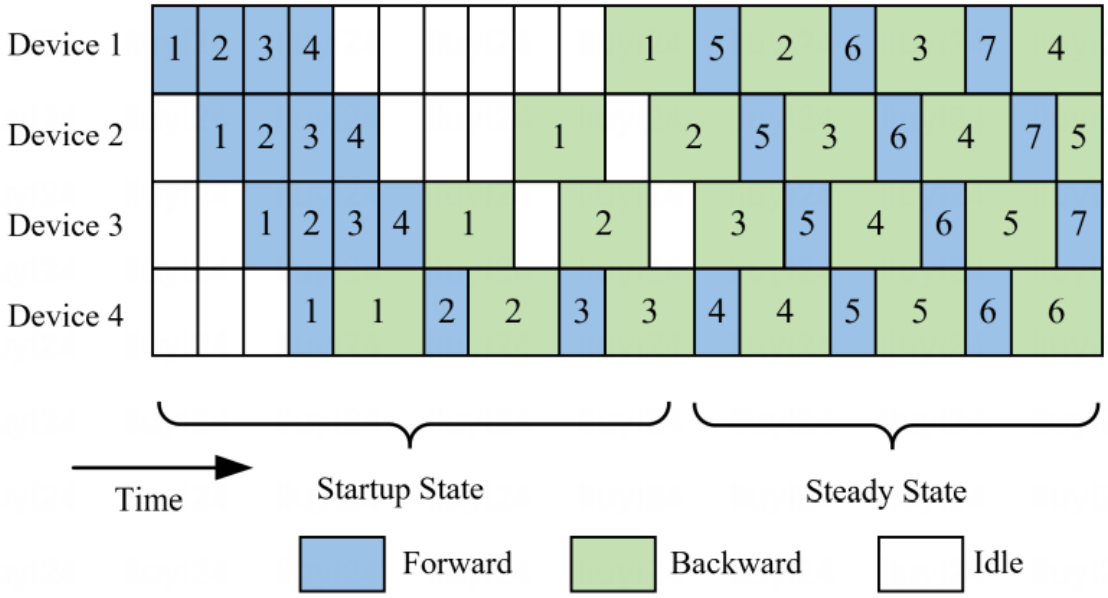


- 缺点：需要等所有的micro-batch前向传播完成后，才会开始反向传播，导致各个device的空闲时间依旧较长

[1] Wei J, Tay Y, Bommasani R, et al. Emergent abilities of large language models[J]. arXiv preprint arXiv:2206.07682, 2022.

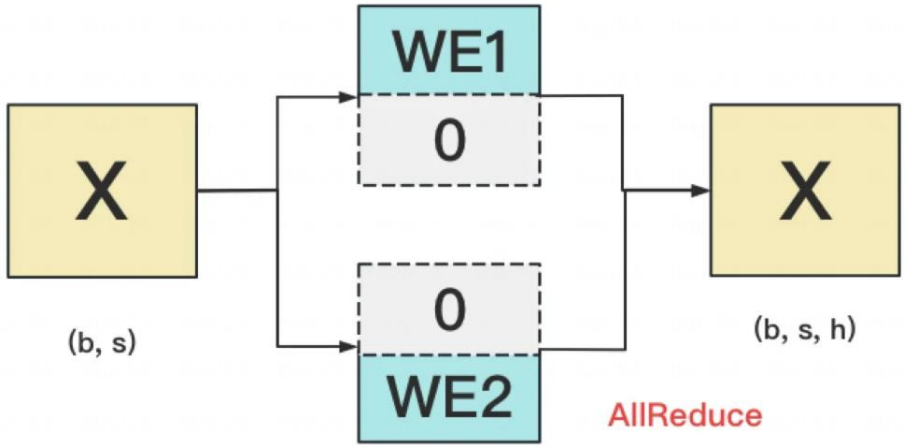
2.2.4 PP: 1 Forward 1 Backward (1F1B)

- 前向计算的中间值需要等到对应的后向计算完成后才能释放
- 提高后向计算的优先级，当每个micro-batch完成前向传播之后，立即开始后向传播

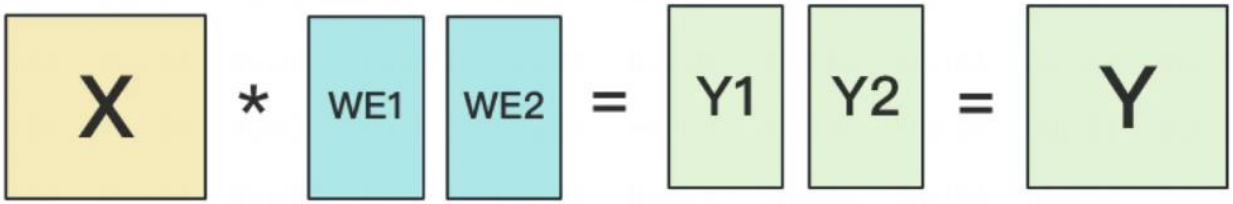


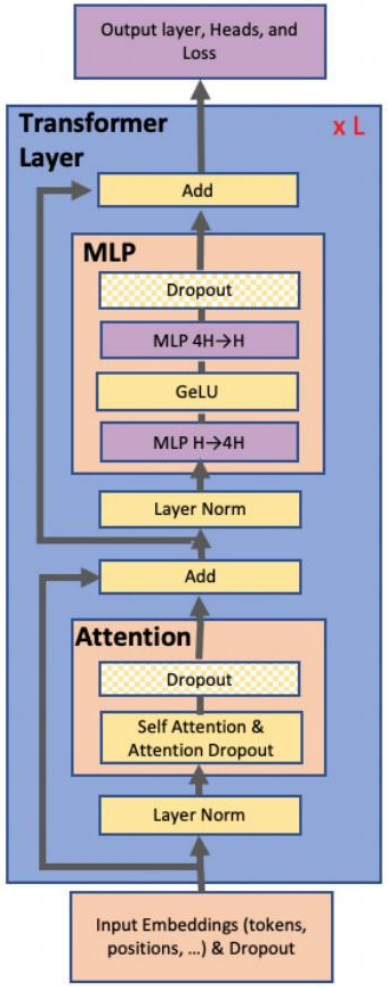
2.2.4 张量并行（TP）

- 针对Transformer中的主要结构（Embedding、MLP、Attention)设计，对参与运算的张量，沿着某个维度进行切分到不同的GPU上
- 输入层Embedding：将word embedding拆分到各个GPU



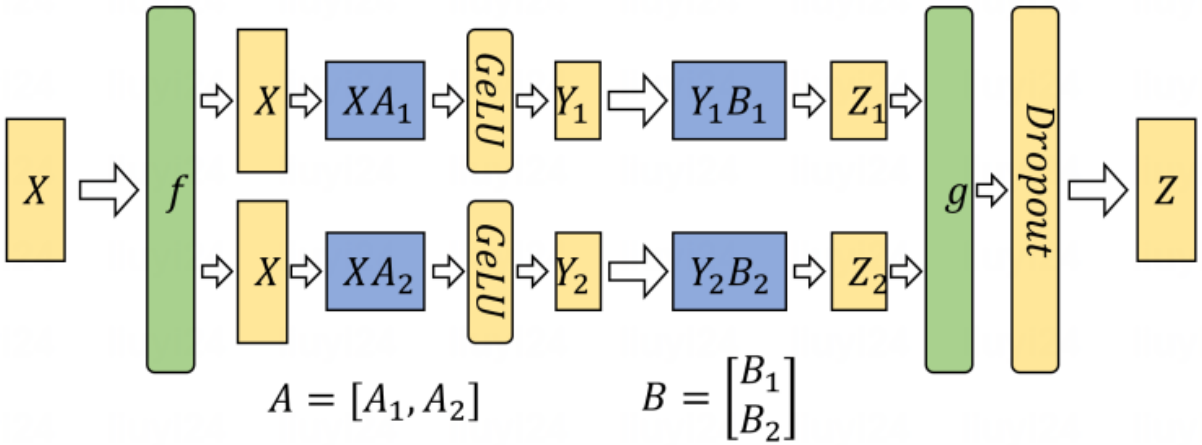
- 输出层Embedding：和输入层Embedding共享同一参数





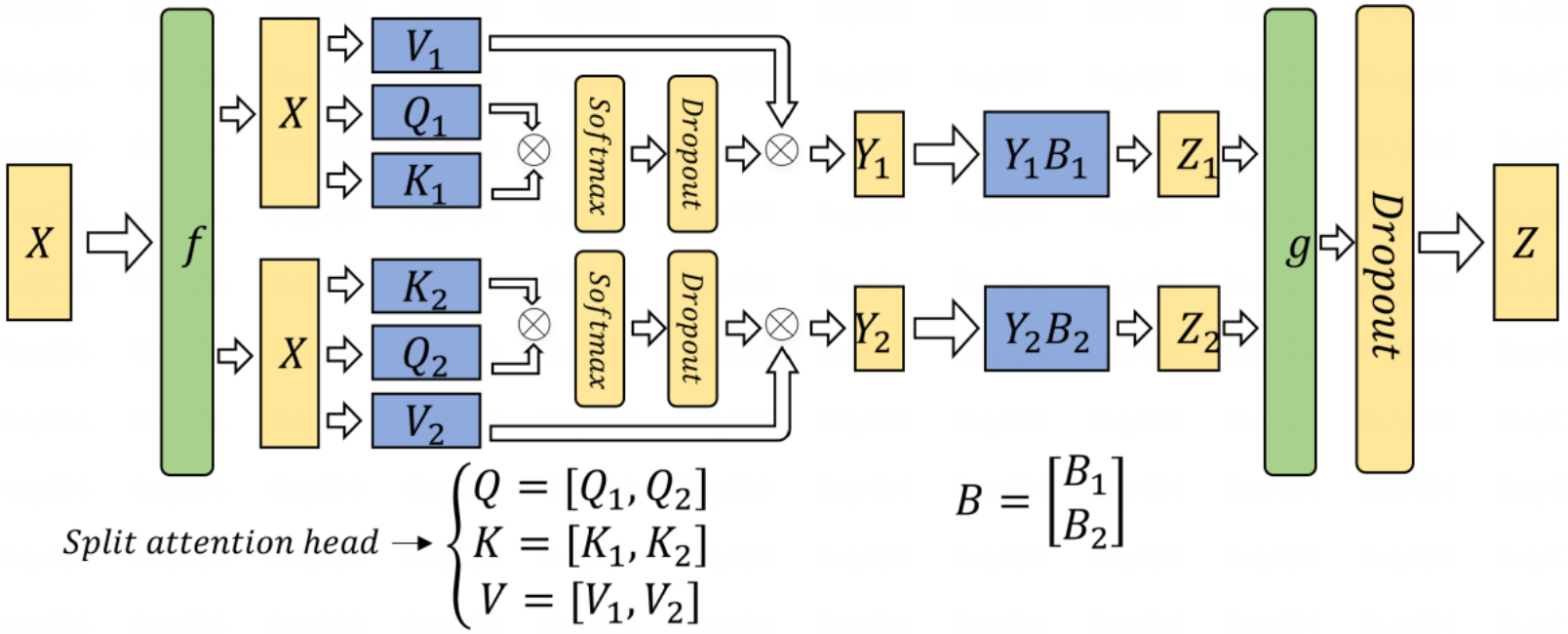
2.2.4 TP: MLP层

- MLP-1：$Y = GeLu(XA)$
  - 按列切分 $A = [A_1, A_2]$ $XA = [XA_1, XA_2]$
  - $Y = [Y_1, Y_2] = [GeLu(XA_1), GeLu(XA_2)]$
- MLP-2：$Z = Dropout(YB)$
  - 按行切分 $B = [B_1, B_2]^T$ $Y = [Y_1, Y_2]$
  - $YB = Dropout(Y_1B_1 + Y_2B_2)$



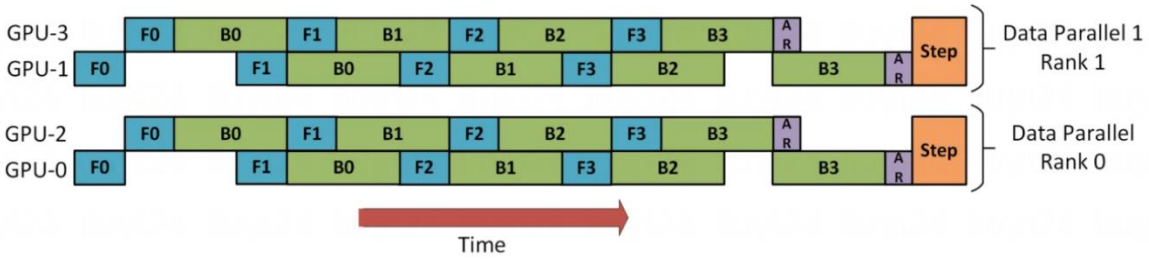
2.2.4 TP: Attention层

- 多头注意力：沿着列方向根据每个头的维度切分，把每个头的参数放到一块GPU上，最后再将结果concat起来

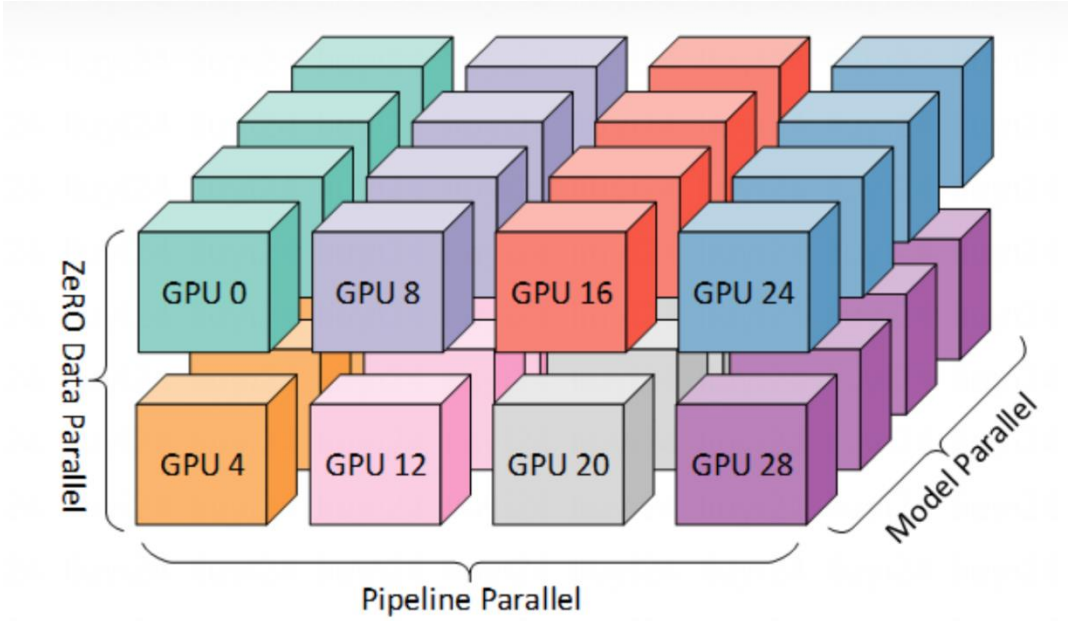


2.2.5 3D-Parallelism: DP+PP+TP

- 同一个节点内TP，跨节点间PP，最外层（若干个节点作为一组）DP



dp-pp-2d



dp-pp-tp-3d

目录

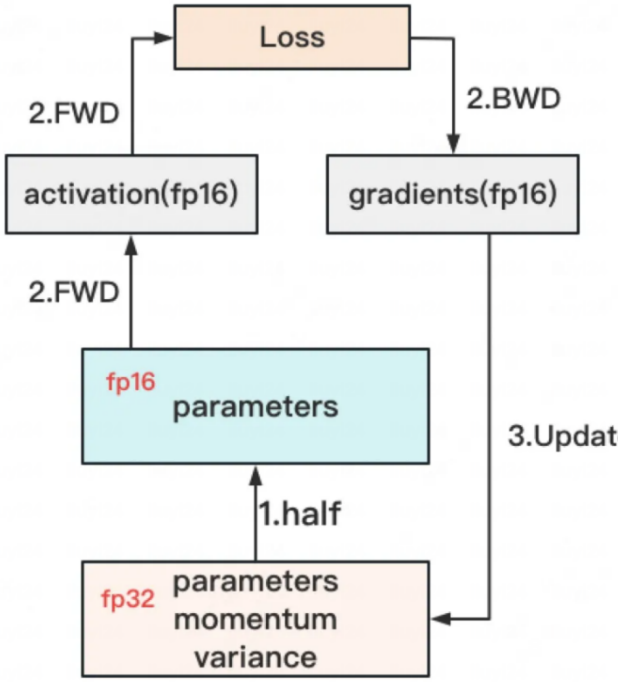
- 介绍
- 分布式训练 (Distributed Training)
- 内存优化技术 (Memory-Saving Technologies)
  - 混合精度训练 (Mix Precision Training)
  - 激活函数重演 (Activation Checkpointing)
  - ZeRO优化机制 (Zero Redundancy Optimizer)
- 参数高效微调方法 (Parameter-Efficient Fine-Tuning, PEFT)
- 国内外大模型训练进展

3.1 混合精度训练

- 大模型训练的过程中，GPU需要存储的内容：
  - **optimizer states**：Adam优化算法中的momentum和variance
  - **gradients**：模型梯度
  - **parameters**：模型参数W

$$
\begin{aligned}
g_t &= \nabla \hat{L}(\theta_t) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \hat{m}_t
\end{aligned}
$$

- 前后向传播用fp16/bf16，参数更新用fp32



训练中所需存储的参数大小：

$$
\underbrace{2+4}_{\text{weights}} + \underbrace{2+4}_{\text{gradients}} + \underbrace{4+4}_{\text{Adam states}} = 20 \text{ bytes}.
$$

因为在训练时可能会用到梯度累积，
所以会保留一个fp32的梯度

3.2 激活函数重演

- 前向计算时选择性的丢弃部分产生的中间激活值
- 当反向传播需要使用这些激活值时，再依据上一个保存的激活值来计算出这些需要的激活值来帮助完成反向传播
- 时间换空间：通过重复计算，减少训练过程中所需的存储空间

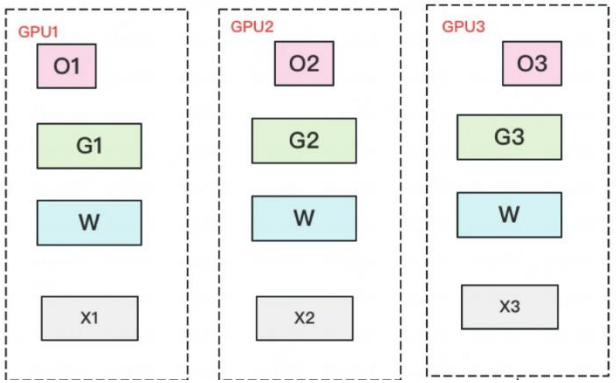
3.3 ZeRO is all you need

- 训练中参数并不会每时每刻都用到：
  - Adam的优化器状态O只在最终做update时才用到
  - 数据并行DP中梯度G只在最后做AllReduce和update时才用到
  - 模型参数W只在做forward和backward的那一刻才用到

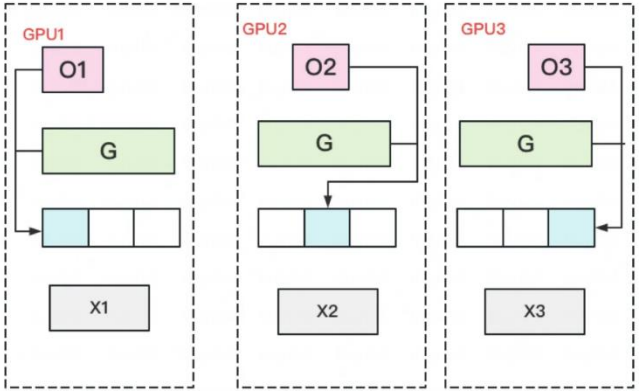
- Zero Redundancy Optimizer：对参数进行最细粒度的**零冗余**切分，每个GPU只维护部分参数，计算或者更新需要的时候再聚合得到完整的参数

3.3.1 ZeRO-DP Stage 1

- P_{os}优化器状态分割：将optimizer states分成若干份，每块GPU上各自维护一份
- 将一个batch的数据分成3份，每块GPU各吃一份，做完一轮forward和backward后，各得一份梯度G

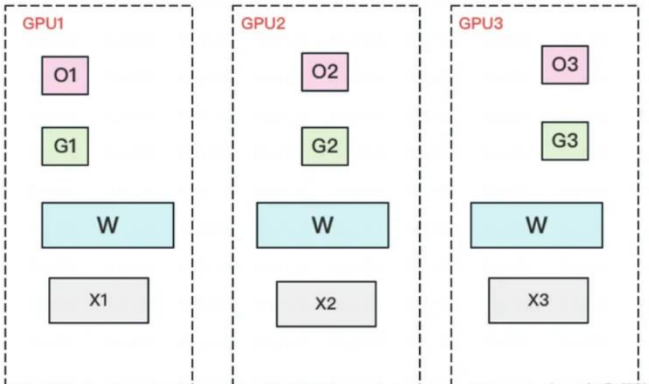


- 对梯度G做一次AllReduce，得到完整的梯度G，对相应部分的参数W进行更新
- 对参数W做一次All-Gather，从别的GPU上把更新好的其余部分的W取回

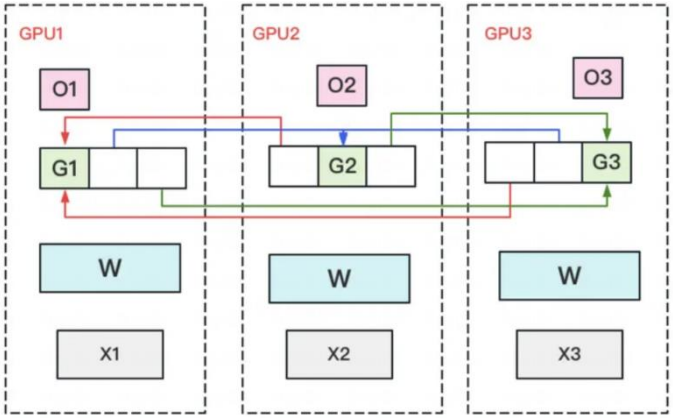


3.3.2 ZeRO-DP Stage 2

- $P_{os}+P_g$优化器状态与梯度分割：更进一步，我们把梯度也拆开，每个GPU各自维护一块梯度
- 将一个batch的数据分成3份，每块GPU各吃一份，做完一轮forward和backward后，各得一份梯度G

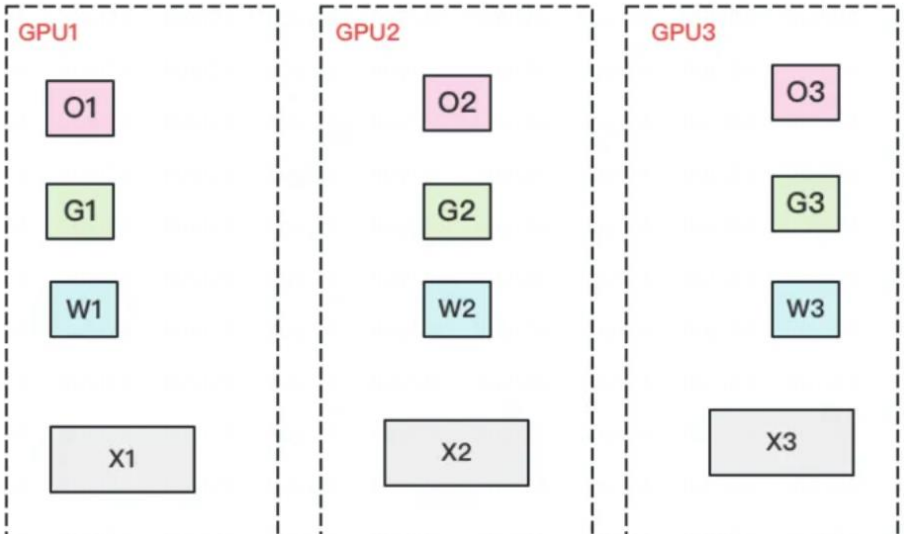


- 对梯度做一次**Reduce-Scatter**，保证每个GPU上所维持的那块梯度是聚合梯度，去更新相应的W
- 同理，对W做一次**All-Gather**



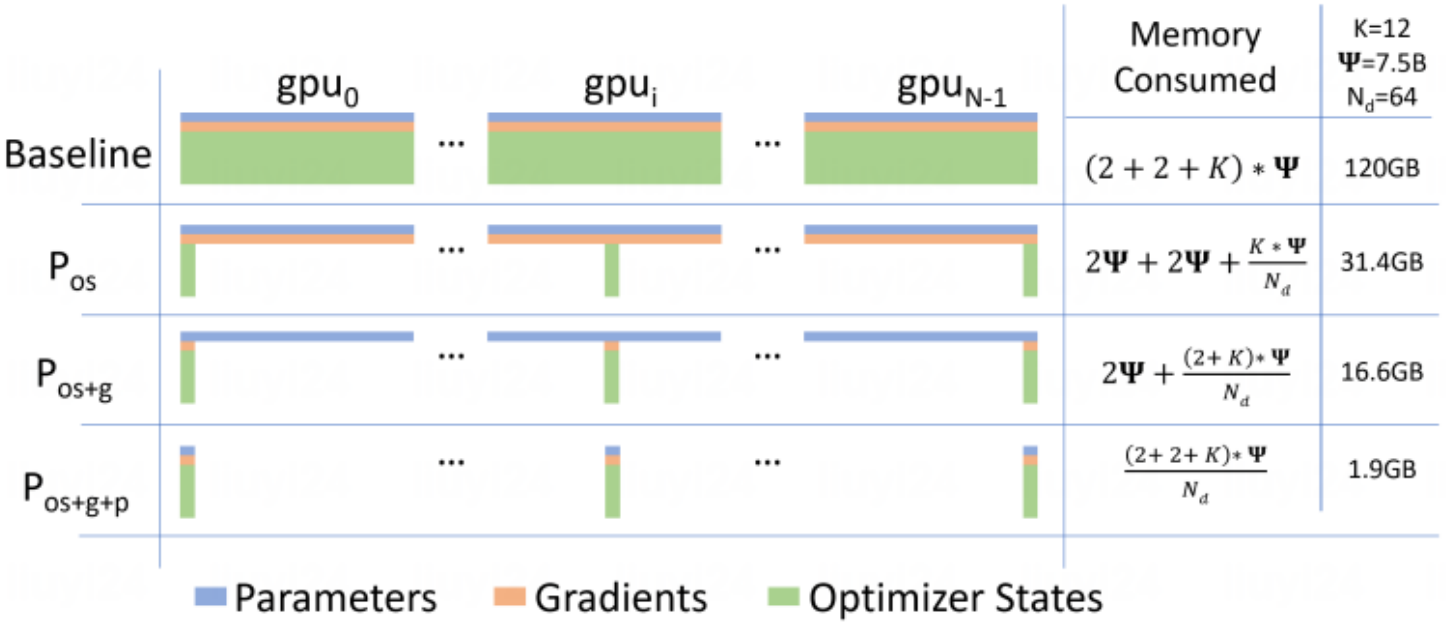
3.3.3 ZeRO-DP Stage 3

- $P_{os}+P_g+P_w$优化器状态、梯度与参数分割：每块GPU各自维持对应的optimizer states，gradients和parameters
- Forward/Backward时，对W做一次**All-Gather**，得到一份完整的W，用完即丢弃
- Backward后算得一份完整的梯度G，做一次**Reduce-Scatter**聚合自己维护的那部分梯度，聚合操作结束后，立刻把不是自己维护的G抛弃
- 用自己维护的O和G，更新W。由于训练过程中每块GPU只维护部分W，因此无需再对W做任何AllReduce操作



3.3.4 ZeRO-DP 存储开销

- K: 优化器状态
- N_d: GPU数量
- 和Baseline (朴素DP)相比，ZeRO-DP 3阶段用**1.5倍**的通讯开销，换回近**120倍**的显存



3.3.4 ZeRO-DP VS MP

- 同样对模型参数进行切分，ZeRO-DP和模型并行MP的区别在哪里?
- ZeRO-DP是**模型并行的形式，数据并行的实质**
  - MP在训练时每个并行节点只使用**部分参数进行计算**
  - 尽管ZeRO只维护部分参数，在计算时需要把各GPU的参数**聚合**起来，进行**完整的计算**
- 区别于MP中按**层切分**的PP和按**维度切分**的TP， ZeRO-DP会对每个参数进行**零冗余**切分（i.e. 对12张GPU和一个768维的tensor，每个GPU存放64维参数)，将存储**均匀分摊到每个GPU**上。但同时也会带来**更高昂的通信开销**

3.3.5 ZeRO-DP + PP + TP

- 当 ZeRO-DP 与 PP (以及 TP) 结合时，它通常只启用 ZeRO 阶段 1，只对优化器状态进行分片

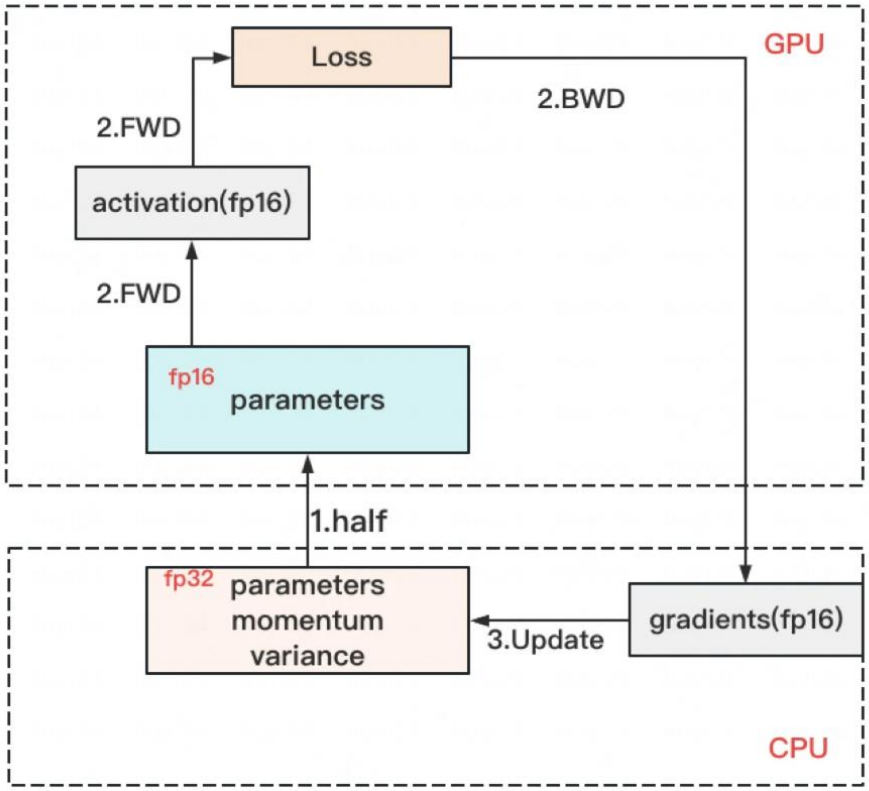
- 无法使用ZeRO-DP的阶段2 (切分梯度)和阶段3 (切分参数)结合模型并行

```
def __init__(self, has_bool_tensors=False, *super_args, **super_kwargs):
    super().__init__(*super_args, **super_kwargs)
    assert isinstance(self.module, PipelineModule), "model must base PipelineModule"

    assert self.zero_optimization_stage() < 2, "ZeRO-2 and ZeRO-3 are incompatible with pipeline parallelism"
```

3.3.6 ZeRO-CPU Offload

- 显存不够，内存来凑：把要存储的大头weight updates卸载(offload)到CPU上，把计算部分FWD+BWD放到GPU上

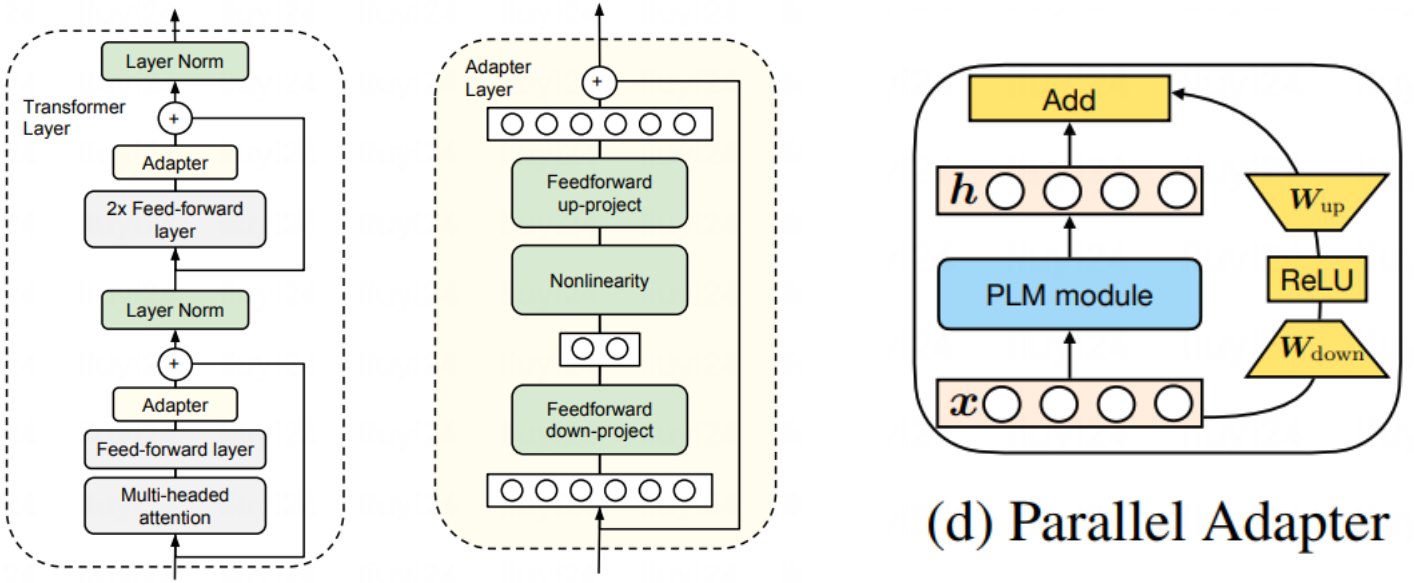


目录

- 介绍
- 分布式训练 (Distributed Training)
- 内存优化技术 (Memory-Saving Technologies)
- 参数高效微调方法 (Parameter-Efficient Fine-Tuning, PEFT)
  - Adapter-tuning
  - Prefix-tuning
  - LoRA
- 国内外大模型训练进展

4.1 Adapter-Tuning

- 将较小的神经网络层或模块插入预训练模型的每一层，这些新插入的神经模块称为adapter（适配器），下游任务微调时也只训练这些适配器参数
- 包括 Series Adapter（串行） 和 Parallel Adapter（并行）



- 缺陷：引入额外的计算，带来推理延迟问题

4.2 Prefix-Tuning

- Prefix-Tuning：在模型输入前添加一个连续的且任务特定的向量序列，固定 PLM 的所有参数，只更新优化特定任务的 prefix
- P-Tuning：利用少量连续的 embedding 参数作为 prompt
- Prompt-Tuning：固定整个模型参数，只允许将额外k个可更新的tokens前置到输入文本中
- 缺陷：Prefix-Tuning性能随可训练参数规模非单调变化，减少用于处理下游任务的序列长度

4.3.1 LoRA

- 虽然模型参数众多，但其实模型主要依赖低秩维度 (low intrinsic dimension)
- 在原始模型增加一个旁路，做一个降维再升维的操作，训练的时候固定模型参数，只训练降维矩阵A与升维矩阵B

具体来看，假设预训练的矩阵为 $W_0 \in \mathbb{R}^{d\times k}$ ，它的更新可表示为：

$$W_0 + \Delta W = W_0 + BA, B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$$

其中秩r << min(d,k)。

对于 $h = W_0 x$ ，它的前向计算变为：

$$h = W_0 x + \Delta W x = W_0 x + BAx = (W_0 + BA)x$$

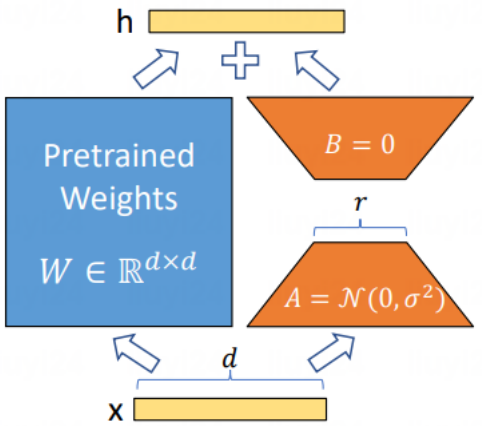
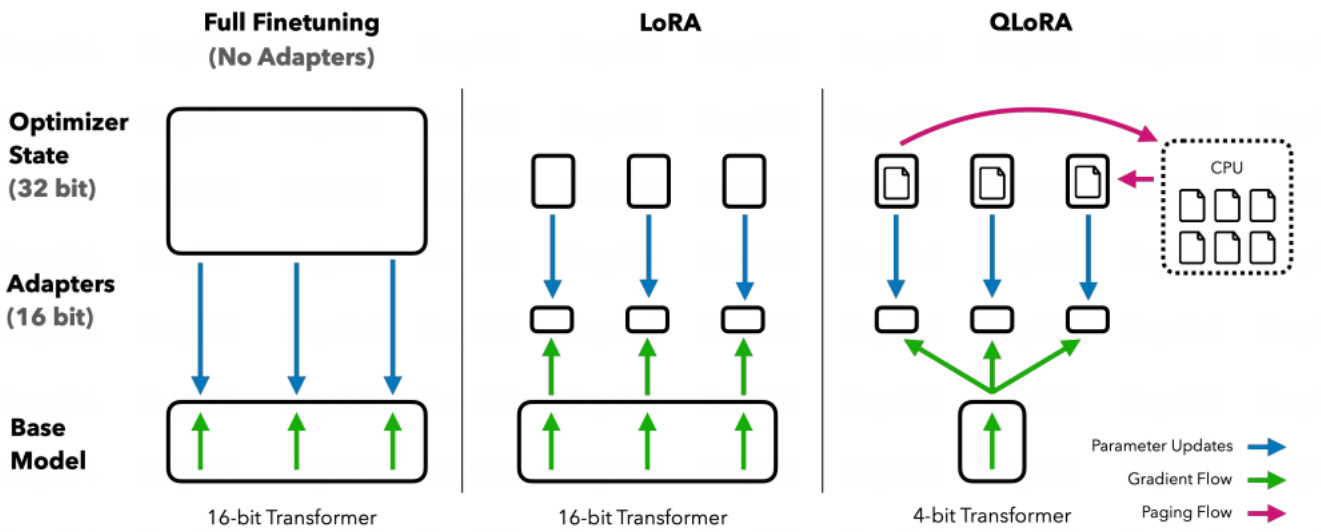


Figure 1: Our reparametrization. We only train A and B.

- 推理时将模型参数和LoRA参数合并即可
- 对于用Adam训练的Transformer，在秩远小于维度的情况下可**将内存减少2/3**，且不需要存储模型参数的优化器状态。同时可大大加快训练速度
- 可与Zero-DP 3阶段结合使用，使用4张A100 80G即可训练65B的LLaMA

4.3.2 QLoRA

- 基于LoRA对模型进行量化：模型用4bit加载，训练时把数值反量化到bf16后进行训练
- 能够在单个48GB显存的GPU上微调65B参数的LLAMA

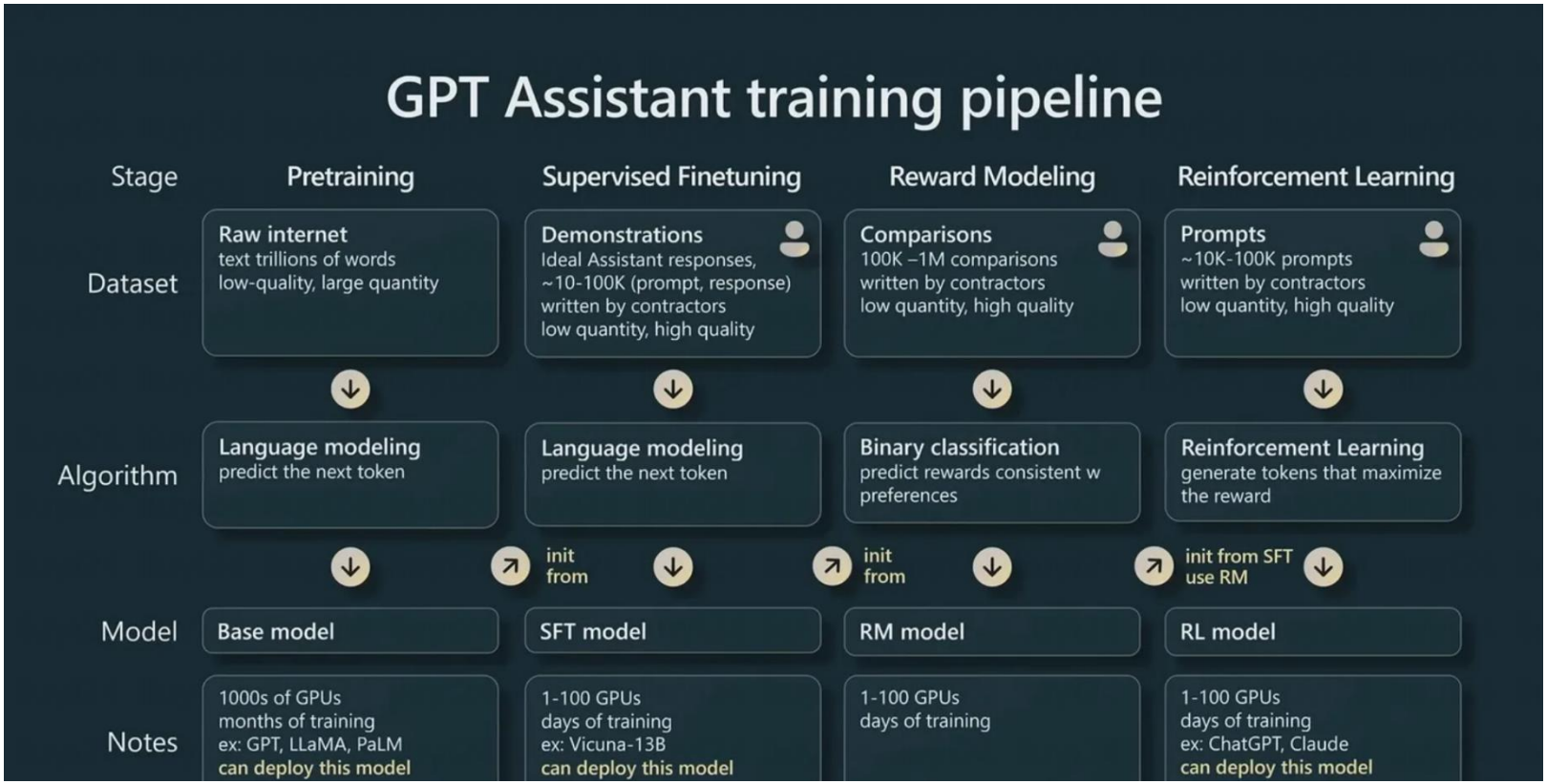


目录

- 介绍
- 分布式训练 (Distributed Training)
- 内存优化技术 (Memory-Saving Technologies)
- 参数高效微调方法 (Parameter-Efficient Fine-Tuning, PEFT)
- 国内外大模型训练进展
  - 大模型训练范式
  - 大模型训练框架
  - 开源大模型训练
  - 国内大模型训练进展

5.1 大模型训练范式

- Pretraining → SFT → Reward Modeling → RLHF



5.2 大模型训练框架

- DeepSpeed: 用于分布式模型训练的框架
- Megatron-DeepSpeed: BLOOM、GLM-130B等主流大模型都是基于Megatron-DeepSpeed的3D-Parallelism训练

组件	DeepSpeed	Megatron-LM
ZeRO 数据并行	是	
张量并行		是
流水线并行	是	
BF16 优化器	是	
CUDA 融合核函数		是
数据加载器		是

- DeepSpeedChat: 实现类似ChatGPT模型的端到端RLHF训练

GPU SKUs	OPT-1.3B	OPT-6.7B	OPT-13.2B	OPT-30B	OPT-66B	OPT-175B
1x V100 32G	1.8 days					
1x A6000 48G	1.1 days	5.6 days				
1x A100 40G	15.4 hrs	3.4 days				
1x A100 80G	11.7 hrs	1.7 days	4.9 days			
8x A100 40G	2 hrs	5.7 hrs	10.8 hrs	1.85 days		
8x A100 80G	1.4 hrs($45)	4.1 hrs ($132)	9 hrs ($290)	18 hrs ($580)	2.1 days ($1620)	
64x A100 80G	31 minutes	51 minutes	1.25 hrs ($320)	4 hrs ($1024)	7.5 hrs ($1920)	20 hrs ($5120)

5.3.1 开源大模型训练

- BLOOM
  - 使用59 种语言，共3500亿token的数据集，384张A100 80G训练3-5个月
  - 模型参数：300M~176B
  - 训练使用Megatron-DeepSpeed框架，ZeRO-DP (stage 1) + TP + PP

- LLaMA
  - 模型参数：7B 13B 33B 65B
  - Pretraining → SFT
  - 语言以英语和拉丁语系为主，在包含1.4T个token的数据集上使用2048张A100 80G训练大约21天

5.3.2 开源大模型训练：ChatGLM

- 中英双语，中文场景下性能超越ChatGPT
- 模型参数：6B (开源) 130B (未开源，听说可以花几百万买模型。。)
- 走完了GPT训练的完整流程 (Pretraining → SFT → Reward Modeling → RLHF)

5.4 国内大模型训练进展

- Pretraining → SFT → Reward Modeling → RLHF：清华唐杰ChatGLM，最好的中文大模型
- Pretraining → SFT：百度NLP文心一言，阿里达摩院通义千问
- 主流范式：使用BLOOM/LLaMA/ChatGLM作为基模型，基于Self-Instruction技术从GPT-4获取高质量的Prompt用于SFT训练
  - 美团NLP中心: 完成百亿LLaMA的完整训练，正在做千亿BLOOM的RLHF
  - 快手MMU: 完成百亿LLaMA的SFT训练

@快手MMU自研大模型聊天机器人 将下面的代码用金庸武侠小说的风格来写一个故事: def quick_sort(arr):
if len(arr) <= 1:
return arr
pivot = arrllen(arr) // 2]
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
return quick sort(left) + middle + quick_sort(right)

@简一 在江湖中，有一门绝技名为“快刀”，乃是一种神奇而强大的武功。江湖中人人皆知，但无人能掌握。只有一位名叫“张无忌”的少年，在江湖中闯荡多年，最终习得了此绝技。

张无忌年轻英俊，聪明过人，他的父亲是江湖上赫赫有名的“倚天剑”，母亲则是“屠龙刀”的传人。从小便在父母亲的指导下，学习各种武学和兵器。

一日，张无忌在江湖上闯荡时，偶遇了一位名叫“杨过”的少年。杨过是江湖上另一个传奇人物，精通“神雕侠侣”的武学，且有“神雕”的陪伴。两人一见如故，决定结为兄弟，共同闯荡江湖。

不久后，两人在江湖上结识了三位大侠，分别是“黄蓉”、“郭靖”和“段誉”。五人志同道合，决定一起闯荡江湖，为民除害。

在一次冒险中，他们遭遇了一群邪恶的敌人，这股势力名为“逍遥派”，首领是“东方不败”。东方不败的武功高强，且掌握了一种名为“快刀”的绝技。这使得逍遥派在江湖上无人敢惹，成为了江湖上的一股强大势力。

为了对抗逍遥派，五人决定研究“快刀”的奥秘。经过一段时间的努力，他们终于发现了“快刀”的精髓所在。原来，“快刀”并非一种武功，而是一种神奇的算法，能够快速地对数据进行排序。这种算法名为“快速排序”，乃是江湖上一种神奇而强大的武功。

掌握了“快速排序”的五人，实力大增。他们决定联合起来，挑战逍遥派。经过一番激战，五人终于战胜了逍遥派，将东方不败击败，为民除害。从此，江湖上再也没有人敢小觑五人的实力。
耗时：18077 ms

谢谢大家！

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS