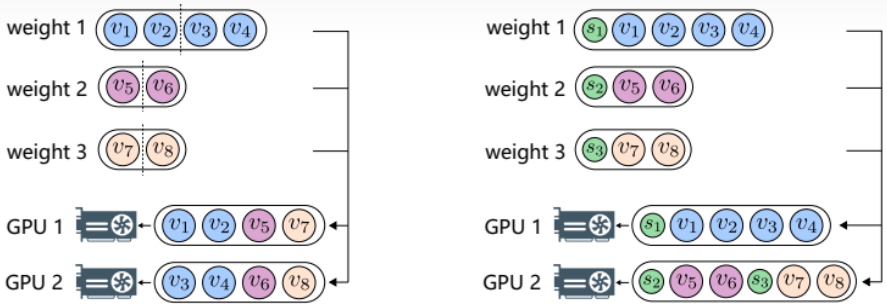


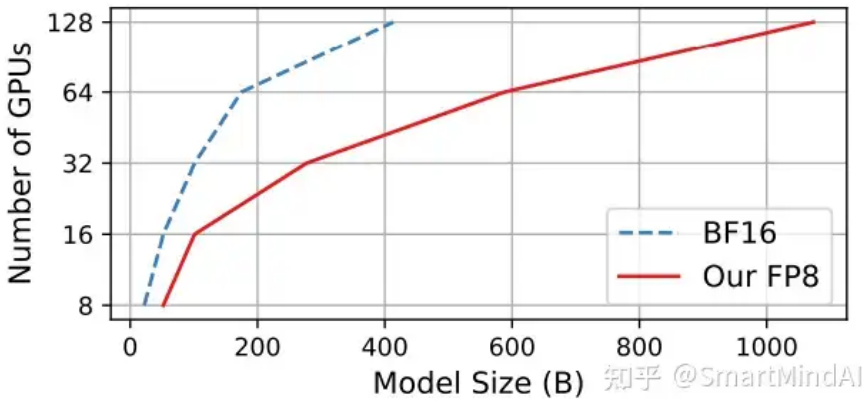
微软2023LLM新作FP8-LM，创业者福音，实现大模型训练成本降低50%《用FP8训练大模型》

SmartMindAI
专注搜索、广告、推荐、大模型和人工智能最新技术，欢迎关注我
已关注

论文原文《FP8-LM: Training FP8 Large Language Models》

Introduction

大型语言模型(LLMs)在语言理解和生成方面展示了前所未有的能力，如GPT-4、Palm和OPT等，但训练成本极高。例如，PaLM需要6144个TPUv4芯片来训练一个540B模型，而GPT-3 175B在预训练时需要数千petaflop/s-天的计算量。因此，降低LLMs的训练成本，特别是对于下一代超智能模型的扩展至关重要。低精度训练是降低成本最有希望的方向之一，因为它可以提供高速度、小内存占用和低通信开销。



随着Nvidia H100 GPU的发布，FP8成为下一代低精度数据类型，如H100白皮书和FP8-DL所示。理论上，与当前的16位和32位浮点混合精度训练相比，FP8可以实现2倍的速度提升、50% -75%的内存成本节约和50% -75%的通信成本节约。 不幸的是，目前对FP8训练的支持很少且有限。唯一可用的框架是Nvidia Transformer Engine（TE），但该框架仅将FP8应用于GEMM计算，并且仍然保留使用高精度（例如FP16或FP32）的主权重和梯度。因此，对于端到端的速度提升、内存和通信成本节省非常有限，没有充分揭示FP8的优势。

为了解决这个问题，我们提出了一种极其优化的FP8混合精度框架，用于LLM训练。该框架的核心思想是将FP8计算、存储和通信渗透到大型模型训练的整个过程中，使前向和后向传递都使用低精度FP8，从而与以前的框架相比，大大减少了系统工作负载。 具体而言，我们设计了三个优化级别，利用FP8来简化混合精度和分布式训练。这三个级别以递增的方式逐步整合8位集体通信、优化器和分布式并行训练。更高的优化级别表示在LLM训练期间使用更多的FP8。此外，对于大规模训练，例如在数千个GPU上训练GPT-175B，我们的框架提供了FP8低比特并行，包括张量、管道和序列并行，为下一代低精度并行训练铺平了道路。

使用FP8训练LLM并不简单，我们面临数据下溢或上溢等挑战，以及由FP8数据格式固有的更窄动态范围和更低精度引起的量化误差。这些挑战可能导致数值不稳定和不可逆的发散。为解决这些问题，我们提出两种技术：精度解耦和自动缩放以防止关键信息的丢失。前者将数据精度对权重、梯

和上溢情况。 为验证提出的FP8低精度框架的有效性，我们将其应用于GPT风格的模型训练，包括预训练和监督微调(SFT)。

实验结果表明，与流行的BF16混合精度训练方法相比，我们的FP8方法显著降低了实际内存使用量（例如，GPT-7B降低27%，GPT-175B降低42%），以及权重梯度通信开销（降低63%至65%）。在不改变任何超参数（例如学习率和权重衰减）的情况下，使用FP8训练的模型在预训练和下游任务中的性能与使用BF16高精度的模型相当。在GPT-175B模型的训练过程中，我们的FP8混合精度框架将训练时间减少了17%，同时在H100 GPU平台上消耗的内存减少了21%。更重要的是，随着模型规模的扩大，使用低精度FP8实现的成本降低将会进一步提高，如图1所示。对于微调，我们采用FP8混合精度指令调优和强化学习与人类反馈（RLHF）来更好地将预训练的大型语言模型与最终任务和用户偏好对齐。

具体来说，我们使用公开的用户共享指令数据对预训练模型进行微调。经过FP8混合精度调整的模型在AlpacaEval和MT-Bench基准测试上的性能与使用半精度BF16的模型相当，同时训练速度提高了27%。此外，FP8混合精度在RLHF中具有相当大的潜力，这个过程需要在训练期间加载多个模型。通过在训练中使用FP8，主流的RLHF框架AlpacaFarm可以减少46%的模型权重和62%的优化器状态内存消耗。这进一步证明了我们的FP8低精度训练框架的多功能性和适应性。

- 提出了一种新型的FP8混合精度训练框架，该框架以附加的方式逐步解锁8位权重、梯度、优化器和分布式训练。这个8位框架可以作为现有16/32位混合精度对等体的简单替换，无需对超参数和训练收据进行任何更改。此外，我们还提供了一个PyTorch实现，可以在几行代码中实现8位低精度训练。
- 提出了FP8方案，并将其应用于GPT预训练和微调（即SFT和RLHF），证明了其在各种模型规模上的潜力，从7B到175B参数。我们为流行的并行计算范式配备了FP8支持，包括张量、管道和序列并行性，使得FP8可以用于训练大型基础模型。我们开源了第一个FP8 GPT训练代码库，基于Megatron-LM 实现。

我们希望FP8框架的发布能为新一代低精度训练系统树立新的典范，该系统致力于大型基础模型。

FP8 LLMs

混合精度计算已被广泛用于LLM训练，以提高计算和内存效率。最流行的混合精度方案是FP16-FP32和BF16-FP32。然而，由于FP16的数值范围有限，FP16-FP32方案在训练大型模型时存在不稳定性。因此，社区现在通常采用BF16-FP32来训练LLMs，例如Megatron-Turing NLG-530B、Bloom-175B和Gopher。BF16具有宽动态范围，在保持数值稳定的同时，与全精度FP32的性能相匹配。此外，与FP32相比，BF16使用的位数减半，从而大大减少了内存占用，同时提高了计算效率。目前，FP8低精度训练的支持是有限的。Nvidia TE仅支持Transformer线性层的FP8计算，而其他所有操作，如权重更新和梯度同步，仍然使用更高的精度。为了解决这个问题，我们提出了一种极致优化的FP8混合精度策略，用于LLM训练。该策略包括三个关键方面：FP8通信、FP8优化器和FP8分布式训练。通过整合这些方面，175B GPT-3等LLM模型的训练可以充分利用FP8低精度的优势，提高训练效率。

FP8 Gradient and All-Reduce Communication

现有的混合精度训练方法通常使用16位或32位数据类型进行梯度的计算和存储，导致整个训练过程中的集体通信具有高带宽要求。我们发现直接将FP8应用于梯度会导致精度的降低。根本问题在于低比特全减操作引起的下溢和上溢问题。具体来说，在全减过程中，跨GPU聚合梯度有两种标准方法：预缩放和后缩放。预缩放将第i个GPU上计算的梯度g_i在求和之前除以GPU的总数（即N），其公式为：

$$g = g_1/N + g_2/N + \cdots + g_N/N.$$

当N非常大时，除法可能导致数据下溢，尤其是对于FP8低精度梯度表示。为了缓解这个问题，后置缩放（post-scaling）在梯度收集过程中首先进行梯度求和，然后在除法缩放时进行除法。

$$g = (g_1 + g_2 + \cdots + g_N)/N.$$

采用后缩放方法，将梯度保持在FP8数据类型的最大值附近，有效缓解了underflow问题。但在聚合梯度时，该方法遇到了overflow问题。 为解决前缩放和后缩放方法的underflow和overflow问

知乎

$$g_i' = \mu \cdot g_i.$$

我们对g_i'的梯度值进行统计分析，以量化在FP8表示范围内达到最大可行值的比例。如果最大值的比例超过0.001%，则后续训练步骤中，μ设为1/2，以降低溢出风险。若比例持续低于阈值，则在1000个训练步骤内将μ指数级增加到2，以降低下溢风险。 FP8集体通信的另一障碍在于管理每个梯度张量相关的张量缩放因子。当前NCCL实现缺乏考虑额外张量缩放因子的全减操作能力。同时，有效的实现极具挑战性，特别是考虑到NCCL梯度求和是在子张量级别上进行的。当考虑到张量缩放因子的更新时，复杂性显著增加。为解决此问题，我们提出一种新机制，使用共享标量来缩放FP8梯度跨GPU。具体来说，将第i个GPU中的权重梯度存储在名为(g_i', s_i')的缩放张量中，其中g_i'为FP8张量，s_i'为缩放因子。实际权重梯度为g_i'/s_i'。在进行所有GPU上的梯度张量的全减操作之前，我们首先收集所有GPU上每个梯度张量的缩放因子s_i'并计算全局最小缩放因子s_g'如下：

$$s_g' = \min\left(s_1', s_2', \ldots, s_N'\right),$$

全局最小缩放因子 s_g' 在GPU之间共享，用于统一GPU之间梯度张量的缩放。这样，与同一权重关联的所有梯度张量都使用相同的共享缩放因子进行量化，以便在所有GPU上使用FP8格式。

$$g_i'' = \mathrm{FP8}\left(s_g' \cdot \left(g_i'/s_i'\right)\right).$$

这种做法可以有效减少通信开销，只传输单个标量s_g'。由于输入张量共享相同的缩放因子，并行缩放因子无需考虑，可执行标准的NCCL all-reduce操作。最终收集到的梯度如下：
$g = g_1'' + g_2'' + \cdots + g_N'', \qquad s = N \cdot s_g'$, g 表示最终聚合的梯度，s 是相应的缩放因子。

理论上，重新调整梯度总和 g 的缩放因子相当于将 g 除以 N。通过实现上述分布式和自动缩放的双重策略，我们成功实现了 FP8 低位梯度通信，同时保持了模型精度。这种方法还涉及将梯度存储在 FP8 中，并在 FP8 中进行通信，从而减少 GPU 内存使用和通信带宽消耗。

FP8 Optimizer

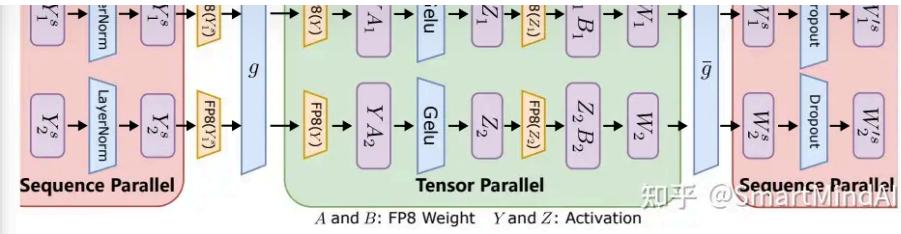
在大型语言模型（LLM）训练中，Adam及其变体是常用的优化方法，它们维护模型权重的副本、梯度、一阶和二阶梯度矩，以便进行模型更新。使用Adam优化器的混合精度训练通常以32位浮点数格式存储主权重、梯度和梯度矩，以保持数值稳定性。因此，在训练过程中，Adam优化器每个参数消耗16字节的内存。

$$\underbrace{4}_{\text{master weights}} + \underbrace{4}_{\text{gradients}} + \underbrace{4 + 4}_{\text{Adam states}} = 16 \text{ bytes}.$$

当模型规模很大时，Adam中变量的内存消耗将成为瓶颈。先前的研究工作 已经表明，将优化器中变量的精度降低到16位会导致训练数十亿规模模型的准确性下降（BF16缺乏所需的精度，而FP16的动态范围有限。

在这些挑战下，流行的混合精度训练方法依赖于使用FP32全精度作为主权重、梯度和梯度时刻。）。这促使我们评估优化器中的哪些变量应该分配高精度，哪些变量可以适应低精度。 为了阐明这一点，我们将把数据精度对优化器中变量的影响解开，并调查哪些变量可以分配较低的精度，即"精度解耦"。我们发现了一个指导原则：梯度统计可以使用较低的精度，而主权重需要高精度。

更具体地说，一阶梯度矩可以容忍高的量化误差并可以被分配低精度的FP8，而二阶矩需要更高的精度，如我们在第部分所分析的那样。这是因为在Adam的模型更新过程中，梯度的方向比其大小更重要。具有张量缩放的FP8可以有效地保留一阶矩的高精度张量分布，尽管它在一定程度上引入了精度下降。计算二阶矩的平方可能会导致由于通常小的梯度值而出现数据下溢。因此，分配16位的高精度是必要的，以保持数值准确性。 另一方面，我们发现保持主权重的高精度是很重要的。根本原因是训练过程中权重更新有时会变得非常小或非常大，主权重的高精度有助于防止在更新权重时丢失信息，确保更稳定和准确的训练。



在实现中，主权重有两个可行的选项：使用FP32全精度或具有张量缩放的FP16。具有张量缩放的FP16在不损害精度的同时提供了节省内存的优势。因此，我们的默认选择是在优化器中使用具有张量缩放的FP16来存储主权重。我们的FP8混合精度优化器在训练期间每参数消耗6字节的内存：

$$\underbrace{2}_{\text{master weights}} + \underbrace{1}_{\text{gradients}} + \underbrace{1+2}_{\text{Adam states}} = 6 \text{ bytes}.$$

这种新的低比特优化器将内存减少了2.6倍，与之前的解决方案相比，如式所示。值得注意的是，这是第一个用于LLM训练的FP8优化器。实验表明，FP8优化器可以在各种规模上保持模型准确性，范围从1.25亿到1750亿参数。

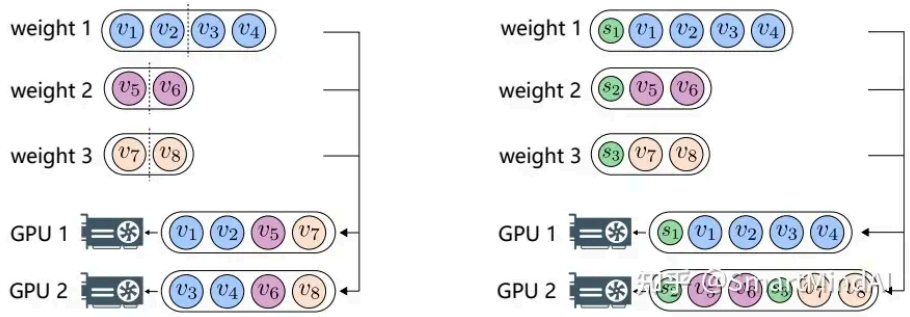
FP8 Distributed Parallel Training

大型语言模型（LLM）的训练，例如GPT-3，需要分布式学习策略来在GPU上进行并行化。常用的策略包括数据并行、张量并行、流水线并行和序列并行。每种并行策略都有自己的优点，并且在现有系统中以互补的方式使用 。对于这些策略的FP8支持，数据并行或流水线并行不需要进行任何特定的修改，因为它们在将数据批量或模型层分割成跨设备的片段时，不涉及额外的FP8计算和通信。 张量并行将模型的单个层分割到多个设备上，使得权重的shard、梯度和激活张量位于不同的GPU上，而不是单个GPU上。为了使张量并行具有FP8功能，我们将分片的权重和激活张量转换为FP8格式，用于线性层计算，使前向计算和后向梯度集体通信都使用FP8。

另一方面，序列并行将输入序列分成多个块，并且将子序列馈送到不同的设备以节省激活内存。如图 所示，序列并行和张量并行在Transformer模型的不同部分并行执行，以充分利用可用内存并提高训练效率。在序列并行区域和张量并行区域之间有一个转换器g，用于在前向传递中收集所有序列分片（或后向传递中减少张量段的散布）。我们在g之前添加了一个FP8数据类型转换，使得所有收集操作（或减少散布操作）使用FP8低位激活来节省跨GPU的通信成本。

此外，零冗余优化器（ZeRO）是大型模型训练中另一种常用的分布式学习技术。ZeRO的核心思想是在设备上对模型状态进行遮蔽，以便每个设备只持有训练步骤所需的一部分数据（例如，主权重、梯度和优化器状态）。为了减少内存消耗，ZeRO方法通常将单个张量分成多个分区，并将它们分发到不同的设备上。直接将FP8应用于ZeRO是不可行的，因为难以处理与FP8分区相关的缩放因子。每个张量的缩放因子应该与FP8分区一起分布。

为了解决这个问题，我们实现了一个新的FP8分布方案，该方案将每个张量作为整体跨设备分布，而不是像ZeRO中那样将其分割成多个子张量。 具体来说，我们的方法首先根据大小对模型状态的张量进行排序，然后根据每个GPU的剩余内存大小将张量分配给不同的GPU。分配遵循的原则是剩余内存较大的GPU在接收新的分布式张量时具有更高的优先级。这样，张量的缩放因子可以与张量一起平滑分布，同时降低通信和计算复杂度。图展示了具有和不具有缩放因子的ZeRO张量分区的不同之处。



我们评估了所提出的FP8混合精度训练方法在GPT风格的LLM上的有效性，包括从1.25亿到1750亿参数的广泛模型规模。为了进行性能消融实验，我们将使用FP8训练的GPT模型与使用半精度BF16和全精度FP32训练的GPT模型进行比较。为了进行一般性评估，我们进行了涵盖FP8低比特预训练和微调的实验，同时考虑了指令调整和人类偏好对齐。

Algorithm 1 Greedy Distribution Algorithm for ZeRO

Input: FP8 tensors with their corresponding scaling factors: $T = \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$, where s denotes scaling factors while t represents 8-bit tensors. The size of each tensor: $C = \{c_1, c_2, \ldots, c_n\}$.
Output: Partitions representing scaling tensors assigned to each GPU.
1: Sort T in descending order of their sizes to get $T' = \{(s'_1, t'_1), (s'_2, t'_2), \ldots, (s'_n, t'_n)\}$ and $C' = \{c'_1, c'_2, \ldots, c'_n\}$, where $c'_1 \geqslant c'_2 \geqslant \cdots \geqslant c'_n$.
2: Initialize memory usage $u_j = 0$ and partition $p_j = \emptyset$ for each GPU G_j.
3: **for** $i = 1$ to n **do**
4: $\quad j \leftarrow \arg\min_j u_j$ ▷ Find the GPU $j \in [1, m]$ with the least memory usage.
5: $\quad p_j \leftarrow p_j \cup \{(s'_i, t'_i)\}$ ▷ Assign (s'_i, t'_i) to G_j.
6: $\quad u_j \leftarrow u_j + c'_i$ ▷ Update the memory usage of G_j.
7: **end for**
8: **return** Partitions $P = \{p_1, p_2, \ldots, p_m\}$

Experimental Setup

Training Dataset

我们的预训练数据由多个开源语言集合构建，包括CommonCrawl、The Pile、C4、OpenWebText、CC-NEWS、CC-Stories、Redpajama和Wikipedia等来源。我们应用模糊去重方法来增强数据质量。对于使用人类反馈的强化学习，我们使用的训练数据是Anthropic的有益和无害数据集与Open-Assistant数据集的组合。训练框架和相关配置与公开可用的AlpacaFarm对齐。

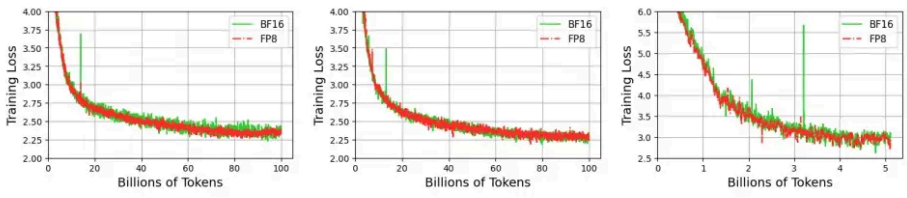
Model Configuration

我们采用了仅解码器Transformer架构，并整合了几个改进方案，以提高模型效率和效果。其中包括：

1. 旋转位置嵌入（RoPE）：这一增加使我们能够捕捉到绝对和相对位置信息，尤其是在推断更大的上下文窗口时，性能得到增强。

2. Flash注意力：标准注意力实现受到内存访问的瓶颈限制。Flash Attention提出了一种IO感知的精确注意力算法，该算法使用平铺来减少HBM访问量，实现了实质性的加速。

我们使用提出的FP8优化器对模型进行训练，该优化器基于Adam构建，采用解耦的权重衰减，遵循常见的实践。其中衰减率β_1 = 0.9, β_2 = 0.95, 权重衰减= 0.1。学习率计划是余弦式的，最终学习率是最大学习率的10%。我们总共训练了100B个令牌的模型，批量大小为4M个令牌，输入序列长度设置为2048。模型预热进行了1000次迭代。列出了模型配置和相应的训练设置的详细信息。训练在Azure NDv5 H100 GPU平台上进行。



(a) GPT-7B (b) GPT-13B (c) GPT-175B

Figure 4: A comparison between FP8 and BF16: Analyzing the training loss of GPT models with the parameters ranging from 7 billion to 175 billion.

	HS	Lambada	BoolQ	PIQA	COPA	Winogrande	Arc-C	Arc-E	ObQA	Avg
GPT-7B model zero-shot performance										
BF16	61.3	61.4	61.2	75.0	79.0	58.5	32.9	59.7	36.4	58.4
FP8	60.0	61.8	62.0	74.2	78.0	59.8	32.9	58.7	34.6	58.0
GPT-13B model zero-shot performance										
BF16	64.8	64.9	63.4	75.9	82.0	61.0	35.2	61.5	40.6	61.0
FP8	64.1	63.4	63.9	76.2	81.0	61.6	34.9	61.3	36.8	60.4

Table 2: Zero-shot performance on downstream tasks. The models are trained with either the standard BF16 mixed-precision scheme (Shoeybi et al., 2019) or the proposed FP8 low-precision scheme.

知乎

Model Performance

我们比较了使用FP8混合精度训练的模型与使用BF16训练的模型的表现。在图1中，展示了7B、13B和175B参数的GPT模型的预训练损失。使用FP8和BF16训练的模型的训练配置和超参数保持一致。唯一的区别在于所使用的混合精度方案。如图1所示，损失曲线几乎重叠在一起。结果明确地表明，所提出的FP8混合精度方案可以在各种模型规模上实现与更普遍使用的高精度BF16方案相当的性能。

此外，我们在广泛的下游任务上评估了预训练模型，包括HellaSwag（HS）、Lambada、BoolQ、PIQA、COPA、Winogrande、Arc和OpenbookQA（ObQA）。如表1所示，FP8预训练模型与BF16预训练模型相比，表现出相当的零样本性能。这一结果表明，使用FP8低精度进行预训练的模型在准确性和内在上下文学习能力方面与高精度模型相当。此外，我们还利用提出的FP8混合精度方法进行指令遵循的LLM微调。

为了公平比较，我们遵循与Vicuna-v1.1相同的指令调整设置，该设置采用开源的LLaMA-7B作为微调的基础模型。图5显示了微调损失，其中BF16和FP8曲线明显重叠。与此同时，我们的FP8微调模型对Davinci-003的胜率与使用BF16半精度微调的Vicuna-v1.1相当，如表7所示。这表明我们的FP8低比特训练方案不仅适用于预训练阶段，也适用于下游微调任务，具有广泛的适用性。

此外，我们还进一步将提出的FP8混合精度方案应用于人类反馈强化学习（RLHF），这是一个更复杂的过程，旨在使LLM与用户偏好保持一致。我们遵循与AlpacaFarm相同的训练设置，AlpacaFarm是一个用于LLM对齐的最新RL框架，我们使用PPO算法优化策略模型。唯一的区别在于混合精度训练方案的选择，即BF16与FP8。我们从图6和表8中报告的结果观察到内存使用量有了显著的减少，例如模型权重减少了31.8%，优化器状态减少了62.5%。因此，我们可以推断出FP8能够复制BF16混合精度用于RLHF训练。这强调了我们的FP8低比特训练解决方案更广泛的适用性和通用性。

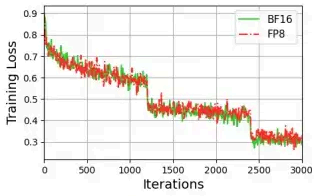


Figure 5: SFT training loss.

Mixed-precision	System Performance		Model Performance	
	GPU Mem. (GB)	Throughput	AlpacaEval	MT-Bench
BF16	51.1	103	66.15	5.75
FP8	44.0(-13.9%)	131(+27.2%)	67.20	5.70

Table 3: A comparison between FP8 and BF16 for SFT. For system performance, we report results of GPU memory usage and training throughput. For model performance, we present the win-rate against Davinci-003 on AlpacaEval and GPT-4 judged scores on MT-Bench.

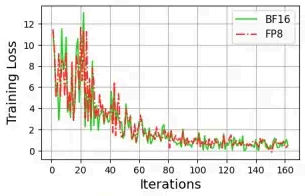


Figure 6: RLHF training loss.

Mixed-precision	Memory Usage (MB)		Model Performance	
	Weights	Optimizer States	AlpacaEval	MT-Bench
BF16	15,082	15,116	72.05	6.16
FP8	10,292(-31.8%)	5,669(-62.5%)	72.42	6.04

Table 4: A comparison of FP8 and BF16 RLHF alignment. Memory usage is assessed with a focus on weights and optimizer states, while model performance is evaluated on AlpacaEval considering win-rate against Davinci-003 and MT-Bench using GPT-4 judged scores.

System Performance

我们评估了FP8混合精度的系统级性能，并考虑了通信效率、内存利用率和整体速度，同时强调了成本节约。我们的方法采用8位梯度，在GPU之间的所有reduce集体通信中。理论上，与主流的32位方案相比，这将使通信成本降低75%（尽管BF16混合精度计算使用16位精度的梯度，但仍然对所有reduce通信采用32位精度）。

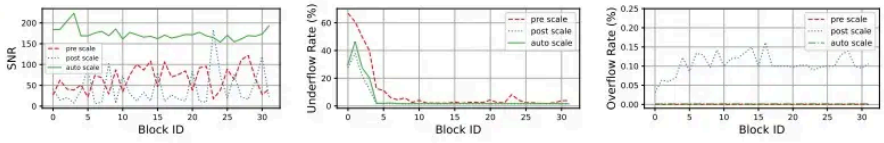
在GPT模型训练期间，由于系统传输损失的影响，实际降低幅度在63%到65%之间，如表所示。另外值得注意的是，最近的Nvidia Transformer Engine仍然依赖于全精度FP32进行集体通信，导致我们的FP8解决方案有相同程度的降低。在相同批量大小的GPT模型训练中，FP8混合精度相较于BF16可减少内存占用27%到42%，如表1所示。这些减少归功于FP8梯度和优化器的引入。此

外，与TE相比，我们的解决方案在模型大小为GPT-7B、13B和175B时，实现了34.2%、35.4%和44.8%的额外内存减少。尽管TE使用了FP8计算，但其优化器和梯度使用了高精度，导致更多内存消耗。节省的内存可用于训练更大批量或更长序列。例如，使用32个具有80G内存容量的H100 GPU时，我们的方法可训练具有4096个令牌上下文的模型，最多容纳1750亿个参数，而TE只能容

此外，与流行的BF16方案相比，我们的FP8混合精度方案在GPT-175B模型上的训练速度提高了64%。FP8混合精度训练的模型FLOPS利用率为H100 GPU上的32.0%，比TE高17.2%。这些发现提供了大量证据表明我们的FP8方案有效地节约了内存，减少了大型模型训练过程中的通信成本，并最终提高了最新H100 GPU平台上的系统利用率效率。

Model	TP	PP	DP	Micro BS	Mixed Precision	GPU Mem. (GB)	Throughput (#samples/s)	TFLOPS	MFU (%)	Weight-related Comm. Rate (%)	Weight-related Comm. Volume (GB)
GPT-7B	1	1	32	2	BF16	69.6	158.1	442	44.7	9.2	37.2
				2	FP8 (TE)	77.2	219.4	613	31.0	9.5	37.2
				2	FP8 (Ours)	50.8 (-27%)	196.2 (+24%)	547	27.6	6.3	13.9 (-63%)
				4	FP8 (Ours)	69.0	215.6 (+36%)	602	30.4	8.3	13.9 (-63%)
GPT-13B	2	1	16	2	BF16	68.4	78.2	415	42.0	11.4	34.3
				2	FP8 (TE)	73.8	111.8	593	30.0	7.9	34.3
				2	FP8 (Ours)	47.7 (-30%)	99.3 (-27%)	498	25.9	5.8	12.4 (-64%)
				4	FP8 (Ours)	65.4	113.1 (+45%)	600	30.3	6.7	12.4 (-64%)
GPT-175B	8	4	4	1	BF16	63.4	22.5	388	39.2	8.8	23.4
				1	FP8 (TE)	66.7	31.4	541	27.3	4.3	23.4
				1	FP8 (Ours)	36.8 (-42%)	24.3 (+8%)	418	21.1	1.6	8.2 (-65%)
				4	FP8 (Ours)	52.3	36.8 (+64%)	634	32.0	8.9	8.2 (-65%)

Table 5: System-level performance on Nvidia H100 GPUs 80G. Here, TP, PP, and DP represent tensor, pipeline, and data parallelism respectively. BS indicates batch size, while MFU denotes model FLOPs utilization. Weight-related communication contains the all-gather operator on weights and the reduce-scatter operator on weight gradients.



(a) SNR (Signal to Noise Ratio) (b) Underflow rate (c) Overflow rate

Figure 7: Comparing different strategies, *i.e.*, pre-scaling, post-scaling, and auto-scaling, for FP8 gradient all-reduce. We investigate SNR, underflow rate, and overflow rate across different Transformer blocks. The experiment is conducted using a GPT-7B model with a data parallelism factor of 128.

Ablation Study

我们对大型语言模型(LLM)的FP8混合精度训练策略进行消融实验，并在表格和图中报告性能。消融实验在GPT模型上进行，架构和训练设置详见表格。重要的是，我们的消融研究为在LLM训练中有效利用8位数据类型提供了指导方针，有助于未来低位模型训练的研究。

优化器： 我们消融了AdamW优化器中变量减少精度的影响。我们选择BF16混合精度优化器作为基准，因其已广泛应用于现有LLM训练框架。表列出变量的减少精度设置，图描绘了相应的训练损失。我们观察到：

1）FP8 master weight会导致性能下降，而FP16可保持与FP32相同的精度（参考红色#2与青色#0和蓝色#1），但需使用张量缩放。这表明master weight对精度很敏感。这可归因于master weight在更新权重时的作用，其往往会表现出小幅度，需高精度以保持精度。

2）第二阶梯度矩比第一阶更敏感，因为平方计算容易造成下溢并导致精度下降。使用FP8进行第二阶梯度矩计算可能导致训练损失发散（参考图）。 并行性： 在我们的FP8 LLM训练框架中，引入了FP8低比特转换器以减少跨GPU的激活通信成本，包括序列并行性和张量并行性。

在此，我们进行了一项分析实验，计算GPT模型训练期间与激活相关的通信量，并在中报告了相关数字。观察到与使用BF16的原始方法相比，我们的FP8并行方案使与激活相关的通信成本减少了33%。此外，在ZeRO分布式训练中，将每个FP8张量及其相关缩放因子作为一个整体分布，而非将张量分割成跨GPU的片段。此策略不仅节省了更多的GPU内存，还保持了GPU之间的平衡内存负载，如所示。

知乎

FP32 #0	FP32	FP32	FP32	FP32+FP32
BF16 #1	BF16	FP32	FP32	FP32+FP32
FP8 #2	FP8	FP8	FP16	FP8+FP16
FP8 #3	FP8	FP8	FP8	FP8+FP16
FP8 #4	FP8	FP8	FP16	FP8+FP8

Table 6: Precision decoupling for the variables within the optimizer. Here, our focus is on ablating the master weight and optimizer states, as these components are precision sensitive. The optimizer states include both first-order and second-order gradient moments. Note that the FP16 master weight uses tensor scaling.

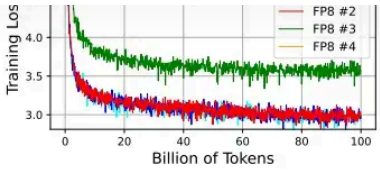


Figure 8: Training losses of GPT-125M models with the settings ~~ablated~~ in Tab. 6. The loss curve for FP8 #4 has diverged.

Conclusion

我们探索了LLM的8位训练。我们提出了一种新的FP8混合精度训练框架，该框架以递增的方式整合了8位集体通信、优化器和分布式并行训练。据我们所知，这是第一项将FP8计算、存储和通信渗透到大型语言模型训练全过程的工作。大量实验表明，在各种规模的GPT模型训练中，提出的方法有效地减少了通信开销并降低了内存利用率。在未来的工作中，我们计划增加FP8 GPT模型的大小和训练步骤，并使用我们的8位混合精度方案进行进一步训练。此外，我们还计划将提出的FP8方案用于训练多模态大型模型，并探索在各种边缘设备（如智能手机）上部署低位LLM。

编辑于 2023-11-04 10:51 · IP 属地北京

大模型 LLM（大型语言模型） chat GPT

理性发言，友善互动

还没有评论，发表第一个评论吧

推荐阅读

科研实习丨微软WizardLM大模型团队招聘大模型研究型/...

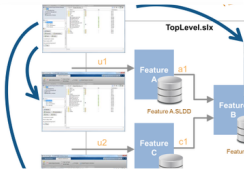
英国学术科... 发表于英国学术科...

湾区应届生薪资鄙视链流出，Meta：别骂了

WSTCareer

微软钦点OpenAI备胎：GPT-4级大模型上线即挤爆，成本仅...

量子位 发表于量子位



VCU/BMS基于模型开发Simulink Project之团队

屌丝小蚂蚁 发表于汽车技