NLP方向总结-显存优化

显存占用

模型参数(parameters)

与模型大小有关，层数，隐藏神经元等，就是weight和bias的占用

开启fp16

模型权重，模型中间结果，和梯度均用fp16存储，权重weights需要用 fp32 的格式额外备份一次，更新权重时使用fp32的即可

基本上这3部分显存减半

为了保证实际运算过程中的精度，一般还会配合动态放缩技术

多GPU训练时

模型拆分，按照模型宽度（张量并行）或深度进行拆分（将不同层放在不同的GPU上）

算子融合

将若干个 CUDA 上的运算合成一个运算，本质上是减少了 CUDA 上的显存读写次数

https://pytorch.org/docs/stable/generated/torch.jit.script.html

模型参数的梯度(gradients)

开启fp16

优化器状态(optimizer states)

例如，Adam优化器使用每个参数的exp_avg和exp_avg_sq状态。因此，Adam优化器的内存消耗至少是模型大小的两倍

多GPU并行训练

Optimizer State Sharding: 优化器状态分片，优化器的使用显存减小一半

半精度Adam统计

8 bit Optimizer

低精度量化方式存储优化器状态的优化器

将优化器状态量化至 8 bit 的空间，并通过动态的浮点数表示来降低量化的误差

模型中间结果(intermediate results)，也可以叫中间激活,特征图

开启fp16

修改输入

图片大小

文本长度

batch_size 大小

batch_size变小后可以使用梯度累加来模拟大的batch_size，例如batch_size=8, accumulation_steps=2, 实际的batch_size就是16

梯度checkpoint

通过在前向传递完成前丢弃中间激活来节省内存,在反向过程中，这些激活会根据需要再次被重新计算,以时间换取内存

https://arxiv.org/abs/1604.06174

https://pytorch.org/docs/stable/checkpoint.html

多GPU并行时，使用DDP方式，即数据并行

其它方式

不需要的梯度计算的信息，放到CPU上，例如loss值，

删掉临时创建的不需要的tensor

in-place运算

ZeRO

微软的 DeepSpeed 团队解决数据并行的中存在的内存冗余问题所提出的解决方法

常驻在每块GPU上的数据可以分为三部分：模型参数，模型梯度和优化器参数，注意到由于每张GPU 上都存储着完全相同的上述三部分参数，我们可以考虑每张卡上仅保留部分数据，其余的可以从其他 GPU 上获取

https://engineering.fb.com/2021/07/15/open-source/fsdp/

https://pytorch.org/docs/stable/fsdp.html

ZeRO-Offload

主要思想是将部分训练阶段的模型状态 offload 到内存，让 CPU 参与部分计算任务

只将模型更新的部分下放至 CPU 计算，即让 CPU 充当 Parameter Server 的角色

GPU进行计算梯度，反向传播，CPU更新模型参数，然后把参数给GPU

多GPU分布式训练的划分

模型并行：不同的GPU输入相同的数据

流水线并行（Pipeline Parallel）

按照深度进行拆分（将不同层放在不同的GPU上）

按照模型宽度（张量并行）放到不同GPU

但朴素的流水线并行实现会导致GPU使用率过低（因为每块GPU都要等待之前的GPU计算完毕才能开始计算）

解决方案：GPipe和PipeDream

GPipe 方法的核心思想便是输入的minibatch划分成更小的micro-batch

PipeDream采取了类似异步梯度更新的策略，即计算出当前 GPU 上模型权重的梯度后就立刻更新，无需等待整个梯度回传完毕

数据并行DDP: Distributed Data Parallel：

不同的GPU输入不同的数据，运行相同的完整的模型

每个GPU上存储的模型、优化器状态是完全相同的

相当于增大了batch-size，

每个GPU训练step结束后，将每块GPU的损失梯度求平均

张量并行（Tensor Parallel）

在训练大模型的时候，通常一块GPU无法储存一个完整的模型。张量并行便是一种使用多块GPU存储模型的方法

张量并行是针对模型中的张量进行拆分，将其放置到不同的GPU上

对于矩阵A有按列拆解和按行拆解两种方式

Y=AX=X[A1,A2]=[XA1,XA2]

y=[X1,X2] [A1 A2] = X1A1+X2A2

可以将矩阵A1和A2分别放置到两块不同的GPU上，让两块GPU分别计算两部分矩阵乘法，最后再在两张卡之间进行通信便能得到最终的结果。

https://arxiv.org/abs/2104.05343

https://arxiv.org/abs/2105.14500

https://arxiv.org/abs/2105.14450

https://arxiv.org/abs/2105.13120

训练步骤

模型定义：定义了模型的网络结构，产生模型参数；

前向传播：执行模型的前向传播，产生中间激活值,激活值就是上一层模型的输出。

后向传播：执行模型的后向传播，产生梯度；

梯度更新：执行模型参数的更新，第一次执行的时候产生优化器状态。

在模型定义完之后，2~4循环执行。

论文

Training Deep Nets with Sublinear Memory Cost

显存优化算法

本文主要关注降低存储中间结果(特征图）和梯度的内存消耗

本文还是使用的静态图进行的计算，对于torch和tf2.0需要重新考虑

本文的主要思想是用时间换空间

显存占用

模型参数

eg：Weight和bias等

优化器参数

梯度

本文优化方向

模型每层输入和输出

本文优化方向

本文优化显存方法：图1第3部分

in-place运算。直接将输出值存储到输入值的存储器中

内存共享。不再需要的中间结果所使用的内存可以被回收，并在另一个节点使用。

放弃一些中间计算结果，并在需要时从额外的前向计算中恢复。

丢掉的是低运算成本的结果

实现思路：图2

用一个活跃度计数器表明该节点是否被关联依赖，当仅剩唯一关联，即计数器等于1时，用inplace运行，当计数器为0时，该节点内存被回收

实际的表现是前向传播时，哪些节点是否需要被镜像保留

其它

量化Quantization

模型量化压缩技术

量化是通过从一个数据类型到另一个数据类型的 “四舍五入 ”来完成的，会导致信息损失，是一种有损的压缩，看似微小的错误在通过模型的各层传播时往往会积累和增长，并导致性能下降

BFloat16

BFloat16是一种浮点数格式，它用于表示16位二进制浮点数。它的名称来自于“Brain Floating Point format”，因为它是由谷歌公司的Brain团队开发的。与传统的浮点数格式（如IEEE 754）相比，BFloat16使用更少的位数来表示一个浮点数，这在机器学习和人工智能等计算密集型应用中非常有用，因为它可以减少内存使用和加速计算。

BFloat16的格式是1位符号位，8位指数和7位尾数。它可以表示的范围和精度与IEEE 754单精度浮点数（32位）相似，但是它不能表示与IEEE 754双精度浮点数（64位）相同的精度和范围。BFloat16广泛应用于机器学习框架和芯片设计中，例如TensorFlow和Google的TPU芯片。

可以表示的数字范围比较小，但是相对于FP16而言精度更高。在TensorFlow和Google的TPU芯片等机器学习框架和芯片中广泛使用。
可以用于进行混合精度计算，其中权重和梯度使用FP32或更高精度的浮点数，而激活使用BFloat16，从而提高计算速度和减少内存使用。

FP16

使用16位二进制格式来表示实数，其中5位用于指数，10位用于尾数，1位用于符号。
可以表示的数字范围和精度相对于BFloat16而言略低。
在深度学习等计算密集型任务中，使用FP16可以大大减少内存使用，加快计算速度。
可以用于进行混合精度计算，从而提高计算速度和减少内存使用。

BFloat16相对于FP16而言精度更高，但是支持程度较低，而FP16则具有更广泛的支持程度，但是精度略低

相对于FP16，失去了3比特的精度

FP32

FP16是16位浮点数格式，FP32是32位浮点数格式。

使用16位二进制格式来表示实数。
可以表示的数字范围比较小，但是精度相对较低。
在深度学习等计算密集型任务中，使用FP16可以大大减少内存使用，加快计算速度。
硬件支持程度有限，一些较旧的GPU可能不支持FP16。

使用32位二进制格式来表示实数。
可以表示的数字范围更广，精度更高。
在大多数应用中被广泛使用，因为它提供了足够的精度和数字范围。
硬件支持程度广泛，几乎所有现代CPU和GPU都支持FP32。

指数，尾数

科学计数法表示一个实数，包含一个基数（在计算机中通常为2）和指数，例如：
1.23 x 10^4 表示为 12300，其中基数为10，指数为4。
在计算机中，指数表示实数的幂次，尾数表示实数的小数部分。FP16中的5位指数可以表示-15到+16之间的指数，其中一部分用于表示正的指数，一部分用于表示负的指数，10位尾数可以表示小数点后的10位二进制数，用于表示实数的小数部分。

例如，考虑一个实数1.25，它可以写成1.25 = 1.01 x 2^0，其中指数为0，尾数为0.01（十进制下为0.25）。在FP16中，这个实数可以写成以下二进制表示：

符号位：0（正数）
指数位：01110（14，表示0+15）
尾数位：0100000000（512/1024，表示0.5+0.25=0.75）
将这些位组合起来，得到以下16位二进制数：
0011101001000000。

Huggingface 的LLM.int8()

GPU设备上加载一个模型，每十亿个参数在float32精度下要花费4GB，float16要花费2GB，int8要花费1GB

如果你使用AdamW优化器,每个参数需要8个字节，10亿个参数需要8GB的GPU内存

FP32被称为全精度（4字节），而BF16和FP16被称为半精度（2字节）。在此基础上，int8（INT8）数据类型由8字节表示组成，可以存储2^8个不同的值（对于有符号的整数，在[0, 255]或[-128，127]之间）

如果模型是1760亿参数，用半精度推理，那么需要的GPU内存是1760亿* 2byte = 352GB

BLOOM-1760亿参数需要8个80GB的A100进行推理，如果微调，需要72个

使用LLM.int8()的BLOOM-176B比fp16版本慢了大约15%到23%