Subject: CS336, Lecture 2, Pytorch, Resource Accounting

Date: from July 26, 2025 to August 12, 2025

Contents

Introduction

Youtube：Stanford CS336 Language Modeling from Scratch, Lecture 2: Pytorch, Resource Accounting.

本节主要讲解内存计算问题，首先介绍了 float 32, float 16 等数据类型. 随后介绍 PyTorch 中的 tensor 这一重要的数据类型. 最后举例介绍了在模型训练中各个部分所需要的计算量，并介绍了浮点运算利用率这一指标以衡量硬件计算效率. 重点如下：

1. 在 PyTorch 中 tensor 是对已分配内存的指针，很多操作无需新占用内存
2. 大型矩阵乘法在深度学习所需计算量最大
3. 浮点运算利用率 MFU $= \frac{\text{actual FLOP/s}}{\text{promised FLOP/s}}$
4. 前向传播所需计算量：$2 \times$ (# tokens) $\times$ (# parameters)
5. 反向传播所需计算量：$4 \times$ (# tokens) $\times$ (# parameters)

Memory Accounting

Basic Knowledge – Floating-point

Bit. 二进制位(binary digit)，是信息的最小单位.
Byte. 字节(Byte, B)，1 Byte = 8 Bit. 1 KB = 1024 byte, 1 MB = 1024 KB, 1 GB = 1024 MB

float 32. 也被称为 fp32 或 single precision 或 full precision[a]. 一个 float 32 表示的数需要 4 bytes = 32 bits. 第1位表示正负(0 正 1 负)，中间8位表示指数，后23位储存有效数位. 如图1所示，符号位为 0，表示"+"，指数从右往左计算，为 $2^2 + 2^3 + 2^4 + 2^5 + 2^6 = 124$，需要减去偏移量 127 得 -3，尾数部分根据归一化法实际为 $1 + 2^{-2} = 1.25$，故最终得到表示的 10 进制数为 $(+1) \times 1.25 \times 2^{-3} = 0.15625$.

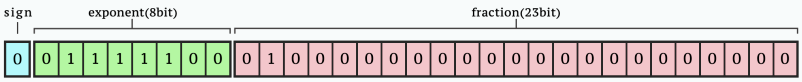

Figure 1: Float 32

float 16. 也被称为 fp16 或 half precision，需要 2 bytes. 但是其动态范围(dynamic range)小，容易造成上溢(overflow)和下溢[b](underflow)，因此不适合表示过大和过小的数.

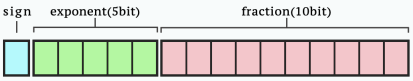

Figure 2: Float 16

bfloat16. 即 brain floating point，由 Google Brain 于 2018 年提出，其使用与 fp16 相同的内存但有 fp32 相同的动态范围. 由于在深度学习中更关注动态范围，因此在前馈计算中常用. 但事实表明存储优化器状态和参数仍需要 fp32.

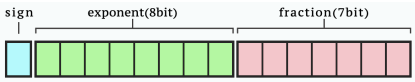

Figure 3: bfloat 16

fp8. 由 Nvidia 于 2022 年提出，其可以动态调配精度和动态范围侧重，在 H100 中支持.

mixed precision training. 在深度学习训练中，对于 pipline 中各个部分（如前向传播、反向传播、优化器等）所需要的最低精度可能不同，例如使用 float 32 用于计算 attention，使用 bfloat 16 计算前向传播.

[a] full precision 这一说法有时会产生误解，在科学计算中 fp32 精度并不算高，但在深度学习中已然足够.
[b] 例如 1e-8 使用 fp16 表示是 0.

Compute Accounting

Tensor

Tensor 是深度学习中重要的数据类型，在 PyTorch 中 tensor 实际上是已对分配内存的指针(pointer)，在内存中看起来实际上像一个长数组，例如图4 所示的 4×4 tensor 按照 `stride[1]` 方向展开就像图5 所示的长数组.

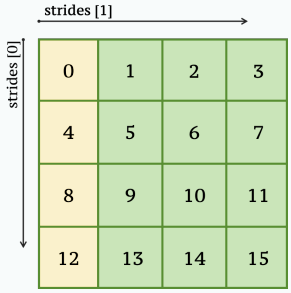

Figure 4: 4×4 An tensor example, where `tensor.stride[0] = 4`

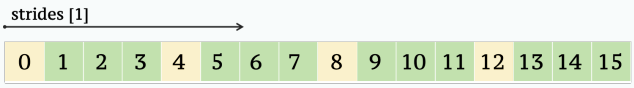

Figure 5: Tensor in stride[1] direction

在索引 tensor 中元素 (x, y)时，只需要在图5 所示的长数组中找到第 N 个元素即可，索引计算方式为[a]:

$$N = x \times \text{stride}[1] + y \times \text{stride}[0] \tag{1}$$

在 PyTorch 中，很多对于 tensor 的操作实际上只是在创建新的视图(view)，而无需重新分配内存，这被称为切片(slice)，详见 Slicing, Indexing, and Masking.

Note 1. 大部分对 *tensor* 的操作默认其是连续的*(contiguous)*，即按照图5 的方式按顺序连续索引，但例如转置(`tensor.t()` / `tensor.transpose(1, 0)`)和原 *tensor* 共享内存，导致在图5 中“跳跃”进行遍历，因此其不连续*(non-contiguous)*，导致报错.

[a] 注意这里 tensor 和数组中索引都是从 0 开始

Create Tensor

计算量取决于所使用的硬件(CPU / GPU). 例如使用 float 32 创建一 32×32 的 tensor 所需要的计算量为 $32 \times 32 \times 4 = 4096$ bytes. 由于大型矩阵在 cpu 与 gpu 之间的转移非常消耗内存和计算量，因此最好直接使用 `torch.·(·,·,device = "cuda:0")` 进行创建.

```
    # how to create data on gpu>
    # 1. create data on cpu and then move it to gpu
    x = torch.zeros(32, 32)
    y = x.to("cuda:0")
    assert y.device == torch.device("cuda", 0)

    # 2. create data directly on gpu
    z = torch.zeros(32, 32, device = "cuda:0")
```

Tensor Operations flops

一次浮点数运算(**FLOP**, floating-point operation)是指一次基本计算，包括加法(″+″)和乘法(″×″). 注意区分如下两个简写:

1. **FLOPs.** floating-point operations，用以衡量计算量[a].
2. **FLOP/s.** floating-point operations per second，用以衡量硬件计算速度[b].

Example 1 (Linear Model). 对于如下一个简单的线性模型(矩阵乘法)，由于最终矩阵有 $B \times K$ 个元素，计算每个元素需要 D 次乘法和 D 次加法，因此计算量为

$$2 \times B \times D \times K \text{ FLOPs.}$$

```
if torch.accelerator.is_available():
    device = torch.accelerator.current_accelerator()
x = toch.ones(B, D, device = device)
z = toch.ones(D, K, device = device)
y = x @ w
```

此线性模型中，B 为数据量，$(D\ K)$ 为参数量，因此前向传播过程中计算量约为

$$2 \times (\#\ \textit{tokens or}\ \#\ \textit{data points}) \times (\#\ \textit{parameters}) \tag{2}$$

Example 2 (Other Operations). 1. 两个 $m \times n$ 矩阵相加需要 mn *FLOPs*

2. 对于 $m \times n$ 矩阵的逐元素*(elementwise)*运算需要 $O(mn)$ *FLOPs*
3. 一般来说深度学习中计算量消耗最大的就是矩阵乘法运算

Model FLOPs utilization(MFU). 一般来说实际运算中硬件的 FLOP/s 都达不到生产商标注的 FLOP/s，二者的比值被称为浮点运算利用率:

$$\text{MFU} = \frac{\text{actual FLOP/s}}{\text{promised FLOP/s}} \tag{3}$$

MFU 通常难以接近 90%，因为在使用中还有一部分性能用于通信等开销. MFU 大于 50% 被认为硬件表现较好，且当矩阵乘法占据计算量的主导时 MFU 会变大.

[a] 训练 GPT-3 需要 3.13×10^{23} FLOPs, GPT-4 需要 3×10^{25} FLOPs.

[b] FLOP/s 取决于硬件和数据类型，例如 H100 ≫ A100，bfloat 16 ≫ float 32. FLOP/s 也会写为**FLOPS**.

Gradients

考虑一个简单的线性模型:

- **Model:** data $x(B \times D) - w_1(D \times D) \to h_1 - w_2(D \times K) \to h_2 \to$ loss
- **Loss:** $\frac{1}{2}(xw - 5)^2$
- **Activation:** $h_1 = xw_1, h_2 = h_1 w_2$

前向传播所需要的计算量为

$$2 \times B \times D \times D + 2 \times B \times K \times D = 2 \times (\text{\# tokens}) \times (\text{\# parameters}) \tag{4}$$

在反向传播中，以计算 w_2 的梯度 $\text{grad}(w_2) = \frac{\text{d loss}}{\text{d } w_2}$ 为例:

$$\text{grad}(w_2)[j,k] = \sum_{i=1}^{B} h_1[i,j] \times \text{grad}(h_2)[i,k] \tag{5}$$

因此计算量为 $2 \times B \times D \times K$.

同理 $\text{grad}(h_1) = \sum_{k=1}^{K} w_2[i,j] \times \text{grad}(h_2[i,k])$，计算量为 $2 \times B \times D \times K$，总计算量为

$$4 \times B \times D \times K = 4 \times (\text{\# tokens}) \times (\text{\# parameters}) \tag{6}$$

First updated: 24 January, 2025
Last updated: 12 May, 2025

References