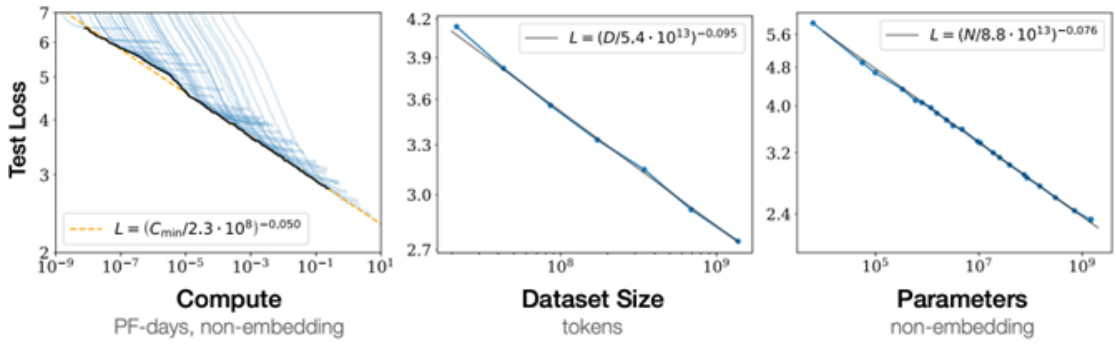
Qlora

通过lora可以一定程度上减少参数量，但是因为scaling law的存在，没法一味减少。

> **Scaling Law 定义：**
>
> 
>
>
> 对于Decoder-only的模型，计算量C(Flops)、模型参数量N（non-embedding parameter count）、数据大小D(token数)三者满足: C≈6ND (小模型不准)。
>
> 最近的说法：数据足够多，小模型也能媲美大模型。因为模型的参数量直接决定了部署的成本，通常在训练过程中是过饱和的，真正的问题在于推理过程中

Quantization

what

why

因为神经网络对低精度很鲁棒

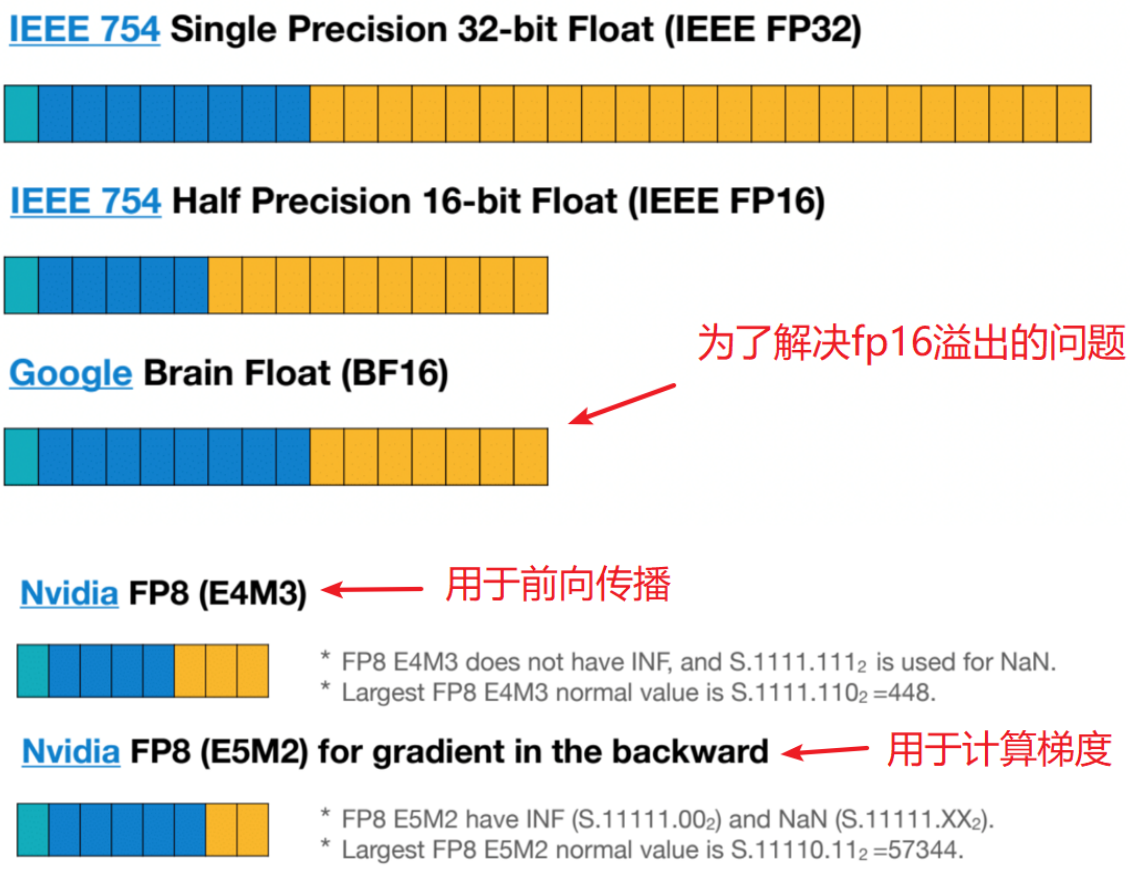
how

如何用低精度的数据去表示高精度的数据?

int8：

fp8：

fp32：



因为训练过程中比较容易溢出。

为了防止训练不稳定，只在计算MLP的过程中才会用到fp8，而在其他层依然使用bf16（layer norm甚至会用到fp32）

前向过程中的E4M3和后向过程中的E5M2是如何转换的?

在用fp8训练时，会随时维护一套半精度的weight（就是bf16格式的weights和bf16格式的gradients），计算完前向传播之后，就把前向传播的结果立马转换为bf16

用fp8做训练并不能节省显存

因为为了训练的稳定性，需要在显存里维护一套高精度的权重和梯度，这样训练才可以收敛。

那我们为什么要这样做呢?

模型在推理过程中用低精度可以节省带宽

因为如果用低精度去做训练，我们的训练速度是可以更快的（更好的tensor core）

关于fp8如何做训练：

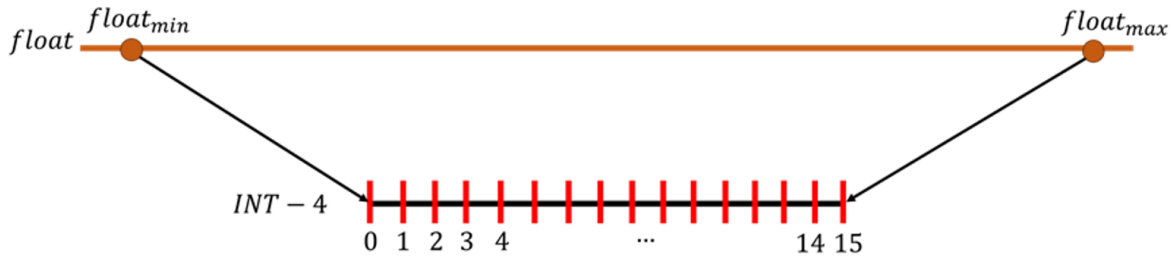
https://developer.nvidia.com/zh-cn/blog/nvidia-gpu-fp8-training-inference/

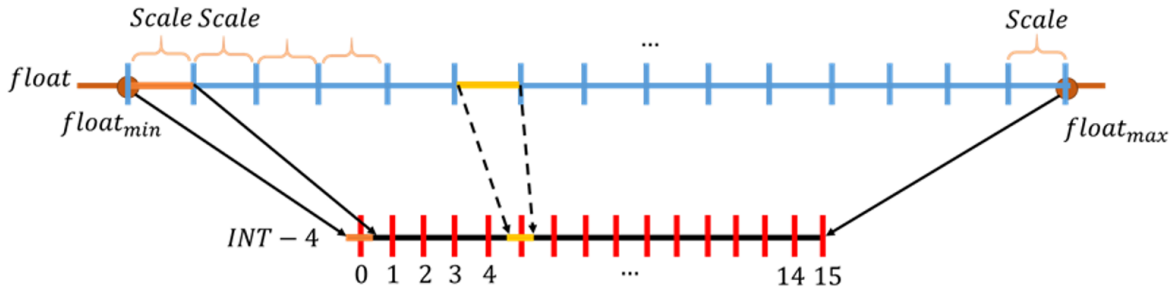
什么是量化：

用低精度表示高精度参数，并且尽量不影响训练和推理的结果

怎么做量化?

映射：





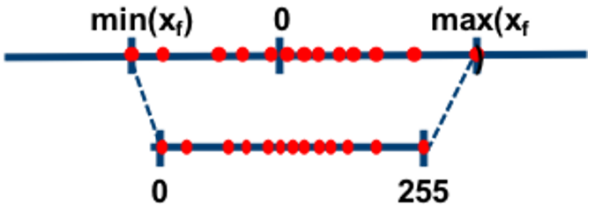
常见的量化的方式：

从不常用到常用来介绍吧：

缩放系数要用原始精度去记录

非对称量化（精度较高）：

需要记录两个超参数，缩放因子和零点（偏移）



缩放因子 零点

$$x_q = round\left((x_f - min_{x_f})\underbrace{\frac{2^n - 1}{max_{x_f} - min_{x_f}}}_{q_x}\right) = round(q_x x_f - \underbrace{min_{x_f} q_x}_{zp_x}) = round(q_x x_f - zp_x)$$

x_f: Original floating-point tensor
x_q: Quantized tensor
q_x: Scale factor
zp_x: Zero-point

对称量化：

不再考虑最大值和最小值，直接考虑绝对值的最大值

只要记录缩放因子

不同的量化粒度：

极端情况下对每个参数做量化，其实就相当于没有做，因为每个参数都要存一个缩放因子

如果对每个矩阵做量化，就是每个矩阵存一个缩放因子即可

weight-only

也对activation做量化，虽然准确度可能会降低，但是可以用低精度的tensor core ，可以加速

LLM.int8()

非均匀量化

好处：精度变高

坏处：更多的存储空间去记录分桶的情况

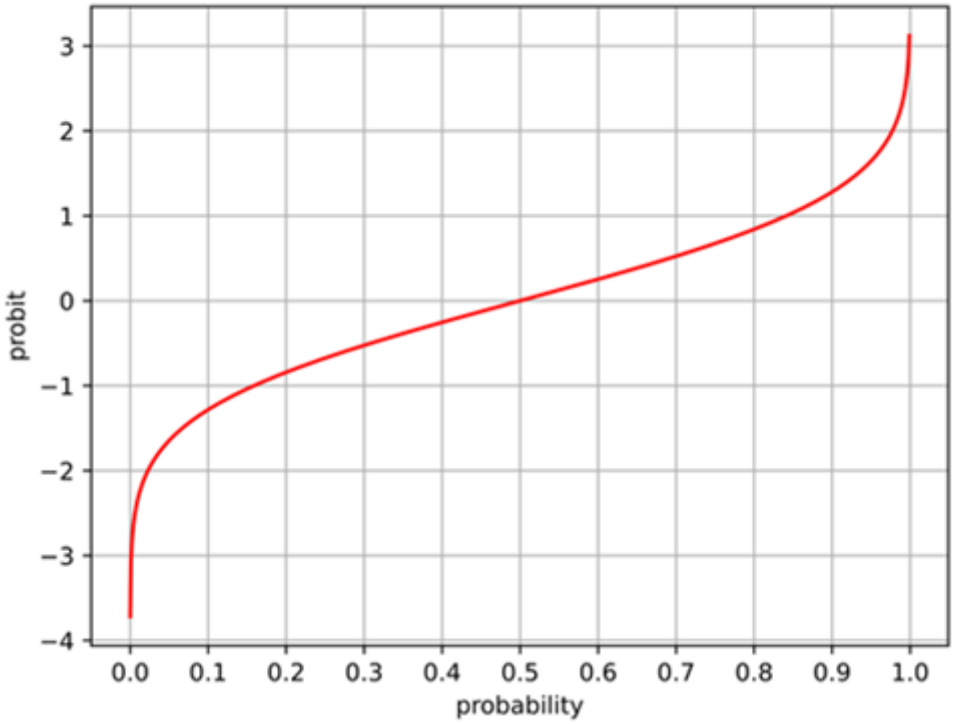
Qlora

可以在4090上微调一个70B的模型，穷人首选

1. 4-bit NormalFloat Quantization 分桶

2. Double Quantization 因为缩放系数要和原始精度一样，所以对缩放系数也进行量化

3. Paged Optimizers 把GPU的显存load到CPU的内存上



为了保证0这个点的精度，把0单独拿出来作为一个桶，保证0一定会出现在最后的结果里

Qlora

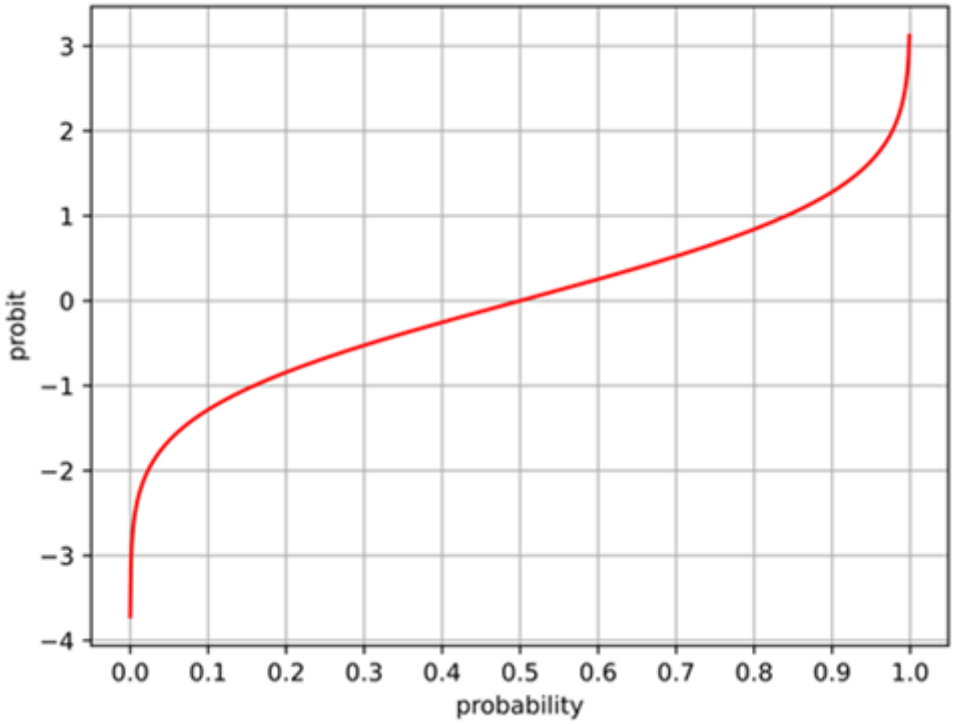
提出了一种高效的微调方式，能在单个48GB GPU上进行65B参数模型的微调，同时保持16位微调任务的完整性能。

主要贡献

- **4-bit NormalFloat (NF4)**: Quantile Quantization，一种信息论上最优的数据类型，用于正态分布的权重量化，确保输入向量落入到每个量化区间的值的数量相同。

论文证明了预训练神经网络权重通常遵循以0为中心的正态分布，因此可以通过缩放σ使分布适应我们的数据范围（本文采用[-1, 1]）。

N(0,1)的quantile function如下：



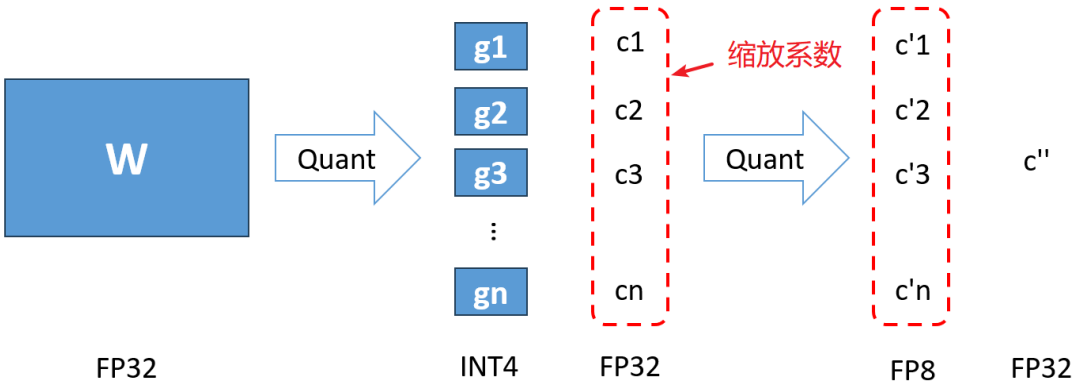
比如（0.9，1.2）这个点就表示x<1.2的概率是0.9。

通过估计N(0, 1)分布的2^k+1分位数，来获取k bit分位数量化的q：

$$q_i = \frac{1}{2}\left(Qx\left(\frac{i}{2^k+1}\right) + Qx\left(\frac{i+1}{2^k+1}\right)\right),$$

- **双重量化 (Double Quantization)**: 量化缩放系数（又叫量化常数），通过双重量化减少内存占用。因为缩放系数要和原始精度一样，所以对缩放系数也进行量化。

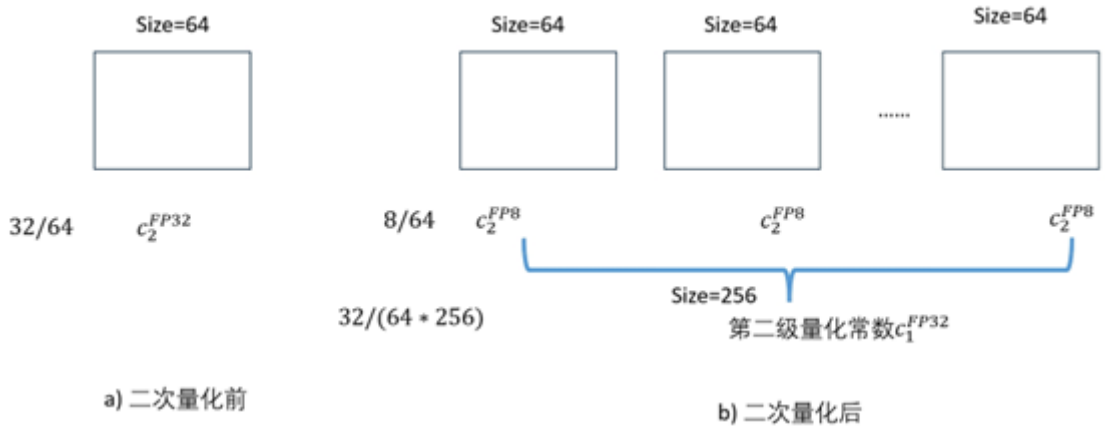
  具体来说，先进行4-bit量化，再对缩放系数进行8-bit量化，减少了每个参数的平均内存占用。



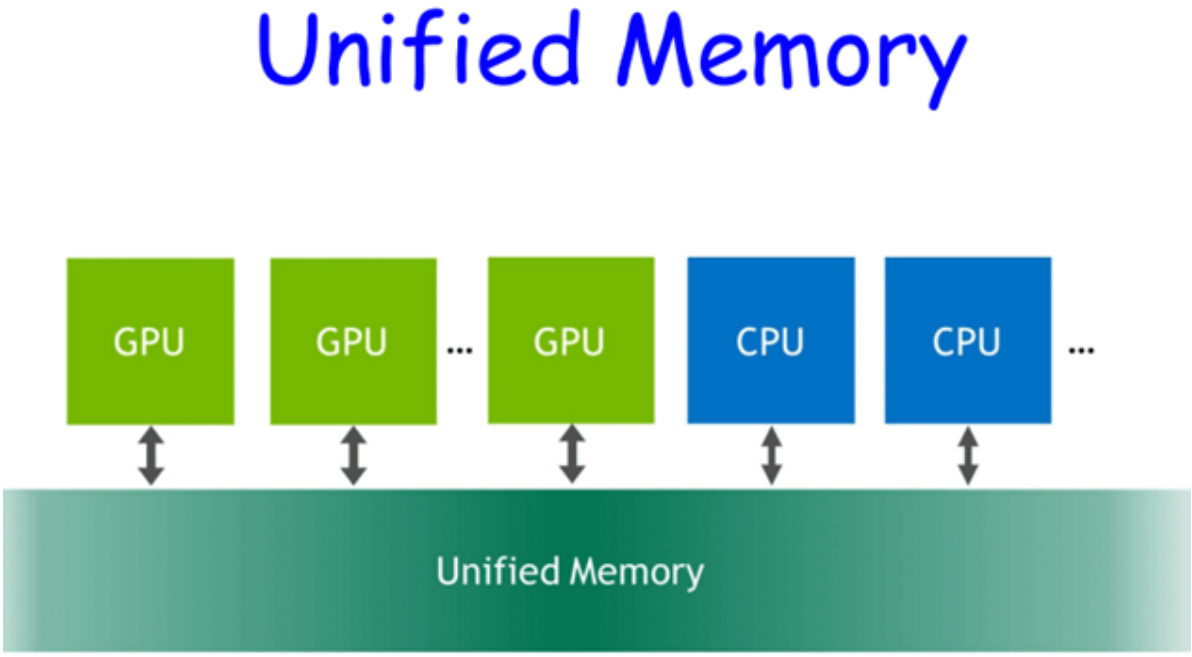
第二次量化的方式可以随意选择，本文中使用E4M3均匀量化。

二次量化的目的是为了节省量化常数占用的空间。为了降低异常值对量化的影响，量化的粒度一般比较小，这需要存储大量的量化常数。例如当粒度为64，且采用32位的量化常数时，对于每个参数相当于增加32/64=0.5个bit，用于存储量化常数。

对于第一次量化的量化常数c_2^{FP32}，将其作为第二次量化的输入，产生新的第一次量化常数c_2^{FP8}和第二次量化常数c_1^{FP32}（这里选择8是出于模型性能考虑）。第二次量化的粒度为256，即将256个fp32变成了fp8，此外还需要存储第二次压缩的fp32，即$8/64 + 32/(64 * 256)$。



- **分页优化器 (Paged Optimizers)**: 使用 NVIDIA 统一内存特性，在CPU和GPU之间进行页传输。当GPU内存不足时，将部分状态转移到CPU RAM 中，并在优化器更新步骤需要内存时分页回到GPU内存中。



整体步骤

单个线性层的QLoRA如下：

$$\mathbf{Y}^{\mathrm{BF16}} = \mathbf{X}^{\mathrm{BF16}} \mathrm{doubleDequant}(c_1^{\mathrm{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\mathrm{NF4}}) + \mathbf{X}^{\mathrm{BF16}} \mathbf{L}_1^{\mathrm{BF16}} \mathbf{L}_2^{\mathrm{BF16}}, \tag{5}$$

在进行前向计算时，首先通过doubleDequant函数把原始模型的参数反量化成fp16，然后加上LoRA的低秩分解矩阵。LoRA的参数不量化而是通过反向传播优化，原始模型参数固定所以可以量化。
doubleDequant(·)定义为：

$$\mathrm{doubleDequant}(c_1^{\mathrm{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \mathrm{dequant}(\mathrm{dequant}(c_1^{\mathrm{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\mathrm{4bit}}) = \mathbf{W}^{\mathrm{BF16}}, \tag{6}$$

首先通过c_1^{FP32}反量化得到每个块的fp32量化常数，在基于这个量化常数的NF4做反量化得到原始参数。

对于参数更新，只需要适配器权重对误差的梯度$\frac{\partial E}{\partial L_i}$，而不需要4位权重的梯度$\frac{\partial E}{\partial W}$。然而，计算$\frac{\partial E}{\partial L_i}$需要通过方程（5）进行，这里通过链式法则会隐含使用$\frac{\partial E}{\partial W}$，这就需要把$W^{\mathrm{NF4}}$反量化成$W^{\mathrm{BF16}}$。

总的来说，QLoRA有一个存储数据类型（通常是4位NormalFloat）和一个计算数据类型（16位BrainFloat）。我们将存储数据类型量化为计算数据类型以执行前向和反向传播，但我们仅计算使用bf16的LoRA参数的权重梯度。