

# QLoRA

## QLoRA

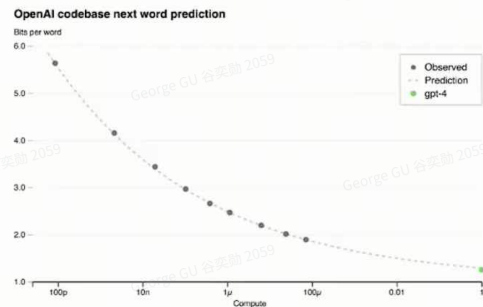
### Scaling Law

计算量FLOP（假设算 $a*b$ 和 $b*c$ 矩阵的乘积，FLOP为 $2abc$ ）与数据集大小和模型参数量成正比。

Scaling Law指出可以通过在小计算量时得到损失值，预测大计算量的损失。

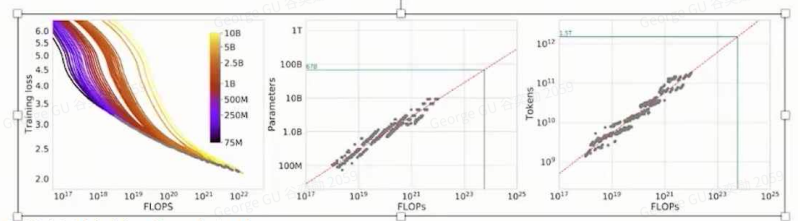
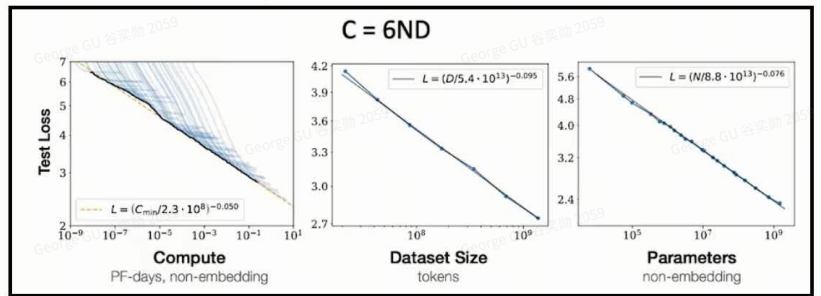
## Extra: Scaling Law

损失值随着数据量、参数量 and 计算量增大而减小。



**Figure 1.** Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

*Having a sense of the capabilities of a model **before training** can improve decisions around alignment, safety, and deployment.*



在计算量固定时，选择小模型大数据量得到的损失更小。

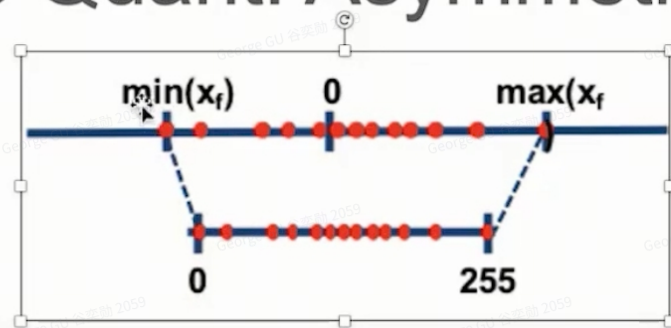
$$\hat{L}(N, D) \triangleq E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}.$$

训练时高精度，推理时低精度。训练时维护一套高精度的参数，只有到MLP层才用低精度。推理时才全换成低精度

### 量化方法

非对称量化，需要记录缩放因子 $q_x$ 和零点值 $zp_x$ 。

# How to Quant: Asymmetric Quant



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

$x_f$ : Original floating-point tensor

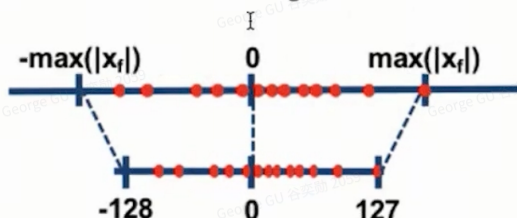
$x_q$ : Quantized tensor

$q_x$ : Scale factor

$z_{p_x}$ : Zero-point

对称量化，记录缩放因子  $q_x$ ，使用更普遍，精度较低（当数据非对称分布时，会导致部分区间浪费）

# How to Quant: Symmetric Quant



$$x_q = \text{round}(q_x x_f)$$

$x_f$ : Original floating-point tensor

$x_q$ : Quantized tensor

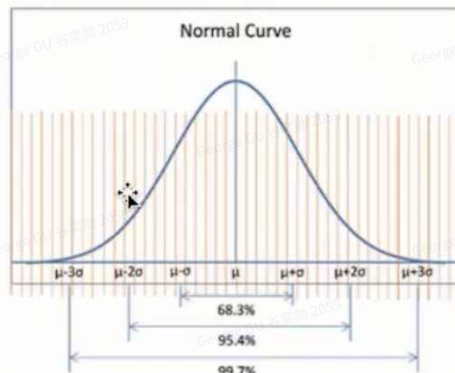
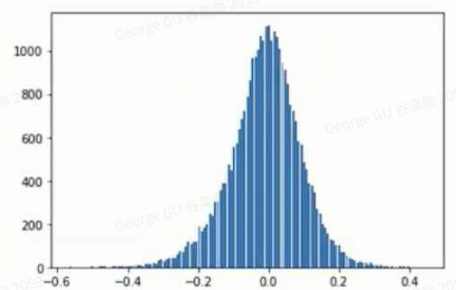
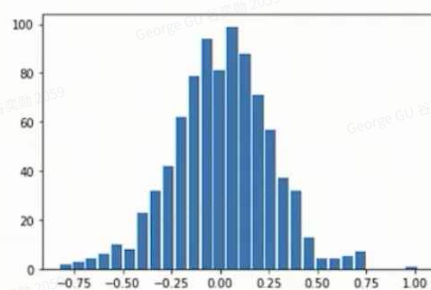
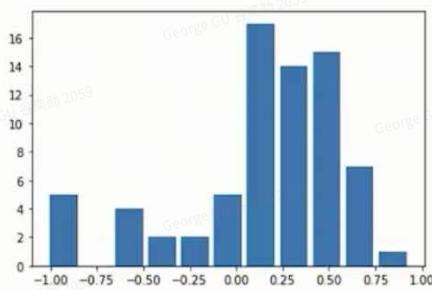
$q_x$ : Scale factor

	Full Range	Restricted Range
Quantized Range	$[-\frac{N_{bins}}{2}, \frac{N_{bins}}{2} - 1]$	$[-(\frac{N_{bins}}{2} - 1), \frac{N_{bins}}{2} - 1]$
8-bit example	$[-128, 127]$ (As shown in image above)	$[-127, 127]$
Scale Factor	$q_x = \frac{(2^n - 1)/2}{\max(abs(x_f))}$	$q_x = \frac{2^{n-1} - 1}{\max(abs(x_f))}$

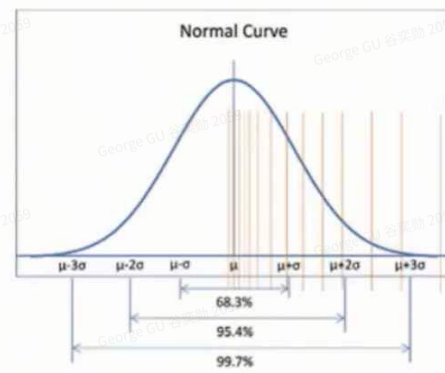
$$N_{bins} = 2^n - 1$$

非均匀量化，针对原始数值分布不均匀的情况，需要记录每一个桶的位置，参数更多。

# Non-uniform Quantization



均匀量化

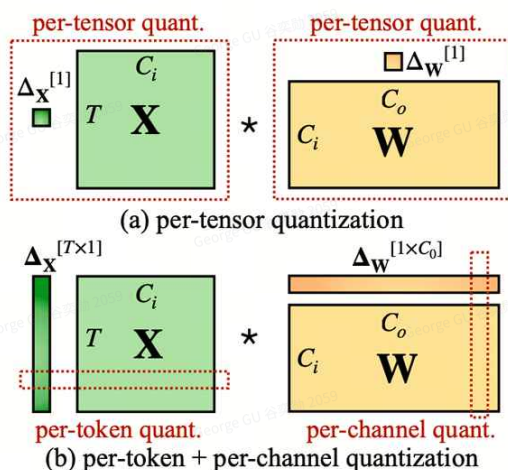
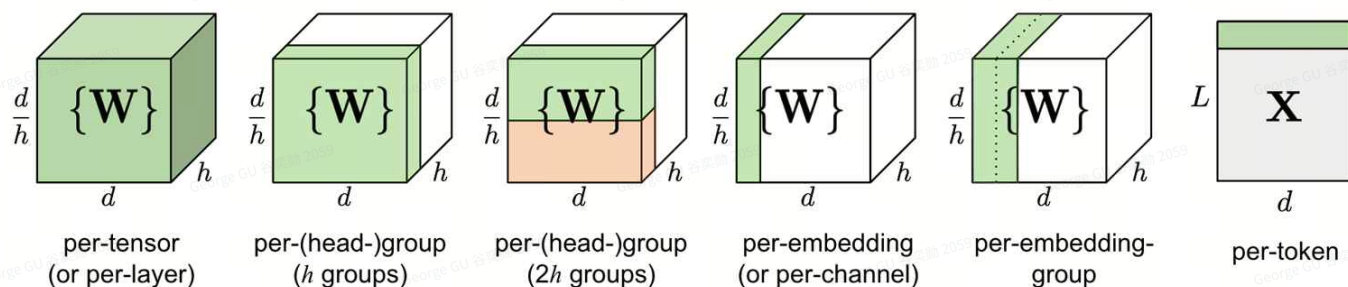


非均匀量化

Weight-only: 使用时将低精度存储到显存，在做矩阵乘法时还原成高精度  $x_f$  .

量化的粒度：粒度大时损失的值会更多，且异常值影响到的参数更多。粒度小需要存储更多的scale factor。

# Quant in different granularity



LLM.int8，去除异常值再进行量化？

正态变成均匀，再用标准int8量化，不用分桶？

在测试数据集中，获取大概范围，确定scale factor？

## 计算资源消耗分布

假设用70B的模型训练，计算资源消耗在3部分：

- 梯度和参数：
- 优化器：Adam需要维护一阶和二阶动量
- 激活函数：在反向传播时，需要前向传播的activation的值。8192表示一个token对应的向量长度，80表示80个transformer，4表示32bit对应4byte，12表示一个transformer中的要过的层数。可以通过将激活参数存到CPU解决。

# 计算资源都消耗在了哪里？

```
model = Model()  
optimizer = Adam(model.parameters())
```

Gradients and Parameters

$70 * 10^9 * 4 * 2 \text{ Byte} = 521.5 \text{ GB}$

```
for batch, (X, y) in enumerate(dataloader):  
    # Compute prediction and loss  
    pred = model(X)  
    loss = loss_fn(pred, y)  
  
    # Backpropagation  
    optimizer.zero_grad()  
    loss.backward()  
    optimizer.step()
```

Optimizer States,

i) the time averaged momentum and

ii) variance of the gradients to compute the updates.

代码Recap: adam的实现

$70 * 10^9 * 4 * 2 \text{ Byte} = 521.5 \text{ GB}$

Activations

Activation Checkpointing / BF16 Training

$8192 * 80 * 4 * 12 \text{ Byte} / \text{Token} = 30 \text{ MB} / \text{Token}$

Temporary buffers

Memory Fragmentation

LoRA能大量减少梯度和激活值需要的计算资源，但参数仍然占据很大空间。

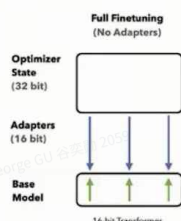
## 正文

QLoRA通过减少每个参数对应的byte数，实现计算量的减少。（神经网络对于低精度的数据是很鲁棒的）

## Finetuning Cost

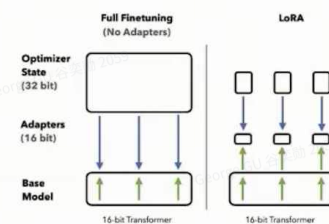
Finetuning cost per parameter:

- Weight: 16 bit
- Weight gradient: 16 bit
- Optimizer state: 64 bit
- 12 byte per parameter



Finetuning cost per parameter:

- Weight: 16 bits
- Weight gradient: ~0.4bit
- Optimizer state: ~0.8bit
- Adapter weights: ~0.4bit
- 17.6 bits per parameter



Finetuning cost per parameter:

- Weight: 4 bit
- Weight gradient: ~0.4 bit
- Optimizer state: ~0.8 bit
- Adapter weights: ~0.4 bit
- 5.2 bit per parameter

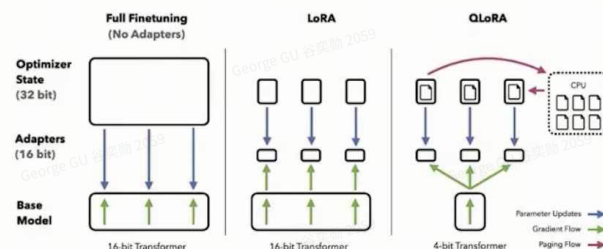


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

- 使用Quantile Quantization，一种在信息理论上最优的数据类型，确保输入向量落入到每个量化区间的值的数量相同。

QLoRA论文证明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  $\sigma$  使分布适应我们的数据范围（本文采用[-1, 1]）。



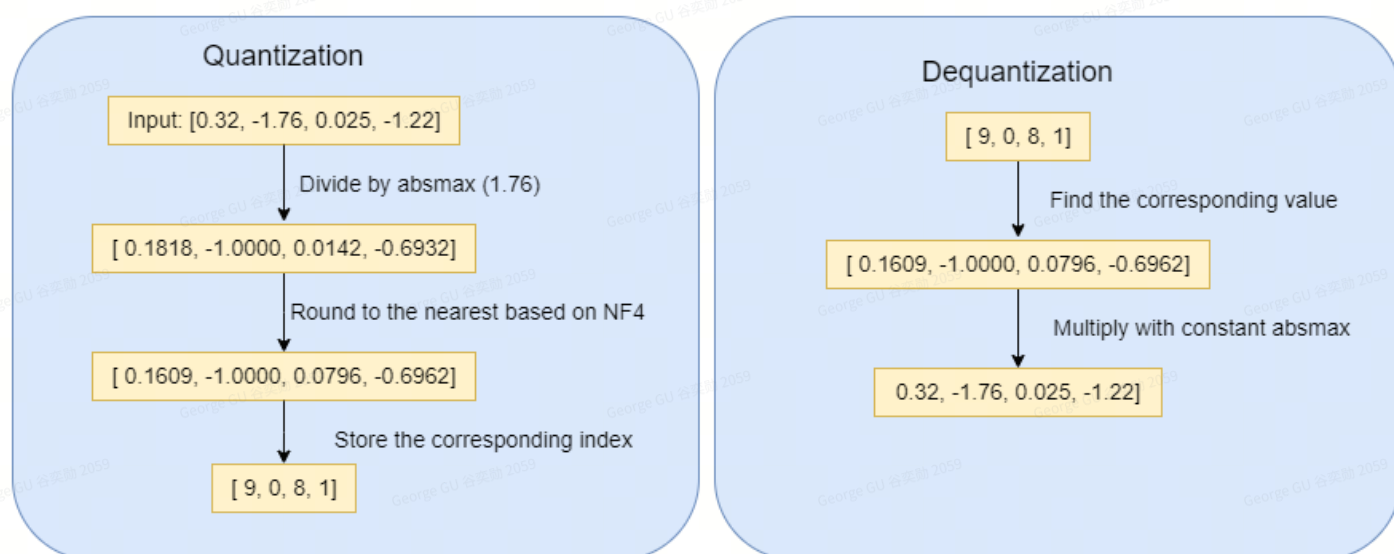
## NF4量化与反量化

针对输入向量 $X=[0.32, -1.76, 0.025, -1.22]$ ，采用对称量化，找到 $\text{absmax}(X)=1.76$ ，将 $X$ 归一化成 $[0.1818, -1, 0.0142, -0.6932]$ 。依据NF4= $[-1.0000, -0.6962, -0.5251, -0.3949, -0.2844, -0.1848, -0.0911, 0.0, 0.0796, 0.1609, 0.2461, 0.3379, 0.4407, 0.5626, 0.7230, 1.0000]$ 进行舍入.NF4的获取后面介绍。

注意尽管0.0142距离0.0更近，但是在神经网络中通常用0进行padding，为了区分padding和较小的数，不能将较小的数舍入为0。舍入后保存在NF4中对应的下标，日0.1609对应9。

反量化则是乘以 $\text{absmax}(X)=1.76$ ，反量化存在一定误差，如 $0.1609 \times 1.76 = 0.283184$ ，右图中展示的是理想中不存在误差的情况。

NF4 =  $[-1.0000, -0.6962, -0.5251, -0.3949, -0.2844, -0.1848, -0.0911, 0.0000, 0.0796, 0.1609, 0.2461, 0.3379, 0.4407, 0.5626, 0.7230, 1.0000]$



总结一下，分为3个步骤。

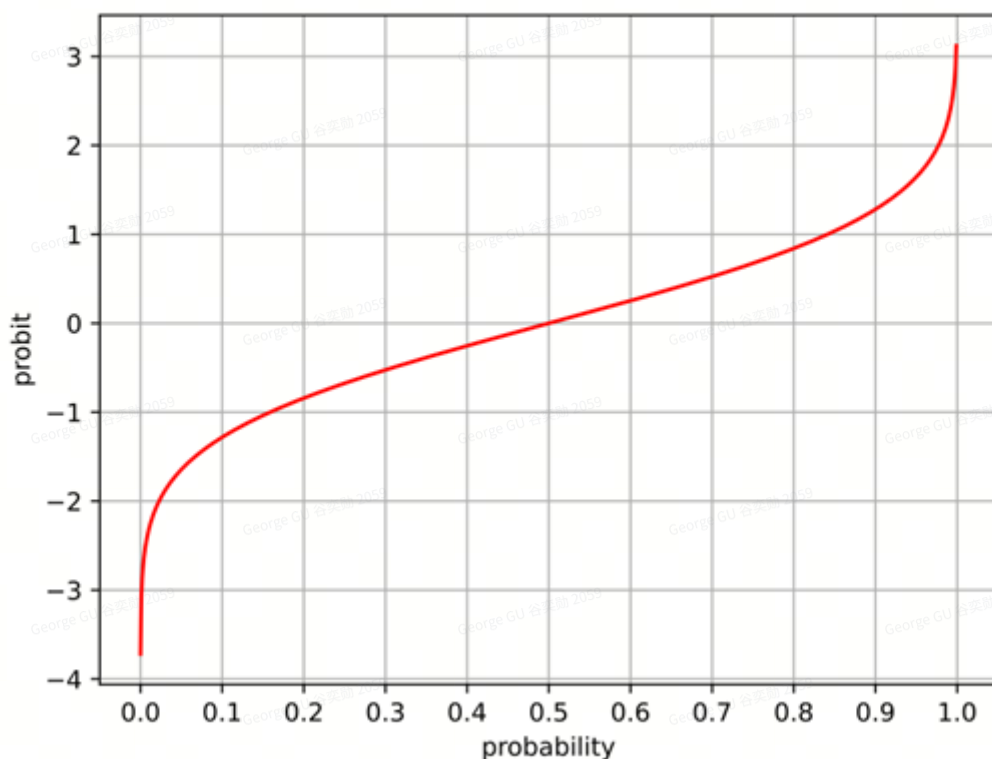
1. 计算出量化常数 $N$ （在非对称量化中 $N=\text{absmax}(X)$ ），将输入向量 $X$ 映射到目标量化区间 $D$ （例如 $[-1, 1]$ ）中。该区间的分位函数为 $Q_X$ 。
2. 对于 $X/N$ 中的每个元素，找到分位函数 $Q_X$ 中计算到的最相近的值 $q_i$
3. 将 $q_i$ 对应的索引 $i$ 存到输出向量 $T^Q$ 中。

## 分位数量化

最理想的情况是，分位函数中的每一个数都以相同概率被用到到。例如采用NF4做为分位函数，输入 $X$ 长度为1600，其中的16个值每个被选中100次。

为了实现这一特性，可以将概率分布函数 $f_X$ 分为 $2^k$ 个区间（ $k$ 表示量化后的一个数字占 $k$  bit），每个区间都有相同的面积，这些区间的中点即为 $q$ 。

为了找到 $q$ 值，就需要找到具有相等概率质量的累积分布函数 $F_X$ 的 $2^k$ 个不重复 $x$ 值，这些值可以通过使用其反函数，即分位函数 $Q_X = F_X^{-1}$ 找到。下图为一个以0为中心的正态分布的累积分布函数的反函数 $F_X^{-1}$ ，横轴表示概率，纵轴表示值， $(0.5, 0)$ 表示值小于0的概率为0.5。



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

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

#### NF4

对输入数据，经常用0进行padding，以保持特定的空间维度或确保数据的对齐。因此对零的精确表示确保了这些padding能够准确地表示。对称的 $k$ 位量化意味着对于值零没有精确的表示。因此NF4采用非对称的量化，分别考虑正值和负值的分位数，然后合并它们。在负的部分创建  $2^{k-1}$  个值，在正的部分创建  $2^{k-1} + 1$  个值，然后删除重叠的0。

1.选择一个比较小的 $\delta = \frac{1}{2}\left(\frac{1}{32} + \frac{1}{30}\right)$ 。我们可以计算出 $1 - \delta = 0.9677083333333334$ 。

2.计算均匀pacing的8个数,  $p_1, \dots, p_8$ , 其中 $p_1 = \delta, p_8 = 1/2$ 。

3.找到它们高斯CDF函数 $\Phi$ 的逆，也就是计算 $\hat{q}_i = \Phi^{-1}(p_i)$ 。

```
1 delta = 1/2*(1/32+1/30)
2 print(torch.linspace(delta, 0.5, 8))
3 print(norm.ppf(torch.linspace(delta, 0.5, 8)))
4 输出:
```

```

5 tensor([0.0323, 0.0991, 0.1659, 0.2327, 0.2996, 0.3664, 0.4332, 0.5000])
6 [-1.84813142 -1.28665578 -0.97040379 -0.72985929 -0.52568489 -0.34148556
7  -0.16827238  0.          ]

```

4. 计算9个均匀paced概率值 $r_8, \dots, r_{16}$ , 其中 $r_8 = 1/2, r_{16} = 1 - \delta$ 。

5. 计算 $\hat{q}_i = \Phi^{-1}(r_i)$ 。

```

1 print(torch.linspace(0.5, 1-delta, 9))
2 print(norm.ppf(torch.linspace(0.5, 1-delta, 9)))
3 结果:
4 tensor([0.5000, 0.5585, 0.6169, 0.6754, 0.7339, 0.7923, 0.8508, 0.9092,
5         0.9677])
6 [0.          0.14707497 0.29742005 0.45484772 0.6245116  0.81448966
7  1.03979003 1.33611882 1.84813166]

```

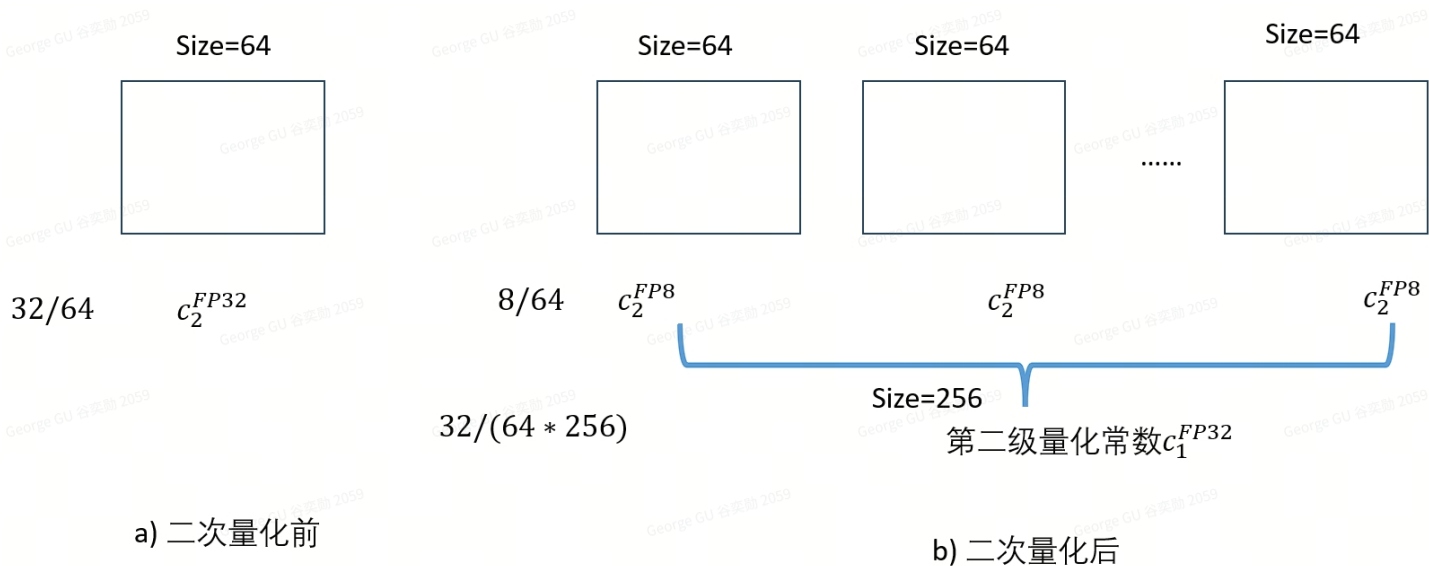
将两部分q合并，去掉重复的0，就得到了N（0，1）分布的NF4。然后将q的值归一化到[-1, 1]范围内，就得到了[-1, 1]区间内的NF4。

## 二次量化

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 $8/64 + 32/(64 * 256)$ 。





## 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。

## QLoRA

单个线性层的QLoRA如下：

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

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( $\cdot$ )定义为：

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

首先通过  $c_1^{\text{FP32}}$  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  $\frac{\partial E}{\partial W}$ ，这就需把  $\mathbf{W}^{\text{NF4}}$  反量化成  $\mathbf{W}^{\text{BF16}}$ 。

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16位BrainFloat的LoRA参数的权重梯度。