论文名称：2305.14314v1 QLoRA: Efficient Finetuning of Quantized LLMs
论文团队：华盛顿大学
github：https://github.com/artidoro/qlora

1. 问题背景

- 微调大型语言模型（如65B参数的LLaMA模型）非常消耗资源，通常需要超过780GB的GPU内存。
- 现有的量化方法虽然能减少内存占用，但仅适用于推断阶段，在训练期间会失效。

2. 解决措施

- **4-bit NormalFloat**：为正态分布数据设计的信息理论上最优的量化数据类型。
- **双重量化**：通过量化量化常量来节省内存。
- **分页优化器**：通过使用NVIDIA统一内存来避免在处理长序列的mini-batch时出现的梯度检查点内存峰值，优化内存使用效率。

3. QLoRA 优势

- **预训练模型量化**：将预训练模型量化到4-bit ，然后添加一组可学习的低秩适配器权重（Low-rank Adapter weights）。
- **内存效率**：将微调65B参数模型的平均内存需求从>780GB降低到<48GB，且不降低运行时或预测性能。
- 使用消费级GPU在12小时内训练，使用专业GPU在24小时内达到与ChatGPT相当的99.3%性能。

4. 预备知识：量化

4.1 量化定义

在大模型训练过程中，量化是一种降低模型计算复杂度和存储需求的技术。量化的核心思想是将模型的权重和激活值从高精度表示（如32位浮点数）转换为低精度表示（如8位整数），从而减少计算和内存使用，同时尽量保持模型性能。

量化可以分为两种主要类型：**权重量化** 和 **激活量化**。

① **权重量化**：将模型权重从高精度表示（例如32位浮点数）转换为低精度表示（例如8位整数）。

**② 激活量化：将模型激活值（即每一层的输出）从高精度表示（例如32位浮点数）转换为低精度表示（例如8位整数）。

量化通常包含以下步骤：

- **选择量化方案**：确定使用 **对称量化** 或 **非对称量化** ，对称量化使用统一的缩放因子和零点，而非对称量化则针对每个权重或激活值使用不同的缩放因子和零点。
- **计算缩放因子（scale）和零点（zero-point）**：用于将浮点数映射到整数范围。
- **应用量化公式**：将浮点数转换为整数表示。

4.2 非对称量化

案例：从32位浮点数转换为8位整数的具体步骤

第1步：确定范围

假设我们有一组浮点数权重：

$w = [0.1, -0.2, 0.3, -0.4, 0.5]$

我们需要确定这些浮点数的最小值和最大值：
$w_{min} = -0.4$
$w_{max} = 0.5$

第2步：计算缩放因子和零点

缩放因子计算如下：

$\text{scale} = \frac{w_{max} - w_{min}}{2^8 - 1}$

在8位量化中，整数范围为0到255，所以有：

$\text{scale} = \frac{0.5 - (-0.4)}{255} = \frac{0.9}{255} \approx 0.00353$

第3步：应用量化公式

将浮点数转换为8位整数使用以下公式：

$w_{int} = \text{round}\left(\frac{w - w_{min}}{\text{scale}}\right)$

逐个计算每个浮点数的量化值：

$w_{int}[0] = \text{round}\left(\frac{0.1 - (-0.4)}{0.00353}\right) = \text{round}(141.36) = 141$

$w_{int}[1] = \text{round}\left(\frac{-0.2 - (-0.4)}{0.00353}\right) = \text{round}(56.65) = 57$

$w_{int}[2] = \text{round}\left(\frac{0.3 - (-0.4)}{0.00353}\right) = \text{round}(197.73) = 198$

$w_{int}[3] = \text{round}\left(\frac{-0.4 - (-0.4)}{0.00353}\right) = \text{round}(0) = 0$

$w_{int}[4] = \text{round}\left(\frac{0.5 - (-0.4)}{0.00353}\right) = \text{round}(254.95) = 255$

所以，量化后的8位整数表示为：

$w_{int} = [141, 57, 198, 0, 255]$

第4步：反量化（Dequantization）

为了验证量化的效果，可以进行反量化，将整数重新映射回浮点数表示：

$w{*float*} = *w*{int} \times \text{scale} + w_{min}$

计算每个量化整数的反量化值：

$w_{float}[0] = 141 \times 0.00353 + (-0.4) \approx 0.09773$

$w_{float}[1] = 57 \times 0.00353 + (-0.4) \approx -0.198$

$w_{float}[2] = 198 \times 0.00353 + (-0.4) \approx 0.29894$

$w_{float}[3] = 0 \times 0.00353 + (-0.4) = -0.4$

$w_{float}[4] = 255 \times 0.00353 + (-0.4) \approx 0.49915$

反量化后的浮点数与原始浮点数相比，存在一定的误差，但基本保持了原始值的分布。

通过上述步骤，完成了从32位浮点数到8位整数的量化过程，并通过反量化验证了量化的效果。

4.3 对称量化

【QLoRA论文中使用的量化方法】

对称量化使用绝对值的最大值进行标准化，将数据映射到 [-127, 127] 范围。

假设我们有一个简单的一维浮点数张量 X 作为输入，它包含以下8个元素：

$X = [1.5, -3.0, 2.0, 4.5, -2.5, 1.0, 3.5, -4.0]$

我们需要对这个张量进行块状8位量化。为了简化示例，我们将每个块的大小 B 设为4，这意味着我们将张量 X 分成两个块进行独立的量化。

第1步: 分块

将 X 划分为两个块：

$X_1 = [1.5, -3.0, 2.0, 4.5]$

$X_2 = [-2.5, 1.0, 3.5, -4.0]$

步骤 2: 计算量化常数

对于每个块，我们需要计算量化常数 c_i ，这里使用绝对最大值来确定量化比例。

对于 X_1 ：

$\text{absmax}(X_1) = 4.5$

$c_1 = \frac{127}{4.5} \approx 28.222$

对于 X_2：

$\text{absmax}(X_2) = 4.0$

$c_2 = \frac{127}{4.0} = 31.75$

步骤 3: 应用量化

使用公式 $X_{\text{Int8}} = \text{round}(c \cdot X_{\text{FP32}})$ 对每个块进行量化。

对于 X_1：

$X_{1,\text{Int8}} = \text{round}(28.222 \cdot [1.5, -3.0, 2.0, 4.5]) = \text{round}([42.333, -84.666, 56.444, 127]) = [42, -85, 56, 127]$

对于 X_2：

$X_{2,\text{Int8}} = \text{round}(31.75 \cdot [-2.5, 1.0, 3.5, -4.0]) = \text{round}([-79.375, 31.75, 111.125, -127]) = [-79, 32, 111, -127]$

步骤 4: 反量化

使用公式 $X_{FP32} = \frac{X_{\text{Int8}}}{c}$ 对每个块进行反量化。

对于 X_1：

$X_{1, \text{FP32}} = \frac{[42, -85, 56, 127]}{28.222} \approx [1.488, -3.011, 1.985, 4.499]$

对于 X_2：

$X_{2, \text{FP32}} = \frac{[-79, 32, 111, -127]}{31.75} \approx [-2.488, 1.008, 3.496, -4.004]$

通过块状量化，我们成功地将张量分成独立的块，并且为每个块计算出适当的量化和反量化值，这样做可以更准确地捕捉和还原原始数据，特别是在数据分布不均匀时。

4.4 对称 VS 非对称量化

（1）对称量化：

- **量化过程**：使用绝对值的最大值进行标准化，将数据映射到 [-127, 127] 范围。
- **公式**： $X_{Int8} = \text{round}\left(\frac{127}{\text{absmax}(X_{\text{FP32}})} \times X_{FP32} \right)$
- **特点**：假设数据的分布是对称的，因此只需要考虑一个缩放因子，即最大绝对值。

（2）非对称量化：

- **量化过程**：考虑数据的实际最小值和最大值，计算缩放因子，将数据映射到 [0, 255] 范围。

- **公式**： $w_{int} = \text{round}\left(\frac{w - w_{min}}{\text{scale}} \right)$
- **特点**：考虑了数据的实际最小值和最大值，适用于数据分布不对称的情况。

4.5 分块量化

分块量化（Chunk-based Quantization）的作用主要是通过将张量划分为较小的块，分别对每个块进行量化，从而提高量化后的精度和模型性能。

（1）分块量化的作用

① 提高精度：

- 当张量中包含的数值范围较大时，使用统一的量化常数可能无法精确表示所有值。通过将张量划分为多个较小的块，可以使每个块内的数值范围相对较小，从而提高量化的精度。

② 减少量化误差：

- 不同块中的数值可能有不同的统计特性，使用全局量化常数会忽略这些差异。分块后，每个块都能独立确定最适合的量化常数，从而减少全局量化带来的误差积累。

③ 适应不同数据分布：

- 对于数据分布不均匀的张量，某些部分可能包含极端值，而其他部分则可能比较平缓。分块量化可以更好地适应这种情况，使得量化更加灵活和准确。

（2）是否必须使用分块量化?

分块量化并不是必须的，是否使用取决于具体的应用场景和需求：

① 必须的场景：

- 当模型需要部署在资源受限的环境中，且量化精度对模型性能有较大影响时，分块量化是一个有效的选择。
- 在处理具有较大动态范围或复杂分布的数据时，分块量化可以显著提高精度。

② 不必须的场景：

- 对于较小的模型或数据分布较均匀的情况，直接使用全局量化常数可能已经足够，分块量化带来的额外复杂度和计算开销可能不值得。
- 如果对量化精度的要求不高，且模型性能对精度不敏感，可以省略分块量化步骤。

5. QLoRA 微调

QLoRA 通过以下技术实现4-bit 微调：

1️⃣ 4-bit NormalFloat（NF4）量化

2️⃣ 双重量化

3️⃣ **分页优化器**：用于避免大模型单机微调过程中因梯度检查点引起的内存超出错误。

存储与计算数据类型

- **存储数据类型**：通常是4-bit 精度。
  - 在存储时，数据通常会被压缩到更低的位宽，以减少存储空间的占用。这种4-bit的精度在某些应用中是足够的，并且可以显著降低存储和传输的成本。
- **计算数据类型**：通常是BFLOAT16。
  - 在使用QLoRA（Quantized Low Rank Adaptation）权重张量时，首先需要将这些张量量化为BFLOAT16格式。量化是指将高精度的浮点数转换为较低精度的浮点数，以减少计算和存储的开销。
  - 之后，使用16位的矩阵乘法进行计算。这种方法在保持一定精度的同时，能显著提升计算效率。

5.1 4-bit NormalFloat量化

- **理论基础**：
  - 4-bit NormalFloat量化是基于分位数量化，这种量化方法理论上能确保每个量化区间有相等数量的输入张量值。分位数量化通过估算输入张量的分位数来实现，从而通过信息论角度来达到最优优化。
- **计算方法**：首先估算 $2^k + 1$ 个分位数以获得 k 位量化的数据类型，然后将张量标准差调整至与数据类型匹配，以便进行常规的量化。
  - 具体步骤包括将张量标准化到 $[-1, 1]$ 区间，并进行绝对最大值重缩放。
- **量化公式**：

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

  其中， Q_X 是标准正态分布 $N(0, 1)$ 的分位数函数。
- **优化设计**：为了确保有一个离散的零点，并使用所有的 2^k 位表示类型，统一这些分位数集合，移除两个集合中共同出现的零点。这种方法被称为 k 位 NormalFloat（NFK），理论上对于以零为中心的正态分布数据是最优的。

>>> 举例讲解 <<<

```
案例-01_QLoRA 算法原理讲解示例.ipynb
```

5.2 双重量化 (Double Quantization)

“双重量化”是一种通过量化量化常数来进一步节省内存的技术。这个过程涉及两个步骤：

1. **第一步量化**：使用32位或更少的位数进行初始量化，但这会增加每个参数的内存开销。

2. **第二步量化**：将第一步的量化常数作为输入，进行第二次量化。这里使用的是8位浮点数，并采用256的块大小进行第二步量化。

具体计算示例为，使用64的块大小，从每个参数平均0.5位减少到0.127位，节约了0.373位的内存。为了使第二次量化的数值集中在零点附近，还会从第一步的量化常数中减去均值。

>>> 举例讲解 <<<

```
案例-01_QLoRA 算法原理讲解示例.ipynb
```

5.3 分页优化器 (Paged Optimizers)

“分页优化器”利用NVIDIA的统一内存特性，自动管理CPU和GPU之间的内存使用，特别是在GPU内存不足的情况下。当GPU内存不足时，优化器状态会被自动换出到CPU RAM中，必要时再换回GPU内存。这样做有助于避免因内存不足而导致的计算错误或性能下降。

6. 实验

实验一：16位、8位和4-bit 微调方法的比较

- **目的**：评估QLoRA微调与全模型16位微调在RoBERTa和T5模型上的表现。
- **模型规模**：125M至3B参数。
- **数据集**：GLUE和Super-NaturalInstructions。
- **结果**：16位、8位和4-bit 适配器方法均能复制完全微调的16位基线性能，说明通过适配器微调后的量化，损失的性能可以完全恢复。

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

实验二：高参数模型的4-bit QLoRA微调

- **目的**：测试 4-bit QLoRA 在7B至65B参数模型上的性能。
- **模型**：LLaMA 7B至65B。
- **数据集**：Alpaca和FLAN v2，使用MMLU基准进行评估。
- **结果**：双重量化的NF4完全恢复了16位LoRA MMLU的性能；与FP4相比，QLoRA微调的NF4提升了约1个百分点。

Table 4: Mean 5-shot MMLU test accuracy for LLaMA 7-65B models finetuned with adapters on Alpaca and FLAN v2 for different data types. Overall, NF4 with double quantization (DQ) matches BFloat16 performance, while FP4 is consistently one percentage point behind both.

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

- **4-bit QLoRA与16位全微调性能匹配**：4-bit QLoRA的性能与16位LoRA全微调相当，尤其是在使用NF4数据类型时。
- **NF4的优越性**：NF4显示出比FP4更高的量化精度，并在各项评估中优于FP4。
- **效率与精度的平衡**：研究表明，在有限的推理资源和微调资源预算下，提高模型的参数数量而降低其精度是有益的。

7. 基于 PEFT 进行案例实战

案例实战-01_QLoRA 算法原理讲解示例.ipynb

案例实战-02_QLoRA案例实战.ipynb

8. 开源项目 qlora 实战

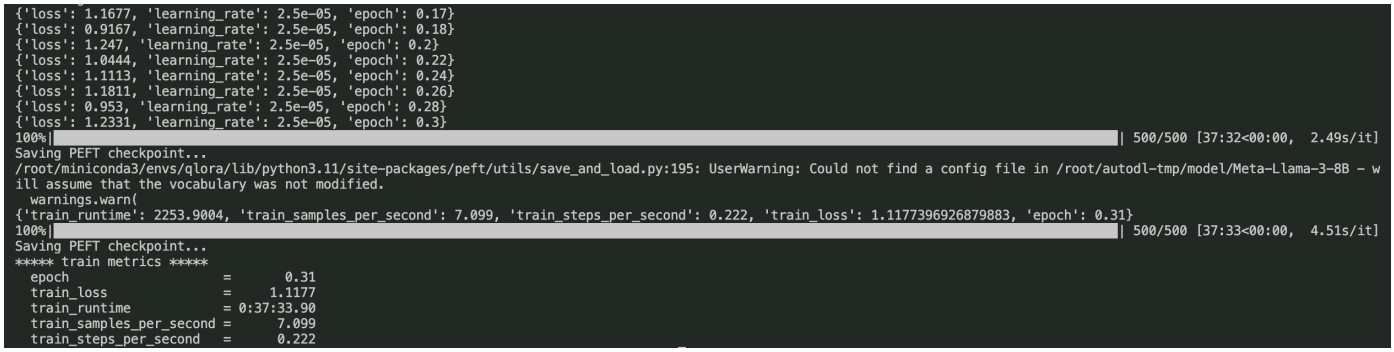
8.1 环境配置

```
conda create -n qlora python=3.11
conda init bash && source /root/.bashrc
conda activate qlora
conda install ipykernel
ipython kernel install --user --name=qlora   # 设置kernel，--user表示当前用户，qlora为虚拟环境

# 注意：pytorch 版本为 2.1.0 ，其它版本可能都有潜在问题。
```

8.2 基于llama3进行QLoRA微调

```
python qlora.py --model_name_or_path /root/autodl-tmp/model/Meta-Llama-3-8B --dataset alpaca --output_dir ./output --optim paged_adamw_8bit --per_device_train_batch_size 8 --gradient_accumulation_steps 4 --max_steps 500 --learning_rate 2.5e-5 --do_train True --logging_steps 30 --group_by_length True
```



8.3 基于 debug 方式熟悉整个QLoRA微调流程

8.4 基于微调模型和Gradio进行Web端交互

```
# 运行代码脚本

gradio_chatbot.py
```