



QLoRA: Efficient Finetuning of Quantized LLMs

QLoRA: Efficient Finetuning of Quantized LLMs

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit

 <https://arxiv.org/abs/2305.14314>




2023 NeurIPS oral presentation

[Openreview]

QLoRA: Efficient Finetuning of Quantized LLMs

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit

 <https://openreview.net/forum?id=OUIFPHEgJU¬elid=6fMfTThibD>



Abstract

Full 16-bit fine-tuning task performance를 보여주면서도 48GB GPU 하나만으로 65B parameter model을 fine-tuning할 수 있는 **효과적인 fine-tuning 기법인 QLoRA**를 제안한다.

- QLoRA는 4-bit quantized pretrained language model를 통해 gradient를 LoRA로 backpropagation한다이를 통해 만들어진 model family인 Guanaco는 SOTA에 준하는 성능을 보여준다.

QLoRA는 performance를 희생하지 않고도 메모리를 절약할 수 있는 방법들을 도입한다.

a. 4-bit NormalFloat (NF4)

- 새로운 data type을 의미하며 이론적으로 distributed weights에서 가장 optimal한 구조

b. Double Quantization by quantizing the quantization constants

c. Paged Optimizers

QLoRA는 small high-quality dataset을 통해 SOTA 결과를 만들어내며 이전 SOTA보다 더 작은 모델을 사용할 때보다도 더 좋은 성능을 보이는 결과가 종종 관찰된다.

또한, QLoRA 연구 과정에서, 현재 chatbot의 성능을 검증하는 benchmark들이 크게 trustworthy하지 않다는 것 또한 발견했다.

lemon-picked analysis를 통해 Guanaco가 ChatGPT와 비교했을 때 어디서 실패를 하는 지를 분석하기도 한다.

1. Introduction

LLM의 크기가 커질수록 fine-tuning하기 어려워지고, 최근 등장한 quantization도 훈련 도중에 끊긴다거나 inference에만 사용할 수 있다는 단점들이 존재한다.

우리는 어떠한 performance의 하락도 없이 quantized 4-bit model을 fine-tuning하는 것이 가능하다는 것을 발견했다.

- 따라서, pre-trained model을 4-bit로 quantize할 수 있는 novel high-precision technique을 사용하여 모델을 quantization한다.
- 이 후, 학습 가능한 작은 크기의 Low-rank Adapter weights를 추가한다.
 - quantized weights를 통해 gradient를 backpropagation해서 tuning할 수 있는 Low-rank adapter weights

65B parameter model을 fine-tuning하는 데에 있어서 780GB 이상 필요했던 메모리를 48GB 이하로 줄이는 것과 동시에 ChatGPT의 성능에 97.8%에 달할 만큼 fine-tuning을 효과적으로 진행할 수 있다.

크게 3가지의 주된 innovations이 존재한다.

- **4-bit NormalFloat**

- 정규 분포에 대해서 4-bit intergers / floats 보다 훨씬 더 좋은 실험적 결과를 도출해내는 optimal quantization data type을 의미한다.
- **Double Quantization**
 - quantization constants를 quantize하는 방법
- **Paged Optimizers**
 - 긴 sequence length를 가진 mini-batch를 처리할 때 발생하는 memory spike의 gradient checkpointing을 avoid

다양한 fine-tuning을 시도해보았으며 QLoRA에 대해서 다음과 같은 분석을 할 수 있었다.

- **data quality가 size보다 더 중요하다**는 점
- **dataset suitability가 size보다 더 중요하다**는 점
 - strong Massive Multitask Language Understanding (MMLU) benchmark의 성능이 반드시 Vicuna chatbot benchmark 성과와 직결되는 것이 아니라는 발견

더 나아가, chatbot performance를 평가하는 데에 있어서 사람이 평가한 것과 모델이 평가하는 것에 대한 추가 분석을 진행할 수 있었다.

- tournament 방식으로 평가를 진행했으며, 대체로 모델로 평가한 것과 사람이 평가한 것이 일치하는 것을 보였으나 완전히 불일치하는 결과도 확인할 수 있었다
- **결국, 사람과 모델의 평가 사이에 불확실성이 존재하기 때문에 human-annotation 대신에 cheap alternative를 사용하는 것에도 불확실성이 존재한다.**

2. Background

Block-wise k-bit Quantization

Quantization이란, 많은 정보를 가진 representation → 적은 정보의 representation으로 input을 discretizing

ex. 32-bit float → 8-bit Integers (더 많은 bit를 가지는 data type을 fewer bit data type으로 변환)

우리가 변환하고자 하는 low-bit data type의 entire range를 모두 활용하기 위해서, input data type을 target data type으로 normalization을 통해 rescaling을 하는 것이 일반적이다.

- normalization은 input element의 absolute maximum으로 진행한다.
 - 보통 tensor의 형태로 이루어져 있다.
- 예를 들어, 32-bit Floating Point (FP32)를 Int8 tensor로 변환하고자 하면 다음과 같이 quantizing 할 수 있다.

$$\mathbf{X}^{\text{Int8}} = \text{round} \left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})} \mathbf{X}^{\text{FP32}} \right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}), \quad (1)$$

여기서 c 는 *quantization constant* or *quantization scale*이라고 한다.

Dequantization은 (2)번 수식처럼 진행한다. (1)번 수식의 반대이다.

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}} \quad (2)$$

이런 방법의 문제점은 input에 매우 큰 크기의 outlier 같은 것이 포함되어 있을 때, quantization bins가 적절하게 활용되지 않는다는 점이다. 즉, 몇몇 구간 (bins)에는 숫자가 거의 할당되지 않는다는 것을 의미한다.

이 문제를 방지하기 위해, 일반적으로 input tensor를 독립적으로 quantized된 block 형태로 chunk하는 방법이다.

- block은 독립적으로 quantized되며 각자의 quantization constant를 가진다.
- ex. input tensor $\mathbf{X} \in \mathbb{R}^{b \times h} \rightarrow n$ 개의 contiguous blocks (size B)
 - input tensor를 flattening & linear segment를 slicing ($n = (b \times h)/B$)

Low-rank Adapters

full parameter를 고정해둔 채로 memory requirements를 줄여서 fine-tuning할 수 있는 기법을 의미한다.

gradient descent동안 gradient들은 fixed pre-trained model을 통해 adapter에 전해지고, 이를 통해 optimization이 진행된다.

$$\mathbf{Y} = \mathbf{XW} + s\mathbf{XL}_1\mathbf{L}_2, \quad (3)$$

Memory Requirement of Parameter-Efficient Fine-tuning

LoRA를 adapter의 크기와 개수로 살펴보자.

LoRA의 memory footprint는 매우 작기 때문에 total memory를 많이 늘리지 않고도 여러 개의 adapter를 사용함으로써 performance를 향상 시킬 수 있다.

- LoRA는 PEFT 방법으로 디자인 되었으므로, 대부분의 LLM fine-tuning의 memory footprint는 activation gradient로부터 오는 것이지, LoRA parameter로 온 것이 아니다.
- **즉, LoRA adapter를 더 많이 사용하는 것은 전체적인 training memory footprint를 늘리지 않고도 성능을 높여주는 근거가 된다.**

3. QLoRA Finetuning

QLoRA는 low-precision storage data type (4-bit) + computation data type (BFloat16)을 포함하고 있다.

즉, 실제 QLoRA weight tensor가 사용될 때는 tensor를 BFloat16으로 dequantize하고 16-bit에서 matrix multiplication을 수행한다는 것이다.

4-bit NormalFloat Quantization

▼ Quantile Quantization이란?

기존 quantization은 데이터 구간을 최솟값과 최댓값 사이로 균등하게 나눈 반면에, quantile quantization은 데이터 분포를 고려하여 구간을 나누는 것.

만약 정규 분포라면, 평균 값 근처에 많은 값들이 모여 있기 때문에 중앙 부분을 더 세밀하게 나누고 평균 값과 먼 부분들을 크게 크게 나누는 quantization을 의미한다.

- ❖ Estimate $2^k + 1$ quantiles for k bits

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

e.g., 4-bit integer (16 levels evenly spaced) reference: <https://www.youtube.com/watch?v=TPcXVJ1VSRl>

-1.0, -0.8667, -0.7333, -0.6, -0.4667, -0.3333, -0.2, -0.0667, 0.0667, 0.2, 0.3333, 0.4667, 0.6, 0.7333, 0.8667, 1.0

- ❖ Given a data, normalize its values to have zero mean and into the $[-1, 1]$ range

▪ e.g., 32 bit floating-point number, 0.5678

만약에, 32-bit floating pointer number를 4-bit로 바꾸고자 한다. 0.5678이라는 값을 4-bit integer로 일정하게 나눈 값들을 보면, 0.6과 0.4677 사이에 속하며 round를 통해 0.6으로 mapping된다. 13번째 값이 되므로, 따라서 13으로 mapping이 되는 방식을 의미한다.

NormalFloat (NF) data는 Quantile Quantization으로 구축되었으며, 정보 이론적으로 매우 optimal한 data type이다.

- **각 quantization bin이 input tensor로부터 동일한 개수의 value를 할당 받는다.**
- Quantile quantization은 empirical cumulative distribution function을 통해 input tensor의 quantile을 추정하여 작동한다.
 - empirical cumulative distribution function : 반복된 시행을 통해 확률 변수가 일정 값을 넘지 않을 확률을 유추하는 함수

기존 quantile estimates은 정확도를 높이려면 cost가 많이 소모되며, 반대로 SRAM 같이 빠른 approximation을 사용할 경우 중요한 outlier들을 놓쳐 오류 확률이 높아지는 문제점을 가진다.

하지만, input tensor가 고정된 quantization constant를 가지게 되면, 동일한 개수로 quantile할 수 있으므로 계산 효율성이 높다.

사전 학습된 neural network는 일반적으로 std가 σ 인 zero-centered normal distribution을 가지기 때문에, 모든 weight들을 σ 로 scaling한 하나의 고정된 distribution으로 바꿀 수 있다.

- 변환된 distribution은 NF data type의 range에 정확하게 맞아 떨어진다.
 - arbitrary range $[-1,1]$ 이며 data type을 위한 quantile 뿐만 아니라 neural network의 weight들도 normalization이 필요하다.

정보 이론적으로 optimal한 data type은 zero-mean normal distribution + arbitrary standard deviations σ + range $[-1,1]$ 이다. 이 data type들은 다음과 같이 계산된다.

1. k-bit quantile quantization data type을 가지는 $2^k + 1$ quantiles을 추정
2. 이 data type을 $[-1,1]$ range로 normalization
3. input weight tensor를 absolute maximum rescaling을 통해 $[-1,1]$ range로 normalizing하여 quantize

⇒ data type을 위한 quantile과 input weight tensor 모두 normalizing하는 것을 확인할 수 있음.

이를 통해 weight range와 data type range를 맞추면, quantize할 수 있다. 3번 과정이 weight tensor의 std를 rescaling하는 것과 동일하다.

2^k 개의 values가 되는 q_i 를 다음과 같이 계산한다. (즉, q_i 가 2^k 개)

$$q_i = \frac{1}{2} \left(Q_X \left(\frac{i}{2^k + 1} \right) + Q_X \left(\frac{i+1}{2^k + 1} \right) \right), \quad (4)$$

- $Q_X(\cdot)$: standard normal distribution의 quantile function

symmetric k-bit quantization은 zero를 정확한 representation으로 표현할 수 없기 때문에, zero-valued elements를 error없이 quantize하고 padding을 quantize할 수 있는 중요한 역할을 수행할 수 없다.

- 따라서, asymmetric data type을 만들어 사용한다. (4)번식.
 - 2^{k-1} 은 negative part, $2^{k-1} + 1$ 은 positive part로 표현하는 2개의 range에서 quantiles을 추정하며, 이 둘 모두에서 zero가 되는 부분에선 둘 중 하나를 삭제하는 방식
 - 이렇게 하면 0에 대한 값도 충분히 표현 가능할 것
 - 이를 통해 각 quantization bins에서 value들이 동일한 기댓값을 갖게 되는데, 이 zero-centered normally distributed data에서 정보 이론적으로 optimal하기 때문이다.

[-1.0, -0.6961928009986877, -0.5250730514526367, -0.39491748809814453, -0.28444138169288635, -0.18477343022823334, -0.09105003625154495, 0.0, 0.07958029955625534, 0.16093020141124725, 0.24611230194568634, 0.33791524171829224, 0.44070982933044434, 0.5626170039176941, 0.7229568362236023, 1.0]

전체 NF4의 값

- 0을 기준으로 +- 모두 적은 단위로 quantization → -1,1에 가까울수록 크게 quantile

Double Quantization

▼ Quantization constant란?



ChatGPT

양자화 상수(quantization constants)는 데이터 양자화 과정에서 사용되는 고정된 값들로, 데이터의 값 범위를 조정하거나, 양자화된 값들 사이의 간격을 정의하는 데 사용됩니다. 양자화는 연속적인 값을 더 작은, 고정된 수의 이산적인 값들로 매핑하는 과정입니다. 이러한 과정은 데이터 저장소 요구사항을 줄이고, 컴퓨터 시스템 내에서의 처리를 용이하게 하기 위해 주로 사용됩니다. 양자화 상수는 이 과정에서 중요한 역할을 하며, 다음 두 가지 주요 목적을 가집니다:

1. **스케일링 (Scaling) 및 범위 조정 (Range Adjustment):** 양자화 상수는 입력 데이터를 특정 범위 내로 스케일링하는 데 사용됩니다. 예를 들어, 신경망 가중치나 활성화 출력 값 같은 데이터를 -1과 1 사이의 값으로 조정할 수 있습니다. 이렇게 스케일링하는 과정에서 사용되는 고정된 비율이나 값들이 바로 양자화 상수입니다.
2. **값 간격 (Value Interval) 정의:** 양자화 과정에서는 연속적인 값들을 이산적인 값들로 매핑해야 합니다. 이 때, 어떤 값들을 어떤 이산적인 값에 할당할지를 결정하는 기준이 필요한데, 이 기준을 설정하는 데 사용되는 고정된 값들 역시 양자화 상수입니다. 예를 들어, 8비트 양자화에서는 256개의 가능한 값이 있으며, 이 때 실제 값의 범위를 이 256개의 값에 할당하는 규칙이 양자화 상수에 의해 결정됩니다.

양자화 상수의 정확한 값은 양자화를 수행하는 구체적인 알고리즘과 설정에 따라 달라집니다. 이러한 상수들은 데이터의 원본 분포를 가능한 한 잘 보존하면서, 동시에 목표하는 메모리 및 처리 성능 요구사항을 충족하기 위해 세심하게 선택되어야 합니다. 양자화 상수의 선택은 양자화된 데이터의 정밀도와 성능, 그리고 나아가 전체 시스템의 효율성에 큰 영향을 미칩니다.



- 연속적인 값을 이산화된 값으로 변형하는 것이 대표적이며, FP32 → INT8로 mapping하는 것이 가장 일반적인 예시이다.

다시 정의

- **quantization한 bit를 다시 원래 source bit로 되돌리기 위한 constant를 의미한다.**

Double Quantization은 quantization constants를 한번 더 quantizing하는 과정을 의미한다.

- 작은 block size는 더 정확한 4-bit quantization을 수행할 수 있으나 memory overhead가 증가한다는 단점이 있다.
- 예를 들어, 32-bit constants에 64 block size를 사용한다면, quantization constants는 평균 0.5 bits per parameter를 추가하는 것이다. → c_2^{FP32}

더 자세하게 살펴보면, double quantization은 quantization constants 첫 번째 quantization인 c_2^{FP32} 을 2번째 quantization의 input으로 사용한다.

두 번째 quantization을 거쳐 나온 quantization constants = c_1^{FP32}

- 8-bit floats + 256 block size를 사용하면 기존 32-bit constants에 64 block size를 사용했을 때 **0.5bits** ⇒ $8/64 + 32/(64 \times 256) = \mathbf{0.127 \text{ bits per parameters}}$

Paged Optimizers

▼ CPU RAM과 disk 사이의 paging

- 메모리 관리를 위해 사용되는 기법 중 하나로, 운영 체제가 CPU에서 사용하는 메인 메모리(RAM)와 보조 저장 장치(디스크) 사이에서 데이터를 이동시키는 과정
- 이 방법은 메인 메모리의 효율적 사용을 가능하게 하며, 프로그램이 실제 메모리 크기보다 더 많은 메모리를 사용할 수 있게 만듦
- Paging은 메모리를 동일한 크기의 블록(페이지)으로 나누고, 이 페이지들을 필요에 따라 RAM과 디스크 사이에서 교환하는 방식

GPU가 Out-Of-Memory (OOM) 문제로 processing에 오류가 생기는 상황을 방지하기 위해, CPU와 GPU 사이를 page-to-page transfers를 통해 자동적으로 메모리 관리하는 방법을 도입.

- Optimizer states를 위한 paged memory를 할당하여 GPU의 OOM이 발생하면 CPU RAM에 자동적으로 저장해줬다가 다시 optimizer update step이 되면 GPU memory로 이동

QLoRA

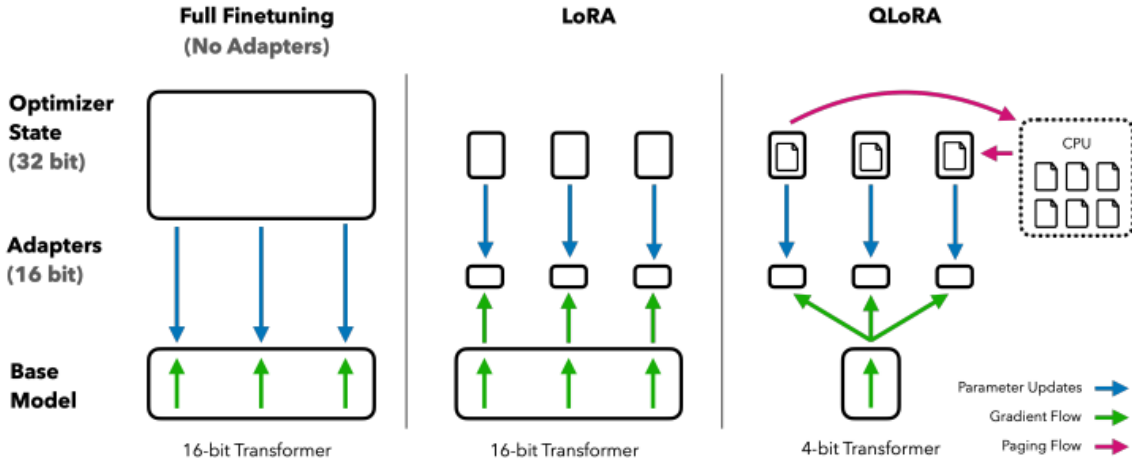


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

위에서 언급된 기법들을 통해, QLoRA를 single LoRA adapter를 가진 quantized base model 안에 single linear layer로 정의한다.

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

where $\text{doubleDequant}(\cdot)$ is defined as:

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

- parameter update를 위해서는 원래 adapter weight에 대한 error의 gradient만 계산하면 되고 4-bit weight에 대한 계산은 할 필요가 없다.
 - $\frac{\partial E}{\partial L_i}$: weight에 대한 gradient
 - $\frac{\partial E}{\partial \mathbf{W}}$: 4-bit weights
- 하지만, (6)번 과정에서도 볼 수 있듯이 error의 gradient를 계산하는 과정이 4-bit weight의 계산 과정을 포함하고 있음을 의미한다.

즉, 2가지의 data type을 사용한다.

- **4-bit NormalFloat** : **storage data type**이며 데이터를 효율적으로 저장할 수 있다.
- 16-bit BrainFloat : computation data type이며 forward, backward pass에서 더 정확한 계산 효율을 보여준다.

4. QLoRA vs Standard Finetuning

LoRA가 full fine-tuning과 동일한 성능을 내는 지, NormalFloat4와 standard Float4의 비교 분석을 다룬다.

Experimental setup

encoder / encoder-decoder / decoder-only 모델로 QLoRA와 full-fine tuning을 비교.

이 때, paged optimizers가 긴 sequence length의 mini-batch만을 처리할 때만 paging이 발생하는 것을 발견했으나, 16 batch size일 때는 일반적인 optimizer와 동일한 속도를 보여줌.

→ 추 후, paging process가 어느 부분에서 slow down을 유발하는 지에 대한 연구가 필요 할 것.

Default LoRA hyperparameters do not match 16-bit performance

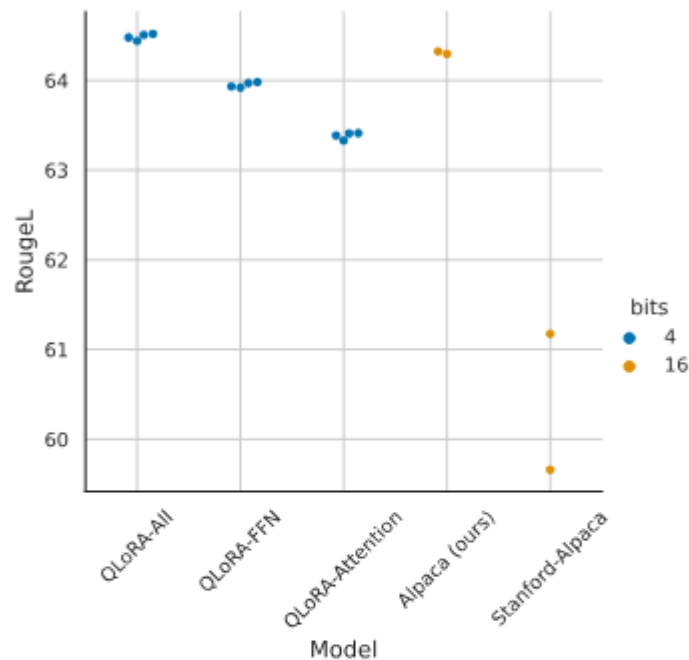


Figure 2: RougeL for LLaMA 7B models on the Alpaca dataset. Each point represents a run with a different random seed. We improve on the Stanford Alpaca fully finetuned default hyperparameters to construct a strong 16-bit baseline for comparisons. Using LoRA on all transformer layers is critical to match 16-bit performance.

- LoRA tuning에 영향을 주는 것은, LoRA adapter를 얼마나 많이 사용했는지이다.
 - QLoRA-ALL (모든 transformer block에 사용한 것) > QLoRA-FFN (FFN에만 LoRA 사용)
- 따라서, rank나 bits가 영향을 주는 것은 아니라는 점을 발견

4-bit NormalFloat yields better performance than 4-bit Floating Point

▼ Figure 3 & Table 3

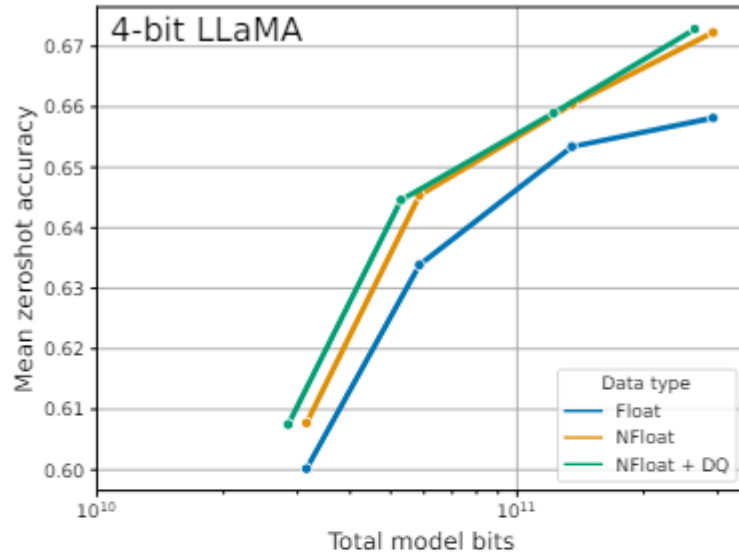


Figure 3: Mean zero-shot accuracy over Wino-grande, HellaSwag, PiQA, Arc-Easy, and Arc-Challenge using LLaMA models with different 4-bit data types. The NormalFloat data type significantly improves the bit-for-bit accuracy gains compared to regular 4-bit Floats. While Double Quantization (DQ) only leads to minor gains, it allows for a more fine-grained control over the memory footprint to fit models of certain size (33B/65B) into certain GPUs (24/48GB).

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 Model	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
	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

NF4를 사용하는 것이 일반적인 FP4와 Int4를 사용하는 것보다 더 좋은 성능을 보여준다는 것을 실험 결과로 확인할 수 있었다.

k-bit QLORA matches 16-bit full finetuning and 16-bit LoRA performance

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.

LLaMA Size Dataset	Mean 5-shot MMLU Accuracy								Mean
	7B		13B		33B		65B		
	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

- RoBERTa + T5 model (125M ~ 3B)에 대한 실험 결과 (Table 3)를 통해, 부정확한 quantization으로 인한 performance lost는 quantization 이 후, adapter fine-tuning을 통해 완전히 복구할 수 있다는 것을 확인
- double quantization을 사용한 NF4가 16-bit LoRA performance를 완전히 따라 잡을 수 있다는 것을 Table 4를 통해 확인할 수 있다.

Summary

- 4-bit QLoRA with NF4 data type으로 16-bit full fine-tuning performance를 재현
 - NF4 >> FP4
 - double quantization이 performance를 감소 시키지 않는다.

기타 실험 및 평가

Table 5: MMLU 5-shot test results for different sizes of LLaMA finetuned on the corresponding datasets using QLoRA.

Dataset	7B	13B	33B	65B
LLaMA no tuning	35.1	46.9	57.8	63.4
Self-Instruct	36.4	33.3	53.0	56.7
Longform	32.1	43.2	56.6	59.7
Chip2	34.5	41.6	53.6	59.8
HH-RLHF	34.9	44.6	55.8	60.1
Unnatural Instruct	41.9	48.1	57.3	61.3
Guanaco (OASST1)	36.6	46.4	57.0	62.2
Alpaca	38.8	47.8	57.3	62.5
FLAN v2	44.5	51.4	59.2	63.9

- QLoRA를 사용한 Guanaco가 다른 fine-tuning 모델 및 foundation model에도 뒤처지지 않는 performance를 보이고 있다는 것을 보여줌.

8. Limitations and Discussion

- 4-bit base model + LoRA를 통해 16-bit full fine-tuning performance를 재현
- 하지만, 33B와 65B scale에서는 full fine-tuning performance를 따라가지 못했으므로 future work로 제시

또한, fine-tuning model에 대한 evaluation 방법에 대한 여러가지 의문점 제기

- 일부 벤치 마크 데이터들의 결과만 제시함으로써 다른 벤치 마크에 대해서도 좋은 성능을 보일 지에 대해서는 의문
- chatbot benchmark 또한 Guanaco를 비롯한 여러 성능들을 확인하는 데에 있어서 불분명

마지막으로, 다른 bit-precision에 대해서 평가해보지 않았다는 점

- 3-bit base model이나 혹은 다른 adapter method를 사용해보지 않았다는 점.

9. Broader Impacts

QLoRA를 통해 LLM을 단일 GPU로 fine-tuning할 수 있으며 QLoRA의 방법을 통해서 LLM을 mobile phone으로도 구현 할 수 있을 것이라는 잠재적인 가능성을 확인할 수 있었음.