QDyLoRA: Quantized Dynamic Low-Rank Adaptation for Efficient Large Language Model Tuning

Hossein Rajabzadeh[1,2], **Mojtaba Valipour**[1,2], **Marzieh Tahaei**[2],
Hyock Ju Kwon[1], **Ali Ghodsi**[1], **Boxing Chen**[2], and **Mehdi Rezagholizadeh**[2]
[1]University of Waterloo
[2]Huawei Noah's Ark Lab
{hossein.rajabzadeh, mojtaba.valipour, hjkwon, ali.ghodsi}@uwaterloo.ca,
{mehdi.rezagholizadeh, marzieh.tahaei, boxing.chen}@huawei.com

Abstract

Finetuning large language models requires huge GPU memory, restricting the choice to acquire Leger language models. While the quantized version of the Low-Rank Adaptation technique, named QLoRA, significantly alleviates this issue, finding the efficient LoRA rank is still challenging. Moreover, QLoRA is trained on a pre-defined rank and, therefore, cannot be reconfigured for its lower ranks without requiring fine-tuning steps. This paper proposes QDyLoRA-Quantized Dynamic Low-Rank Adaptation-, as an efficient quantization approach for dynamic low-rank adaptation. QDyLoRA combines the advantages of QLoRA with Dynamic LoRA to efficiently finetune LLMs on a set of pre-defined LoRA ranks. QDyLoRA enables fine-tuning Falcon-40b for ranks 1 to 64 on a single 32GiG V100-GPU through one round of fine-tuning. Experimental results show that QDyLoRA is competitive to QLoRA and outperforms when employing its optimal rank.

1 Introduction

The popularity of adopting Large Language Models (LLMs) across a diverse range of downstream tasks has rapidly increased over the past two years. Fine-tuning LLMs has become necessary to enhance their performance and introduce desired behaviors while preventing undesired outputs . However, as the size of these models increases, fine-tuning costs become more expensive. This has led to a large body of research that focuses on improving the efficiency of the fine-tuning stage Liu et al. [2022], Mao et al. [2021], Hu et al. [2021], Edalati et al. [2022], Sung et al. [2022].

Low-rank adapter (LoRA) Hu et al. [2021] is a well-known, parameter-efficient tuning (PET) method that reduces memory requirements during fine-tuning by freezing the base model and updating a small set of trainable parameters in form of low-rank matrix multiplication added to matrices in the base model. However, the memory demand during fine-tuning remains substantial due to the necessity of a backward pass through the frozen base model during stochastic gradient descent.

Recent research has thus focused on further reducing memory usage by designing new parameter-efficient modules that can be tuned without necessitating gradients from the base models Sung et al. [2022]. Alternatively, researchers have explored combining other efficiency strategies with parameter-efficient tuning methods Kwon et al. [2022], Dettmers et al. [2023].

Among these approaches, QLoRA Dettmers et al. [2023] stands out as a recent and highly efficient fine-tuning method that dramatically decreases memory usage. It enables fine-tuning of a 65-billion-parameter model on a single 48GB GPU while maintaining full 16-bit fine-tuning performance.

37th Conference on Neural Information Processing Systems - ENLSP (NeurIPS 2023).

QLoRA achieves this by employing 4-bit NormalFloat (NF4), Double Quantization, and Paged Optimizers as well as LoRA modules.

However, another significant challenge when utilizing LoRA modules is the need to tune their rank as a hyperparameter. Different tasks may require LoRA modules of varying ranks. In fact, it is evident from the experimental results in the LoRA paper that the performance of models varies a lot with different ranks, and there is no clear trend indicating the optimal rank. On the other hand, any hyperparameter tuning for finding the optimal rank contradicts the primary objective of efficient tuning and is not feasible for very large models. Moreover, when deploying a neural network on diverse devices with varying configurations, the use of higher ranks can become problematic for highly sensitive devices due to the increased parameter count. To address this, one typically has to choose between training multiple models tailored to different device configurations or determining the optimal rank for each device and task. However, this process is costly and time-consuming, even when using techniques like LoRA.

DyLoRA Valipour et al. [2022], is a recent PET method that aims to address theses challenges by employing dynamic Low-Rank Adapter (DyLoRA). Inspired by nested dropout, this method aims to order the representations of the bottleneck at low-rank adapter modules. Instead of training LoRA blocks with a fixed rank, DyLoRA extends training to encompass a spectrum of ranks in a sorted manner. The resulting low-rank PET modules not only provide increased flexibility during inference, allowing for the selection of different ranks depending on the context, but also demonstrate superior performance compared to LoRA, all without imposing any additional training time.

In this paper, we employ the DyLoRA PET method in conjunction with the quantization scheme utilized in the QLoRA work, resulting in QDyLoRA. QDyLoRA has all the aforementioned benefits of DyLoRA but with significant memory reduction both during training and at inference through 4-bit quantization. We utilize QDyLoRA for efficient fine-tuning of LLaMA-7b, LLaMA-13b, and Falcon-40b models across ranks ranging from 1 to 64, all on a single 32GB V100 GPU. Once tuned, we determine the optimal rank by inferring the model on the test set. Our results reveal that the optimal rank can be quite low, yet it outperforms QLoRA.

1.1 Related Work

Low-rank PET methods These methods aim to fine-tune pre-trained LLMs for specific tasks while minimizing computational and memory resources. Low-rank adaptation techniques were inspired by Aghajanyan et al. [2020], demonstrating that pre-trained language models possess a low intrinsic dimension. Since then, several works have explored the incorporation of trainable parameters in the form of low-rank up-projection/down-projection during fine-tuning. In Houlsby et al. [2019], the Adapter module includes a down projection, a non-linear function, an up projection, and a residual connection. These modules are sequentially inserted after the feed-forward network (FFN) or attention blocks. Additionally, He et al. [2021] extends the Adapter concept by introducing trainable modules that run in parallel (PA) with the original PLM module. As a result of this extension, PA has demonstrated improved performance compared to the original Adapter method. One notable approach among these techniques is LoRA Hu et al. [2021], which introduces low-rank up-projection/down-projection into various matrices within a PLM. This method offers efficient inference by seamlessly integrating the adapter module into the original model's weight matrices.

Quantization aware PET methods Alpha-Tuning Kwon et al. [2022], aims to combine parameter-efficient adaptation and model compression. Alpha-Tuning achieves this by employing post-training quantization, which involves converting the pre-trained language model's full-precision parameters into binary parameters and separate scaling factors. During adaptation, the binary values remain fixed for all tasks, while the scaling factors are fine-tuned for the specific downstream task.

QLoRA Dettmers et al. [2023] is a more recent quantization-aware PET that combines a low-rank adapter with 4-bit NormalFloat (NF4) quantization and Double Quantization (DQ) of the base model to optimize memory usage. NF4 ensures an optimal distribution of values in quantization bins, simplifying the process when input tensors have a fixed distribution. DQ further reduces memory overhead by quantizing quantization constants. To manage memory during gradient checkpointing, QLoRA employs Paged Optimizers, utilizing NVIDIA's unified memory feature for efficient GPU memory management. These techniques collectively enable high-fidelity 4-bit fine-tuning while effectively handling memory constraints.

Dynamic PET methods DyLoRA paper Valipour et al. [2022] introduces a novel approach for training low-rank modules to work effectively across a range of ranks simultaneously, eliminating the need to train separate models for each rank. Inspired by the concept of nested dropout, the authors propose a method for organizing the representations within low-rank adapter modules. This approach aims to create dynamic low-rank adapters that can adapt well to various ranks, rather than being fixed to a single rank with a set training budget. This is achieved by dynamically selecting ranks during training, allowing for greater flexibility without the need for extensive rank searching and multiple model training sessions.

Algorithm 1 QDyLoRA - Training and Inference

```
Require: $r \in [r_{min}, r_{max}]$; $i$: the number of training iterations; $\alpha$: a scaling factor; $p_B$: probability
  distribution function for rank selection; $X \in \mathbb{R}^{d \times n}$ : all input features to LoRA; $W_0 \in \mathbb{R}^{m \times d}$
  the original frozen pre-trained weight matrix, $W_{dw} \in \mathbb{R}^{r \times d}$; $W_{up} \in \mathbb{R}^{m \times r}$; $Q$: Quantizer; $\mathbb{L}^{DY}_{\downarrow b}$:
  objective function given truncated weights
  Initialization:
  $W_0^{NF4} = Q(W_0)$ // Quantize $W_0$ to NF4
  Iterations:
  while t < $i$ do:
    Forward:
    $b \sim p_B(.)$ // sample a specific rank, during test is given
    $W_{dw\downarrow b} = W_{dw}[:b,:]$ // truncate down-projection matrix
    $W_{up\downarrow b} = W_{up}[:,:b]$ // truncate up-projection matrix
    $W_0^{DDequant-NF4} = \frac{W_0^{NF4}}{c_2^{FP8}/c_1^{FP32}}$ // dequantized the chunks of the parameters that are needed
    $h = W_0^{DDequant-NF4} X^{BF16} + \frac{\alpha}{b} W^{BF16}_{up\downarrow b} W^{BF16}_{dw\downarrow b} X^{BF16}$ // calculate the LoRA output
    Backward:
    $W^{BF16}_{dw\downarrow b} \leftarrow W^{BF16}_{dw\downarrow b} - \eta \nabla_{W^{BF16}_{dw\downarrow b}} \mathcal{L}^{\mathcal{DY}}_{\downarrow b}$
    $W^{BF16}_{up\downarrow b} \leftarrow W_{up\downarrow b} - \eta \nabla_{W^{BF16}_{up\downarrow b}} \mathcal{L}^{\mathcal{DY}}_{\downarrow b}$
  end while
```

2 Proposed Method: Quantized DyLoRA

Following DyLoRA notations Valipour et al. [2022], we define a truncated weight $W_{\downarrow b} \in \mathbb{R}^{r \times d}$ as $W[: b, :]$. Assume we have a set of input features $X \in \mathbb{R}^{d \times n}$, a set of pre-trained weights W_0, and a given range of desired ranks represented by $r \in [r_{min}, r_{max}]$ that we want the model to operate with, and a dynamic objective function $L^{DY}_{\downarrow b}$ that can evaluate a truncated sub-model. Then we can use the following equation to calculate the forward pass of the model at each iteration.

$$h = W_0^{DDequant-NF4} x^{BF16} + \frac{\alpha}{b} W^{BF16}_{up\downarrow b} W^{BF16}_{dw\downarrow b} x^{BF16} \tag{1}$$

where α is the LoRA scalar, and b is the chosen rank by the $p_B(.)$ during training stage.

Following QLoRA Dettmers et al. [2023], we used 4-bit Normal Float (NF4) for storing the double quantized pre-trained weights. As all the computations need to be calculated in BFloat16 precision, DDequant-NF4 will dequantize the stored data. Similar to Dettmers et al. [2023], we have:

$$W_0^{DDequant-NF4} = \frac{W_0^{NF4}}{c_2^{FP8}/c_1^{FP32}} \tag{2}$$

where c_1^{FP32} and c_2^{FP8} are quantization constants introduced in Dettmers et al. [2023]. After this process, we can update the dynamic LoRA parameters using:

$$\begin{aligned} W^{BF16}_{dw\downarrow b} &\leftarrow W^{BF16}_{dw\downarrow b} - \eta \nabla_{W^{BF16}_{dw\downarrow b}} \mathcal{L}^{\mathcal{DY}}_{\downarrow b} \\ W^{BF16}_{up\downarrow b} &\leftarrow W_{up\downarrow b} - \eta \nabla_{W^{BF16}_{up\downarrow b}} \mathcal{L}^{\mathcal{DY}}_{\downarrow b} \end{aligned} \tag{3}$$

Table 1: A comparison between QLoRA and QDyLoRA on the MMLU benchmark, reporting 5-shot test results for LLMs of varying sizes. QDyLoRA is evaluated on ranks [1,2,4,8,16,32,64] and the best rank is reported in brackets.

Dataset	LLaMA-7b		LLaMA-13b		Falcon-40b	
	QLoRA	QDyLoRA	QLoRA	QDyLoRA	QLoRA	QDyLoRA
Alpaca	38.8 [64]	39.7 [16]	47.8 [64]	47.6 [8]	55.2 [64]	57.1 [4]
OASST1	36.6 [64]	36.8 [16]	46.4 [64]	47.2 [8]	56.3 [64]	56.7 [4]
Self-Instruct	36.4 [64]	37.2 [8]	33.3 [64]	41.6 [4]	51.8 [64]	51.1 [4]
FLAN-v2	44.5 [64]	45.9 [4]	51.4 [64]	52.1 [8]	58.3 [64]	60.2 [4]

Table 2: Comparing the performance of QLoRA and QDyLoRA across different evaluation ranks. Both models receives the same training settings. Maximum LoRA rank is set to 64. Falcon-40b is adopted as the base LLM. Exact matching and Bleu-score are used as evaluation measurements for GSM8k and Web-GLM, respectively.

Data	Method	Rank						
		1	2	4	8	16	32	64
Web-GLM	QLoRA	19.9	19.9	19.9	33.8	35.2	52.7	54.3
	QDyLoRA	43.3	**56.0**	54.9	53.3	53.3	50.5	50.2
GSM8k	QLoRA	8.9	8.91	8.9	15.1	20.5	22.6	28.1
	QDyLoRA	21.4	25.3	28.2	**30.6**	29.8	28.5	27.4

Algorithm 1 describes the workflow of our proposed QDyLoRA in detail.

3 Experiments and Evaluation

This section evaluates the efficiency and efficacy of QDyLoRA through several instruct-fine-tuning tasks. The first experiment compares QDyLoRA with QLoRA on Massively Multitask Language Understating (MMLU) benchmark Hendrycks et al. [2020], consisting of more than 50 different tasks, spanning from fundamental mathematics and U.S. history to computer science and law. As shown in Table 1[1], we finetune LLaMA-7b, LLaMA-13b, and Falcon40b on different datasets, Alpaca Taori et al. [2023], OASST1 Köpf et al. [2023], Self-Instruct Wang et al. [2022], and FLAN-v2 Chung et al. [2022], using QLoRA and QDyLoRA techniques. We use the same training budget and maximum LoRA rank [2] for each technique. The results show that QDyLoRA achieves a superior performance by finding the optimal rank.

The second experiment provides a more in-depth comparison between QLoRA and QDyLoRA. In particular, we fairly finetuned Falcon-40b on WebGLM Liu et al. [2023] and GSM8k Cobbe et al. [2021] benchmarks, and compared their test performances across different ranks. As described in Table 2, QDyLoRA attains superior performance, notably when employing its optimal ranks (Rank 2 for Web-GLM and Rank 8 for GSM8k). Furthermore, QDyLoRA exhibits consistent superiority over QLoRA, particularly at lower ranks.

4 Conclusion

QDyLoRA offers an efficient and effective technique for LoRA-based fine-tuning LLMs on downstream tasks. Eliminating the need for fine-tuning multiple models to find the optimal LoRA rank and offering the possibility of fine-tuning larger LLMs are two main advantages of QDyLoRA. The experimental results demonstrated that the optimal rank for QDyLoRA can be surprisingly low, yet it consistently outperforms QLoRA. QDyLoRA provides greater flexibility for deploying LLMs in

[1] The same settings as the original QLoRA work are applied here.

[2] The maximum LoRA rank is fixed to 64. While QLoRA's rank is always fixed, QDyLoRA can split the training across ranks in range 1 to 64.

various contexts and represents a promising step towards making fine-tuning large language models more accessible and efficient.

References

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35:12991–13005, 2022.

Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. *arXiv preprint arXiv:2210.03858*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations–democratizing large language model alignment. *arXiv preprint arXiv:2304.07327*, 2023.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. Webglm: Towards an efficient web-enhanced question answering system with human preferences. *arXiv preprint arXiv:2306.07906*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

5 Supplementary Material

5.1 Hyperparameters

This section provides an overview of the hyperparameters and experimental configurations, detailed in Table 3.

Model	Parameter	Value
BERT-Base	Optimizer	AdamW
	Warmup Ratio	0.06
	Dropout	0.1
	LR Scheduler	Linear
	Batch Size	32 (RoBertA) / 8 (Bert)
	Epochs	30 (RoBertA) / 3,6 (Bert)
	Learning Rate (LR)	2e-5 (RoBertA / 6e-6 (Bert)
	Weight Decay	0.1
	Max Sequence Length	512
	Seeds	[10, 110, 1010, 42, 4242]
	GPU	Tesla V100-PCIE-32GB
MobileNetV2	Model	"google/mobilenet_v2_1.4_224"
	Optimizer	AdamW
	LR Scheduler	Linear
	Batch Size	128
	Seeds	4242
	Epochs	60 $\times$# Models
	GPU	8 $\times$ Tesla V100-PCIE-32GB
cPreResNet20	Optimizer	SGD
	Criterion	Cross Entropy
	LR Scheduler	cosine_lr
	Batch Size	128
	Seed	40
	Momentum	0.9
	Weight Decay	0.0005
	LR	0.1
	Epochs	[200,400,600,800]
	Gradient Accumulation	[1,2,3,4]

Table 3: All the hyperparameters that have been used throughout our study for different experiments. If we didn't mention a parameter specifically, it means we utilized the default value of the HuggingFace Transformers v'4.27.0.dev0'. [4]. Otherwise, we highlighted any exception in the main text.