MoRA: High-Rank Updating for Parameter-Efficient Fine-Tuning

Anonymous authors
Paper under double-blind review

Abstract

Low-rank adaptation (LoRA) is a popular parameter-efficient fine-tuning (PEFT) method for large language models (LLMs). In this paper, we analyze the impact of low-rank updating, as implemented in LoRA. Our findings suggest that the low-rank updating mechanism may limit the ability of LLMs to effectively learn and memorize new knowledge. Inspired by this observation, we propose a new method called MoRA, which employs a square matrix to achieve high-rank updating while maintaining the same number of trainable parameters. To achieve it, we introduce the corresponding non-parameter operators to reduce the input dimension and increase the output dimension for the square matrix. Furthermore, these operators ensure that the weight can be merged back into LLMs, which enables our method to be deployed like LoRA. We perform a comprehensive evaluation of our method across five tasks: instruction tuning, mathematical reasoning, continual pretraining, memory and pretraining. Our method outperforms LoRA on memory-intensive tasks and achieves comparable performance on other tasks.

1 Introduction

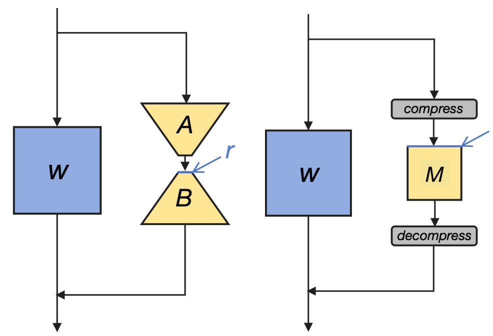
As the size of language models increases, parameter-efficient fine-tuning (PEFT) Houlsby et al. (2019) has emerged as a popular technique to adapt these models to specific downstream tasks. Compared to Full Fine-Tuning (FFT), which updates all model parameters, PEFT modifies only a small part of the parameters. For example, it can achieve similar performance with FFT by updating less than 1% of the parameters in some tasks Hu et al. (2021), which significantly reduces the memory requirements for the optimizer and facilitates the storage and deployment of fine-tuned models.

Among the existing PEFT methods, Low-Rank Adaptation (LoRA) Hu et al. (2021) is particularly prevalent for LLMs. LoRA enhances performance over other PEFT methods such as prompt tuning Lester et al. (2021) or adapters Houlsby et al. (2019) by updating parameters via low-rank matrices. These matrices can be merged into the original model parameters, thereby avoiding additional computational costs during inference. There are numerous methods that aim to improve LoRA for LLMs. However, most methods primarily validate their efficiency based on GLUE Wang et al. (2018), either by achieving better performance or by requiring fewer trainable parameters. Recent methods Liu et al. (2024); Meng et al. (2024); Zhu et al. (2024) leverage instruction tuning task such as Alpaca Wang et al. (2024) or reasoning tasks like GSM8K Cobbe et al. (2021) to better evaluate their performance on LLMs. However, the diverse settings and datasets used in the evaluation complicate the understanding of their progression.

In this paper, we conduct a comprehensive evaluation of LoRA across various tasks under the same settings, including instruction tuning, mathematical reasoning, and continual pretraining. We find that LoRA-like methods demonstrate similar performance across these tasks and they perform comparably to FFT in instruction tuning but fall short in mathematical reasoning and continual pretraining. Among these tasks, instruction tuning primarily focuses on interacting with the format, rather than acquiring knowledge and capabilities, which are learned almost entirely during pretraining Zhou et al. (2024). We observe that LoRA is easily adapted to follow response formats in instruction tuning but struggles with other tasks that require enhancing knowledge and capabilities through fine-tuning.

One plausible explanation for this limitation observed with LoRA could be its reliance on low-rank updates Lialin et al. (2023). The low-rank update matrix, ΔW, struggles to estimate the full-rank updates in FFT, particularly in memory-intensive tasks like continual pretraining that require memorizing domain-specific knowledge. Since the rank of ΔW is significantly smaller than the full rank, this limitation restricts capacity to store new information via fine-tuning. Moreover, current variants of LoRA cannot alter the inherent characteristic of low-rank updates. To validate this, we conducted a memorization task using pseudo-data to assess the performance of LoRA in memorizing new knowledge. We found that LoRA performed significantly worse than FFT, even with a large rank such as 256.

Given these observations, we introduce a method called MoRA, which employs a square matrix as opposed to low-rank matrices, aiming to maximize the rank in ΔW while maintaining the same number of trainable parameters. For instance, when utilizing 8 rank with the hidden size 4096, LoRA employs two low-rank matrices $A \in \mathbb{R}^{4096\times 8}$ and $B \in \mathbb{R}^{8\times 4096}$, with $rank(\Delta W) \leq 8$. Under same number of parameters, our method uses a square matrix $M \in \mathbb{R}^{256\times 256}$, with $rank(\Delta W) \leq 256$, as depicted in Figure 1. Notably, our method exhibits a greater capacity than LoRA with a large rank. To decrease the input dimension and increase the output dimension for M, we develop corresponding non-parameter operators. Furthermore, these operators and M can be substituted by a ΔW, ensuring our method can be merged back into LLM like LoRA.



(a) LoRA ($r = 8$) (b) MoRA ($r = 256$)

Figure 1: An overview of our method compared to LoRA under **same** number of trainable parameters. W is the frozen weight from model.r represents the rank in two methods.

Our contributions are as follows:

1. We introduce MoRA, a novel method that employs a square matrix instead of low-rank matrices in LoRA to achieve high-rank updating, while maintaining the same number of trainable parameters.
2. We discuss four kinds of non-parameter operators of MoRA to reduce the input dimension and increase the output dimension for the square matrix, while ensures that the weight can be merged back into LLMs.
3. We evaluate MoRA across five tasks: memory, instruction tuning, mathematical reasoning, continual pretraining, and pretraining. Our method outperforms LoRA on memory-intensive tasks and achieves comparable performance on other tasks, which demonstrates the effectiveness of high-rank updating.

2 Related Work

2.1 LoRA

LoRA is one of the most popular PEFT methods for fine-tuning LLM, owing to its broad applicability and robust performance in comparison to other methods. To approximate the updated weight ΔW in FFT, LoRA employs two low-rank matrices for its decomposition. By adjusting the rank of these two matrices, LoRA can accordingly modify the trainable parameters. As a result, LoRA can merge these matrices after fine-tuning without incurring the inference latency associated with FFT. There are many methods to further improve LoRA, particularly for the application in LLMs. DoRALiu et al. (2024) further decomposes the original weight into magnitude and direction components and uses LoRA to update the direction component. LoRA+Hayou et al. (2024) employs different learning rates for the two low-rank matrices to improve learning efficiency. ReLoRALialin et al. (2023) integrates LoRA into the LLM during training to increase the rank of the final ΔW.

2.2 Fine-Tuning with LLMs

Despite the impressive performance of LLMs with in-context learning, certain scenarios still necessitate fine-tuning, which can be broadly categorized into three types. The first type, instruction tuning, aims to better align LLMs with end tasks and user preferences, without significantly enhancing the knowledge and capabilities of LLMs Zhou et al. (2024). This approach simplifies the process of dealing with varied tasks and understanding complex instructions. The second type involves complex reasoning tasks such as mathematical problem-solving Collins et al. (2023); Imani et al. (2023); Yu et al. (2023), where general instruction tuning often falls short in handling complex, symbolic, multi-step reasoning tasks. To improve the reasoning abilities of LLMs, the majority of research focuses on creating corresponding training datasets, either by leveraging larger teacher models like GPT-4 Fu et al. (2023), or by rephrasing questions along a reasoning path Yu et al. (2023). The third type, continual pretraining Cheng et al. (2023); Chen et al. (2023); Han et al. (2023); Liu et al. (2023), aims to enhance the domain-specific capabilities of LLMs. Unlike instruction tuning, it necessitates fine-tuning to augment the corresponding domain-specific knowledge and capabilities.

However, most variants of LoRA Kopiczko et al. (2023); Lialin et al. (2023); Dettmers et al. (2024); Zhu et al. (2024) predominantly employ instruction tuning or text classification tasks from GLUE Wang et al. (2018) to validate their efficacy on LLMs. Given that instruction tuning requires the least capacity for fine-tuning compared to other types, it may not accurately reflect the effectiveness of LoRA variants. To better evaluate their methods, recent works Meng et al. (2024); Liu et al. (2024); Shi et al. (2024); Renduchintala et al. (2023) have employed reasoning tasks to test their methods. But the training sets used are often too small for LLMs to effectively learn reasoning. For instance, some methods Meng et al. (2024); Renduchintala et al. (2023) utilize the GSM8K Cobbe et al. (2021) with only 7.5K training samples. Compare to the SOTA method with 395K training samples Yu et al. (2023), this small training set achieves worse performance on reasoning and makes it hard to evaluate the effectiveness of these methods.

3 Analysis the Influence of Low-rank Updating

The key idea of LoRA Hu et al. (2021) involves the use of low-rank updates to estimate full-rank updates in FFT. Formally, given a pretrained parameter matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA employs two low-rank matrices to calculate the weight update ΔW:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{1}$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ represent the low-rank matrices in LoRA. To ensure that $\Delta W = 0$ at the beginning of training, LoRA initializes A with a Gaussian distribution and B with zero. Due to the low-rank decomposition of ΔW into BA, the $rank(\Delta W) \leq r$. The weight update in LoRA exhibits a markedly low rank, $r \ll \min(d, k)$, in comparison to the full-rank updating in FFT. Low-rank updating by LoRA shows on-par performance with full-rank updating in some tasks such as text classification or instruction tuning Liu et al. (2024); Meng et al. (2024). However, for tasks like complex reasoning or continual pretraining, LoRA tends to show worse performance Liu et al. (2023).

Based on these observations, we hypothesize that low-rank updating easily leverages the original knowledge and capabilities of LLMs to solve tasks but struggles with tasks that require enhancing the knowledge and capabilities of LLMs.

To substantiate this hypothesis, we examine the differences between LoRA and FFT in terms of memorizing new knowledge through fine-tuning. In order to circumvent leveraging the original knowledge of the LLM, we randomly generate 10K pairs of Universally Unique Identifiers (UUIDs), each pair comprising two UUIDs with 32 hexadecimal values. The task requires the LLM to generate the corresponding UUID based on the input UUID. For instance, given a UUID such as "205f3777-52b6-4270-9f67-c5125867d358", the model should generate the corresponding UUID based on 10K training pairs. This task can also be viewed as a question-answering task, while the knowledge indispensable for accomplishing it is exclusively from the training datasets rather than the LLM itself.

For the training settings, we employ LLaMA-2 7B as base model, utilizing 1,000 pairs per batch and conducting 100 epochs. For the LoRA, we apply low-rank matrices to all linear layers and search learning rate from {1e-4,2e-4,3e-4} to enhance performances.

We conduct the experiment on LoRA using various ranks $r \in \{8, 16, 32, 64, 128, 256\}$. For the FFT, we directly use a learning rate of 3e-5. Based on Figure 2, we observe low-rank updating are hard to memorizing new knowledge compared to FFT. Although constantly increasing the rank of LoRA can alleviate this problem, the gap still exists.

In contrast to the memory task, we also evaluate the performance gap between LoRA and FFT on instruction tuning, which merely introduces new knowledge. Similar to previous results Meng et al. (2024); Zhu et al. (2024), we also find that LoRA matches the performance of FFT with small rank $r = 8$ in Table 2. This indicates that LoRA can easily leverage the original knowledge of LLMs by fine-tuning like FFT.

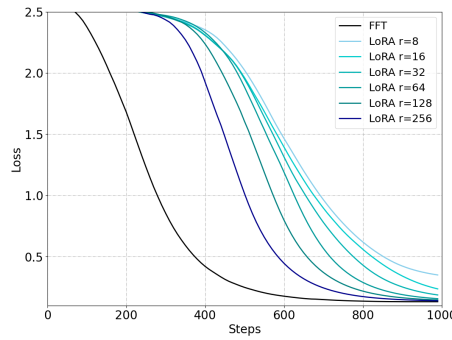


Figure 2: Performance of memorizing UUID pairs through fine-tuning with FFT and LoRA.

4 Method

Based on the above analysis, we propose a new method to alleviate the negative effects of low-rank updating. The main idea of our method is to utilize the same trainable parameters as much as possible to achieve a higher rank in ΔW. Consider the pretrained weight $W_0 \in \mathbb{R}^{d\times k}$. LoRA uses two low-rank matrices A and B with $(d + k)r$ total trainable parameters for rank r. Under same trainable parameters, a square matrix $M \in \mathbb{R}^{\hat{r}\times\hat{r}}$ where $\hat{r} = \lfloor \sqrt{(d + k)r} \rfloor$ can achieve the highest rank due to $r \ll \min(d, k)$.

To accomplish this, we need to reduce the input dimension and increase the output dimension for M. Formally,

$$h = W_0 x + f_{\text{decomp}}\left(M f_{\text{comp}}\left(x\right)\right) \tag{2}$$

where $f_{\text{comp}} : \mathbb{R}^k \rightarrow \mathbb{R}^{\hat{r}}$ denotes the function that decreases the input dimension of x from k to $\hat{r}$, and $f_{\text{decomp}} : \mathbb{R}^{\hat{r}} \rightarrow \mathbb{R}^d$ represents the function that enhances the output dimension from $\hat{r}$ to d. Furthermore, these two functions ought to be non-parameterized operators and expected to execute in linear time corresponding to the dimension. They should also have corresponding function, $f_{\overline{\text{comp}}} : \mathbb{R}^{\hat{r}\times\hat{r}} \rightarrow \mathbb{R}^{\hat{r}\times k}$ and $f_{\overline{\text{decomp}}} : \mathbb{R}^{\hat{r}\times k} \rightarrow \mathbb{R}^{d\times k}$, to transform M into ΔW. For any x, the following should hold:

$$f_{\text{decomp}}\left(M f_{\text{comp}}\left(x\right)\right) = \Delta W x, \forall x \in \mathbb{R}^k \tag{3}$$

where $\Delta W = f_{\overline{\text{decomp}}}\left(f_{\overline{\text{comp}}}\left(M\right)\right)$. If Eq. 3 holds, M can be losslessly expanded to ΔW based on f_{comp} and f_{decomp}. This allows our method to merge back into the LLM like LoRA.

For the design of f_{comp} and f_{comp}, we explore several methods to implement these functions. One straightforward method is truncating the dimension and subsequently add it in corresponding dimension. Formally, this can be represented as:

$$\begin{aligned} f_{\text{comp}}\left(x\right) &= x_{1:\hat{r}} \\ f_{\text{decomp}}\left(x\right) &= \begin{bmatrix} x \\ \mathbf{0} \end{bmatrix} \end{aligned} \tag{4}$$

and the corresponding ΔW is:

$$\Delta W = \begin{bmatrix} M & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{5}$$

However, this method leads to a significant loss of information during compression and only modifies a segment of the output by appending a zero vector during decompression. To improve it, we can share the rows and columns of M to achieve a more efficient compression and decompression. Formally, this can be represented as:

$$\begin{aligned} f_{\text{comp}}\left(x\right) &= \left[\sum\nolimits_{j\in g_i} x_j\right]_{i=1}^{r} \\ f_{\text{decomp}}\left(x\right) &= \left[x_{\widetilde{g}'_i}\right]_{i=1}^{d} \end{aligned} \tag{6}$$

Here, g and g' represent predefined groups that share the same row and column in M, respectively. The $j \in g_i$ indicates that the j-th dimension belongs to the i-th group in g. The term $\widetilde{g}'_i$ is the reverse of g'_i, referring to the i-th dimension associated with the $\widetilde{g}'_i$-th group in g'. The corresponding ΔW is as follows:

$$\Delta W_{i,j} = M_{\widetilde{g}'_i, \widetilde{g}_j} \tag{7}$$

Sharing rows and columns can be efficient for larger ranks such as $r = 128$ or $r = 256$, as only a few rows or columns in ΔW share a common row or column. For instance, considering to $\Delta W \in \mathbb{R}^{4096\times 4096}$ for $r = 128$, which has $\hat{r} = 1024$ and $M \in \mathbb{R}^{1024\times 1024}$. In this situation, only 4 rows or columns share the same row or column. Conversely, for smaller ranks such as $r = 8$, where $\hat{r} = 256$, it requires average 16 rows or columns in a group to share the same row or column in M. It can lead to inefficiencies due to the significant information loss during compression in Eq. 6.

To enhance performance for smaller ranks, we reshape x instead of directly compressing it, to preserve the input information. In this context, $f_{\text{comp}}(x) : \mathbb{R}^k \to \mathbb{R}^{n\times\hat{r}}$ and $f_{\text{decomp}} : \mathbb{R}^{n\times\hat{r}} \to \mathbb{R}^d$. Corresponding f_{comp}, f_{decomp} and ΔW are as follows:

$$\begin{aligned} f_{\text{comp}}(x) &= \begin{bmatrix} x_{1:\hat{r}} & x_{\hat{r}:2\hat{r}} & \cdots & x_{(n-1)\hat{r}:n\hat{r}} \end{bmatrix} \\ f_{\text{decomp}}(x) &= \text{concat}(x) \\ \Delta W &= \begin{bmatrix} M & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & M & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & M \end{bmatrix} \end{aligned} \tag{8}$$

where concat(x) refers to concatenate the rows of x into a vector. For simplicity, we omit the padding and truncation operators in above functions and focus on the case where $d = k$. In comparison to sharing columns and rows, this method incurs additional computational overhead by reshaping x into $\mathbb{R}^{n\times\hat{r}}$ instead of $\mathbb{R}^{\hat{r}}$. However, given that the size of M is significantly smaller than W_0, this additional computation is very small for rank like 8. For instance, when fine-tuning the 7B model with rank of 8 ($\hat{r} = 256$), this method is only 1.03 times slower than the previous methods.

Inspired by RoPE Su et al. (2024), we can further refine this method by incorporating rotation operators into f_{comp} to augment the expressiveness of M by enable it to differentiate between various $x_{i\hat{r}:(i+1)\hat{r}}$ by rotating them. We can modify Eq. 8 as follows:

$$\begin{aligned} f_{\text{comp}}(x) &= \begin{bmatrix} a^1 & a^2 & \cdots & a^{n-1} \end{bmatrix} \\ \Delta W &= \begin{bmatrix} P^1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & P^2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & P^{n-1} \end{bmatrix} \end{aligned} \tag{9}$$

where a^i and P^i represent the corresponding values of $x_{i\hat{r}:(i+1)\hat{r}}$ and M post-rotation, respectively. Following RoPE, we use a $\hat{r}\times\hat{r}$ block diagonal matrix to achieve the rotation. However, our method use rotation information to enable M distinguish the $x_{i\hat{r}:(i+1)\hat{r}}$ instead of token position in RoPE. We can define a^i and P^i as follows:

$$\begin{aligned} a^i &= \begin{bmatrix} R_{\theta_1,i} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & R_{\theta_2,i} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & R_{\theta_{\frac{\hat{r}}{2}},i} \end{bmatrix} x_{i\hat{r}:(i+1)\hat{r}} \\ P^i &= M \begin{bmatrix} R_{\theta_1,i} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & R_{\theta_2,i} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & R_{\theta_{\frac{\hat{r}}{2}},i} \end{bmatrix} \end{aligned} \tag{10}$$

where $\theta_j = 10000^{-2(j-1)/\hat{r}}$ and $R_{\theta_j,i} \in \mathbb{R}^{2\times 2}$ is a rotation matrix:

$$R_{\theta_j,i} = \begin{bmatrix} \cos i\theta_j & -\sin i\theta_j \\ \sin i\theta_j & \cos i\theta_j \end{bmatrix} \tag{11}$$

5 Experiment

We evaluate our method on various tasks to understand the influence of high-rank updating. In Section 5.1, we evaluate our method with LoRA and our method on memorizing UUID pairs to show the benefit of high-rank updating on memorizing. In Section 5.2, we reproduce LoRA, LoRA variants and FFT on three fine-tuning tasks: instruction tuning, mathematical reasoning and continual pretraining. In Section 5.3, we compare our method with LoRA and ReLoRA on pretraining by training transformer from scratch.

5.1 Memorizing UUID Pairs

We first compare our method with LoRA and FFT on memorizing UUID pairs to demonstrate improvements through high-rank updating. Following the training settings in Section 3, we search learning rate from $\{5e\text{-}5, 7e\text{-}5, 1e\text{-}4\}$ and use decompress and compress functions in Eq. 8, sharing rows and columns in M. Due to use one matrix M instead of two matrices A and B, we can directly initialize M with zeros. For the predefined groups g and g', we group every adjacent $\hat{r}$ rows or columns together. The training loss is presented in Figure3. Our method shows significant improvements over LoRA with the same number of trainable parameters, benefiting from high-rank updating. We also report character-level accuracy at various training steps in Table 1. MoRA requires fewer training steps to memorize these UUID pairs compared to LoRA. Compared to FFT, MoRA with 256 rank can achieve similar performance and both method can memorize all UUID pairs in 500 steps.

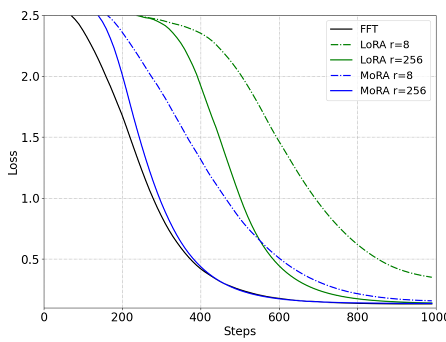


Figure 3: Performance of memorizing UUID pairs with LoRA and our method on rank 8 and 256.

	Rank	300	500	700	900
FFT	-	42.5	100	100	100
LoRA	8	9.9	10.0	10.7	54.2
MoRA	8	10.1	15.7	87.4	100
LoRA	256	9.9	70.6	100	100
MoRA	256	41.6	100	100	100

Table 1: Character-level accuracy of memorizing UUID pairs by generating the value of corresponding key in 300, 500, 700 and 900 training steps.

5.2 Fine-tuning Tasks

5.2.1 Setup

We evaluate our method across three fine-tuning tasks for large language models (LLMs): instruction tuning, mathematical reasoning, and continual pretraining. For these tasks, we select high-quality corresponding datasets to test both LoRA and our method. In instruction tuning, we utilize Tülu v2 Ivison et al. (2023), a blend of several high-quality instruction datasets, containing 326k filtered samples. We assess instruction performance using the MMLU Hendrycks et al. (2020) in both zero-shot and five-shot settings. For mathematical reasoning, we employ the MetaMath Yu et al. (2023) with its 395k samples to enhance mathematical reasoning capabilities and also use GSM8K Cobbe et al. (2021) and MATH Hendrycks et al. (2021) for further evaluation. In continual pretraining, we adapt an LLM to the biomedicine and finance using PubMed abstracts from the Pile Gao et al. (2020) and finicial news, complemented by data preprocessing methods from AdaptLLM Cheng et al. (2023) to boost performance. We report the average performance of corresponding tasks for continual pretraining. More details can be found in Appendix C.

		Instruction Tuning		Mathematical Reasoning		Continual Pretraining	
Method	**Rank**	**MMLU 0**	**MMLU 5**	**GSM8K**	**MATH**	**BioMed.**	**Finance**
FFT	-	50.6	51.3	66.6	20.1	56.4	69.6
LoRA	8	50.2	51.5	**64.6**	15.1	52.3	64.0
LoRA+	8	49.2	51.1	64.1	**15.8**	52.2	64.9
ReLoRA	8	49.3	50.2	61.5	14.5	46.3	61.0
AsyLoRA	8	**50.3**	**52.2**	64.5	15.0	52.5	63.5
DoRA	8	50.2	51.5	64.5	14.6	52.5	63.9
MoRA (Ours)	8	49.7	51.5	64.2	15.4	**53.3**	**67.1**
LoRA	256	49.7	50.8	67.9	**19.9**	54.1	67.3
LoRA+	256	49.2	51.3	**68.2**	17.1	54.2	66.7
ReLoRA	256	-	-	64.0	18.1	52.9	57.9
AsyLoRA	256	**50.1**	**52.0**	66.9	19.3	54.1	66.9
DoRA	256	49.6	51.1	67.4	19.5	54.2	66.0
MoRA (Ours)	256	49.9	51.4	67.9	19.2	**55.4**	**68.7**

Table 2: Performance of FFT, LoRA, LoRA variants and our method on instruction tuning, mathematical reasoning and continual pretraining tasks.

5.2.2 Baselines and Implements

For LoRA-like methods and MoRA, we conducted experiments at $r = 8$ and $r = 256$, and reproduce following methods across three tasks: FFT, LoRA, LoRA+ Hayou et al. (2024), AsyLoRA Zhu et al. (2024), ReLoRA Lialin et al. (2023) and DoRA Liu et al. (2024). LoRA+ enhances the learning rate of matrix B in LoRA to facilitate efficient feature learning based on theoretical analysis. We search the corresponding the hyperparameter λ from $\{2,4\}$. AsyLoRA also analyzes asymmetry in the A and B matrices, and we adopted their initialization strategy. ReLoRA proposes a method to merge low-rank matrices into the model during training to increase the rank of ΔW. we search merge steps from $\{$1k, 2k$\}$ and use 50 steps restarts warmup. DoRA leverages weight decomposition to enhance performance as a robust baseline. For FFT, we follow the settings proposed by corresponding datasets. For MoRA, we employed rotation operators as outlined in Eq. 9 to implement compression and decompression for $r = 8$, and for $r = 256$, we utilized shared rows and columns as specified in Eq. 6 and group every adjacent $\hat{r}$ rows or columns together. The details hyperparameters about fine-tuning can be found in Appendix A.

5.2.3 Results and Analysis

We present the results of fine-tuning tasks in Table 2. We report the results of MMLU with zero-shot and 5-shot settings for instruction tuning, GSM8K and MATH for mathematical reasoning, and average performance on biomedical tasks and financial tasks for continual pretraining.

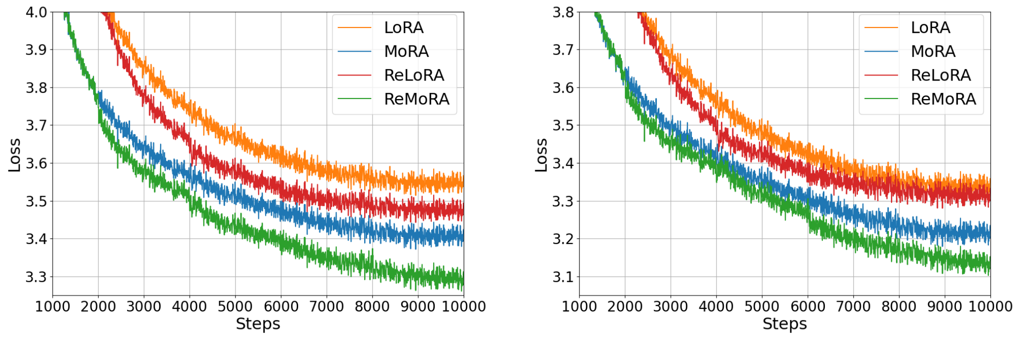
MoRA perform on par with LoRA in instruction tuning and mathematical reasoning. Benefit from high-rank updating to memorize new knowledge, MoRA outperforms LoRA on both biomedical and financial domains for continual pretraining.

We also find that LoRA variants exhibit similar performances on these fine-tuning tasks as compared to LoRA. Although AsyLoRA achieves the best performance in instruction tuning, it demonstrates poor performance in mathematical reasoning. For ReLoRA, merging low-rank matrices during training can harm performance, particularly at the the high rank like 256.

Consider the difference between three tasks, they show different requirements for fine-tuning capabilities. For instruction tuning, which does not learn new knowledge from fine-tuning, rank 8 is enough to achieve performance similar to FFT. For mathematical reasoning, rank 8 is unable to match FFT performance. However, increasing the rank from 8 to 256 can eliminate the performance gap. For continual pretraining, LoRA with rank 256 still underperforms FFT.

5.3 Pretraining

To understand the influence of high-rank updating, we train transformer from scratch on the C4 datasets Raffel et al. (2020). For the model architeture, we use LLaMA-based model with RM-



(a) Pretraining loss at 250M models.

(b) Pretraining loss at 1.3B models.

Figure 4: Pretraining loss with LoRA and MoRA on 250M and 1B models from scratch. Both LoRA and MoRA use same amount of trainable parameters with $r = 128$. ReMoRA and ReLoRA refer to merge MoRA or LoRA back to the model during training to increase the rank of ΔW.

SNorm Zhang & Sennrich (2019), SwiGLU Shazeer (2020) and RoPE Su et al. (2024), testing two sizes: 250M and 1.3B parameters. For the hyperparameters, we use 10k steps, 1024 batch size, 512 sequence length and applying rank $r = 128$ for LoRA and our methods follow Lialin et al.. We also leave modules layernorm or embeddings, which do not apply LoRA, unfrozen. To better show the difference between high-rank and low-rank updating, we reproduce LoRA and other methods without full-rank training warmup. For MoRA, we use Eq. 6 as compression and decompression functions by sharing columns and rows.

We also combine merge-and-reint in ReLoRA with our method called ReMoRA by merging M back into the original parameters during training to increase the rank of ΔW. However, if we directly merge M with g and g' in Eq. 6, the final rank of ΔW is unchanged due to the same expand pattern. To solve this problem, we can change g and g' after merging to ensure the rank of ΔW increasing. More details about ReMoRA can be found in Appendix B. For the hyperparameters corresponding to ReLoRA and ReMoRA, we merge every 2k steps and use 50 steps restarts warmup with optimizer reseting and jagged scheduler.

We show pretraining loss in Figure 4 and corresponding perplexity on C4 validation dataset in Table 3. Our method show better performance on pretraining compared to LoRA and ReLoRA with same amount of trainable parameters. Benefiting from high-rank updating, ReMoRA also achieves more improvements on MoRA compared to ReLoRA, which demonstrates the effectiveness of merge-and-reint strategy in ReMoRA.

	250M	1.3B
LoRA	33.40	28.56
MoRA (Ours)	28.54	25.25
ReLoRA	32.19	27.80
ReMoRA (Ours)	26.74	23.34

Table 3: Perplexity on C4 validation dataset.

6 Analysis

6.1 Influence of Decompression and Compression

To explore the impact of decompression and compression functions in MoRA, we report the performance on GSM8K using various methods: truncation, sharing, decoupling, and rotation in Table 4. Among these methods, truncation shows the worst performance due to the significant information loss during compression. Sharing can achieve better performance than truncation by leveraging the shared rows or columns to preserve the input information. But in the case of $r = 8$, sharing shows worse performance than decouple and rotation due to the large number of sharing rows or columns, as we discussed in Section 4. Rotation is more efficient than decouple, due to the rotation information can help the square matrix to distinguish the input information.

	f_{comp}, f_{decomp}	$r = 8$	$r = 256$
Truncation	Eq. 4	59.5	66.6
Sharing	Eq. 6	62.5	67.9
Decouple	Eq. 8	63.6	67.8
Rotation	Eq. 9	64.2	67.9

Table 4: Influence of decompression and compression functions on $r = 8$ and $r = 256$ on GSM8K.

6.2 High-rank Updating

To demonstrate the impact of high-rank updating on the rank of ΔW, we analyzed the spectrum of singular values for the learned ΔW on 250M pretraining 250M model. We present the average count of singular values exceeding 0.1 across all layers for ΔW_q, ΔW_k, ΔW_v, ΔW_o, ΔW_{up}, ΔW_{down}, and ΔW_{gate} in Figure 5 following Lialin et al. (2023). MoRA and ReMoRA exhibit a substantially higher number of significant singular values compared to LoRA and ReLoRA, highlighting the effectiveness of our methods in increasing the rank of ΔW. We find the quantity of singular values shown in Figure 5 can be correlated with the perplexity metrics listed in Table 3. Moreover, MoRA, without merge-and-reint strategy in ReLoRA and ReMoRA, can achieve a lower perplexity than ReLoRA along with a higher significant singular values.

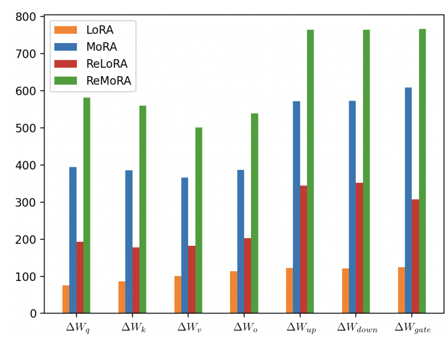


Figure 5: The number of singular values >0.1 in ΔW on the 250M pretraining model.

6.3 Training Speed and Memory Usage

Regarding training time and GPU memory usage, we benchmark LoRA and MoRA on the same hardware. For training settings, we run these methods on a single GPU with a sequence length of 1024, applying LoRA and MoRA to all linear layers of the 7B parameter model. The results are reported in Table 5. For $r = 256$, MoRA uses almost the same time and memory as LoRA, benefiting from the non-parameterized operators. Interestingly, we find that MoRA uses even less GPU memory than LoRA. However, for $r = 8$, MoRA employs Eq. 9 to compress and decompress input features, making it approximately 1.15 times slower than LoRA during fine-tuning.

	Training Speed	**Memory Usage**
	$r = 8$	
LoRA	1.92	16.0GB
MoRA	1.67	16.0GB
	$r = 256$	
LoRA	1.56	31.9GB
MoRA	1.54	31.8GB

Table 5: Comparison of training speed (steps/second) and memory usage with LoRA and MoRA with rank 8 and 256.

7 Conclusion

In this paper, we analyze the impact of low-rank updating through LoRA and observe it struggles with memory-intensive tasks, which also limits the performance of current LoRA variants. To overcome this limitation, we introduce MoRA, a method that utilizes non-parameterized operators for high-rank updating. Within the MoRA framework, we explore various methods to implement decompression and compression functions. Performance comparisons indicate that MoRA matches LoRA in instruction tuning and mathematical reasoning, and exhibits superior performance in continual pretraining and memory tasks. Additionally, we conduct pretraining experiments to further demonstrate the effectiveness of high-rank updating and show superior results compared to ReLoRA.

REFERENCES

Wei Chen, Qiushi Wang, Zefei Long, Xianyin Zhang, Zhongtian Lu, Bingxuan Li, Siyuan Wang, Jiarong Xu, Xiang Bai, Xuanjing Huang, et al. Disc-finllm: A chinese financial large language model based on multiple experts fine-tuning. *arXiv preprint arXiv:2310.15205*, 2023.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. *arXiv preprint arXiv:2210.03849*, 2022.

Daixuan Cheng, Shaohan Huang, and Furu Wei. Adapting large language models via reading comprehension. *arXiv preprint arXiv:2309.09530*, 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.

Katherine M Collins, Albert Q Jiang, Simon Frieder, Lionel Wong, Miri Zilka, Umang Bhatt, Thomas Lukasiewicz, Yuhuai Wu, Joshua B Tenenbaum, William Hart, et al. Evaluating language models for mathematics through interactions. *arXiv preprint arXiv:2306.01694*, 2023.

Franck Dernoncourt and Ji Young Lee. Pubmed 200k rct: a dataset for sequential sentence classification in medical abstracts. *arXiv preprint arXiv:1710.06071*, 2017.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. Specializing smaller language models towards multi-step reasoning. In *International Conference on Machine Learning*, pp. 10421–10430. PMLR, 2023.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Tianyu Han, Lisa C Adams, Jens-Michalis Papaioannou, Paul Grundmann, Tom Oberhauser, Alexander Löser, Daniel Truhn, and Keno K Bressem. Medalpaca–an open-source collection of medical conversational ai models and training data. *arXiv preprint arXiv:2304.08247*, 2023.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient Low Rank Adaptation of Large Models. 3, 2024. URL `http://arxiv.org/abs/2402.12354`.

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.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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*, pp. 2790–2799. PMLR, 2019.

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.

Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A Smith, Iz Beltagy, et al. Camels in a changing climate: Enhancing lm adaptation with tulu 2. *arXiv preprint arXiv:2311.10702*, 2023.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2567–2577, 2019.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Stack more layers differently: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. Www'18 open challenge: financial opinion mining and question answering. In *Companion proceedings of the the web conference 2018*, pp. 1941–1942, 2018.

Pekka Malo, Ankur Sinha, Pekka Korhonen, Jyrki Wallenius, and Pyry Takala. Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65(4):782–796, 2014.

Xiangdi Meng, Damai Dai, Weiyao Luo, Zhe Yang, Shaoxiang Wu, Xiaochen Wang, Peiyi Wang, Qingxiu Dong, Liang Chen, and Zhifang Sui. Periodiclora: Breaking the low-rank bottleneck in lora optimization. *arXiv preprint arXiv:2402.16141*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-lora: Enhacing parameter efficiency of lora with weight tying. *arXiv preprint arXiv:2311.09578*, 2023.

Julio Cesar Salinas Alvarado, Karin Verspoor, and Timothy Baldwin. Domain adaption of named entity recognition to support credit risk assessment. In Ben Hachey and Kellie Webster (eds.), *Proceedings of the Australasian Language Technology Association Workshop 2015*, pp. 84–90, Parramatta, Australia, December 2015. URL `https://aclanthology.org/U15-1010`.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Reslora: Identity residual mapping in low-rank adaption. *arXiv preprint arXiv:2402.18039*, 2024.

Ankur Sinha and Tanmay Khandait. Impact of news on the commodity market: Dataset and results. In *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 2*, pp. 589–601. Springer, 2021.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36, 2024.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez de Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. *arXiv preprint arXiv:2402.16842*, 2024.

A HYPERPARAMETERS

We report the hyperparameters in Table 6. The hyperparameters for Tülu v2 and MetaMath are following their papers Yu et al. (2023); Ivison et al. (2023). Additionally, we search for the optimal learning rate for LoRA across different tasks and report the best performance. We are able to reproduce the results in Yu et al. (2023); Ivison et al. (2023) with LoRA and even achieve better performance. For the hyperparameters of MoRA, we remove the α parameter from LoRA and use the same hyperparameters, except for the learning rate. The learning rate selection for MoRA with Eq. 6 may differ from that of LoRA. Due to the shared rows and columns in Eq. 6, MoRA exhibits a larger gradient norm, so we employ a smaller learning rate.

Dataset	Method	r	α	LR	LR Scheduler	Warmup	Epochs	Batch size	f_{comp}, f_{decomp}
Tülu v2	FFT	-	-	2e-5	cosine	500	2	128	-
	LoRA-like	8	16	{1e-4,2e-4,3e-4}	cosine	500	2	128	-
	MoRA	8	-	{1e-4,2e-4,3e-4}	cosine	500	2	128	Eq. 9
	LoRA-like	256	128	{1e-4,2e-4,3e-4}	cosine	500	2	128	-
	MoRA	256	-	{3e-5,5e-5,7e-5}	cosine	500	2	128	Eq. 6
MetaMath	FFT	-	-	2e-5	cosine	300	3	128	-
	LoRA-like	8	16	{1e-4,2e-4,3e-4}	cosine	300	3	128	-
	MoRA	8	-	{1e-4,2e-4,3e-4}	cosine	300	3	128	Eq. 9
	LoRA-like	256	128	{1e-4,2e-4,3e-4}	cosine	300	3	128	-
	MoRA	256	-	{3e-5,5e-5,7e-5}	cosine	300	3	128	Eq. 6
BioMed./Fiance	FFT	-	-	3e-5	linear	150	1	128	-
	LoRA-like	8	16	{3e-4,4e-4,5e-4}	linear	150	1	128	-
	MoRA	8	-	{3e-4,4e-4,5e-4}	linear	150	1	128	Eq. 9
	LoRA-like	256	128	{3e-4,4e-4,5e-4}	linear	150	1	128	-
	MoRA	256	-	{5e-5,7e-5,1e-4}	linear	150	1	128	Eq. 6

Table 6: Hyperparameters for fine-tuning on three datasets.

B IMPLEMENTATION OF REMORA

We introduce detial implementation of ReMoRA in pretraining. In this case, we simply define two kinds of g. The first kind is grouping every adjacent $\hat{r}$ rows or columns together following the defined in fine-tuning, the first groups can be represented as $\{1, 2, \ldots, \hat{r}\}$. The second kind is grouping every neighboring k of the rows or columns together, the first groups can be represented as $\{1, 1+k, \ldots, 1+\hat{r}k\}$. We propose a example code about compression and decompression functions in Algorithm **1** and **2**. After merging, we can change the group type from 0 to 1 or 1 to 0.

Algorithm 1 Compression

function COMPRESS(x, $\hat{r}$, $type$)
    # $x \in \mathbb{R}^{bsz \times l \times k}$: Input tensor
    # $y \in \mathbb{R}^{bsz \times l \times \hat{r}}$: Output tensor
    # $type \in \{0, 1\}$: Group type 0 or 1
    padding x to make k divisible by $\hat{r}$
    if $type = 0$ **then**
        y = x.view($bsz, l, k/\hat{r}, \hat{r}$).sum(dim=2) # first type of group
    else
        y = x.view($bsz, l, \hat{r}, k/\hat{r}$).sum(dim=3) # second type of group
    end if
    return y
end function

C DOWNSTREAM TASKS OF CONTINUAL PRETRAINING

For biomedcine, we use PubMedQA Jin et al. (2019), RCT Dernoncourt & Lee (2017), USMLE Jin et al. (2021), and selecting biomedicine subjects from MMLU to evaluate the performance. For finance, following BloombergGPT Wu et al. (2023),we use ConvFinQA Chen et al. (2022), NER Salinas Alvarado et al. (2015), Headline Sinha & Khandait (2021), FiQA SA Maia et al. (2018) and FPB Malo et al. (2014). We report the detail performance of these tasks following:

Algorithm 2 Decompression

1: **function** DECOMPRESS(x, $\hat{r}$, $type$)
2: # $x \in \mathbb{R}^{bsz \times l \times \hat{r}}$: Input tensor
3: # $y \in \mathbb{R}^{bsz \times l \times d}$: Output tensor
4: # $type \in \{0, 1\}$: Group type 0 or 1
5: **if** $type = 0$ **then**
6: y = repeat(x, $d/\hat{r}$, dim=2) # first type of group
7: **else**
8: y = repeat-interleave(x, $d/\hat{r}$, dim=2) # second type of group
9: **end if**
10: truncate y to $\mathbb{R}^{bsz \times l \times d}$
11: **return** y
12: **end function**

	r	**PubMedQA**	**USMLE**	**BioMMLU**	**RCT**	**Avg.**
FFT	-	74.1	41.2	47.5	62.7	56.4
LoRA	8	73.1	34.9	45.3	54.9	51.9
MoRA	8	73.3	34.7	45.3	59.9	53.3
LoRA	256	73.8	39.7	46.0	56.9	54.1
MoRA	256	74.4	40.4	46.1	60.6	55.4

Table 7: Performance on biomedical tasks.

	r	**ConvFinQA**	**FiQA SA**	**Headline**	**NER**	**FPB**	**Avg.**
FFT	-	44.4	78.8	82.3	68.1	74.3	69.6
LoRA	8	44.5	76.2	72.4	61.6	65.1	64.0
MoRA	8	45.8	76.6	76.3	68.9	68.2	67.1
LoRA	256	41.4	78.3	83.0	66.8	66.7	67.3
MoRA	256	47.7	76.3	83.4	68.0	68.1	68.7

Table 8: Performance on finicial tasks.