LoRAM: Low-Rank Adaptation of Large Language Models on Manifold

Xiaowen Jiang, Xun Wang & Sebastian U. Stich
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
{xiaowen.jiang,xun.wang,stich}@cispa.de

Abstract

Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning (PEFT) method, has gained remarkable popularity in recent years. By freezing pretrained weights and injecting the product of two trainable low-rank matrices into certain layers of the model, LoRA significantly reduces the number of trainable parameters while introducing no additional inference latency. From an optimization perspective, the original domain consists of bounded-rank matrices, which LoRA parametrizes using the standard LR factorization. However, this parametrization has unfavorable theoretical properties, including a highly non-smooth optimization landscape and the absence of fast local convergence guarantees. In this work, we explore two alternative techniques with stronger theoretical properties for fine-tuning large models: (i) direct optimization over the set of fixed-rank matrices and (ii) optimization over bounded-rank matrices using a smooth parameterization via desingularization. Both approaches leverage well-established Riemannian manifold geometry, and we employ Riemannian Adam with coordinate-wise stepsize as the optimization algorithm. The resulting methods have comparable memory and computation complexity to LoRA optimized with Adam. We show superior performances of them on fine-tuning LLaMA for commonsense reasoning tasks.

1 Introduction

Due to the exceptional generalization capabilities of large-scale pretrained models, fine-tuning has become an essential technique for adaptation to a wide range of downstream tasks. Traditional full-parameter fine-tuning methods can be computationlly expensive and memory inefficient for certain tasks especially when the number of downstream tasks increase. To tackle this challenge, parameter-efficient fine-tuning (PEFT) methods have gained significant attention (Houlsby et al., 2019).

Among them, Low-Rank Adaptation (LoRA) (Hu et al., 2022) has emerged as one of the most popular approaches due to its simplicity and ability to reduce the number of trainable parameters while maintaining inference efficiency. By freezing the pretrained weights and injecting a trainable bounded-rank matrix into selected layers, LoRA effectively adapts large models with minimal additional overhead. Specifically, LoRA parametrizes the set of bounded-rank matrices via the standard LR factorization, which is the product of two trainable low-rank matrices.

While this parametrization is easy to implement, it has several unfavorable theoretical properties. Specifically, the optimization landscape induced by LR parametrization can be highly non-smooth, makeing optimization challenging and potentially hindering convergence. Furthermore, the absence of a local Polyak-Łojasiewicz (PŁ) condition may prevent fast local convergence guarantees (Rebjock & Boumal, 2024). On the experimental side, a capacity gap may still exist between LoRA and more advanced methods (Zhuo et al., 2024; Hu et al., 2023; Liu et al., 2024).

In this work, we explore two alternative approaches with stronger theoretical properties for optimizing over the set of bounded-rank matrices and apply them to fine-tuning large models. The first approach is to directly optimize over the set of fixed-rank matrices, which forms a smooth submanifold of a linear space (Vandereycken, 2013). The second approach optimizes over an alternative smooth parameterization of bounded-rank matrices using a desingularization (Khrulkov &

Oseledets, 2018). Rebjock & Boumal (2024) have shown that this parameterized set is a smooth manifold and has endowed it with a family of Riemannian structures.

Contributions. Leveraging well-established Riemannian optimization techniques (Absil et al., 2008; Boumal, 2023; Boumal et al., 2014), we employ the Riemannian Adam optimizer with coordinate-wise step sizes to fine-tune large neural networks on these two smooth manifolds. The resulting methods maintain comparable memory and computational complexity to LoRA optimized with Adam. We validate their effectiveness through fine-tuning experiments on LLaMA3.1-8B (Touvron et al., 2023) for the commonsense reasoning tasks, We demonstrate their superior performance compared to LoRA in terms of generalization accuracy and convergence speed. In particular, adding nuclear norm regularization to the second approach further enhances its performance. Our findings highlight the potential of Riemannian-based optimization techniques for improving PEFT methods.

Related work. We discuss the closely related work in the next section and defer the reader to the Appendix for additional references.

Problem formulation and background. In this work, we consider adapter-based fine-tuning methods by introducing additional trainable modules into the original frozen parameters. Following Hu et al. (2022), we inject a trainable matrix $\mathbf{W} \in \mathbb{R}^{m\times n}_{\leq r}$ (with rank at most r) into a pre-trained weight $\mathbf{W}_0$, so that after fine-tuning, the updated weight becomes $\mathbf{W}_0 + \mathbf{W}$. Compared to directly training $\mathbf{W}$, this method can significantly improve the memory efficiency if r is sufficiently small. Suppose we add q such matrices, then the fine-tuning task is equivalent to solving a constrained optimization problem of the form:

$$\min_{(\mathbf{W}_i)_{i=1}^q} f(\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_q) \quad \text{subject to} \quad \mathbf{W}_i \in \mathbb{R}^{m_i\times n_i}_{\leq r_i},\ \forall i \in [n]\ , \tag{1}$$

for some $r_i \leq \min\{m_i, n_i\}$, where f is the expected loss function for the downstreaming task.

However, the feasible set of problem (1) is a non-convex non-smooth algebraic variety, which can be computationally hard in general (Gillis & Glineur, 2011). There exist several ways to tackle this challenge. The most prevalent one is by using LR parametrization: $\varphi(\mathbf{L}_1, \mathbf{R}_1, ..., \mathbf{L}_q, \mathbf{R}_q) = (\mathbf{L}_i\mathbf{R}_i^T, ..., \mathbf{L}_q\mathbf{R}_q^T)$ where $\mathbf{L}_i \in \mathbb{R}^{m_i\times r_i}$ and $\mathbf{R}_i \in \mathbb{R}^{n_i\times r_i}$. Then solving problem (1) is equivalent to minimizing $g = f \circ \varphi$ over $(\mathbf{L}_i, \mathbf{R}_i)_{i=1}^n$, which is the idea behind the celebrated method LoRA. However, this parametrization has several unfavorable theoretical properties. For instance, the function g can become highly non-smooth even if f has Lipschitz gradient. This can make optimization challenging and hinder convergence. Moreover, lacking local Polyak–Łojasiewicz (PŁ) condition of LR parametrization can also prevent fast local convergence. More discussions and details can be found in (Rebjock & Boumal, 2024) and Section 2.3 from (Khrulkov & Oseledets, 2018).

Apart from LR parametrization, there exist other techniques with stronger theoretical properties to optimize problem (1). However, how to efficiently apply them to large-model fine-tuning tasks and how their practical performance compares to LoRA remain unclear.

Notations: $\mathrm{St}(n, r) := \{\mathbf{X} \in \mathbb{R}^{n\times r} : \mathbf{X}^T\mathbf{X} = \mathbf{I}_r\}$, $\mathrm{diag}(r) = \{\text{diagonal matrices with size } r \times r\}$.

2 Low-Rank Adaptation on Manifold

In this section, we describe two methods for fine-tuning large neural networks on manifolds with low-rank structures. Both of them are based on well-established Riemannian geometry.

Fixed-rank matrices as embedded geometry (Vandereycken, 2013). Instead of considering the set of bounded-rank matrices, the classical approach is to optimize over the set of fixed-rank matrices: $\mathbf{W}_i \in \mathbf{R}^{m_i\times n_i}_{r_i}$, for any $i \in [q]$. The whole search space a is a well-known smooth manifold that can be endowed with a well-defined Riemannian structure (Boumal, 2023). Consequently, we can apply the stochastic Riemannian gradient methods that generate a sequence of points constrained to the manifold. Empirically, we found that equipping Riemannian Adam with a coordinate-wise step size often outperforms the single step size approach described in (Bécigneul & Ganea, 2018), which, in turn, performs better than Riemannian SGD. The full descriptions of the algorithm we implement can be found in Algorithm 1. We next discuss its memory and computation cost.

In what follows, we assume $q = 1$ for simplicity. For any $\mathbf{W} \in \mathbb{R}^{m\times n}_r$, we can represent and store it by $\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ where $\mathbf{U} \in \mathrm{St}(m, r)$, $\mathbf{V} \in \mathrm{St}(n, r)$ and $\mathbf{S} \in \mathrm{diag}(r)$. This requires $mr + nr + r^2$

memory storage. The main step is to compute the Riemannian gradient of $\nabla f(\mathbf{W})$, which is represented by a tuple $(\mathbf{U}_p, \mathbf{M}, \mathbf{V}_p)$ such that $\mathbf{U}_p = \nabla f(\mathbf{W})\mathbf{V} - \mathbf{U}\mathbf{M}$, $\mathbf{M} = \mathbf{U}^T\nabla f(\mathbf{W})\mathbf{V}$ and $\mathbf{V}_p = \nabla f(\mathbf{W})^T\mathbf{U} - \mathbf{V}\mathbf{M}^T$. Note that $\nabla f(\mathbf{W})$ is not required to be computed explicitly. Instead, it is sufficient to obtain $\nabla f(\mathbf{W})\mathbf{V}$ and $\nabla f(\mathbf{W})^T\mathbf{U}$ via automatic differentiation (Novikov et al., 2022). Specifically, the matrices $\mathbf{U}, \mathbf{S}, \mathbf{V}$ do not require gradient computations and we create trainable matrices $\mathbf{A} = \mathbf{U}\mathbf{S} \in \mathbb{R}^{m\times r}$ and $\mathbf{B} = \mathbf{0}^{n\times r}$ and compute $f(\mathbf{W}) = f(\mathbf{A}\mathbf{V}^T + \mathbf{U}\mathbf{B}^T)$ during the forward pass. Then after the backward pass, the gradients of $\mathbf{A}$ and $\mathbf{B}$ are equivalent to $\nabla f(\mathbf{W})\mathbf{V}$ and $\nabla f(\mathbf{W})^T\mathbf{U}$ respectively, which are used to compute $(\mathbf{U}_p, \mathbf{M}, \mathbf{V}_p)$ and perform the optimization step. The procedure is described in Algorithm 2. Next, We compute the momentum which requires the storage of the past tuple $(\mathbf{U}_p, \mathbf{M}, \mathbf{V}_p)$ and its element-wise product $(\mathbf{U}_p \odot \mathbf{U}_p, \mathbf{M} \odot \mathbf{M}, \mathbf{V}_p \odot \mathbf{V}_p)$. The next point is then obtained by performing a retraction step (Algorithm 3) along the direction of the negative rescaled momentum. Finally, we transport the current momentum vector to the tangent space of the new point (Algorithm 4).

In total, the method requires $2mr + 2nr + r^2$ memory for storing the parameters (which can be reduced to $mr + nr + r^2$ if the memory for $\mathbf{A}$ and $\mathbf{B}$ is freed at each iteration). Additionally, storing the optimizer states requires $2mr + 2nr + 2r^2$ memory, The whole storage is nearly the same as the requirement for LoRA. For each optimization step, the product of two matrices of size $\mathbf{R}^{m\times r}$ and $\mathbf{R}^{n\times r}$ is computed for several times, which scales up to mnr. Additionally, in the retraction operator, we need to compute QR factorizations of two thin matrices, and the SVD for a small matrix of size $\mathbf{R}^{2r\times 2r}$, which has time complexity $(m+n)r^2$.

Desingularization of bounded-rank matrices (Khrulkov & Oseledets, 2018; Rebjock & Boumal, 2024). Khrulkov & Oseledets (2018) proposed another smooth parametrization of the set of bounded-rank matrices, called *desingularization*, which is given by:

$$\mathcal{M} = \{(\mathbf{W}, \mathbf{P}) \in \mathbb{R}^{m\times n} \times \mathrm{Gr}(n, n-r) : \mathbf{W}\mathbf{P} = 0\}\,, \tag{2}$$

where Gr is the standard Grassmann manifold (Bendokat et al., 2024) and $\mathbf{W}$ has rank at most r. The original problem 1 is equivalent to solving $g = f \circ \varphi$ with $\varphi(\mathbf{W}, \mathbf{P}) = \mathbf{W}$. Compared with LR parametrization, the manifold $\mathcal{M}$ (2) has several favorable theoretical properties such as having smooth and bounded fibers, which allows to obtain global and local convergence guarantee. We use the Riemannian tools provided in (Rebjock & Boumal, 2024) and consider again the Riemannian Adam with coordinate-wise stepsize. The full algorithm is described in Alogirithm 5.

For any $(\mathbf{W}, \mathbf{P}) \in \mathcal{M}$, we can represent it by $\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ and $\mathbf{P} = \mathbf{I} - \mathbf{V}\mathbf{V}^T$ where $\mathbf{U} \in \mathrm{St}(m, r), \mathbf{V} \in \mathrm{St}(n, r)$ and $\mathbf{S} \in \mathrm{diag}(r)$. Therefore, the memory storage of $\mathbf{U}, \mathbf{S}, \mathbf{V}$ is the same as fixed rank matrices. The Riemannian gradient can be represented as a tuple $(\mathbf{K}, \mathbf{V}_p)$ where $\mathbf{K} = \nabla f(\mathbf{W})\mathbf{V} \in \mathbb{R}^{m\times r}$ and $\mathbf{V}_p = (\mathbf{I} - \mathbf{V}\mathbf{V}^T)\nabla f(\mathbf{W})^T\mathbf{U}\mathbf{S}\mathbf{S}(\alpha)^{-1} \in \mathbb{R}^{n\times r}$ with $\mathbf{S}(\alpha) = 2\alpha\mathbf{I} + \mathbf{S}^2$ and α is usually set to be 0.5. We can use the same trick as before to compute $\nabla f(\mathbf{W})\mathbf{V}$ and $\nabla f(\mathbf{W})^T\mathbf{U}$ via automatic differentiation (See Algorithm 6). To compute the momentum update, we need to store the previous tuples $(\mathbf{K}, \mathbf{V}_p)$ and $(\mathbf{K} \odot \mathbf{K}, \mathbf{V}_p \odot \mathbf{V}_p)$. In the retraction operator (Algorithm 7), a thin QR decomposition of a matrix of size $\mathbf{R}^{n\times r}$ and a thin SVD for a $\mathbb{R}^{m\times r}$ matrix are needed, which has the time complexity of order $(m+n)r^2$. In total, the memory and computation complexity of Algorithm 5 are at the same scale of Algorithm 1.

Nuclear norm regularization. If needed, a nuclear norm penalty can be explicitly added to the loss function to encourage a solution with small singular values (Bach, 2008). Consider the loss function $g(\mathbf{W}) = f(\mathbf{W}) + \lambda \left\|\mathbf{W}\right\|_*$ where $\lambda > 0$ and $\left\|\mathbf{W}\right\|_* := \mathrm{trace}(\mathbf{S})$ where $\mathbf{U}, \mathbf{S}, \mathbf{V}$ are the singular value decomposition of $\mathbf{W}$. We can use $-\lambda\mathbf{U}\mathbf{V}^T$ as the negative subgradient of the function $\lambda \left\|\mathbf{W}\right\|_*$ and add this direction separately to the retraction operator similar to decoupled weight decay for AdamW (Loshchilov & Hutter, 2019). This is described on line 4 of Algorithm 7.

Non-LoRAM parameters. For all other parameters that are not low-rank and require full fine-tuning, such as bias vectors or the final layers of the classifier, we optimize them separately using the standard AdamW optimizer.

Memory storage of the fine-tuned model. After fine-tuning, for both parametrizations, we can free the memory of $\mathbf{A}$ and $\mathbf{B}$ and merge $\mathbf{U}\mathbf{S}$ into a matrix $\mathbf{U}'$. Thus we are only required to store $\mathbf{U}'$ and $\mathbf{V}^T$ for each task. During the forward pass, we use $\mathbf{W}^{\text{new}} = \mathbf{W}_0 + \mathbf{U}'\mathbf{V}^T$ as the updated weight. Therefore, the memory storage is the same as LoRA.

For the second desingularization approach, since it may converge to a solution with rank less than r, we can first identify and remove singular vectors associated with singular values below a predefined threshold before merging, further reducing the memory cost.

Diagonal Scaling. Similar to DORA (Liu et al., 2024), one can enhance the expressivity of the solution by introducing scaling matrices. For example, the updated weight can be parameterized as $\mathbf{W}^{\text{new}} = \mathbf{D}_1\mathbf{W}_0\mathbf{D}_2 + \mathbf{W}$ where $\mathbf{D}_1 \in \text{diag}(m)$ and $\mathbf{D}_2 \in \text{diag}(n)$ are additional two trainable parameters. Since letting $\mathbf{D}_1 = \mathbf{I}_m$ and $\mathbf{D}_2 = \mathbf{I}_n$ recover the previous parametrization, the new solution should in principle be better. This simple approach can be independently incorporated into LoRA and LoRAM. We leave this exploration in the future work.

3 Experiments

In this section, we evaluate LoRAM against LoRA by fine-tuning LLaMA3.1-8B (Touvron et al., 2023) on commonsense reasoning tasks. We conduct the experiments using four A100-40G GPUs.

The commonsense reasoning benchmark consists of eight sub-tasks, each with predefined training and testing sets. Following the setup of Hu et al. (2023); Liu et al. (2024), we combine the training datasets from all eight tasks to form a unified training dataset and evaluate performance on the individual test set. We inject trainable parameters into three attention matrices as well as up and down projection matrices. In addition, we make several changes to create a better training setup.

We first increase the validation set (part of the training dataset) from 160 to 24000 to ensure a more robust solution. We then choose the best model based on the evaluation accuracy instead of the evaluation loss, as the former is more stable than the latter for this task. Finally, we only compute the cross entropy loss for the answer tokens.

In what follows, we use LoRAM-FR to denote Algorithm 2 running on the manifold of fixed-rank matrices and LoRAM-BR to refer to Algorithm 6 for optimizing the desingularization of bounded-rank matrices. We use the minibatch size of 16 and context length of 256. We run 3 epochs of all the methods and evaluate the performance every 640 iterations. The learning rate of all the methods is linearly decreased to 0. The result can be found in Table 1.

Methods	rank	lr	ARC-c	ARC-e	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	OBQA	Average
LoRA-previous [a]	32	3×10^{-4}	71.2	84.2	70.8	85.2	79.9	91.7	84.3	87.4	80.8
LoRA	16	10^{-4}	83.6	93.5	74.8	89.1	80.7	94.3	88.6	87.4	86.5
LoRAM-FR	16	5×10^{-5}	83.0	92.2	74.5	**91.0**	**83.1**	95.4	89.2	88.2	87.1
LoRAM-BR [b]	16	10^{-4}	84.2	**93.8**	**76.4**	90.0	81.2	**95.7**	**89.7**	87.8	87.3
LoRA	32	10^{-4}	84.7	93.1	75.5	89.9	81.7	95.5	89.1	**88.4**	87.2
LoRAM-FR	32	3×10^{-5}	84.6	93.5	76.0	90.5	81.7	95.5	**89.7**	**88.4**	87.5
LoRAM-BR [b]	32	10^{-4}	**85.3**	93.4	76.2	**91.0**	81.8	95.5	**89.7**	87.8	**87.6**

[a] The performance of LoRA reported in previous papers (Liu et al., 2024; Si et al., 2025) fine-tuned on LLaMA3-8B.
[b] We use $\lambda = 10^{-3}$ for nuclear norm regularization . The performance gets worse if $\lambda = 0$.

Table 1: Performance comparison (test accuracy) of different methods for fine-tuning LLaMA3.1-8B on commonsense reasoning tasks where the last column is the average over the number of tasks.

We see a significant improvement of LoRA compared to the statstics reported in previous papers (line 1), validating the effectiveness of the training setup. Moreover, LoRAM can further improve the performance and test accuracy of both rank options. The convergence comparisons can be found in Figure 1 in the Appendix.

4 Conclusion and Future Work.

In this work, we propose to fine-tune large models on smooth manifolds with low-rank structures. Future work includes extensive empirical studies on various downstream tasks, adding additional diagonal scaling parameters, and using quantization strategies to further improve memory efficiency.

REFERENCES

P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008. ISBN 978-0-691-13298-3.

Francis R Bach. Consistency of trace norm minimization. *The Journal of Machine Learning Research*, 9:1019–1048, 2008.

Gary Bécigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. *arXiv preprint arXiv:1810.00760*, 2018.

Thomas Bendokat, Ralf Zimmermann, and P-A Absil. A grassmann manifold handbook: Basic geometry and computational aspects. *Advances in Computational Mathematics*, 50(1):6, 2024.

Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023. doi: 10.1017/9781009166164. URL https://www.nicolasboumal.net/book.

Nicolas Boumal, Bamdev Mishra, P-A Absil, and Rodolphe Sepulchre. Manopt, a matlab toolbox for optimization on manifolds. *The Journal of Machine Learning Research*, 15(1):1455–1459, 2014.

Nicolas Gillis and François Glineur. Low-rank matrix approximation with weights or missing data is np-hard. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1149–1165, 2011.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

Valentin Khrulkov and Ivan Oseledets. Desingularization of bounded-rank matrix sets. *SIAM Journal on Matrix Analysis and Applications*, 39(1):451–471, 2018.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Cheng Lin, Lujun Li, Dezhi Li, Jie Zou, Wei Xue, and Yike Guo. Nora: Nested low-rank adaptation for efficient fine-tuning large models. *arXiv preprint arXiv:2408.10280*, 2024.

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. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Grigory Malinovsky, Umberto Michieli, Hasan Abed Al Kader Hammoud, Taha Ceritli, Hayder Elesedy, Mete Ozay, and Peter Richtárik. Randomized asymmetric chain of lora: The first meaningful theoretical framework for low-rank adaptation. *arXiv preprint arXiv:2410.08305*, 2024.

Alexander Novikov, Maxim Rakhuba, and Ivan Oseledets. Automatic differentiation for riemannian optimization on low-rank matrix and tensor-train manifolds. *SIAM Journal on Scientific Computing*, 44(2):A843–A869, 2022. doi: 10.1137/20M1356774. URL https://doi.org/10.1137/20M1356774.

Quentin Rebjock and Nicolas Boumal. Optimization over bounded-rank matrices through a desingularization enables joint global and local guarantees. *arXiv preprint arXiv:2406.14211*, 2024.

Chongjie Si, Zhiyi Shi, Shifan Zhang, Xiaokang Yang, Hanspeter Pfister, and Wei Shen. Unleashing the power of task-specific directions in parameter efficient fine-tuning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=RYrJqz44p4.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Bart Vandereycken. Low-rank matrix completion by riemannian optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013. doi: 10.1137/110845768. URL https://doi.org/10.1137/110845768.

Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of lora: Efficient fine-tuning of language models via residual learning. *arXiv preprint arXiv:2401.04151*, 2024.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned lora for fine-tuning foundation models. *arXiv preprint arXiv:2402.02347*, 2024.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

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.

Terry Yue Zhuo, Armel Zebaze, Nitchakarn Suppattarachai, Leandro von Werra, Harm de Vries, Qian Liu, and Niklas Muennighoff. Astraios: Parameter-efficient instruction tuning code large language models. *arXiv preprint arXiv:2401.00788*, 2024.

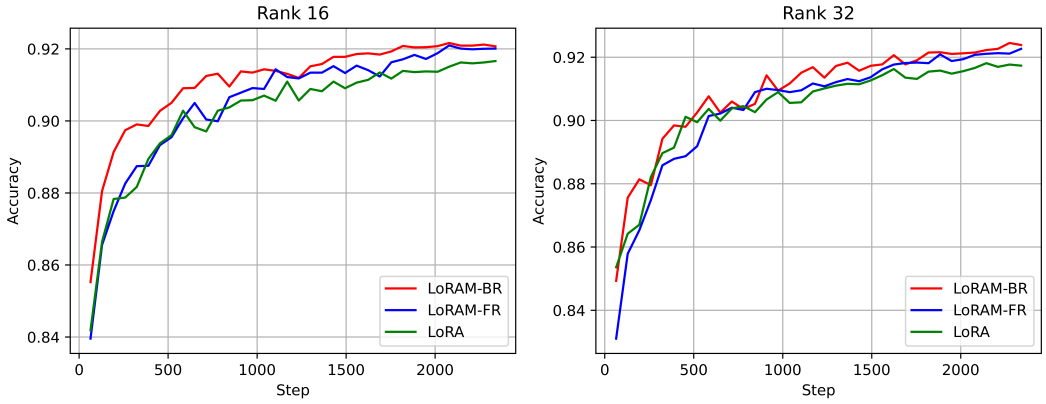

Figure 1: Convergence (validation accuracy) comparisons of LoRAM against LoRA on for fine-tuning LLaMA 3.1-8B on commonsense reasoning tasks.

Algorithm 1 RAdam for fixed-rank matrices applied to fine-tuning task

Input $\{\eta_k\}$, $\beta_1 > 0$, $\beta_2 > 0$, $\varepsilon > 0$
Initialize $\mathbf{U}_i \in \mathrm{St}(m_i, r_i)$, $\mathbf{V}_i \in \mathrm{St}(n_i, r_i)$, $\mathbf{S}_i = \mathbf{0}^{r_i \times r_i}$, $\mathbf{w}_{i,-1} = (\mathbf{0}^{m_i \times r_i}, \mathbf{0}^{r_i \times r_i}, \mathbf{0}^{n_i \times r_i})$, $\mathbf{v}_{i,-1}^{\mathbf{U}_p} = \mathbf{0}^{m_i \times r_i}$, $\mathbf{v}_{i,-1}^{\mathbf{M}} = \mathbf{0}^{r_i \times r_i}$, $\mathbf{v}_{i,-1}^{\mathbf{V}_p} = \mathbf{0}^{n_i \times r_i}$, $\forall i \in [q]$. Let $\mathbf{v}_{i,k} = (\mathbf{v}_{i,k}^{\mathbf{U}_p}, \mathbf{v}_{i,k}^{\mathbf{M}}, \mathbf{v}_{i,k}^{\mathbf{V}_p})$.
for $k = 0, 1, 2, \ldots$ **do**
  for each $i \in [q]$ **do**
    $\mathbf{g}_{i,k} = (\mathbf{U}_p^{i,k}, \mathbf{M}_{i,k}, \mathbf{V}_p^{i,k})$ (By Algorithm 6)
    $\mathbf{m}_{i,k} = \beta_1 \mathbf{w}_{i,k-1} + (1-\beta_1)\mathbf{g}_{i,k}$
    $\mathbf{v}_{i,k}^{\mathbf{U}_p} = \beta_2 \mathbf{v}_{i,k-1}^{\mathbf{U}_p} + (1-\beta_2)\mathbf{U}_p^{i,k} \odot \mathbf{U}_p^{i,k}$
    $\mathbf{v}_{i,k}^{\mathbf{M}} = \beta_2 \mathbf{v}_{i,k-1}^{\mathbf{M}} + (1-\beta_2)\mathbf{M}^{i,k} \odot \mathbf{M}^{i,k}$
    $\mathbf{v}_{i,k}^{\mathbf{V}_p} = \beta_2 \mathbf{v}_{i,k-1}^{\mathbf{V}_p} + (1-\beta_2)\mathbf{V}_p^{i,k} \odot \mathbf{V}_p^{i,k}$
    $(\mathbf{U}_{i,k+1}, \mathbf{S}_{i,k+1}, \mathbf{V}_{i,k+1}) = R_{(\mathbf{U}_{i,k}, \mathbf{S}_{i,k}, \mathbf{V}_{i,k})}\Big(-\eta_k \mathbf{m}_{i,k} / (\sqrt{\mathbf{v}_{i,k}} + \varepsilon)\Big)$ [1]
    (By Algorithm 3)
    $\mathbf{w}_{i,k} = \mathrm{PT}_{(\mathbf{U}_{i,k}, \mathbf{S}_{i,k}, \mathbf{V}_{i,k}) \to (\mathbf{U}_{i,k+1}, \mathbf{S}_{i,k+1}, \mathbf{V}_{i,k+1})}(\mathbf{m}_{i,k})$ (By Algorithm 4)

A Appendix

More Related work. Recent studies have investigated various LoRA variants from multiple perspectives. A line of research focuses on understanding and improving the optimization of LoRA. Hayou et al. (2024) propose adjusting and assigning different learning rates to the two adapter matrices. Zhang & Pilanci (2024) introduces a small $r \times r$ preconditioner to the gradient updates. Xia et al. (2024) suggest periodically merging the low-rank matrices into the frozen pre-trained weights. Zhu et al. (2024); Malinovsky et al. (2024) explore training one component and freezing the other in the product of two matrices to tackle the non-smooth optimization landscape. However, in practice, LoRA is trained with coordinate-wise adaptive stepsize, which is incomparable to the theoretical settings of these works. Another line of research focuses on modifying LoRA to balance expressivity and efficiency. Some works introduce additional trainable parameters to enhance the model's expressivity (Liu et al., 2024; Lin et al., 2024), while others reduce the number of trainable parameters to further optimize memory usage (Kopiczko et al., 2024). Zhang et al. (2023) propose to add regularization to adaptively choose the rank for each layer based on SVD decomposition.

[1] The operation $/$, $+$ and $\sqrt{\cdot}$ is element-wise.
[2] The operation $/$, $+$ and $\sqrt{\cdot}$ is element-wise.

Algorithm 2 Computation of Riemannian gradient for fixed-rank matrices

Input: $(\mathbf{U}_i, \mathbf{S}_i, \mathbf{V}_i)_{i=1}^q$ with $\mathbf{U}_i \in \mathrm{St}(m_i, r_i)$, $\mathbf{V}_i \in \mathrm{St}(n_i, r_i)$ and $\mathbf{S}_i = \mathrm{diag}(r_i)$ (with `requires_grad=False`), $\mathbf{A}_i = \mathbf{U}_i\mathbf{S}_i$ and $\mathbf{B}_i = \prime^{n_i \times r_i}$ (both with `requires_grad=True`)
for $i \in [q]$ in parallel **do**
  Compute $\mathbf{W}_i = \mathbf{A}_i\mathbf{V}_i^T + \mathbf{U}_i\mathbf{B}_i^T$ $(\mathbf{W}_i = \mathbf{U}_i\mathbf{S}_i\mathbf{V}_i^T)$
  Inject $\mathbf{W}_i$ to the corresponding large matrix
Perform `forward` and `backward` pass
Set $\mathbf{M}_i = \mathbf{U}_i^T\mathbf{A}_i$.`grad`, $\mathbf{U}_p^i = \mathbf{A}_i$.`grad`$-\mathbf{U}_i\mathbf{M}_i$ and $\mathbf{V}_p^i = \mathbf{B}_i$.`grad`$-\mathbf{V}_i\mathbf{M}_i^T$
Return $(\mathbf{U}_p^i, \mathbf{M}_i, \mathbf{V}_p^i)_{i=1}^q$

Algorithm 3 Retraction operator for fixed-rank matrices

Input: points $(\mathbf{U}_i, \mathbf{S}_i, \mathbf{V}_i)_{i=1}^q$, vectors $(\mathbf{U}_p^i, \mathbf{M}_i, \mathbf{V}_p^i)_{i=1}^q$.
for $i \in [q]$ in parallel **do**
  Compute thin QR factorizations of the left and right matrices:
  $\mathbf{Q}_U^i\mathbf{R}_U^i = [\mathbf{U}_i\ \mathbf{U}_p^i]$, $\mathbf{Q}_V^i\mathbf{R}_V^i = [\mathbf{V}_i\ \mathbf{V}_p^i]$
  Compute SVD of a small matrix: $\tilde{\mathbf{U}}_i\tilde{\mathbf{S}}_i\tilde{\mathbf{V}}_i^T = \mathbf{R}_U^i \begin{bmatrix} \mathbf{S}_i + \mathbf{M}_i & \mathbf{I}_{r_i} \\ \mathbf{I}_{r_i} & 0 \end{bmatrix} (\mathbf{R}_V^i)^T$
  Update $\mathbf{U}_i^+ = \mathbf{Q}_U^i\tilde{\mathbf{U}}_i[:, :r]$, $\mathbf{S}_i^+ = \tilde{\mathbf{S}}_i[:r, :r]$ and $\mathbf{V}_i^+ = \mathbf{Q}_V^i\tilde{\mathbf{V}}_i[:, :r]$
Return $(\mathbf{U}_i^+, \mathbf{S}_i^+, \mathbf{V}_i^+)_{i=1}^q$

Algorithm 4 Parallel transport for fixed-rank matrices

Signature: $\mathrm{PT}_{(\mathbf{U},\mathbf{S},\mathbf{V})\to(\mathbf{U}^+,\mathbf{S}^+,\mathbf{V}^+)}(\mathbf{U}_p, \mathbf{M}, \mathbf{V}_p)$
Input: points $(\mathbf{U}, \mathbf{S}, \mathbf{V})$, $(\mathbf{U}^+, \mathbf{S}^+, \mathbf{V}^+)$, vector $(\mathbf{U}_p, \mathbf{M}, \mathbf{V}_p)$
$\mathbf{M}^+ = (\mathbf{U}^+)^T(\mathbf{U}\mathbf{M} + \mathbf{U}_p)\mathbf{V}^T\mathbf{V}^+ + (\mathbf{U}^+)^T\mathbf{U}\mathbf{V}_p^T\mathbf{V}^+$
$\mathbf{U}_p^+ = (\mathbf{U}\mathbf{M} + \mathbf{U}_p)\mathbf{V}^T\mathbf{V}^+ + \mathbf{U}\mathbf{V}_p^T\mathbf{V}^+ - \mathbf{U}^+\mathbf{M}^+$
$(\mathbf{V}_p^+)^T = (\mathbf{U}^+)^T(\mathbf{U}\mathbf{M} + \mathbf{U}_p)\mathbf{V}^T + (\mathbf{U}^+)^T\mathbf{U}\mathbf{V}_p^T - \mathbf{M}^+(\mathbf{V}^+)^T$
Return $(\mathbf{U}_p^+, \mathbf{M}^+, \mathbf{V}_p^+)$

Algorithm 5 RAdamW for bounded-rank matrices applied to fine-tuning task

Input $\{\eta_k\}$, $\lambda \geq 0$, $\beta_1 > 0$, $\beta_2 > 0$, $\varepsilon > 0$
Initialize $\mathbf{U}_i \in \mathrm{St}(m_i, r_i)$, $\mathbf{V}_i \in \mathrm{St}(n_i, r_i)$, $\mathbf{S}_i = \mathbf{0}^{r_i \times r_i}$, $\mathbf{w}_{i,-1} = (\mathbf{0}^{m_i \times r_i}, \mathbf{0}^{n_i \times r_i})$, $\mathbf{v}_{i,-1}^{\mathbf{K}} = \mathbf{0}^{m_i \times r_i}$, $\mathbf{v}_{i,-1}^{\mathbf{V}_p} = \mathbf{0}^{n_i \times r_i}$, $\forall i \in [n]$. Let $\mathbf{v}_{i,k} = (\mathbf{v}_{i,k}^{\mathbf{K}}, \mathbf{v}_{i,k}^{\mathbf{V}_p})$.
for $k = 0, 1, 2, \ldots$ **do**
  for each $i \in [n]$ **do**
    $\mathbf{g}_{i,k} = (\mathbf{K}_{i,k}, \mathbf{V}_p^{i,k})$ (By Algorithm 6)
    $\mathbf{m}_{i,k} = \beta_1\mathbf{w}_{i,k-1} + (1-\beta_1)\mathbf{g}_{i,k}$
    $\mathbf{v}_{i,k}^{\mathbf{K}} = \beta_2\mathbf{v}_{i,k-1}^{\mathbf{K}} + (1-\beta_2)\mathbf{K}_{i,k} \odot \mathbf{K}_{i,k}$
    $\mathbf{v}_{i,k}^{\mathbf{V}_P} = \beta_2\mathbf{v}_{i,k-1}^{\mathbf{V}_p} + (1-\beta_2)\mathbf{V}_p^{i,k} \odot \mathbf{V}_p^{i,k}$
    $(\mathbf{U}_{i,k+1}, \mathbf{S}_{i,k+1}, \mathbf{V}_{i,k+1}) = R^{\lambda}_{(\mathbf{U}_{i,k}, \mathbf{S}_{i,k}, \mathbf{V}_{i,k})}\Big(-\eta_k\mathbf{m}_{i,k}/(\sqrt{\mathbf{v}_{i,k}} + \varepsilon)\Big)$ [2]
    (By Algorithm 7)
    $\mathbf{w}_{i,k} = \mathrm{PT}_{(\mathbf{U}_{i,k}, \mathbf{S}_{i,k}, \mathbf{V}_{i,k})\to(\mathbf{U}_{i,k+1}, \mathbf{S}_{i,k+1}, \mathbf{V}_{i,k+1})}(\mathbf{m}_{i,k})$ (By Algorithm 8)

Algorithm 6 Computation of Riemannian gradient for desingularization of bounded-rank matrices

Input: $(\mathbf{U}_i, \mathbf{S}_i, \mathbf{V}_i)_{i=1}^q$ with $\mathbf{U}_i \in \mathrm{St}(m_i, r_i)$, $\mathbf{V}_i \in \mathrm{St}(n_i, r_i)$ and $\mathbf{S}_i = \mathrm{diag}(r_i)$ (with `requires_grad=False`) $\mathbf{A}_i = \mathbf{U}_i\mathbf{S}_i$ and $\mathbf{B}_i = \mathbf{0}^{n_i \times r_i}$ (both with `requires_grad=True`), $\alpha > 0$ (e,g, $\alpha = 0.5$)
for $i \in [q]$ in parallel **do**
    Compute $\mathbf{W}_i = \mathbf{A}_i\mathbf{V}_i^T + \mathbf{U}_i\mathbf{B}_i^T$ $\quad (\mathbf{W}_i = \mathbf{U}_i\mathbf{S}_i\mathbf{V}_i^T)$
    Inject $\mathbf{W}_i$ to the corresponding large matrix
Perform `forward` and `backward` pass
Set $\mathbf{K}_i = \mathbf{A}_i$.`grad`, and $\mathbf{V}_p^i = (\mathbf{I} - \mathbf{V}_i\mathbf{V}_i^T)\mathbf{B}_i$.`grad`$\mathbf{S}_i(2\alpha\mathbf{I} + \mathbf{S}_i^2)^{-1}$
Return $(\mathbf{K}_i, \mathbf{V}_p^i)_{i=1}^q$

Algorithm 7 Q-factor retraction operator with nuclear norm regularization

Input: points $(\mathbf{U}_i, \mathbf{S}_i, \mathbf{V}_i)_{i=1}^q$, vectors $(-\mathbf{K}_i, -\mathbf{V}_p^i)_{i=1}^q$, and constant $\lambda \geq 0$
for $i \in [q]$ in parallel **do**
    Compute a thin QR decomposition $\mathbf{V}_i - \mathbf{V}_p^i = \mathbf{Q}_i\mathbf{R}_i$
    Compute the matrix $\mathbf{W}_i = \big(\mathbf{U}_i\mathbf{S}_i - \mathbf{K}_i - \lambda\mathbf{U}_i)\big)\mathbf{V}_i^T\mathbf{Q}_i - (\mathbf{U}_i\mathbf{S}_i)(\mathbf{V}_p^i)^T\mathbf{Q}_i$
    Compute thin SVD for $\mathbf{W}_i = \bar{\mathbf{U}}_i\bar{\mathbf{S}}_i\mathbf{H}_i^T \in \mathbb{R}^{m\times r}$
    Set $\mathbf{U}_i^+ = \bar{\mathbf{U}}_i, \mathbf{S}_i^+ = \bar{\mathbf{S}}_i, \mathbf{V}_i^+ = \mathbf{Q}_i\mathbf{H}_i$
Return $(\mathbf{U}_i^+, \mathbf{S}_i^+, \mathbf{V}_i^+)_{i=1}^q$

Algorithm 8 Parallel transport for bounded-rank matrices

Signature: $\mathrm{PT}_{(\mathbf{U},\mathbf{S},\mathbf{V})\to(\mathbf{U}^+,\mathbf{S}^+,\mathbf{V}^+)}(\mathbf{K}, \mathbf{V}_p)$
Input: points $(\mathbf{U}, \mathbf{S}, \mathbf{V})$, $(\mathbf{U}^+, \mathbf{S}^+, \mathbf{V}^+)$, vector $(\mathbf{K}, \mathbf{V}_p)$, $\alpha > 0$
$\mathbf{K}^+ = [\mathbf{K}, \mathbf{U}\mathbf{S}]\,[\mathbf{V}, \mathbf{V}_p]^T\,\mathbf{V}^+$
$\mathbf{V}_p^+ = (\mathbf{I} - \mathbf{V}^+(\mathbf{V}^+)^T)\big([\mathbf{V}, \mathbf{V}_p][\mathbf{K}, \mathbf{U}\mathbf{S}]^T\mathbf{U}^+\mathbf{S}^+ + 2\alpha(\mathbf{V}_p\mathbf{V}^T + \mathbf{V}\mathbf{V}_p^T)\mathbf{V}^+\big)\big(2\alpha I + (\mathbf{S}^+)^2\big)^{-1}$
Return $(\mathbf{K}^+, \mathbf{V}_p^+)$