Enhancing Fine-Tuning Efficiency of LLMs Through Gradient Subspace Tracking

Sahar Rajabi **Sirisha Rambhatla**
Critical ML Lab, University of Waterloo
Waterloo, ON, Canada
{srajabi, srambhatla}@uwaterloo.ca

Abstract

Training and fine-tuning Large Language Models (LLMs) require substantial computational resources and time due to their large model sizes and optimizer states. To address these challenges and enhance accessibility, several memory-efficient techniques have been introduced. For instance, Low-Rank Adaptation (LoRA) optimizes model weights within a low-rank subspace, while Gradient Low-Rank Projection (GaLore) reduces the memory footprint by projecting gradients into a lower-dimensional space. In this paper, we introduce Gradient Subspace Tracking (SubTrack), a method that restricts optimization to a compact core subspace of the gradient matrices, and efficiently updates its subspace estimation by leveraging estimation errors and previously identified subspaces. Our results show that even with rank-1 updates to the underlying subspace, SubTrack achieves performance comparable to or better than GaLore, while **reducing runtime by an average of 15% and up to 20.56% on some datasets**.

1 Introduction

Large Language Models (LLMs) have achieved state-of-the-art performance across numerous tasks; however, their training and fine-tuning demand substantial resources, making them impractical for many applications. [32, 12, 22, 20, 21, 10, 16]. As a result, there is an acute need to develop memory-efficient methods to democratize their use and mitigate environmental impacts. Several techniques, such as gradient checkpointing [5] and memory offloading [23] have been proposed to lower memory usage. In this context, Parameter-Efficient Fine-Tuning (PEFT) approaches focus on optimizing a subset of model parameters or operating in a lower-dimensional space to reduce memory usage [7, 29, 17, 24, 27, 20, 11]. Notably, LoRA [11] decomposes weight matrices into two low-rank matrices, optimizing parameters in a lower-dimensional space.

Memory requirements are not limited to the trainable parameters, and a significant portion is consumed by the optimizers. To address this, more recently reducing the optimizer parameters have been another area of focus [15, 1, 19, 6, 31, 21, 32, 22]. GaLore [32] reduces memory usage by projecting gradient matrices into a low-rank subspace by performing periodic Singular Value Decomposition (SVD) on the gradient matrix, to get a rank-r estimation of the gradient space. This approach presents several challenges. First, SVD is computationally expensive, and if the gradient does not evolve within a nearly constant subspace, GaLore must increase the frequency of SVD operations. This poses a significant issue since not all layers' gradients converge to a stable subspace early in the training process [12]. Moreover, applying SVD to a single gradient matrix can be influenced by data noise [26], and GaLore does not use 1) the information in the orthogonal space [21] or 2) the previously computed subspaces to alleviate this effect, thus slowing convergence.

To this end, we propose Gradient Subspace Tracking (SubTrack), a Grassmannian-based subspace tracking method that efficiently updates the subspace using rank-1 updates. SubTrack accumulates

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

gradients between two update steps to reduce noise effect and leverages information from the orthogonal complement to enhance subspace estimation through simple linear algebra operations which are more computationally efficient compared to GaLore, as SubTrack does not perform periodic SVD on the main gradient matrices. Additionally, SubTrack dynamically captures changes in the gradient subspace and reduces the jumps in subspace updates, for faster convergence.

2 Related Works

LoRA [11], is a widely recognized method for reducing the number of trainable parameters. It projects the model weights into a lower-dimensional space, which in turn reduces memory requirements. Dettmers et al. [7] employ quantization techniques and paged optimizers on top of LoRA to further reduce memory usage. Yaras et al. [29] introduced Deep LoRA, which utilizes deep matrix factorization to address overfitting and reduce the need for precise tuning of the rank parameter. Several other works also extend LoRA to improve the efficiency of training large models [17, 24, 27]. Miles et al. [20] propose compressing the intermediate activation vectors and then reconstructing them for a proper backpropagation. Additionally, Hao et al. [10] demonstrate that full-parameter fine-tuning is feasible by employing random projections to the gradient matrix by showing that LoRA is essentially a down-projection of gradient.

Several approaches focus on reducing memory consumption in optimizers, as optimizers like Adam [14] are responsible for a substantial portion of memory usage [15, 1, 19, 6, 31]. MicroAdam [21] addresses this by compressing the gradient space while using the resulting error via feedback loops. According to Gur-Ari et al. [9], a substantial portion of gradients tends to lie within a small subspace that remains largely consistent. This observation has been approved by multiple studies, including Schneider et al. [25], Yaras et al. [28]. Gradient Low-Rank Projection (GaLore) [32] leverages this property of gradient space to reduce memory requirements by projecting gradients into a lower-dimensional subspace. This approach has been successfully integrated with other methods to further reduce memory usage during fine-tuning [16]. However, not all layer gradients in an LLM evolve in a low-rank subspace. Jaiswal et al. [12] identify layers with constantly changing gradients where low-rank projection is inefficient. By analyzing the singular values' distribution, they select layers that evolve within a small subspace for fine-tuning, while freezing the others. Gradient Structured Sparsification (Grass) [22] further minimizes memory usage by applying sparse projection matrices, transforming the gradient matrix into a sparse vector space.

When working with high-dimensional data, a common strategy is to project the data into a lower-dimensional space, and there are many studies focusing on cases where the underlying subspace evolves over time. Balzano et al. [2] introduce an incremental method for updating subspaces on the Grassmannian manifold when data is partially observed. Zhang and Balzano [30] and Kasai [13] address the challenge of noisy data in streaming and evolving environments, and Blocker et al. [4] introduced a method for time-varying data based on Geodesics in Grassmannian space.

3 SubTrack: Tracking the Gradient Subspace

Since gradients tend to evolve within a small subspace, compressing the gradient space can effectively reduce the optimizers' memory footprint. However, the gradient's subspace does not always remain stable, and tracking this changes is crucial for optimization purposes. GaLore [32] addresses this by periodically performing SVD on gradient matrices. We propose Gradient Subspace Tracking or SubTrack, a computationally efficient method for tracking gradient subspaces. SubTrack leverages information in the orthogonal space and the previously computed subspace to update the core subspace. The subspace initialization is performed using SVD, as follows

$$G_0 = U\Sigma V^\top \approx \sum_{i=1}^{r} \sigma_i u_i v_i^\top, \quad P_0 = [u_1, u_2, ..., u_r], \quad Q_0 = [v_1, v_2, ..., v_r]. \tag{1}$$

Here, G_0 is an $m \times n$ gradient matrix at step 0, and U, S, and V are its SVD components, with r representing the specified rank. At each optimization step, we project the gradients onto left singular vectors subspace if $m \leq n$, and onto right singular vectors otherwise, to further optimizing the memory usage[32]. The optimization then takes place in this subspace and afterward, the gradient is projected back, enabling full parameter tuning. Henceforth, we assume that $m \leq n$ w.l.o.g.

To utilize the orthogonal space while mitigating the effects of noise, SubTrack computes an accumulated gradient by averaging the gradients between two subspace update steps, as shown below, where T_n and T_{n-1} are the steps in which we update the underlying subspace.

$$G_{acc} = \frac{1}{T_n - T_{n-1}} \sum_{\substack{t= \\ T_{n-1}}}^{T_n} G_t \tag{2}$$

We then frame the problem of identifying the subspace as selecting the appropriate element from the Grassmannian, which is the set of all d-dimensional subspaces within an n-dimensional vector space [3]. Our goal is to minimize the Euclidean distance between the current subspace and the observed accumulated gradient G_{acc} at each update step, with the cost function defined as

$$F(S_t) = \min_A \|S_t A - G_{acc}\|_F^2, \tag{3}$$

where S_t is an orthonormal matrix whose columns span the current subspace and A is the answer of the least square problem. The derivative of this function with respect to S_t is in (4), and $R = G_{acc} - S_t A$ lies in the orthogonal complement of S_t. We then compute the tangent vector ∇F on the Grassmannian manifold for updating the subspace in the given direction [8] as in (5), in which the second equality holds as R is orthogonal to $S_t S_t^\top$.

$$\frac{\partial F}{\partial S_t} = 2(S_t A - G_{acc})A^\top = -2RA^\top \tag{4}$$

$$\nabla F = (I - S_t S_t^\top)\frac{\partial F}{\partial S_t} = \frac{\partial F}{\partial S_t} = -2RA^\top \approx \widehat{U}_F \widehat{\Sigma}_F \widehat{V}_F^\top \tag{5}$$

∇F gives the direction for adjusting the subspace considering the error that lies in the orthogonal complement; however, to keep subspace changes to a minimum; SubTrack first computes a rank-1 estimation of ∇F indicated by its largest singular value and associated singular vectors gained form its SVD, represented as $\hat{U}_F \hat{\Sigma}_F \hat{V}_F^\top$, for updating the subspace. As demonstrated by Edelman et al. [8], Bendokat et al. [3], the subspace can be updated with a step of size η on the Grassmannian using the SVD of associated tangent vector, as shown in Equation 6.

$$S_{t+1}(\eta) = \begin{pmatrix} S_t \widehat{V}_F & \widehat{U}_F \end{pmatrix} \begin{pmatrix} \cos \widehat{\Sigma}_F \eta \\ \sin \widehat{\Sigma}_F \eta \end{pmatrix} \widehat{V}_F^\top + S_t(I - \widehat{V}_F \widehat{V}_F^\top) \tag{6}$$

Using the geometry of Grassmannian manifold, SubTrack effectively tracks the underlying subspace of gradient space, and Algorithm 1 presents the pseudo-code of this method.

4 Experiments

Fin-Tuning Experiment. To ensure a fair comparison of computational efficiency between SubTrack and GaLore, we fine-tuned RoBERTa-Base [18] measuring the corresponding wall-time and performance while keeping all shared hyperparameters consistent. Wall-time is the real-world elapsed time it takes for a process or operation to complete, measured from start to finish, including both the actual runtime and any waiting time for resources or data retrieval. To compare wall-time, RoBERTa-Base was fine-tuned on GLUE tasks for 2500 iterations, with the subspace update interval set to 500 iterations with ranks 4 and 8. Consequently, both methods update the underlying subspace exactly five times, and evaluation steps were excluded to ensure an accurate runtime comparison. For performance evaluation, the model was fine-tuned on these tasks for 30 epochs, with results presented in Table 1. As shown, SubTrack reduces runtime by up to 20.56%. Despite restricting updates to rank-1 changes to the previous subspace, Table 1 demonstrates that SubTrack achieves performance comparable to or even surpassing that of GaLore. Further experimental details are provided in Appendix A.

Runtime Consistency. Figure 1 compares the wall-times of GaLore and SubTrack across subspace update intervals ranging from 50 to 500 while fine-tuning RoBERTa-Base on the COLA task with an NVIDIA T4 GPU. The subspace update interval denotes the number of iterations between two updates; thus, increasing this interval reduces the update frequency. As shown, GaLore's runtime significantly increases with more frequent subspace updates, whereas SubTrack maintains minimal runtime overhead regardless of the update frequency.

Algorithm 1 SubTrack

Require: Sequence of $m \times n$ gradients G_t with $m \leq n$ (w.l.o.g.), step-size η, rank r, subspace update steps k

Initialize Subspace via SVD Decomposition:

$P_0 \leftarrow U[:, :r]$, where $U, \Sigma, V \leftarrow \text{SVD}(G_0)$

$S_0 \leftarrow P_0$ {The initial subspace}

$G_{acc} = 0_{m\times n}$ {To keep the accumulated gradient}

for $t = 1, \ldots, T$ **do**

  if $t \mod k = 0$ **then**

    Prepare accumulated gradients: $G_{acc} = \frac{G_{acc}+G_t}{k}$

    Update subspace:

    $G_{lr} = \arg\min_A \|(S_{t-1}A - G_{acc})\|^2$ {Solving the least square problem}

    $R = G_{acc} - S_{t-1}G_{lr}$ {Computing the residual}

    $\nabla F = -2RG_{lr}^\top \approx \widehat{U}_F \widehat{\Sigma}_F \widehat{V}_F^\top$ {Computing the rank-1 estimation of tangent vector}

    $S_t = (S_{t-1}\widehat{V}_F \quad \widehat{U}_F) \begin{pmatrix} \cos \widehat{\Sigma}_F \eta \\ \sin \widehat{\Sigma}_F \eta \end{pmatrix} \widehat{V}_F^\top + S_{t-1}(I - \widehat{V}_F \widehat{V}_F^\top)$ {Updating the subspace}

    Reset accumulated gradients: $G_{acc} = 0_{m\times n}$

  else

    Keep using previous subspace: $G_{acc} = G_{acc} + G_t$, $S_t = S_{t-1}$

  Return final projected gradient to the optimizer: $S_t^\top G_t$

Table 1: Evaluating SubTrack and GaLore on their performance and runtime when fine-tuning RoBERTa-Base on GLUE tasks for ranks 8. All hyperparameters including scale-factor and subspace update interval are the same. SubTrack achieved better average performance compared to GaLore while spending a considerably less time for fine-tuning. The performance is measured after fine-tuning for 30 epochs. Wall-times are reported in seconds, and are measured after fine-tuning for 2500 iterations, removing the evaluation steps while performing exactly 5 updates for having a fair comparison between these two methods.

		COLA	STS-B	MRPC	RTE	SST-2	MNLI	QNLI	QQP	Avg
GaLore [32] Rank = 4	Perf.	60.34	90.58	92.58	76.53	94.27	87.12	92.20	87.86	85.18
	Time	195.7	195.1	200.2	325.8	185.7	206.8	216.5	190.4	214.5
SubTrack (Ours) Rank = 4	Perf.	61.32	90.64	92.66	77.98	94.15	86.85	91.85	87.50	85.37
	Time	155.7	158.0	172.7	305.7	147.5	175.0	192.2	155.8	182.8
Reduction in Wall-Time		**20.44%**	**19.02%**	**13.74%**	**6.17%**	**20.57%**	**15.38%**	**11.22%**	**18.17%**	**15.59%**
GaLore [32] Rank = 8	Perf.	58.54	90.61	91.30	74.37	94.50	87.34	92.71	87.99	84.67
	Time	188.0	195.6	196.5	328.3	187.1	208.0	217.1	189.4	213.7
SubTrack (Ours) Rank = 8	Perf.	58.54	90.87	91.43	76.53	94.27	87.09	92.49	87.57	84.85
	Time	149.4	159.1	171.2	304.9	148.9	177.1	192.1	156.8	182.4
Reduction in Wall-Time		**20.56%**	**18.68%**	**12.87%**	**7.11%**	**20.37%**	**14.89%**	**11.48%**	**17.21%**	**15.40%**

5 Discussion and Conclusion

We proposed a computationally efficient method that projects gradients into a lower-dimensional subspace, updating this subspace by tracking its changes over time. SubTrack preserves the previously computed subspace and incorporates gradient components from the orthogonal complement to perform rank-1 updates. This approach reduces the frequency of abrupt transitions between iterations and leverages the available information effectively.

In some cases, the extent of changes in the subspace may require updates of rank greater than 1. During our experiments, we observed that applying updates as per (6), using the SVD of the tangent vector from (5), can hinder convergence if the singular values of the tangent vector become very small. Furthermore, simultaneously updating subspace dimensions associated with sufficiently large singular values caused convergence issues in some cases. Therefore, we restricted updates to rank-1 in this paper, as this approach still enabled SubTrack to achieve performance comparable to or better than GaLore, with reduced runtime. In future work, we plan to explore increasing the rank of updates

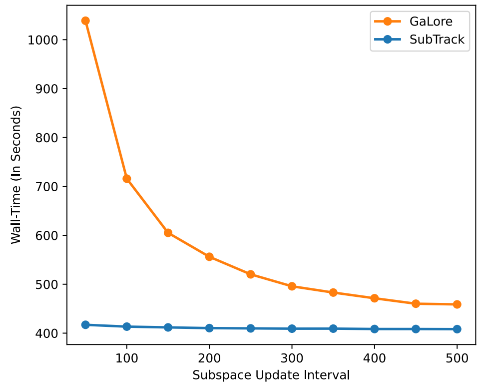

Figure 1: Comparing runtimes of GaLore and SubTrack. RoBERTa-Base is fine-tuned for 10 epochs on COLA. Subspace update intervals range from 50 to 500. Notice that by increasing subspace update interval, the update frequency actually decreases, as it indicates the number of iterations between two subspace update steps.

without compromising convergence. Additionally, dynamically selecting the step size could eliminate the need for manual tuning as a hyperparameter and further enhance convergence.

References

[1] Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. Memory-efficient adaptive optimization, 2019. URL `https://arxiv.org/abs/1901.11150`.

[2] Laura Balzano, Robert Nowak, and Benjamin Recht. Online identification and tracking of subspaces from highly incomplete information, 2011. URL `https://arxiv.org/abs/1006.4046`.

[3] Thomas Bendokat, Ralf Zimmermann, and P.-A. Absil. A grassmann manifold handbook: basic geometry and computational aspects. *Advances in Computational Mathematics*, 50(1), January 2024. ISSN 1572-9044. doi: 10.1007/s10444-023-10090-8. URL `http://dx.doi.org/10.1007/s10444-023-10090-8`.

[4] Cameron J. Blocker, Haroon Raja, Jeffrey A. Fessler, and Laura Balzano. Dynamic subspace estimation with grassmannian geodesics, 2023. URL `https://arxiv.org/abs/2303.14851`.

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL `https://arxiv.org/abs/1604.06174`.

[6] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization, 2022. URL `https://arxiv.org/abs/2110.02861`.

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

[8] Alan Edelman, T. A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints, 1998. URL `https://arxiv.org/abs/physics/9806030`.

[9] Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace, 2018. URL `https://arxiv.org/abs/1812.04754`.

[10] Yongchang Hao, Yanshuai Cao, and Lili Mou. Flora: Low-rank adapters are secretly gradient compressors, 2024. URL `https://arxiv.org/abs/2402.03293`.

[11] 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.

[12] Ajay Jaiswal, Lu Yin, Zhenyu Zhang, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. From galore to welore: How low-rank weights non-uniformly emerge from low-rank gradients, 2024. URL https://arxiv.org/abs/2407.11239.

[13] Hiroyuki Kasai. Fast online low-rank tensor subspace tracking by cp decomposition using recursive least squares from incomplete observations, 2017. URL https://arxiv.org/abs/1709.10276.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[15] Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states, 2023. URL https://arxiv.org/abs/2309.01507.

[16] Pengxiang Li, Lu Yin, Xiaowei Gao, and Shiwei Liu. Owlore: Outlier-weighed layerwise sampled low-rank projection for memory-efficient llm fine-tuning, 2024. URL https://arxiv.org/abs/2405.18380.

[17] Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates, 2023. URL https://arxiv.org/abs/2307.05695.

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

[19] Kai Lv, Yuqing Yang, Tengxiao Liu, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8187–8198, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.445.

[20] Roy Miles, Pradyumna Reddy, Ismail Elezi, and Jiankang Deng. Velora: Memory efficient training using rank-1 sub-token projections, 2024. URL https://arxiv.org/abs/2405.17991.

[21] Ionut-Vlad Modoranu, Mher Safaryan, Grigory Malinovsky, Eldar Kurtic, Thomas Robert, Peter Richtarik, and Dan Alistarh. Microadam: Accurate adaptive optimization with low space overhead and provable convergence, 2024. URL https://arxiv.org/abs/2405.15593.

[22] Aashiq Muhamed, Oscar Li, David Woodruff, Mona Diab, and Virginia Smith. Grass: Compute efficient low-memory llm training with structured sparse gradients, 2024. URL https://arxiv.org/abs/2406.17660.

[23] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

[24] Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-LoRA: Enhancing parameter efficiency of LoRA with weight tying. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8694–8705, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.481. URL https://aclanthology.org/2024.naacl-long.481.

[25] Jan Schneider, Pierre Schumacher, Simon Guist, Le Chen, Daniel Häufle, Bernhard Schölkopf, and Dieter Büchler. Identifying policy gradient subspaces, 2024. URL https://arxiv.org/abs/2401.06604.

[26] Namrata Vaswani, Thierry Bouwmans, Sajid Javed, and Praneeth Narayanamurthy. Robust subspace learning: Robust pca, robust subspace tracking, and robust subspace recovery. *IEEE Signal Processing Magazine*, 35(4):32–55, July 2018. ISSN 1558-0792. doi: 10.1109/msp.2018.2826566. URL http://dx.doi.org/10.1109/MSP.2018.2826566.

[27] Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of lora: Efficient fine-tuning of language models via residual learning, 2024. URL https://arxiv.org/abs/2401.04151.

[28] Can Yaras, Peng Wang, Wei Hu, Zhihui Zhu, Laura Balzano, and Qing Qu. Invariant low-dimensional subspaces in gradient descent for learning deep matrix factorizations. In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*, 2023.

[29] Can Yaras, Peng Wang, Laura Balzano, and Qing Qu. Compressible dynamics in deep overparameterized low-rank learning & adaptation. *arXiv preprint arXiv:2406.04112*, 2024.

[30] Dejiao Zhang and Laura Balzano. Global convergence of a grassmannian gradient descent algorithm for subspace estimation, 2016. URL https://arxiv.org/abs/1506.07405.

[31] Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more, 2024. URL https://arxiv.org/abs/2406.16793.

[32] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL https://arxiv.org/abs/2403.03507.

A Fine-Tuning RoBERTa-Base

To compare the computational efficiency and performance of SubTrack with GaLore, we fine-tuned RoBERTa-Base using the hyperparameters reported in Table 2, which are identical to those reported in the GaLore paper for rank-4 and rank-8 subspaces, with a subspace update interval of 500 iterations.

Table 2: Hyperparameters of fine-tuning RoBERTa-Base.

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	1E-05	1E-05	3E-05	3E-05	1E-05	1E-05	1E-05	1E-05
SubTrack Step Size	0.001	0.001	1.5	0.1	0.0001	0.001	1.0	1.0
Rank Config.				$r = 4$				
α				4				
Max Seq. Len.				512				

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	1E-05	2E-05	2E-05	1E-05	1E-05	2E-05	2E-05	3E-05
SubTrack Step Size	0.001	0.01	15.0	3.0	0.001	0.001	1.0	1.0
Rank Config.				$r = 8$				
α				2				
Max Seq. Len.				512				