RGP: Achieving Memory-Efficient Model Fine-tuning Via Randomized Gradient Projection

Ali Saheb Pasand
Department of Computer Science
McGill University
Montreal, Canada
`ali.sahebpasand@mail.mcgill.ca`

Pouya Bashivan
Department of Physiology
McGill University
Montreal, Canada
`pouya.bashivan@mcgill.ca`

Abstract

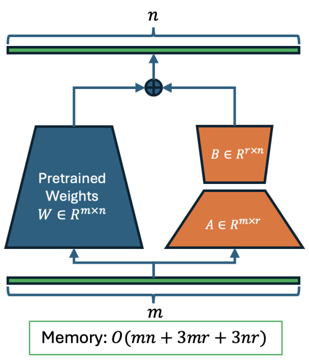
Training and fine-tuning Large Language Models (LLMs) require significant memory due to the substantial growth in the size of weight parameters and optimizer states. While methods like low-rank adaptation (LoRA), which introduce low-rank trainable modules in parallel to frozen pre-trained weights, effectively reduce memory usage, they often fail to preserve the optimization trajectory and are generally less effective for pre-training models. On the other hand, approaches, such as GaLore, that project gradients onto lower-dimensional spaces maintain the training trajectory and perform well in pre-training but suffer from high computational complexity, as they require repeated singular value decomposition on large matrices. In this work, we propose Randomized Gradient Projection (**RGP**), which outperforms GaLore, the current state-of-the-art in efficient fine-tuning, on the GLUE task suite, while being 74% faster on average and requiring similar memory.

1 Introduction

As the size of deep learning models increases, training them becomes more challenging due to the increase in memory and computational costs. The memory requirement increases significantly because the modern optimizers, such as the Adam [9], require tracking multiple gradient states during training. In recent years, multiple solutions have been proposed to address this issue. One of the most acclaimed and widely adopted methods is LoRA [5], which models weight updates as a multiplication of two low-rank matrices. Figure 1 shows a schematic of how a LoRA module is added to a network's weight matrix. Although LoRA has been proven to be effective for fine-tuning, it does not perform well for model pre-training and also it alters the training trajectory [22]. Other methods for reducing the memory required to track state information are based on gradient projection [1, 2, 22]. However, these methods fail to exploit the structure of low-rank gradient matrices [22]. Recently, an alternative method has been proposed (GaLore [22]), which involves projecting gradients to lower dimensions and tracking the state matrices in the resulting lower-rank subspace. This leads to a significant reduction in the memory footprint required to track the optimizer states during training. However, GaLore also exhibits certain drawbacks which can be outlined as follows:

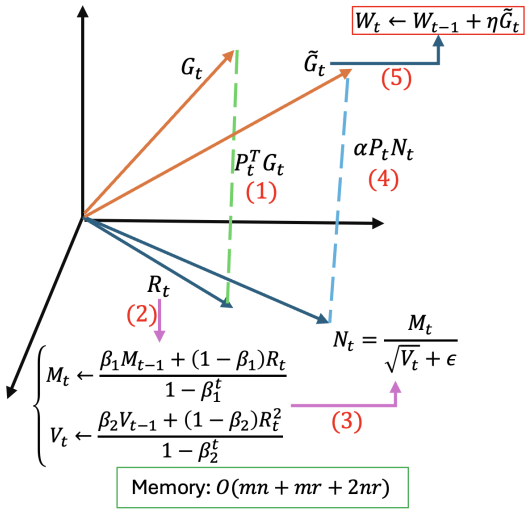
1. It requires frequent calculation of Singular Value Decomposition (SVD) on large matrices.
2. The projected sub-space's rank is fixed during training which is potentially sub-optimal as it has been shown that the rank of gradients continuously decreases during training [22].
3. Projection matrix needs to be calculated and stored. Also, the projection matrix is unstructured and as a result, projection complexity cannot be reduced from the general matrix-vector multiplication complexity.

In this work, we propose Randomized Gradient Projection (**RGP**) as an alternative method with lower computational cost which paves the way for more frequent projection matrix update . Comparing **RGP** with GaLore shows that **RGP** outperforms GaLore in the fine-tuning of NLP (Natural Language Processing) models for solving GLUE (General Language Understanding Evaluation) tasks [18].



n
Pretrained Weights $W \in R^{m\times n}$
$B \in R^{r\times n}$
$A \in R^{m\times r}$
m
Memory: $O(mn + 3mr + 3nr)$

Figure 1: Schematic of Low-Rank Adaptation (**LoRA**). To reduce the memory foot-print, a parallel low-rank module is added to the pre-trained weights. Only those low-rank modules (orange blocks) are trainable and therefore, only momentum and variance of the gradient of them must be tracked.



$W_t \leftarrow W_{t-1} + \eta \tilde{G}_t$
G_t
$\tilde{G}_t$
(5)
$P_t^T G_t$
(1)
$\alpha P_t N_t$
(4)
R_t
(2)
$N_t = \frac{M_t}{\sqrt{V_t} + \epsilon}$
$M_t \leftarrow \frac{\beta_1 M_{t-1} + (1-\beta_1) R_t}{1-\beta_1^t}$
$V_t \leftarrow \frac{\beta_2 V_{t-1} + (1-\beta_2) R_t^2}{1-\beta_2^t}$
(3)
Memory: $O(mn + mr + 2nr)$

Figure 2: RGP Scheme. Without loss of generality, let the gradient matrix $G \in \mathbb{R}^{m\times n}$ have fewer rows than columns. Initially, the full-rank gradient is down-projected into a rank-r subspace, achieved by computing a projection matrix derived from the first r left singular vectors from randomized singular value decomposition on the full-rank gradient matrix. Then, weight updates are computed based on momentum and variance tracked within this reduced subspace. Finally, the gradient is projected back into the original n-dimensional space ($\widetilde{G}$) and utilized for updating weights.

2 Related Work

An active direction for achieving memory-efficient fine-tuning and pre-training is based on employing adaptation modules. Examples include various types of adapter modules that are integrated serially into the model layers [4, 6], or in parallel to them (e.g. LoRA [5]). LoRA's early success, in particular, has inspired numerous variations of adapter modules. Diverse variants of LoRA [14, 16, 20], along with methods for improved multitask fine-tuning [13, 19, 3], have been proposed. Moreover, methods have been suggested to incorporate LoRA modules with dynamic rank [17, 21].

Despite its wide application, LoRA is ineffective for pre-training models due to its rank deficiency, which gave rise to a closely related method called ReLoRA [11]. The core idea of this method is that the rank of the sum of weight updates are less than or equal of the summation of their individual ranks. ReLoRA merges the decomposed matrix with the frozen weight part at specific intervals to prevent divergence and to increase the rank of updates [11]. Methods based on matrix decomposition alter optimization trajectories, in contrast with methods based on gradient projection [22]. However, beside trajectory alteration, ReLoRA suffers for inefficiency due to requiring full-rank warm-ups, limiting its applicability on certain low-budget devices.

Another relevant direction involves projected gradient descent [1], which treats the model as a black box and operates on gradients as vectors, limiting compression levels due to not exploiting the natural matrix structure of weight matrices. Additionally, methods explored for memory reduction during optimization include weight or optimizer state quantization [10, 7, 8]. These techniques can be integrated with the method proposed in this paper without added complexity. Furthermore, another approach involves projecting optimizer states, as seen in Adafactor [15], where **RGP** can be implemented alongside with it, offering additional memory savings.

GaLore is the closest method to our proposed method **RGP**. The idea behind **RGP** is to use randomized down-projection of gradient matrices instead of performing SVD. Due to using techniques from randomized linear algebra, **RGP** is significantly faster than GaLore and as a result, more frequent updates on the projection matrix for the gradient can be calculated. Authors of GaLore paper state that more frequent projection matrix updates degrades the performance. However, we show in the Results section that more frequent updates improve the performance to a specific point, and with a fixed update interval, **RGP** performs better and faster than GaLore.

3 Methodology

The general scheme of our method is shown in Figure 2. The underlying concept of **RGP** is based on projecting full-rank gradients onto a subspace with rank r, where r is significantly smaller than the minimum of the number of rows and columns of the gradient matrix. Subsequently, the states are tracked in this significantly lower-dimensional space. The projection matrix is calculated by performing randomized singular value decomposition on the full-rank matrix of gradients and taking the first r columns as the projection matrix. By performing this projection, the size of the states that needs to be tracked, including momentum and variance in the Adam optimizer, is significantly reduced. Since calculating the projection matrix is time-consuming, it is computed only at intervals T iterations to achieve a balance between performance and execution time.

The memory usage of full-rank training, when using the Adam optimizer, is of the order $O(3mn)$. This includes tracking one $m \times n$ weight matrix, and two $m \times n$ Momentum and Variance state matrices (M and V). In **RGP**, with the same optimizer, the memory usage is of the order $O(mn + mr + 2nr)$. This includes tracking one $m \times n$ weight matrix, one $m \times r$ projection matrix, and two $n \times r$ state matrices (M and V). With a fixed rank r, this memory usage is lower than that of LoRA, which has a memory complexity of order $O(mn + 3mr + 3nr)$. LoRA tracks one $m \times n$ weight matrix, one A matrix with dimension $m \times r$, two states corresponding to it with the same size, and one B matrix with dimension $r \times n$, along with two states corresponding to it with the same size. In randomized SVD, usually more singular vectors compared to the desired output dimension are sampled. If we take k to be the output dimension, which is slightly larger than desired dimension r, the overall complexity of randomized-SVD is of order $O(mnk)$ which will be done every T intervals. This complexity scales linearly with respect to the matrix dimensions. In contrast, GaLore introduces a quadratic complexity $O(\min\{mn^2, nm^2\})$, at every T training iterations.

4 Results

In this section, we investigate the results of employing **RGP** to fine-tune a RoBERTa [12] model on GLUE tasks [18]. The optimizer utilized for all of these datasets is Adam, and all hyperparameters are consistent with those reported in GaLore [22], which are listed in Table 3 in Appendix A.

4.1 Performance Comparison of Different Proposed Methods

Table 1: The final epoch dev set accuracy on GLUE tasks. Results are averaged over 5 runs. For each update interval, the best method is underlined. The best performance across all update intervals and methods is displayed in **bold**.

Method	Update Interval	SST-2	MRPC	CoLA	RTE	MNLI	QNLI	QQP	Avg.
Full-rank	NA	94.57	91.30	62.24	79.42	87.18	92.33	92.28	85.6
GaLore	200	94.42 ± 0.40	92.21 ± 0.22	$\mathbf{\underline{60.82}} \pm 1.08$	77.50 ± 0.36	86.82 ± 0.14	92.44 ± 0.57	90.96 ± 0.07	85.0
	500	$\underline{94.32} \pm 0.56$	$\underline{91.69} \pm 0.29$	59.82 ± 0.67	77.25 ± 0.42	$\underline{86.79} \pm 0.03$	92.15 ± 0.38	$\underline{90.97} \pm 0.06$	84.7
	800	$\underline{94.27} \pm 0.11$	$\underline{91.62} \pm 0.80$	58.76 ± 1.27	77.01 ± 0.41	86.76 ± 0.11	$\underline{92.11} \pm 0.33$	$\underline{90.95} \pm 0.04$	$\underline{84.5}$
RGP	200	$\mathbf{\underline{94.45}} \pm 0.13$	$\mathbf{\underline{92.42}} \pm 0.45$	60.42 ± 0.01	$\mathbf{\underline{78.70}} \pm 0.95$	$\mathbf{\underline{86.90}} \pm 0.07$	$\mathbf{\underline{92.59}} \pm 0.19$	$\mathbf{\underline{90.99}} \pm 0.13$	$\mathbf{\underline{85.2}}$
	500	94.19 ± 0.28	91.66 ± 0.91	$\underline{60.76} \pm 0.78$	$\underline{77.86} \pm 0.54$	86.78 ± 0.01	$\underline{92.27} \pm 0.39$	90.93 ± 0.08	$\underline{84.9}$
	800	94.08 ± 0.17	91.00 ± 0.71	$\underline{59.07} \pm 1.16$	$\underline{77.25} \pm 0.54$	$\underline{86.78} \pm 0.23$	92.08 ± 0.27	90.91 ± 0.05	84.4

We assessed the performance of different methods for fine-tuning models on GLUE tasks. The results are summarized in Table 1. In this table, **RGP** refers to Randomized Gradient Projection with over sampling factor 2 ($k = r + 2$). **RGP** uses the same scaling factor α as GaLore (GaLore hyperparameters are shown in Table 3) in Appendix A.

Based on these results, several observations can be made:

1. Decreasing the interval of updates, more frequent updates improves results which justifies the need for the more efficient method to calculate projection matrix proposed in this paper.
2. The best-performing method in terms of average performance and across the majority of tasks is **RGP**.

4.2 Time and Performance Comparison

As discussed in the previous subsection and shown in Table 1, **RGP** outperforms GaLore on average and in 6 out of 7 tasks. Additionally, it reduces the performance gap between Full-rank training and memory-efficient training to 0.4 percent. This superior performance compared to GaLore is achieved with a substantially more computationally efficient fine-tuning as well. Table 2 quantifies the computational efficiency of RGP, which displays the execution time ratio of GaLore over **RGP**. **RGP** is significantly faster than GaLore in all tasks, with an average relative speedup of 74% over GaLore. This further puts emphasize on the advantages of adopting our randomized algorithm.

Table 2: The ratio of **RGP** training time compared to GaLore for the best performing update interval (T = 200).

Method	SST-2	MRPC	CoLA	RTE	MNLI	QNLI	QQP	Avg.
RGP	1.98x	1.31x	1.42x	1.15x	2.01x	2.23x	2.13x	1.74x

5 Conclusion and Future Work

In this paper, we introduced Randomized Gradient Projection (**RGP**) as an efficient technique for memory-constrained model training, offering minimal computational overhead. Our results demonstrated that **RGP** not only surpasses the state-of-the-art method, GaLore, in performance but also achieves a 74% faster execution time.

Contrary to our initial hypothesis, starting with a higher rank for the down-projection and gradually reducing it did not yield the expected improvements in results B. Investigating the underlying reasons for this unexpected behavior remains an open area for future research. Future work should also focus on exploring projection methods that could further enhance computational efficiency. Specifically, analyzing the performance of fast projection techniques, such as Walsh–Hadamard Projection, could reveal additional improvements in the complexity of **RGP** and similar algorithms.

References

[1] Han Chen, Garvesh Raskutti, and Ming Yuan. Non-convex projected gradient descent for generalized low-rank tensor regression. *Journal of Machine Learning Research*, 20(5):1–37, 2019.

[2] Yudong Chen and Martin J Wainwright. Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees. *arXiv preprint arXiv:1509.03025*, 2015.

[3] Alexandra Chronopoulou, Matthew E Peters, Alexander Fraser, and Jesse Dodge. Adaptersoup: Weight averaging to improve generalization of pretrained language models. *arXiv preprint arXiv:2302.07027*, 2023.

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

[5] 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.

[6] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

[7] Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Advances in Neural Information Processing Systems*, 36, 2024.

[8] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36, 2024.

[11] Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. Relora: High-rank training through low-rank updates. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*, 2023.

[12] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[13] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

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

[15] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[16] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*, 2023.

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

[18] 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.

[19] Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*, 1(2):4, 2022.

[20] 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.

[21] Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv preprint arXiv:2403.09113*, 2024.

[22] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

A Hyperparamaters

The hyperperameters used for experiments are listed in Table 3.

B Ablation Study on the Decay Ratio and Projection Method in Rank-Decaying RGP

As discussed in introduction section, one of GaLore's drawbacks is having the same rank on the space the gradients will be projected on throughout training. The idea of having decaying rank tested for **RGP**. The Rank-Decaying **RGP** method relies on two important parameters to be chosen:

1. Decay Ratio: This parameter determines the rate at which we reduce the rank at each interval.
2. Type of Projection for State Matrices: This parameter specifies the method used for down-projecting the M and V matrices (SVD or RSVD).

The results for the two choices for each of these parameters and different updating intervals are presented in Table 4. In all cases, the starting rank is set to 256. It is noteworthy that decay is terminated when the rank reaches a value of 4 to ensure comparability with GaLore. Additionally, decay occurs at each update interval. Hence, for instance, when the update interval is 200 and the decay ratio is 4, the decay ceases at iteration $3 \times 200 = 600$. However, for an interval of 500, it stops at iteration $3 \times 500 = 1500$. This scheme is adopted to compensate for any performance deficiency in scenarios with larger interval updates. As it can be seen in Table 4, the configuration with a Decay Ratio of 4 and SVD for down-projection outperforms other ones.However, the best perfroming scenario does not outperform GaLore and RGP.

Table 3: Hyperparametrs used for each GLUE task.

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE
Batch Size	16	16	16	32	16	16	16
# Epochs	30	30	30	30	30	30	30
Learning Rate	1×10^{-5}	1×10^{-5}	3×10^{-5}	3×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Rank Config.				$r = 4$			
GaLore α				4			
Max Seq. Len.				512			

Table 4: Performance comparison between different configurations of Rank-Decaying RGP. Where RSVD stands for Randomized SVD. For each update interval, the best method is <u>underlined</u>. The best performance across all update intervals and methods is displayed in **bold**.

Configuration	Update Interval	SST-2	MRPC	CoLA	RTE	Average
DR: 4, SVD	200	<u>**94.42**</u> $\pm$ 0.06	<u>90.73</u> $\pm$ 0.14	56.25 $\pm$ 1.41	<u>**79.66**</u> $\pm$ 1.04	<u>**80.26**</u>
	500	93.88 $\pm$ 0.24	90.84 $\pm$ 0.14	53.65 $\pm$ 2.03	75.21 $\pm$ 1.43	78.39
	800	93.42 $\pm$ 0.42	84.72 $\pm$ 4.83	<u>58.97</u> $\pm$ 1.41	75.20 $\pm$ 1.80	78.08
DR: 4, RSVD	200	94.12 $\pm$ 0.26	85.26 $\pm$ 4.11	<u>**59.74**</u> $\pm$ 1.42	75.33 $\pm$ 4.16	78.61
	500	94.11 $\pm$ 0.47	87.14 $\pm$ 4.48	<u>58.56</u> $\pm$ 1.18	<u>78.34</u> $\pm$ 1.08	<u>79.54</u>
	800	94.11 $\pm$ 0.24	86.86 $\pm$ 5.11	56.59 $\pm$ 0.53	<u>76.77</u> $\pm$ 2.92	<u>78.58</u>
DR: 2, SVD	200	93.77 $\pm$ 0.41	89.21 $\pm$ 1.62	55.22 $\pm$ 0.91	76.58 $\pm$ 2.16	78.67
	500	<u>94.23</u> $\pm$ 0.13	83.10 $\pm$ 5.58	55.67 $\pm$ 0.62	67.62 $\pm$ 4.54	75.15
	800	<u>94.19</u> $\pm$ 0.24	83.68 $\pm$ 4.89	48.29 $\pm$ 3.24	73.40 $\pm$ 3.24	74.89
DR: 2, RSVD	200	94.07 $\pm$ 0.35	88.65 $\pm$ 2.25	57.61 $\pm$ 1.20	77.49 $\pm$ 1.09	79.45
	500	93.89 $\pm$ 0.14	<u>**91.04**</u> $\pm$ 1.11	56.62 $\pm$ 0.13	75.69 $\pm$ 2.20	79.31
	800	93.92 $\pm$ 0.19	<u>89.52</u> $\pm$ 1.46	57.44 $\pm$ 2.34	71.84 $\pm$ 3.44	78.18