SimplexLoRA: Dynamic Rank Updating via Alternating Minimization with Simplex Projection

Igor Shalygin[1,2,*], **Grigorii Davydenko**[1,2,3,*], **Andrey Veprikov**[1,2,3], **Aleksey Antonov**[1,2,4]
Nikolay Bushkov[4], **Stanislav Moiseev**[4], **Aleksandr Beznosikov**[1,2,5]
[1]Moscow Independent Research Institute of Artificial Intelligence (MIRAI)
[2]Basic Research of Artificial Intelligence Laboratory (BRAIn Lab)
[3]SB AI Lab [4]T-Technologies [5]Innopolis University
[*]Equal contribution.

Abstract

The success of large language models in real-world applications relies on parameter-efficient fine-tuning methods such as Low-Rank Adaptation (LoRA), which inserts compact, trainable adapters into selected layers. In this paper, we propose SimplexLoRA, a LoRA variant that adaptively adjusts adapter ranks during fine-tuning based on their learned importance. Unlike standard LoRA, which employs a fixed rank across all layers, SimplexLoRA optimizes per-adapter importance weights on the n-scaled simplex during a short warm-up and then freezes them, dynamically allocating capacity where it is most beneficial without increasing the overall parameter budget or inference cost. We evaluate SimplexLoRA across 12 language understanding, summarization, and reasoning tasks using DeBERTaV3, BART-large, Llama 3.1 8B, and Qwen3-8B. Our results show that SimplexLoRA consistently matches or outperforms standard LoRA and recent adaptive baselines, with the largest gains at low ranks. Our code is available at https://github.com/brain-lab-research/SimplexLoRA.

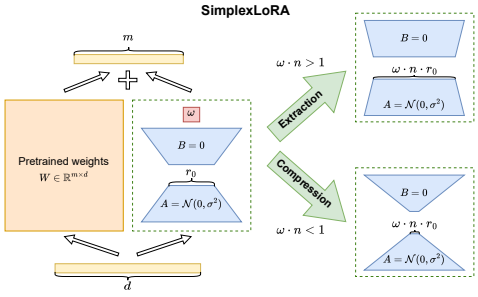


Figure 1: SimplexLoRA framework. Per-adapter importance weights are constrained to the n-scaled simplex, enabling dynamic allocation of a fixed rank budget across layers.

1 Introduction

LoRA (Hu et al., 2021) is a widely used parameter-efficient fine-tuning method (Han et al., 2024) that parameterizes weight updates as $\boldsymbol{W}' = \boldsymbol{W}^0 + \boldsymbol{AB}$, with $\boldsymbol{W}^0 \in \mathbb{R}^{m \times d}$ frozen and trainable $\boldsymbol{A} \in \mathbb{R}^{m \times r}, \boldsymbol{B} \in \mathbb{R}^{r \times d}$, $r \ll m, d$. This approach achieves low inference overhead and substantially reduces memory during fine-tuning.

Despite its efficiency, standard LoRA uses a fixed rank across all layers, ignoring layer-wise differences in convergence and representation learning. Existing adaptive rank allocation methods attempt to address this issue but suffer from the critical drawbacks: (i) substantially more trainable parameters (Sheng et al., 2025), (ii) non-negligible inference-time overhead caused by dynamic pruning or gating mechanisms (Zhang et al., 2025), and (iii) deviations from the standard LoRA pipeline that hinder compatibility (Zhang et al., 2023b). See Section 2 for further discussion.

To address this limitation, we propose SimplexLoRA, which assigns an importance weight to each of the n LoRA adapters, constrained to the n-scaled simplex to enforce a fixed parameter budget. Key properties: (i) ranks learned automatically during fine-tuning; (ii) simplex constraint preserves the LoRA parameter budget; (iii) full LoRA compatibility; (iv) negligible training and inference overhead.

2 Related Work

2.1 Classical low-rank approaches

While LoRA has popularized parameter-efficient fine-tuning, it is not the only method to explore low-rank or structured adaptations. Earlier approaches include LoHA (Nam et al., 2022), which utilizes element-wise (Hadamard) interactions between low-rank matrices to reduce the parameter count without sacrificing expressivity, and Krona

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 51747–51756
October 24-29, 2026 ©2026 Association for Computational Linguistics

(Edalati et al., 2022), which employs the Kronecker product in place of the conventional low-rank matrix product to improve parameter efficiency while maintaining model capacity.

Subsequent methods extended or refined the LoRA framework. VERA (Kopiczko et al., 2023) reduces adapter storage by sharing frozen random projection matrices across layers and learning small scaling vectors. rsLoRA (Kalajdzievski, 2023) replaces the original LoRA scaling factor α/r (Hu et al., 2021) with the rank-stabilized factor $\alpha/\sqrt{r}$. A complementary direction is taken by DoRA (Liu et al., 2024), which decomposes the adapted weight into magnitude and direction components and learns the magnitude together with a low-rank correction of the direction.

Another line of work improves LoRA optimization and numerical efficiency. LoRA+ (Hayou et al., 2024) uses different learning rates for the two adapter matrices, while QLoRA (Dettmers et al., 2023) enables memory-efficient fine-tuning through 4-bit quantization of frozen base-model weights. Both are orthogonal to our adaptive rank allocation method and can be combined with it.

2.2 Dynamic LoRA rank allocation

This topic is of particular relevance in the context of our work, as we focus on adaptive rank learning, enabling the model to automatically adjust the capacity of each adapter during fine-tuning. It has attracted significant attention in recent research (Zhang et al., 2023a,b; Sheng et al., 2025), as exploiting dynamically changing ranks allows models to utilize newly allocated parameters more efficiently.

The SVD-like representation of LoRA adapters proposed in AdaLoRA (Zhang et al., 2023b) replaces the standard two-matrix adapter with the parameterization $\boldsymbol{P}\Lambda\boldsymbol{Q}$ and enables the construction of importance scores for individual singular components. Components outside the global importance budget are masked by setting the corresponding entries of Λ to zero. To encourage the SVD-like structure, the method applies the orthogonality regularizer

$$R(\boldsymbol{P},\boldsymbol{Q}) = \|\boldsymbol{P}^\top\boldsymbol{P}-\boldsymbol{I}\|_F^2 + \|\boldsymbol{Q}^\top\boldsymbol{Q}-\boldsymbol{I}\|_F^2.$$

During training, all three matrices $\boldsymbol{P}$, Λ, and $\boldsymbol{Q}$ are updated simultaneously. Although this parameterization avoids explicitly computing an SVD at every fine-tuning step, maintaining an initial rank above the target budget, regularly evaluating importance scores, and applying orthogonality regularization increase the training complexity compared with standard LoRA.

Several methods manage the rank budget more flexibly. ElaLoRA (Chang et al., 2025) uses SVD-based adaptation and gradient-derived importance scores to prune and expand ranks at scheduled intervals. FlexLoRA (Liu et al., 2026) uses normalized spectral entropy to guide rank pruning and expansion under a global budget. The dynamic-rank DoRA of Mao et al. (2024) decomposes adapters into structured rank-one components and prunes them based on per-component importance. NormAL LoRA (Aastik et al., 2025) uses rank-norm regularization to determine an appropriate rank for each module.

In a similar way, Sheng et al. (2025) proposes a rank-one LoRA series representation of the increment matrix:

$$\Delta\boldsymbol{W} = \boldsymbol{b}_1\boldsymbol{a}_1 + \ldots + \boldsymbol{b}_p\boldsymbol{a}_p = \begin{bmatrix}\boldsymbol{B}_{p-1} & \boldsymbol{b}_p\end{bmatrix}\begin{bmatrix}\boldsymbol{A}_{p-1}\\ \boldsymbol{a}_p\end{bmatrix},$$

where p denotes the number of outer-loop iterations. This representation is applied to several layers, and after each outer-loop step a convergence criterion is evaluated: if the relative change between the $(p-1)$-th and p-th solutions is sufficiently small, the training of $\Delta\boldsymbol{W}$ is terminated. Unlike IncreLoRA, which expands modules according to gradient-based importance scores, AROMA uses the relative Frobenius-norm change between consecutive solutions as a stopping criterion.

Compared with these works, SimplexLoRA does not force adapter ranks to strictly increase or rely on expensive SVD computations. Instead, we allow the ranks to be learned as continuous parameters, enabling the model to respond dynamically to the task's complexity. By leveraging simplex projection and weight freezing, we obtain a direct form of rank adaptation that avoids explicit importance scoring and iterative pruning.

3 SimplexLoRA Framework

For the LoRA method, we introduce trainable layer-wise scaling factors: $\boldsymbol{W}_i^0 + \boldsymbol{A}_i\boldsymbol{B}_i \rightarrow \boldsymbol{W}_i^0 + \omega_i\boldsymbol{A}_i\boldsymbol{B}_i$, where $\omega_i \geq 0$ represents the importance coefficient assigned to the i-th adapter. The weights $\omega = (\omega_1, \ldots, \omega_n)$ are constrained to the n-scaled simplex, where n is the number of LoRA adapters. This preserves LoRA magnitude, with

$\omega = (1, \ldots, 1) = \mathbf{1}$ recovering standard LoRA initialization (Hu et al., 2021).

Motivated by the theoretical analysis in Appendix A, SimplexLoRA formulates the following alternating optimization problem:

$$\min_{\omega \in n \cdot \Delta_{n-1}} \min_{\mathcal{A},\mathcal{B}} \mathcal{L}(\omega, \mathcal{A}, \mathcal{B}) + (\lambda/2)\|\omega - \mathbf{1}\|_2^2, \quad (1)$$

where $\mathcal{A}, \mathcal{B} = \{\boldsymbol{A}_i\}_{i=1}^n, \{\boldsymbol{B}_i\}_{i=1}^n$ denote the sets of all LoRA adapters, $\mathcal{L}$ is the loss, and the term $(\lambda/2)\|\omega - \mathbf{1}\|_2^2$ prevents overfitting of the importance weights ω.

3.1 SimplexLoRA Training Process

The SimplexLoRA algorithm employs an alternating optimization scheme with three phases, repeated for K iterations.

Phase 1. Warm-up. For T steps, we perform standard LoRA fine-tuning by jointly optimizing the adapter parameters $\mathcal{A}, \mathcal{B}$ and the importance weights ω using gradient-based methods.

Phase 2. Simplex projection. The weight vector ω is projected onto the scaled simplex $n \cdot \Delta_{n-1}$ via Euclidean projection (Blondel et al., 2014):

$$\min_{\hat{\omega} \in n \cdot \Delta_{n-1}} \|\omega - \hat{\omega}\|_2^2.$$

Phase 3. Rank update. Adapter ranks are dynamically resized while preserving the total parameter budget: $r_i^{\text{new}} = \lfloor r_0 \cdot \omega_i \rfloor$, where r_0 is the initial (uniform) LoRA rank. When ranks change, the corresponding adapter matrices are resized and reinitialized following the standard LoRA scheme (see Appendix B).

Repeat Phases 1–3 for K iterations.

Phase 4. Standard LoRA training with new ranks. After K repetitions, we fix ω to $\mathbf{1}$ and reinitialize the adapter matrices $\mathcal{A}$ and $\mathcal{B}$ (Appendix B). In this sense Phases 1–3 act as a short rank-search procedure: their role is to determine the per-layer rank allocation, while the warm-up adapter weights are intentionally discarded so that the final stage is functionally identical to standard LoRA with a non-uniform rank profile. Standard training then resumes with these fixed ranks, and any LoRA-based optimization algorithm can be applied.

Scheduler. To ensure that the model resumes training after freezing the weights, we employ a linear scheduler with two training phases (see Figure 2). The first phase includes rank training with reduced learning rate. Once rank training is complete, the learning rate is reset to zero, and the warm-up begins to train the resulting LoRA layers. Note, however, that the learning rate for ω is fixed during training, the scheduler exclusively adjusts the LoRA weights. As a result, the subsequent fine-tuning phase is functionally identical to the standard LoRA training process.

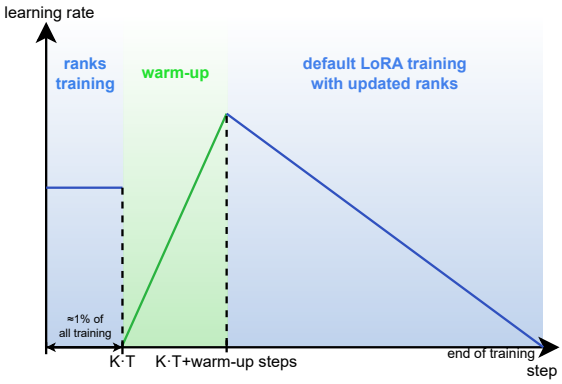


Figure 2: Scheduler for SimplexLoRA parameters.

Discussion. The total number of rank updates, $K \cdot T$, is kept small to avoid rank collapse and to ensure that the majority of training focuses on optimizing the LoRA adapters. In practice, T is negligible compared to the overall training duration. In the ablation study (Section 4.3), we analyze the sensitivity of SimplexLoRA to both hyperparameters K and T, evaluating their impact on performance.

3.2 Motivating Example of Rank Scaling

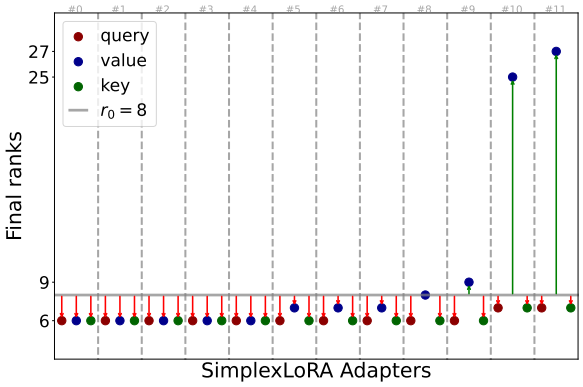


Figure 3: Illustration of the rank distribution of SimplexLoRA.

Figure 3 shows the final rank distribution from fine-tuning DeBERTaV3$_{\text{base}}$ (He et al., 2021) on the MRPC dataset from GLUE (Wang et al., 2018),

Table 1: Comparative performance of "out-of-the-box" algorithms on DeBERTaV3$_{\text{base}}$. The best results are shown in **bold**, and the second-best are underlined. All experiment details are provided in Appendix D.

Method	# Params	MNLI Acc	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc/F1	ALL Avg
`Full Fine-Tuning`	184M (100%)	0.8910	0.9541	0.6806	0.8962/0.8644	**0.9383**	0.8376	0.8848/0.9165	0.8689
`LoRA`$_{r=8}$ (Hu et al., 2021)	442K (0.24%)	0.8797	0.9450	**0.6913**	0.8802/0.8437	0.9301	**0.8448**	0.8897/0.9220	0.8696
`LoHA`$_{r=8}$ (Nam et al., 2022)	884K (0.48%)	0.8560	0.9392	0.6295	0.8674/0.8269	0.9085	0.8051	0.8628/0.9007	0.8440
`Krona`$_{r=32}$ (Edalati et al., 2022)	57.6K (0.03%)	0.7587	0.9289	0.5907	0.8275/0.7816	0.8605	0.7509	0.7500/0.8416	0.7878
`VERA`$_{r=1024}$ (Kopiczko et al., 2023)	1.64M (0.88%)	0.8355	0.9404	0.6252	0.8592/0.8203	0.8975	0.7581	0.8529/0.8962	0.8317
`rsLoRA`$_{r=8}$ (Kalajdzievski, 2023)	442K (0.24%)	0.8814	0.9564	0.6414	0.8853/0.8506	0.9325	0.8267	0.8750/0.9122	0.8624
`DoRA`$_{r=8}$ (Liu et al., 2024)	442K (0.24%)	0.8811	0.9473	0.6748	0.8809/0.8451	0.9295	0.7978	0.8799/0.9139	0.8611
`AdaLoRA`$_{r=8}$ (Zhang et al., 2023b)	664K (0.36%)	0.8390	0.9392	0.6222	0.8534/0.8083	0.8999	0.7834	0.8859/0.9178	0.8388
`NormAL LoRA`$_{r=8}$ (Aastik et al., 2025)	442K (0.24%)	0.8814	0.9328	0.6857	0.8768/0.8403	0.9282	0.8292	0.8870/0.9195	0.8645
`ElaLoRA`$_{r=8}$ (Chang et al., 2025)	442K (0.24%)	0.8853	0.9501	0.6638	0.8979/0.8628	0.9146	0.8031	0.8782/0.9113	0.8630
`FlexLoRA`$_{r=8}$ (Liu et al., 2026)	664K (0.36%)	0.8711	0.9557	0.6331	0.8684/0.8658	0.9343	0.8392	**0.9014**/0.9217	0.8656
`SimplexLoRA`$_{r=8}$	442K (0.24%)	**0.8984**	**0.9587**	0.6572	**0.9124/0.8844**	0.9362	0.8339	0.8987/**0.9289**	**0.8788**

Table 2: Comparative results of top-performing algorithms: `LoRA` (L) and `SimplexLoRA` (S) on DeBERTaV3$_{\text{base}}$ and Llama 3.1 8B. Results are averaged over 3 runs with different seeds. The best results per rank are shown in **bold**. All experiment details are provided in Appendix D.

Model	Rank	Method	MNLI Acc	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc/F1	ALL Avg
DeBERTaV3$_{\text{base}}$	$r=1$	L	0.8912	0.9488	0.6558	0.8905/0.8544	0.9250	0.7943	0.8905/0.9213	0.8562
		S	**0.8913**	**0.9576**	**0.6775**	**0.9018/0.8709**	**0.9317**	**0.8159**	**0.9093/0.9359**	**0.8690**
	$r=2$	L	**0.9004**	0.9413	0.6600	0.9077/0.8631	**0.9369**	0.8068	0.9003/0.9193	0.8629
		S	0.8946	**0.9608**	**0.6658**	**0.9178/0.8842**	0.9338	**0.8252**	**0.9103/0.9314**	**0.8717**
	$r=4$	L	0.8780	0.9562	**0.6809**	0.9153/0.8866	**0.9582**	0.7816	**0.9207/0.9471**	0.8700
		S	**0.9002**	**0.9610**	0.6807	**0.9244/0.8903**	0.9358	**0.8259**	0.9044/0.9322	**0.8756**
	$r=8$	L	0.9030	**0.9641**	0.6965	0.9092/0.8723	**0.9449**	**0.8537**	0.9064/0.9127	0.8804
		S	**0.9047**	0.9619	**0.6978**	**0.9129/0.8901**	0.9447	0.8305	**0.9267/0.9475**	**0.8826**
Llama 3.1 8B	$r=1$	L	**0.8961**	0.9751	**0.7015**	**0.8891**/0.8370	0.9411	0.9016	0.9154/**0.9408**	0.8886
		S	0.8861	**0.9801**	0.6829	**0.8891/0.8379**	**0.9431**	**0.9204**	**0.9278**/0.9404	**0.8898**
	$r=2$	L	0.8996	0.9751	0.6967	**0.8851**/0.8311	**0.9421**	0.9154	0.9204/0.9441	0.8900
		S	**0.8998**	**0.9801**	**0.6981**	**0.8851/0.8316**	0.9381	**0.9353**	**0.9254/0.9466**	**0.8933**
	$r=4$	L	0.8993	0.9800	0.6902	0.8851/0.8286	0.9391	0.9154	**0.9254/0.9462**	0.8899
		S	**0.9004**	**0.9823**	**0.7069**	**0.8891/0.8325**	**0.9421**	**0.9304**	0.9204/0.9429	**0.8941**
	$r=8$	L	0.8995	**0.9851**	0.6928	0.8851/0.8306	0.9421	0.9104	0.9154/0.9419	0.8892
		S	**0.9016**	**0.9851**	**0.7058**	**0.8881/0.8343**	**0.9431**	**0.9353**	**0.9204/0.9429**	**0.8952**

tuning only self-attention layers. The learned distribution reveals a clear pattern: value projection matrices in self-attention layers consistently receive the highest ranks, indicating their critical role. Rather than eliminating ranks of adapters with low weights, the method reduces their magnitude, which helps mitigate underfitting.

4 Experiments

We evaluate `SimplexLoRA` on a diverse set of benchmarks, including GLUE (Wang et al., 2018), XSum (Narayan et al., 2018), CNN/Daily Mail (Chen et al., 2016), GSM8K (Cobbe et al., 2021), BoolQ (Clark et al., 2019), and HellaSwag (Zellers et al., 2019). We exclude GLUE STS-B, whose regression metric requires custom evaluation and which overlaps with the QQP/MRPC similarity tasks. Experiments are conducted using DeBERTaV3$_{\text{base}}$ (He et al., 2021), Llama 3.1 8B (AI@Meta, 2024), Qwen3-8B (Yang et al., 2025), and BART-large (Lewis et al., 2020) models. Hardware and hyperparameters are provided in Appendix D.

4.1 Out-of-the-box Comparison

We benchmark against baselines in default configurations under a number of training iterations. These span foundational PEFT methods (a performance floor) and recent dynamic-rank methods comparable to `SimplexLoRA`:

Standard PEFT methods: LoRA (Hu et al., 2021), LoHA (Nam et al., 2022), Krona (Edalati et al., 2022), rsLoRA (Kalajdzievski, 2023), DoRA (Liu et al., 2024), and VERA (Kopiczko et al., 2023).

Dynamic rank methods: AdaLoRA (Zhang et al., 2023b), NormAL LoRA (Aastik et al., 2025), ElaLoRA (Chang et al., 2025), and FlexLoRA (Liu et al., 2026).

In this setting (Table 1), where adapters are applied only to self-attention layers, full fine-tuning, standard LoRA, and SimplexLoRA emerge as the most robust methods, with SimplexLoRA achieving the best average score at the same 0.24% parameter budget as standard LoRA. Other approaches require extensive hyperparameter tuning to achieve competitive performance. Consequently, we select standard LoRA as our primary baseline for further detailed comparison due to its widespread adoption and stability.

Table 3: Comparative results: LoRA (L) and SimplexLoRA (S) on Qwen3-8B and BART-large. The best results per rank and model are shown in **bold**. All details are provided in Appendix D.

Rank	Method	Qwen3-8B				BART-large	
		GSM8K Acc	BoolQ Acc	HellaSwag Acc	ALL Avg	XSum ROUGE-1	CNN/DailyMail ROUGE-1
$r=1$	L	0.8522	0.655	0.772	0.760	0.372	0.245
	S	**0.8666**	**0.680**	**0.775**	**0.774**	**0.373**	**0.249**
$r=2$	L	**0.8681**	0.662	0.776	0.769	0.374	0.250
	S	0.8673	**0.682**	**0.791**	**0.780**	**0.376**	**0.254**
$r=4$	L	0.8537	0.691	0.777	0.774	0.375	0.255
	S	**0.8719**	**0.700**	**0.780**	**0.784**	**0.378**	**0.256**
$r=8$	L	0.8681	0.691	0.782	0.780	**0.380**	0.257
	S	**0.8741**	**0.699**	**0.790**	**0.788**	0.379	**0.259**

4.2 Comparison with LoRA

Table 2 compares SimplexLoRA with standard LoRA at multiple ranks on DeBERTaV3$_{\text{base}}$ and Llama 3.1 8B (adapters in all linear layers, hyperparameters tuned per method). Across most GLUE tasks, SimplexLoRA consistently outperforms LoRA while maintaining the same computational cost. For generality, we also evaluate Qwen3-8B (GSM8K, BoolQ, HellaSwag) and BART-large (XSum, CNN/DailyMail) in Table 3. As on GLUE, SimplexLoRA matches or outperforms LoRA on most of these tasks, confirming the trend across architectures and task types. The improvement is most pronounced at low ranks ($r \in \{1, 2\}$), where the tight capacity budget of standard LoRA gives adaptive allocation the most room to help.

4.3 Ablation Study

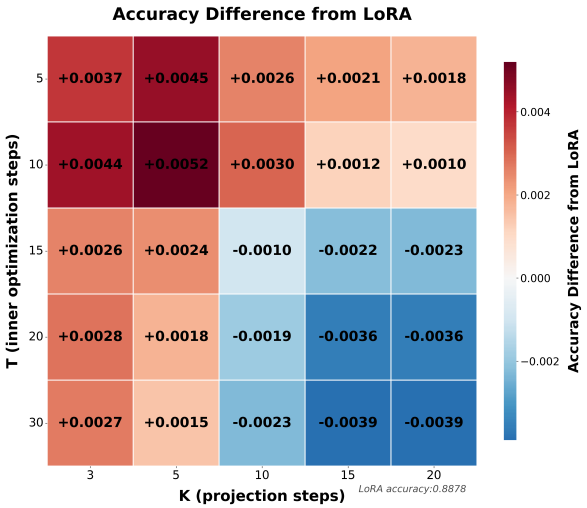


Figure 4: The influence of hyperparameters on SimplexLoRA performance.

As supplementary analysis, we grid-search hyperparameters on MNLI with DeBERTa to assess the robustness (Figure 4). Both methods are trained for 2048 steps using the best configuration from the main experiments; each cell reports the validation-accuracy difference.

The results show that SimplexLoRA is stable under a limited optimization budget ($K \cdot T < 100$, i.e., less than 5% of total training). Very small values (e.g., $K = 3, T = 5$) lead to underfitting due to insufficient rank adaptation, while larger values (e.g., $K \geq 10$ or $T \geq 15$) tend to cause rank overfitting. Beyond these thresholds, we find that increasing the number of projection steps and inner iterations results in a performance plateau, with values showing minimal deviation from the average. This confirms the importance of keeping $K \cdot T$ low to maintain efficiency without performance loss. In all experiments, we use $K = 5$ and $T = 10$, which consistently yields strong performance.

5 Conclusion

We presented SimplexLoRA, a LoRA variant that allocates a fixed rank budget non-uniformly across layers via short alternating optimization on the n-scaled simplex. Across 12 tasks and four backbones up to 8B parameters, SimplexLoRA matches or outperforms standard LoRA and recent adaptive baselines at the same parameter count and inference cost, with the largest gains in the low-rank regime.

Limitations

The limitations of the proposed strategy are as follows. First, SimplexLoRA introduces three additional hyperparameters: the number of iterations between simplex projection steps T, the total number of such projections K, and the weight decay coefficient used for regularization; the ablation in Section 4.3 indicates that the method is robust within a broad range but does not eliminate the need to set these values. Second, the lack of experiments on models substantially larger than 8B parameters limits the generalizability of our findings. Third, while the per-task averages over three seeds consistently favor SimplexLoRA, on a few individual tasks (e.g., CoLA and RTE in the out-of-the-box DeBERTaV3 comparison) standard LoRA remains competitive or slightly better, indicating that the benefit of adaptive rank allocation is task-dependent. Fourth, a fundamental ambiguity remains during training: a small gradient norm in a layer may indicate either that the weights are well-optimized or that the gradients have vanished by the time they reach that layer, which can in principle bias the learned importance weights.

On the ethical side, we see no specific risks beyond those already inherent to fine-tuning large language models: this work studies a parameter-efficient adaptation procedure, introduces no new data sources, uses only publicly available benchmarks with no proprietary or personal data and no human subjects, and reduces rather than increases the compute and memory footprint of fine-tuning.

Acknowledgments

The work of A. Beznosikov was supported by the Russian Science Foundation (project no. 21-71-30005-P).

References

Aastik, Meghana Topu, Chinmay Kulkarni, and Pragya Paramita Sahu. 2025. NormAL LoRA: What is the perfect size? In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 19716–19731, Suzhou, China. Association for Computational Linguistics.

AI@Meta. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Mathieu Blondel, Akinori Fujino, and Naonori Ueda. 2014. Large-scale multiclass support vector machine training via euclidean projection onto the simplex. In *2014 22nd International Conference on Pattern Recognition*, pages 1289–1294.

Huandong Chang, Zicheng Ma, Mingyuan Ma, Zhenting Qi, Andrew Sabot, Hong Jiang, and H. T. Kung. 2025. ElaLoRA: Elastic and learnable low-rank adaptation for efficient model fine-tuning. *arXiv preprint arXiv:2504.00254*.

Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the CNN/Daily Mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2358–2367.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

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

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

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

Ilyas Fatkhullin, Jalal Etesami, Niao He, and Negar Kiyavash. 2022. Sharp analysis of stochastic optimization under global kurdyka-lojasiewicz inequality. *Advances in Neural Information Processing Systems*, 35:15836–15848.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Oliver Hinder, Aaron Sidford, and Nimit Sohoni. 2020. Near-optimal methods for minimizing star-convex functions and beyond. In *Conference on learning theory*, pages 1894–1938. PMLR.

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

Damjan Kalajdzievski. 2023. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*.

Ahmed Khaled, Othmane Sebbouh, Nicolas Loizou, Robert M Gower, and Peter Richtárik. 2023. Unified analysis of stochastic gradient methods for composite convex and smooth optimization. *Journal of Optimization Theory and Applications*, 199(2):499–540.

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. 2018. An alternative view: When does sgd escape local minima? In *International conference on machine learning*, pages 2698–2707. PMLR.

Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2023. VeRA: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. Association for Computational Linguistics.

Muqing Liu, Chongjie Si, and Yuheng Jia. 2026. FlexLoRA: Entropy-guided flexible low-rank adaptation. In *International Conference on Learning Representations*.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. 2024. DoRA: Enhancing parameter-efficient fine-tuning with dynamic rank distribution. *arXiv preprint arXiv:2405.17357*.

Hyeon-Woo Nam, Ye-Bin Moon, and Tae-Hyun Oh. 2022. FedPara: Low-rank hadamard product for communication-efficient federated learning. In *International Conference on Learning Representations*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807.

Hao Nan Sheng, Zhi yong Wang, Mingrui Yang, and Hing Cheung So. 2025. Aroma: Autonomous rank-one matrix adaptation. *Preprint*, arXiv:2504.05343.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP*, pages 353–355. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Dacao Zhang, Kun Zhang, Shimao Chu, Le Wu, Xin Li, and Si Wei. 2025. More: A mixture of low-rank experts for adaptive multi-task learning. *arXiv preprint arXiv:2505.22694*.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *Preprint*, arXiv:2308.12043.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Yi Zhou, Junjie Yang, Huishuai Zhang, Yingbin Liang, and Vahid Tarokh. 2019. Sgd converges to global minimum in deep learning via star-convex path. *arXiv preprint arXiv:1901.00451*.

A SimplexLoRA: theoretical motivation

This section presents a heuristic argument that motivates the simplex-constrained reparameterization of Section 3. We do not claim a formal optimization guarantee in the non-convex setting; the goal is to clarify why importance weights on a simplex are a natural device for redistributing adapter capacity.

Consider a loss function $\mathcal{L}(\boldsymbol{W}^1, \ldots, \boldsymbol{W}^n)$ for a machine learning model with n layers. In the context of fine-tuning an LLM using the LoRA algorithm, the optimization problem can be formulated as

$$\min_{\mathcal{A},\mathcal{B}} \left\{ \mathcal{L}(\mathcal{A}, \mathcal{B}) := \mathcal{L}(\boldsymbol{A}^1\boldsymbol{B}^1, \ldots, \boldsymbol{A}^n\boldsymbol{B}^n) \right\},$$

where $\mathcal{A}$ and $\mathcal{B}$ denote the sets of all LoRA adapters. To generalize this formulation, we introduce a non-negative scaling factor $\omega_i \geq 0$ for each layer i, which can be interpreted as rescaling the adapter matrix $\boldsymbol{A}^i$ by ω_i. For the purposes of the present derivation, we constrain the vector $\omega := (\omega_1, \ldots, \omega_n)$ to lie on the probability simplex Δ_{n-1}, i.e., $\sum_{i=1}^n \omega_i = 1$; the n-scaled simplex used in the algorithm (Section 3) is the same object up to a factor of n, chosen so that the uniform vector $\omega = \mathbf{1}$ recovers standard LoRA. The resulting optimization problem is then

$$\min_{\mathcal{A},\mathcal{B}} \left\{ \mathcal{L}(\omega_1\boldsymbol{A}^1\boldsymbol{B}^1, \ldots, \omega_n\boldsymbol{A}^n\boldsymbol{B}^n) \right\}.$$

Note that this reparameterization does not alter the expressive power of the model, as the scaling factors can be absorbed into the adapter matrices.

A classical simplifying assumption in the optimization literature is that $\mathcal{L}$ is convex (Khaled et al., 2023) with respect to all parameters $\{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^n\}$. For deep networks this is generally false, but convex analysis is still commonly used as a guiding lens: empirical studies report local convexity along training trajectories (Kleinberg et al., 2018; Zhou et al., 2019), and many non-convex results build on convex foundations (Hinder et al., 2020; Fatkhullin et al., 2022). We adopt this assumption only to derive the upper bound below; the resulting prescription on ω is then transferred back to the non-convex setting as a design heuristic, not as a guarantee.

Therefore, if we take $\mathcal{A}^i \cdot \mathcal{B}^i := \{0, \ldots, \boldsymbol{A}^i\boldsymbol{B}^i, \ldots, 0\}$, that is, we fine-tune only the i-th layer, then, by the convexity of $\mathcal{L}$ (Jensen's inequality), we obtain:

$$\begin{aligned}
&\min_{\mathcal{A},\mathcal{B}} \{\mathcal{L}(\mathcal{A}, \mathcal{B})\} \\
&= \min_{\mathcal{A},\mathcal{B}} \left\{ \mathcal{L}(\omega_1\boldsymbol{A}^1\boldsymbol{B}^1, \ldots, \omega_n\boldsymbol{A}^n\boldsymbol{B}^n) \right\} \\
&= \min_{\mathcal{A},\mathcal{B}} \left\{ \mathcal{L}\left(\sum_{i=1}^n \omega_i \cdot \{0, \ldots, \boldsymbol{A}^i\boldsymbol{B}^i, \ldots, 0\} \right) \right\} \\
&\leq \min_{\mathcal{A},\mathcal{B}} \left\{ \sum_{i=1}^n \omega_i \mathcal{L}\left(\mathcal{A}^i \cdot \mathcal{B}^i \right) \right\} \\
&= \sum_{i=1}^n \omega_i \min_{\boldsymbol{A}^i, \boldsymbol{B}^i} \left\{ \mathcal{L}(0, \ldots, \boldsymbol{A}^i\boldsymbol{B}^i, \ldots, 0) \right\}.
\end{aligned}$$

Thus, the original fine-tuning problem is upper-bounded by a weighted sum of n single-layer fine-tuning problems. Minimizing this upper bound over $\omega \in \Delta_{n-1}$ allocates weight to the layer with the smallest single-layer minimum: the i-th coordinate of the resulting ω exceeds the j-th if and only if

$$\min_{\mathcal{A}^i,\mathcal{B}^i} \mathcal{L}(\mathcal{A}^i \cdot \mathcal{B}^i) \leq \min_{\mathcal{A}^j,\mathcal{B}^j} \mathcal{L}(\mathcal{A}^j \cdot \mathcal{B}^j),$$

since otherwise swapping their positions would yield a smaller upper bound. Reading this in reverse, components of ω rank the layers by their single-layer "fitting potential". SimplexLoRA uses this ranking as a proxy for layer importance and reallocates adapter rank accordingly: well-adapted layers receive higher ranks, weaker ones lower. We emphasize that this argument bounds the loss, not the rank distribution itself, so the connection to optimal ranks remains a heuristic; its empirical effectiveness is reported in Section 3 and the ablation in Section 4.3.

Note that to prevent ω from degenerating into a one-hot vector, which is a common issue in simplex-constrained optimization where a single component dominates, one can apply regularization. In our implementation, we use a standard Euclidean regularization term to encourage ω to remain closer to a uniform distribution across all components.

B Adapter Reinitialization

To ensure that the reinitialized adapters do not introduce immediate perturbations to the model's output, we follow the standard LoRA initialization scheme.

- $\boldsymbol{A}^i$ initialized using the Kaiming uniform distribution (He et al., 2015) to preserve the variance of activations: $\boldsymbol{A}^i \sim U\left(-d_{in}^{-0.5}, d_{in}^{-0.5}\right)$,

- $\boldsymbol{B}^i$ initialized to zero.

This combination ensures that the initial product $\boldsymbol{A}^i\boldsymbol{B}^i = \mathbf{0}$, maintaining the model's performance at the start of the final training stage while providing a stable gradient flow for the new parameters.

C Simplex Projections

While softmax and weighted softmax transformations are commonly used for projecting onto the simplex, we observed severe performance degradation during training when applying these methods. Their behavior is highly sensitive to temperature scaling: low temperatures tend to produce nearly one-hot distributions, whereas high temperatures result in almost uniform outputs. Such degeneracies are particularly problematic in our setting, where the goal is to efficiently allocate the rank budget across multiple adapters without inducing excessive sparsity or uniformity. For these reasons, we utilize the standard Euclidean regularization in the optimization problem formulation equation 1, rather than alternatives such as KL divergence.

Table 4: Hyperparameters for out-of-the-box comparison (Table 1).

Hyperparameter	**Value**
Optimizer	AdamW
β_1, β_2	0.9, 0.999
Model learning rate	Tuned (PEFT)
Learning rate (ω)	5
LR warm-up	30
Reg. weight decay	0
Dropout	0.05
Grad. accumulation	6
Training budget	512 iterations
Batch size	32
Scheduler	Linear
Target layers	Query, Key, Value

D Experimental Details

The core algorithm is implemented in PyTorch using the Huggingface library. All experiments are conducted on NVIDIA A100-SXM4-80GB GPUs. Following standard `LoRA` practice, we use the scaling factor α/r with $\alpha = 32$ and dynamically updated ranks r_i^{new} (Hu et al., 2021). To support simplex projection and dynamic rank reallocation, we employ a modified AdamW optimizer (Loshchilov and Hutter, 2019) with specialized update steps for ω, rank resizing, and a custom learning rate schedule.

Table 5: Hyperparameters for DeBERTaV3$_{\text{base}}$ experiments (Table 2).

Hyperparameter	**Value / Grid**
Optimizer	AdamW
β_1, β_2	0.9, 0.999
Model learning rate	$\{3, 5, 8, 10, 30, 50, 80, 100\} \times 10^{-5}$
Learning rate (ω)	5
LR warm-up	$\{0, 20, 30, 50, 70, 100, 200\}$
Reg. weight decay	$\{0, 10^0, 10^{-1}, \ldots, 10^{-6}\}$
Dropout	0.05
Grad. accumulation	6
Training budget	task-specific
Batch size	task-specific
Scheduler	Two-phase (Linear)
Target layers	All linear layers
Task-specific (# epochs / batch size)	
MNLI / SST-2 / QNLI / QQP	(7 / 32), (10 / 32), (5 / 32), (5 / 32)
CoLA / RTE / MRPC	(15 / 64), (50 / 64), (30 / 64)

Table 6: Hyperparameters for Llama 3.1 8B experiments (Table 2).

Hyperparameter	**Value / Grid**
Optimizer	AdamW
β_1, β_2	0.9, 0.999
Model learning rate	$\{3, 5, 8, 10, 30, 50, 80, 100\} \times 10^{-5}$
Learning rate (ω)	5
LR warm-up	50
Reg. weight decay	$\{0, 10^0, 10^{-1}, \ldots, 10^{-6}\}$
Dropout	0.05
Grad. accumulation	1
Training budget	1024 iterations
Batch size	16
Scheduler	Two-phase (Cosine)
Target layers	All linear layers

Training and optimization hyperparameters for each experimental setup are summarized in Tables 4, 5, 6, 7, and 8. The LR warm-up values reported in these tables refer to the learning-rate scheduler and are not `SimplexLoRA` rank-search hyperparameters. For the first table, the learning rate was either adopted from the original publications or selected via a small grid search. For the DeBERTa, Llama, and Qwen setups, the optimal configuration for each task was determined via an independent grid search over the specified hyperparameter ranges. For GSM8K, we use few-shot chain-of-thought prompting with flexible answer extraction during evaluation. All the results were averaged across 3 random seeds.

Table 7: Hyperparameters for Qwen3-8B experiments (Table 3).

Hyperparameter	**Value / Grid**
Optimizer	AdamW
β_1, β_2	$0.9, 0.999$
Model learning rate	$\{3, 5, 8, 10, 30, 50, 80, 100\} \times 10^{-5}$
Learning rate (ω)	5
LR warm-up	350
Reg. weight decay	$\{0, 10^0, 10^{-1}, \ldots, 10^{-6}\}$
Dropout	0.05
Grad. accumulation	1
Training budget	3500 iterations
Batch size	4
Scheduler	Two-phase (Linear)
Target layers	All linear layers

Table 8: Hyperparameters for BART-large experiments (Table 3).

Hyperparameter	**Value / Grid**
Optimizer	AdamW
β_1, β_2	$0.9, 0.999$
Model learning rate	Tuned (PEFT)
Learning rate (ω)	5
LR warm-up	50
Reg. weight decay	10^{-4}
Model weight decay	10^{-1}
Dropout	0.05
Grad. accumulation	16
Training budget	15 epochs
Batch size	8
Scheduler	Two-phase (Linear)
Beam size	8
Max source / target length	$256/160$
Target layers	All linear layers

E Use of AI Assistants

During the preparation of this work, the authors used a LLM to assist with several tasks. Specifically, the LLM was employed for code generation to create data visualization scripts, debugging existing code, and for proofreading the paper to improve grammatical accuracy and stylistic consistency. After using this tool, the authors reviewed and edited the content as needed for the final publication.