LoRA^2 : Multi-Scale Low-Rank Approximations for Fine-Tuning Large Language Models

Jia-Chen Zhang[1], Yu-Jie Xiong[1*], He-Xi Qiu[1], Dong-Hai Zhu[1], Chun-Ming Xia[1]

[1]School of Electronic and Electrical Engineering, Shanghai University of Engineering Science,
333 Longteng Road, Songjiang District, Shanghai, China

Correspondence: xiong@sues.edu.cn

Abstract

Fine-tuning large language models (LLMs) with high parameter efficiency for downstream tasks has become a new paradigm. Low-Rank Adaptation (LoRA) significantly reduces the number of trainable parameters for fine-tuning. Although it has demonstrated commendable performance, updating parameters within a single scale may not be the optimal choice for complex downstream tasks. in this paper, we extend the LoRA to multiple scales, dubbed as LoRA^2. We first combine orthogonal projection theory to train a set of LoRAs in two mutually orthogonal planes. Then, we improve the importance score algorithm, which reduce parameter sensitivity score calculations by approximately 98.5%. By pruning singular values with lower importance scores, thereby enhancing adaptability to various downstream tasks. Extensive experiments are conducted on two widely used pre-trained models to validate the effectiveness of LoRA^2. Results show that it significantly reduces the number of trainable parameters to just 0.72% compared to full fine-tuning, while still delivering highly impressive performance. Even when the parameters are further reduced to 0.17M, it still achieves comparable results to the baseline with 8 times more parameters. Our code is available here: https://anonymous.4open.science/r/LoRA-2-5B4C

1 Introduction

Large Language Models (LLMs) have become the cornerstone of NLP tasks (Devlin et al., 2019; Liu et al., 2021; He et al., 2021; Radford et al., 2019). The powerful emergent abilities (Wei et al., 2022) enables LLMs to adapt to downstream tasks through fine-tuning. The simplest approach is to fine-tune all the parameters of LLMs (Qiu et al., 2020; Liu et al., 2021). However, as the model's parameters gradually expand, PaLM (Chowdhery et al., 2023) contains up to 540 billion parameters;
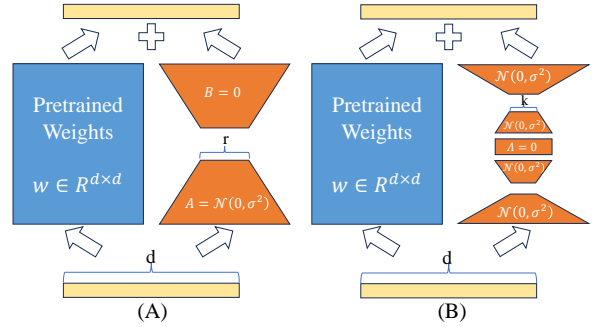

Figure 1: Blue blocks represent frozen parameters, while orange represents trainable parameters. (A) LoRA only utilizes a set of low-rank matrices to approximate increments. (B) LoRA^2 trains a set of low-rank matrices in two mutually orthogonal planes.

GPT-4 (Achiam et al., 2023) contains up to 100 trillion parameters. The massive memory and time resources required to fine-tune all the parameters of these models are completely unacceptable.

To address this issue, LoRA (Hu et al., 2022) propose learning incremental updates to pre-trained weights through the product of two small matrices. LoRA avoids forward propagation latency caused by inserting additional neural modules while demonstrating stable performance. Compared to fine-tuning, only less than 0.5% additional trainable parameters are needed, and training overhead can be reduced by up to 70%. LoRA achieves performance comparable to or even better than fine-tuning. But LoRA still has limitations as it prespecifies the rank r of each incremental matrix identical.

In response to this issue, AdaLoRA (Zhang et al., 2023) dynamically allocates parameter budgets between weight matrices. Sensitivity scores are computed for all parameters of the matrices, which are aggregated into importance scores for singular values. The lowest-ranking singular values are pruned. Such operations enable adaptive manipulation of the rank. Although the performance of AdaLoRA is improved, but results in slower convergence.

---
[*]Corresponding author.

frozen

trainable

matrix multiplication

matrix addition

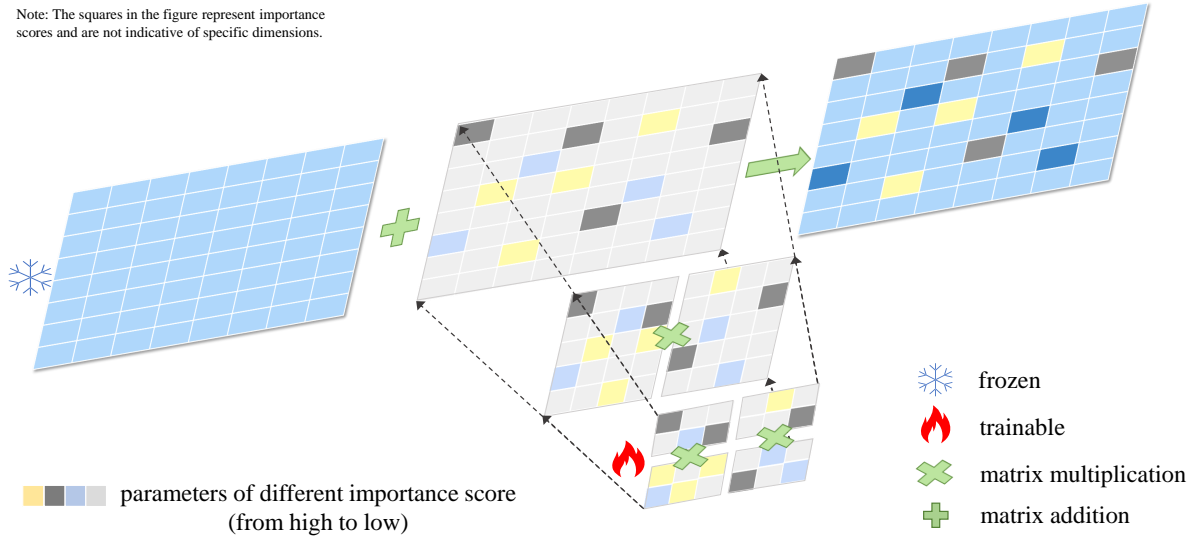parameters of different importance score (from high to low)

Figure 2: An illustration of Multi-Scale Orthogonal Low-Rank Approximations (LoRA^2): Based on the principle of orthogonal projection, We train a set of inherently orthogonal LoRAs on orthogonal planes as the incremental matrices.

In this paper, We first combine orthogonal projection theory to train a set of LoRAs on two different planes and make them orthogonal through dual regularization. The external regularizer minimizes the overlap of LoRA blocks, thereby enlarging the learning space of multi-scale LoRAs. The dual internal regularizers enhance the regularity of parameter updates, accelerating convergence. Similar to AdaLoRA, we adopt dynamic parameter budget allocation method. We extend the computation method of importance scores to adapt to the complex matrices of LoRA^2. Due to the properties of matrix multiplication, where the rows of the preceding matrix must be multiplied by the columns of the succeeding matrix, we add the importance scores of the column matrix to the importance scores of the row matrix. But further discover through reasoning that since the calculation of each singular value's importance score includes the sensitivity scores of all parameters in the column matrix, the sensitivity scores of the column matrix have no impact on parameter pruning. Therefore, we exclude the column matrix calculations when computing importance scores. LoRA^2 adapts to various downstream by pruning with importance scores. Extensive experiments are conducted on various tasks and models to demonstrate the effectiveness of LoRA^2. Specifically, natural language understanding GLUE (Wang et al., 2018) is evaluated using DeBERTaV3-base (He et al., 2023). The findings, including qualitative and quantitative results,

indicate that LoRA^2 outperforms existing methods. The contributions of this paper are as follows:

- We propose LoRA^2, a novel method that trains two internally LoRAs on orthogonal planes. It greatly increases the model's learnable space in a low-rank environment. Compared to full fine-tuning, it reduces the number of trainable parameters to 0.72%.

- We improve the importance score algorithm to accommodate the structure of LoRA^2. It reduce the parameter sensitivity score calculations by approximately 98.5% without causing any degradation in performance. By dynamically allocating parameter budgets based on importance scores, we achieve adaptability across various downstream tasks.

- Extensive experiments are conducted to demonstrate the effectiveness of our method. Particularly, our model could consistently outperform parameter-efficient baselines with fewer parameters on a wide range of downstream tasks.

2 Related Work

Parameter-Efficient Fine-Tuning (PEFT) is a strategy to adapt large-scale pre-trained models effectively and resource-efficiently for specific tasks. In fields such as NLP, pre-trained models like BERT and GPT-3 are highly favored due to their powerful representation learning capabilities. However,

directly fine-tuning these models with all their parameters to fit new tasks often requires significant computational resources and time.

PEFT aims to overcome this challenge by updating only a small subset of crucial parameters in the model, rather than all millions or even billions of parameters. It achieves efficient and precise task transfer. Here are several major PEFT methods:

Adapter Module (Houlsby et al., 2019; Li and Liang, 2021): Inserting compact trainable modules (such as residual blocks) into various layers of the pre-trained model and fine-tuning only these adapters while keeping the original model parameters unchanged. This approach preserves pre-training knowledge while allowing adapters to capture task-specific information.

Prompt Tuning (Sahoo et al., 2024): represents an efficient approach to adapt Transformer-based models to new tasks by optimizing only the prompt vector, rather than adjusting the model weights directly. This method involves inserting a fixed-length trainable vector, known as a prompt, at the input stage, which interacts with the standard fixed inputs to guide the model's output generation in a task-specific manner. The key advantage of prompt tuning lies in its ability to adapt the model to new tasks with minimal disruption to the underlying model architecture. By merely adjusting the prompts, the approach avoids the computationally expensive and potentially destabilizing process of retraining large portions of the model. This results in a significant reduction in the number of parameters that need to be trained, making the process more efficient and scalable.

LoRA (Hu et al., 2022): Fine-tuning by adding low-rank correction terms to the model weight matrices instead of directly updating the entire weight matrix. It utilizes low-rank approximation theory to effectively adjust the model's behavior with smaller parameter increments. formulated as:

$$W = W^{(0)} + \Delta = W^{(0)} + BA, \quad (1)$$

where $\Delta \in \mathbb{R}^{din \times dout}$, $A \in \mathbb{R}^{r \times dout}$, and $B \in \mathbb{R}^{din \times r}$, with $r^{\in (din, dout)}$. The dimensions of din and $dout$ are the same as those of the pre-trained matrix W. During fine-tuning, only A and B are updated. The rank r is chosen to be much smaller than the dimension of W. With less than 0.5% additional trainable parameters, the training overhead can be reduced up to 70%. Therefore, using LoRA to fit the incremental matrix increases

the sparsity of the incremental matrix, and theoretically demonstrates the role of sparsity in model stability. As the compression ratio p decreases, the upper bound also decreases. Ref. (Fu et al., 2023) derives Equation 2 based on the pointwise hypothesis stability (PHS) (Bindel et al., 2002) to prove that using LoRA for fine-tuning implies better stability.

$$E_{S,i \sim U(n)} |\ell(A(S^i), z_i) - \ell(A(S), z_i)|$$
$$\leq \frac{2\rho^2}{(\Lambda_{min} + 2(1-p))n}, \quad (2)$$

$A(S)$ is defined as the model parameters obtained by running algorithm A on data S. $\Lambda_{\min} = \min\{\Lambda_1, \ldots, \Lambda_m\}$. $\ell(\cdot)$ represents the loss function. The variable ρ represents this measure of stability, reflecting the maximum impact of input variations on the output in the loss function. Equation 2 demonstrates that as the sparsity parameter p decreases, the upper bound also decreases. Therefore, sparse models are associated with better stability.

AdaLoRA (Zhang et al., 2023) dynamically allocates parameter budgets among weight matrices based on LoRA, and parametrizes the incremental updates to the pre-trained weight matrices using singular value decomposition:

$$W = W^{(0)} + \Delta = W^{(0)} + P\Lambda Q, \quad (3)$$

where P and Q denote the left and right singular values, and the diagonal matrix Λ contains the top r largest singular values. Initially, Λ is initialized to zero, and P and Q are randomly initialized with Gaussian distribution to ensure $P\Lambda Q = 0$ at the beginning of training. Specifically, the model retains only the top-scoring entries based on their newly proposed metric of sensitivity, heuristically constructed from the product of weight gradients. Although the AdaLoRA method achieved incremental results in benchmark testing, it still exhibits significant redundancy in terms of parameters. The updates to the matrix are irregular, resulting in slow fitting speeds when encountering complex tasks. Here are two unresolved issues: (1) Under low-rank conditions, the model's performance is adversely affected, rendering it unable to adapt effectively to low-resource environments. (2) When $r = 1$, the dynamic allocation of parameter budgets fails to work.

3 Our Method

Our method consists of two main parts: (1) Multi-Scale Orthogonal Approximation. We use multi-scale orthogonal matrices to approximate the incremental matrix and minimize the overlap of LoRA through dual orthogonality constraints. (2) Complex Matrix Importance Pruning. We extend the orthogonality constraints to minimize spatial overlap.

3.1 Multi-Scale Orthogonal Approximation

The overall pipeline of LoRA^2 is illustrated in Figure 2. We utilize Singular Value Decomposition (SVD) to project the parameter increment matrix Δ onto a mutually orthogonal plane. Aiming to reduce model complexity and enhance computational efficiency, while preserving crucial information as much as possible through the singular value matrix Λ. Then, we iteratively apply low-rank matrices. As shown in Figure 2, training two low-rank matrices within two orthogonal plane. Ultimately, the parameter increment matrix is added to the pre-trained parameters to adapt to various downstream tasks, offering a plug-and-play capability. Our forward propagation proceeds as follows:

$$W = W^{(0)} + \Delta = W^{(0)} + P\Lambda Q$$
$$= W^{(0)} + (uv)\Lambda(\mathcal{U}\mathcal{V}), \quad (4)$$

where $v \in \mathbb{R}^{(din,k)}$, $u \in \mathbb{R}^{(k,r)}$, $V \in \mathbb{R}^{(k,dout)}$ and $U \in \mathbb{R}^{(r,k)}$ are two sets of orthogonal LoRA matrices. The variable k is a hyperparameter used to determine the dimension to which the data is projected. The matrix Λ is a diagonal matrix. Λ is initialized with zero while u, v, $\mathcal{U}$ and $\mathcal{V}$ adopt a random Gaussian initialization to ensure $\Delta = 0$ at the beginning of training. LoRA^2 employs regularization terms to increase the orthogonality of the matrices. This is similar to the AdaLoRA, which converts a constrained optimization problem into an unconstrained version, thereby enlarging the representational space of the matrices. To utilize regularization terms in LoRA^2, we expand the scope of the regularization terms. We further introduce regularizers between the uv and $\mathcal{U}\mathcal{V}$ matrices:

$$\begin{cases} R(\mathcal{P}, \mathcal{Q}) = \left\| P^T P - I \right\|_F^2 + \left\| QQ^T - I \right\|_F^2 \\ R(\mathcal{U}, \mathcal{V}) = \left\| \mathcal{U}^T \mathcal{U} - I \right\|_F^2 + \left\| \mathcal{V}\mathcal{V}^T - I \right\|_F^2, \\ R(u, v) = \left\| u^T u - I \right\|_F^2 + \left\| vv^T - I \right\|_F^2 \end{cases}$$
$$(5)$$

dual regularization helps enhance the optimization stability of the matrix. By minimizing the

overlap in LoRA, the learning space of the method is expanded. In Section 4.4, we present an ablation study to demonstrate the effectiveness of our regularization approach.

Algorithm 1 Pruning Algorithm of LoRA^2

1: **Input:** Dataset $\mathcal{D}$; total iterations T.
2: **for** $t = 1, \ldots, T$ **do**
3:     Sample a mini-batch from $\mathcal{D}$;
4:     Compute gradient $\nabla\mathcal{L}(v, \Lambda, \mathcal{V})$;
5:     Compute $\overline{I}_{ij}^{(t)}$ and $\overline{U}_{ij}^{(t)}$ as (?? and ??);
6:     Compute Importance $\mathbb{I}(\Lambda, i)$ as (9);
7:     Compute threshold $\mathcal{I}$, for $i = 1, \ldots, r$;
8:     Mask $(\mathbb{I}(\Lambda, i) < \mathcal{I})$.
9: **end for**
10: **Output:** The fine-tuned parameters (Λ, i).

3.2 Complex Matrix Importance Pruning

In LoRA^2, the matrix Λ is iteratively pruned to adjust the rank after each gradient descent step. We adopt the same rank pruning method as AdaLoRA, by applying the SVD-based adaptation to every weight matrix including W_q, W_k, W_v, W_{f1} and W_{f2} of each transformer layer. We summarize the detailed algorithm in Algorithm 1.

where $0 < \beta_1, \beta_2 < 1$. $\overline{I}^{(t)}$ is the sensitivity smoothed by an exponential moving average, and $\overline{U}^{(t)}$ is the uncertainty term quantified by the local variation between $\overline{I}^{(t)}$ and $I^{(t)}$. They then define the importance as the product of $\overline{I}^{(t)}$ and $\overline{U}^{(t)}$.

since LoRA^2 trains two LoRA blocks at multiple scales, there are four matrices related to singular values. The dimensions of matrices u and $\mathcal{U}$ are the same as those of the singular value matrix Λ, whereas the dimensions of v and $\mathcal{V}$ differ from Λ. Since the number of columns in matrices u and $\mathcal{U}$ and the number of rows in matrices v and $\mathcal{V}$ are all equal to the hyperparameter k, and each parameter in matrices u and $\mathcal{U}$ needs to be multiplied by matrices v and $\mathcal{V}$. We average the importance scores of each column in u and $\mathcal{U}$ and add it to the importance scores of each row in v and $\mathcal{V}$. Our total importance score calculation formula is as follows:

$$C(K, v, ij) = \sum_{j=1}^{K} S_{ij}(v), \quad (6)$$

$$\overline{S}(K, u) = \sum_{i=1}^{K} \sum_{j=1}^{din} S_{ji}(u), \quad (7)$$

$$I(\Lambda, i) = S(\Lambda, i) + \frac{C(K, v, ij) + \overline{S}(K, u)}{K}$$
$$+ \frac{C(K, \mathcal{V}, ij) + \overline{S}(K, \mathcal{U})}{K}, \quad (8)$$

Equation 6 represents the sum of row sensitivity scores for a low-rank matrix v with k rows. Equation 7 represents the sum of the column sensitivity scores for a low-rank matrix u with k columns. $I(\Lambda, i)$ denotes the importance score for the ith singular value Λ, calculated by averaging the sums of row sensitivity scores from matrices v and $\mathcal{V}$, and column sensitivity scores from matrices u and $\mathcal{U}$. Then adding the inherent sensitivity score of the singular value Λ. Since the importance score of each singular value Λ includes the average of the sensitivity scores of all parameters in matrices u and $\mathcal{U}$, the ranking is unaffected. Thus, in practical terms, we can disregard the sensitivities of matrices u and $\mathcal{U}$. Surprisingly, this approach reduces the calculation of parameter sensitivity scores by approximately 98.5%. Our method for calculating importance scores is as follows:

$$\mathbb{I}(\Lambda, i) = S(\Lambda, i) + \frac{C(K, v, ij)}{K} + \frac{C(K, \mathcal{V}, ij)}{K}. \quad (9)$$

4 Experiments

We implement LoRA^2 for fine-tuning DeBERTaV3-base (He et al., 2023) and RoBERTa-large (Liu et al., 2021), We evaluate the effectiveness of the proposed algorithm on natural language understanding tasks from the GLUE benchmark (Wang et al., 2018).

4.1 Experimental Settings

Implementation Details. We use PyTorch (Paszke et al., 2019) to implement all the algorithms. Our implementation is based on the publicly available Huggingface Transformers3 (Wolf et al., 2020) code-base. All the experiments about DeBERTaV3-base (He et al., 2021) are conducted on NVIDIA 4090 laptop GPU and experiments about RoBERTa-large (Liu et al., 2021) are conducted on NVIDIA A800 GPU. Since the total number of parameters in our model is primarily controlled by the hyperparameter K, the rank R has almost no impact on the total amount of trainable parameters. Therefore, we keep the K

of LoRA^2 consistent with the R of the baselines. This allows for comparisons between models with a similar amount of trainable parameters. We mainly use DeBERTaV3-base (He et al., 2023) as the backbone model. Additionally, we also use RoBERTa-large (Liu et al., 2021) for analysis.

Baselines. Our baselines comprise full-parameter fine-tuning and other well-recognized parameter-efficient methods, including Bitfit (Zhang et al., 2022) ,LoRA (Hu et al., 2022) , AdaLoRA (Zhang et al., 2023) and SoRA (Ding et al., 2023).

Datasets. For evaluation, we adopt the GLUE benchmark (Wang et al., 2018), a widely recognized benchmark for natural language understanding, including CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP (Wang et al., 2018), STS-B (Wang et al., 2018), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016) and RTE (Dagan et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2011).

4.2 Results

We first conduct an evaluation on GLUE benchmark. The experimental performance of LoRA^2 and other baselines are shown in Table 1. Our method results indicate that LoRA^2 consistently outperforms the baselines in most tasks. For instance, on the RTE, the accuracy of LoRA^2 reaches 89.53%, which is 2.17% higher than AdaLoRA ($r = 2$). On average, under the condition of $K/R = 1$, LoRA^2 outperforms LoRA and SoRA on the GLUE benchmark by 2.03% and 1.41%. When the parameter amount increases to $K/R = 8$, the performance of LoRA^2 further improves, exceeding LoRA and SoRA by 1.29% and 0.31% on average. Specifically, even when comparing LoRA^2 ($k = 1$) to other baselines with ($r = 8$), it still slightly outperforms the baselines. We also conduct an experiment on RoBERTa-large (Liu et al., 2021) to compare the performance of larger models. LoRA^2 exhibits remarkable capabilities. It achieves results comparable to the 335M parameters (full fine-tuning) while using only 0.4M parameters, thereby achieving a compression rate of 99.97%. Further comparison between LoRA^2 and LoRA reveals that our method improves performance by 2% while reducing parameters by 0.37M. These outcomes confirm that LoRA^2 maintains robust performance in large-scale pre-trained models,

Model	Method	#Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg.
DeB$_{base}^{V3}$	Fine-Tune	184M	69.21	95.64	89.22	92.05/89.31	91.59	89.98/89.95	93.78	82.49	87.82
DeB$_{base}^{V3}$	Bitfit	0.1M	68.70	94.38	87.16	87.86/84.20	89.71	87.45/87.45	91.90	76.12	85.18
DeB$_{base}^{V3}$	AdaLoRA ($r = 2$)	0.32M	70.04	**95.80**	<u>90.44</u>	**91.78/89.16**	<u>91.63</u>	**90.66/90.70**	**94.49**	<u>87.36</u>	<u>88.86</u>
DeB$_{base}^{V3}$	LoRA ($r = 1$)	0.17M	68.60	94.95	88.24	91.20/88.37	91.41	90.09/90.28	93.35	81.29	87.23
DeB$_{base}^{V3}$	SoRA ($r = 1$)	0.12M	<u>70.24</u>	95.14	89.22	91.52/<u>88.73</u>	91.41	90.08/<u>90.41</u>	93.43	83.02	87.85
DeB$_{base}^{V3}$	LoRA^2($k = 1$)	0.17M	**70.63**	<u>95.64</u>	**90.93**	<u>91.62</u>/88.05	**91.69**	90.19/90.38	<u>93.92</u>	**89.53**	**89.06**
DeB$_{base}^{V3}$	AdaLoRA ($r = 8$)	1.27M	71.45	**96.1**	90.69	92.23/<u>89.74</u>	91.84	**90.76/90.79**	**94.55**	88.09	<u>89.31</u>
DeB$_{base}^{V3}$	LoRA ($r = 8$)	1.33M	69.73	95.57	89.71	91.95/89.26	91.86	<u>90.47</u>/90.46	93.76	85.32	88.38
DeB$_{base}^{V3}$	SoRA ($r = 8$)	0.91M	<u>71.48</u>	95.64	**91.98**	**92.39/89.87**	**92.22**	90.35/<u>90.38</u>	<u>94.28</u>	87.77	89.36
DeB$_{base}^{V3}$	LoRA^2($k = 8$)	1.33M	**72.30**	<u>95.76</u>	91.42	92.25/<u>89.74</u>	<u>91.92</u>	90.43/90.23	<u>94.28</u>	**88.45**	**89.68**

Table 1: Test results of LoRA^2 and other baselines on the GLUE benchmark. We report the matched and mismatched accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. We use the same hyperparameters, which are specified in Appendix A. The optimal training values are used as results. We denote the best result in **bold** and <u>underline</u> the second-best result.

effectively demonstrating the versatility of LoRA^2. The results prove that the advantage of LoRA^2 is consistent across different model sizes, achieving results comparable to a baseline with four times the parameter quantity.

4.3 Rank Analysis

For a fixed model, it is inevitable that adapting to different downstream tasks can be challenging. At the same time, the importance of the model's parameters varies. Some parameters contribute minimally to the final outcomes, not only failing to enhance the model's capabilities but also impacting the convergence speed. Therefore, adaptively adjusting the parameter budget is crucial. In this section, we will show and analyze the final rankings of parameters after stable training on four datasets using the LoRA^2 method, as illustrated in Figure 3. The analysis results reveal that even after fitting with a low-rank matrix, the predefined rank still goes far beyond what is required for fine-tuning specific tasks. More than half of the ranks have minimal impact on the final outcome. Comparing different tasks, it is evident that MNLI requires the highest rank, while SST2 demands the lowest. Further analysis reveals that extensive pruning is needed for the top layers, with only the parameters in the FF.W1 layer being significant. In contrast, the demand for incremental parameters increases for the lower layers, demonstrating a clear heavy-tail structure. This phenomenon also indicates that using a constant parameter budget negatively affects the fine-tuning results, necessitating a case-by-case consideration.

Datasets	AdaLoRA	LoRA^2
CoLA	1.01s/it	1.14
SST-2	1.04	1.19
MRPC	1.45	1.68
QQP	1.44	1.64
STS-B	1.05	1.23
MNLI	1.31	1.50
RTE	1.44	1.65
Avg.	335M	0.8M

Table 2: The average training time per epoch on six datasets. For each task, the experiments with AdaLoRA and LoRA^2 have the same batch size 32.

4.4 Orthogonal Constraint Analysis

We evaluate five orthogonal methods on the GLUE benchmark. Experiments use the same hyperparameters, with only the method of orthogonal constraints changing. The experimental results are shown in Table 4. Applying orthogonal constraints simultaneously to $uv\&\mathcal{UV}\&PQ$ achieves the best results, outperforming the other three orthogonal methods by about 0.3%. Further analysis shows that applying orthogonal constraints to either $uv\&\mathcal{UV}$ or PQ yields similar results, and applying orthogonal constraints only to $\mathcal{UV}$ yields the worst results. We believe that constraining only $\mathcal{UV}$ without constraining PQ in LoRA^2 leads to spatial overlap in the training of Multi-Scale LoRA, resulting in reduced learning space. In contrast, applying dual constraints maximizes the orthogonality between matrix planes, thereby maximizing the learnable space.
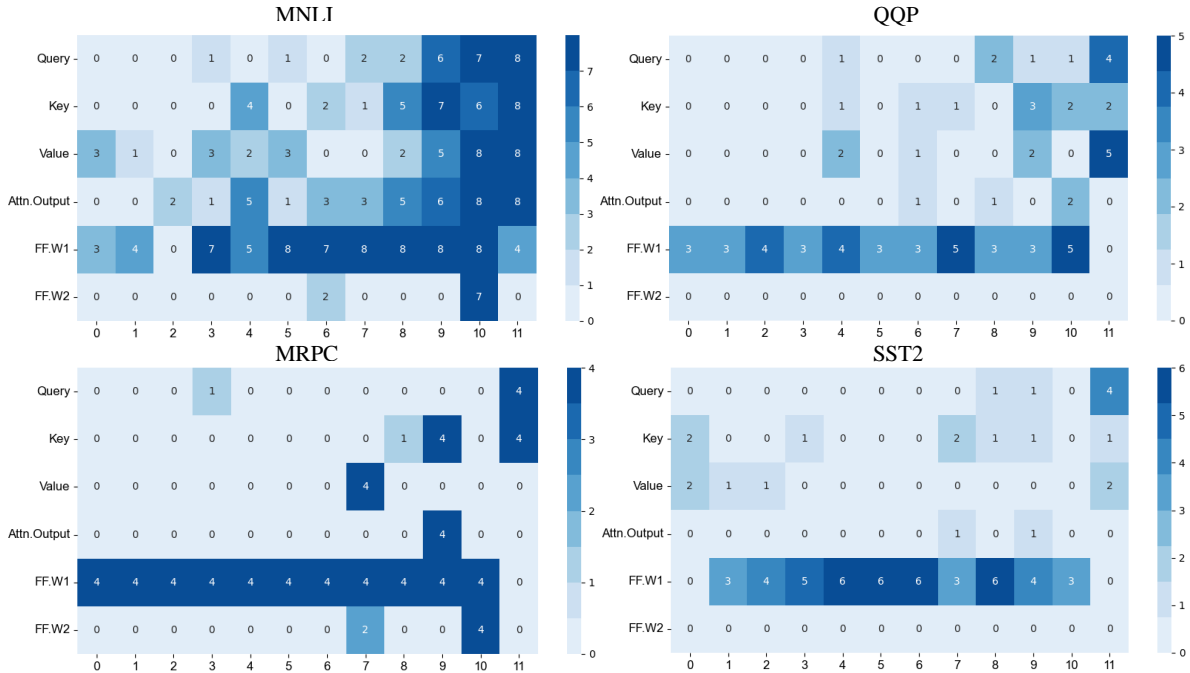
Figure 3: The final rankings after training with LoRA^2 ($r = 8$) on four datasets (i.e., MNLI, QQP, MRPC, and SST2). The X-axis is the index of DeBERTaV3-base layers, and the Y-axis indicates the different layers to which LoRA^2 is applied. The lighter the color, the lower the degree of pruning.

	MRPC	STS-B	MNLI	RTE	Avg.
LoRA$^2_{(uv)}$	90.20	91.61	90.27	89.17	90.31
LoRA$^2_{(\mathcal{UV})}$	90.20	91.57	90.23	89.17	90.29
LoRA$^2_{(uv\&\mathcal{UV})}$	90.44	91.61	90.33	89.53	90.48
LoRA$^2_{(P\&Q)}$	90.44	91.59	90.33	88.81	90.29
LoRA$^2_{(All)}$	**90.93**	**91.69**	**90.38**	**89.53**	**90.63**

Table 3: Results using different regularization methods. LoRA$^2_{(uv)}$ indicates the application of orthogonal constraints to the matrix uv of LoRA^2. *All* represents the simultaneous application of orthogonal constraints to $uv, \mathcal{UV}$, and PQ.

Method	#Params	CoLA	MRPC	STS-B	RTE	Avg.
LoRA$^2_{(Q,K)}$	37.4k	68.66	89.46	**91.69**	88.09	84.53
LoRA$^2_{(Q,V)}$	37.4k	67.70	89.95	91.66	84.84	83.54
LoRA$^2_{(Q,K,V)}$	56.2k	70.44	89.46	91.04	86.28	84.31
LoRA$^2_{(ALL)}$	167.6k	**70.63**	**90.93**	**91.69**	**89.53**	**85.70**

Table 4: The results of applying LoRA^2 ($k = 1$) to different layers. *ALL* represents the output of the query, key, value attention and FF layers respectively. **#Params** refers to the number of trainable parameters.

4.5 Applying LoRA^2 to Different Weights

In this section, we apply LoRA^2 ($k = 1$) to different types of weight matrices to determine how to achieve optimal performance on downstream tasks. We employ the DeBERTaV3-base model for fine-tuning tasks on the CoLA, MRPC, STS-B, and RTE datasets. As shown in Table 3, on the STS-B dataset alone, the results of applying LoRA^2 only to the Q and K matrices are equally excellent.

However, for other tasks, applying LoRA^2 to all weight matrices yields the best results, with an average performance lead of over 1%. This is largely consistent with the situation for LoRA. The results suggest that applying LoRA^2 to all weight matrices can be a beneficial strategy.

7

5 Conclusion

We propose a multi-scale low-rank approximation named LoRA^2, an innovative approach for efficiently fine-tuning large pretrained language models. Building on the basis of SVD, we train LoRAs at multiple scales on mutually orthogonal planes. By dynamically allocating parameter budgets through pruning, LoRA^2 adapts to various downstream tasks. We change the importance score algorithm to accommodate the structure of LoRA^2. It reduce the parameter sensitivity score calculations by approximately 98.5% without causing any degradation in performance. We conduct extensive experiments in the NLP domain, and LoRA^2 achieves performance close to baselines with eight times the number of parameters, demonstrating that LoRA^2 surpasses existing methods.

Limitations

Although LoRA^2 has demonstrated surprising performance on NLP datasets, our research still has some acknowledged limitations. However, recent studies have shown that methods for parameter-efficient fine-tuning can also be applied in the cross-modal domain, and the performance of LoRA^2 in the multimodal field is currently unknown. Additionally, our method only evaluates the fine-tuning results of dual LoRA at multiple scales. For more multi-scale LoRA, we intend to conduct experiments to verify its performance.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, and et al. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Luisa Bentivogli, Peter Clark, Ido Dagan, and et al. 2011. The seventh pascal recognizing textual entailment challenge. *Theory and Applications of Categories*.

David Bindel, James Demmel, William Kahan, and et al. 2002. On computing givens rotations reliably and efficiently. *Association for Computing Machinery Transactions on Mathematical Software*, pages 206–238.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, and et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, pages 1–113.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and et al. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186.

Ning Ding, Xingtai Lv, Qiaosen Wang, and et al. 2023. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4133–4145.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, and et al. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, pages 12799–12807.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and et al. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *Proceedings of the International Conference on Learning Representations*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and et al. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *Proceedings of the International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, and et al. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of Machine Learning Research*, pages 2790–2799.

Edward Hu, Yelong Shen, Phillip Wallis, and et al. 2022. Lora: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations*.

David C. Lay, Steven R. Lay, and Judi J. McDonald. 2016. *Linear Algebra and Its Applications*, 5 edition. Pearson.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 4582–4597.

Zhuang Liu, Wayne Lin, Ya Shi, and et al. 2021. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the Chinese National*

Conference on Computational Linguistics, pages 1218–1227.

Adam Paszke, Sam Gross, Francisco Massa, and et al. 2019. *PyTorch: an imperative style, high-performance deep learning library*.

XiPeng Qiu, TianXiang Sun, YiGe Xu, and et al. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, page 1872–1897.

Alec Radford, Jeffrey Wu, Rewon Child, and et al. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and et al. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, and et al. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *Preprint*, arXiv:2402.07927.

Richard Socher, Alex Perelygin, Jean Wu, and et al. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Alex Wang, Amanpreet Singh, Julian Michael, and et al. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 353–355.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, pages 625–641.

Jason Wei, Yi Tay, Rishi Bommasani, and et al. 2022. Emergent abilities of large language models. *Transactions on Machine Learning Research*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1112–1122.

Thomas Wolf, Lysandre Debut, Victor Sanh, and et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 38–45.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, and et al. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *Proceedings of the International Conference on Learning Representations*.

Qingru Zhang, Simiao Zuo, Chen Liang, and et al. 2022. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *Proceedings of the 39th International Conference on Machine Learning*, pages 26809–26823.

A Datasets

For evaluation, we adaopt the GLUE benchmark (Wang et al., 2018), including CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP (Wang et al., 2018), STS-B (Wang et al., 2018), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016) and RTE (Dagan et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2011). We present the dataset statistics of GLUE in the following table 5.

Dataset	Metric	#Train	#Valid	#Test	#Label
CoLA	Mcc	8.5k	1,043	1,063	2
SST-2	Acc	67k	872	1.8k	2
MRPC	Acc	3.7k	408	1.7k	2
QQP	Acc/F1	364k	40.4k	391k	2
STS-B	Corr	5.7k	1.5k	1.4k	1
MNLI	Acc(m/mm)	393k	20k	20k	3
QNLI	Acc	105k	5.5k	5.5k	2
RTE	Acc	2.5k	277	3k	2

Table 5: Dataset Sizes and Evaluation Metrics in the GLUE Benchmark. "Mcc," "Acc," "F1," and "Corr" denote the Matthews correlation coefficient, accuracy, F1 score, and Pearson correlation coefficient, respectively. "Acc(m/mm)" indicates accuracy results for matched and mismatched datasets within MNLI.

B Sparse Regularization Theory

By using progressive projection matrices, we further increase the compression ratio p of the parameters. Additionally, this enhances the stability of the fine-tuning process. Equations 10(Fu et al., 2023) theoretically demonstrate the role of sparsity in model stability. As the compression ratio p decreases, the upper bound also decreases. Therefore, a sparser model implies better stability.

$$\mathbb{E}_{S,i\sim U(n)}[|\ell(A(S^i), z_i) - \ell(A(S), z_i)|]$$
$$\leq \frac{2\rho^2}{(\Lambda_{min} + 2(1-p))n}, \quad (10)$$

$\mathbb{E}_{S,i\sim U(n)}[\cdot]$ is Pointwise Hypothesis Stability (PHS)(Bindel et al., 2002) which focuses on analyzing the change of model output after a training sample is removed. $\Lambda_{\min} = \min\{\Lambda_1, \ldots, \Lambda_m\}$. $\ell(\cdot)$ represents the loss function. The variable ρ represents this measure of stability, reflecting the maximum impact of input variations on the output in the loss function.

C Orthogonal Projection Theory

It is a fundamental concept in linear algebra with applications across various fields including machine learning, statistics, and computer graphics (Lay et al., 2016). This theory revolves around the idea of projecting a vector onto a subspace in a way that minimizes the distance between the vector and the subspace, effectively finding the closest approximation within that subspace.

Mathematically, consider a vector u in R_n and a subspace $\mathbb{V}$ spanned by vectors $\{v_1, v_2, \ldots, v_k\}$. The orthogonal projection of u onto $\mathbb{V}$, denoted as $\mathbb{P}_{\mathbb{V}}(\mathbf{u})$, is given by:

$$\mathbb{P}_{\mathbb{V}}(\mathbf{u}) = \sum_{i=1}^{k} \frac{\mathbf{u} \cdot \mathbf{v}_i}{\mathbf{v}_i \cdot \mathbf{v}_i} \mathbf{v}_i \quad (11)$$

LoRA[2] enhances the model's learning and representational capabilities by training two mutually orthogonal LoRA blocks. This design allows each LoRA block to capture information in different dimensions, thereby reducing information overlap and increasing the overall efficiency and effectiveness of the model. Additionally, the orthogonal training strategy helps prevent overfitting, making the model more robust when faced with new data.