SARA: Singular-Value Based Adaptive Low-Rank Adaption

Anonymous ACL submission

Abstract

Low-Rank Adaptation (LoRA) as a parameter-efficient fine-tuning (PEFT) method is widely used for not adding inference overhead. It assumes that weight changes during fine-tuning can be approximated by low-rank matrices. Despite the recent progress, existing methods suffer from three drawbacks: 1) Lacking differentiation of ranks for each layer of the model; 2) The rank values need to be manually verified; 3) Ignore the relationship between intrinsic rank and the initial pre-trained matrix. In this work, we first analyze the relationship between the performance of different layers and their pre-trained matrix using SVD. Based on this, we design the **S**ingular-Value Based **A**daptive Low-**R**ank **A**daption (SARA), which adaptively finds the suitable rank for each layer during initialization. Additionally, we explore the Mixture-of-SARA (Mo-SARA), which significantly reduces the number of parameters by fine-tuning only multiple parallel sets of singular values controlled by a router. Extensive experiments on various complex tasks have demonstrated the state-of-the-art performance and parameter efficiency of our methods.

1 Introduction

Large language models have demonstrated impressive generative capabilities, achieving excellent performance across various natural language processing (NLP) tasks (Touvron et al., 2023; Qin et al., 2023; Kojima et al., 2022). However, as the model size increases, the cost of full-parameter fine-tuning to adapt the model to downstream tasks becomes increasingly prohibitive. To address this issue, PEFT methods have garnered increasing attention (Houlsby et al., 2019; Li and Liang, 2021). Among them, the LoRA (Hu et al., 2021) method, which leverages the concept of matrix 'intrinsic rank' by freezing the original model parameters and fine-tuning only a small number of newly added, representative parameters, has been widely adopted.

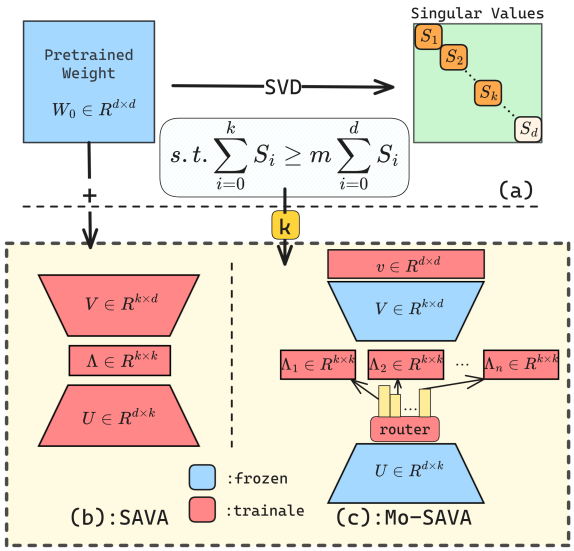

Figure 1: An overview of our methods, (a) performing SVD on the pre-trained weights and determining the number k of values that account for a proportion threshold τ of the total sum of singular values; (b) the method of adding a truncated singular value matrix to the pre-trained weights based on k; and (c) the extreme method of fine-tuning only mixture of parallel singular values. Λ and v, as diagonal matrix, only require a one-dimensional vector for storage.

Its primary advantage is it does not add extra computational overhead during inference.

Existing LoRA-like methods, however, as shown in Table 1, still suffer from three drawbacks: 1) *Lacking differentiation of ranks for each layer of the model.* As different layers in transformers have varying degrees of importance (Jawahar et al., 2019; Tenney et al., 2019; Jawahar et al., 2019). 2) *The rank values need to be manually verified* which fails to determine the most suitable rank for each specific model. 3) *Ignore the relationship between intrinsic rank and the initial pre-trained matrix.* LoRA leverages the concept of intrinsic rank which is related to singular value decomposition (SVD), and the singular diagonal value matrix is characterized by a small number of leading val-

ues accounting for a large proportion of the total sum. We perform SVD on the initial pre-trained model matrices, as shown in Figure 1a. We focus on the number k of values in the singular value that cumulatively account for a certain threshold τ of the total sum and our analysis reveals that the k value of each layer in the model correlates with the performance of that layer! Thus, we argue that k can reflect the most suitable intrinsic rank for each layer easily during initialization as the magnitude of singular values represent the significance of it, and existing works ignore this phenomenon.

To alleviate these aforementioned problems, we propose a **S**ingular-Value Based **A**daptive Low-**R**ank **A**daption (SARA) method, as shown in Figure 1b. SARA calculates the most suitable rank for each layer based on the importance threshold τ during initialization and fine-tunes the newly added truncated singular value matrices. Additionally, we explore an extreme method, Mixture-of-SARA (Mo-SARA), which significantly reduces the number of trainable parameters to the limit. As shown in Figure 1c, Mo-SARA only fine-tunes k diagonal values as well as a diagonal matrix$v$ used to accelerate convergence. They just require a one-dimensional vector for storage to significantly reduce the number of trainable parameters. Moreover, leveraging the concept of Mixture-of-Experts (MoE) (Jacobs et al., 1991), we innovatively train multiple singular value matrices in parallel, to leverage the entire truncated singular value matrix separately, achieving comparable performance.

Experimental results show that our improved methods can adaptively find suitable ranks, achieving better performance even with fewer trainable parameters while retaining the advantages of LoRA and achieving state-of-the-art performance.

In summary, our contributions are:

1.We analyze the interactions between different layers and pre-trained matrices by SVD, discovering more suitable intrinsic rank, providing a new research perspective for the entire PEFT field to address the issue of inter-layer inconsistency.

2.We propose the SARA method, which can adaptively calculate the suitable rank for each layer during initialization, extending the performance of LoRA, and can be combined with other methods.

3.We further propose the Mo-SARA, which explores leveraging the entire SARA process with only singular values and paralleling these values, significantly reducing the number of trainable parameter by an order of magnitude.

Methods	**Rank-Differ**	**Adaptive-Rank**	**Intrinsic Rank &Matrix-Relation**
LoRA	✗	✗	✗
PiSSA	✗	✗	✓
AdaLoRA	✓	✓	✗
DyLoRA	✓	✓	✗
DoRA	✗	✗	✗
VeRA	✗	✗	✗
SARA	✓	✓	✓

Table 1: Comparisons with LoRA-like methods from the perspective of whether assign different ranks to different layers, whether adaptively allocate ranks, and whether consider the relationship between intrinsic rank and pre-trained matrix. The methods include LoRA(Hu et al., 2021), PiSSA (Meng et al., 2024), AdaLoRA(Zhang et al., 2023), DyLoRA(Valipour et al., 2023), DoRA(Liu et al., 2024), VeRA(Kopiczko et al., 2023).

2 Related Works

2.1 PEFT Methods

Traditional PEFT methods focuing on freezing the original pre-trained parameters and fine-tuning only a subset of newly added parameters. Typically, adapters (Houlsby et al., 2019; Patel et al., 2021) involve serially connecting a set of newly added tunable parameters within the model; prefix-tuning add virtual tokens to the model inputs (Li and Liang, 2021); LoRA (Hu et al., 2021) assume that the model parameter matrix only requires fine-tuning a matrix of rank r, and replace the original matrix with two matrices that increase and decrease dimensions, respectively, for fine-tuning.

Among these methods, LoRA is widely used while it can be directly added alongside the original matrix without requiring additional inference time and generally achieves better performance across various tasks. Consequently, the LoRA method has numerous improvements.

2.2 LoRA's Variants

Several works focus on the modification of the structure of LoRA. For example, PiSSA (Meng et al., 2024), which is the most similar method to ours, sets a fixed rank, performs SVD on the matrix, and fine-tunes only the low-rank components. Dora (Liu et al., 2024) improves performance by decomposing the original matrix into weight and direction components and fine-tuning them separately; VeRA (Kopiczko et al., 2023) method reduces the number of trainable parameters based on the LoRA by randomly initializing and freezing the dimensionality expansion/reduction matrices and only fine-tuning two diagonal matrices added after

them. A common issue with these works is that they overlook the fact that each model and each layer's weight matrix has a different intrinsic rank, making a globally uniform rank setting inefficient for fine-tuning.

Other works noticed this issue. AdaLoRA (Zhang et al., 2023) and DyLoRA (Valipour et al., 2023) calculate the suitable rank during training. However, they overlook leveraging the relationship between intrinsic rank and the pre-trained matrix to compute the rank simply and efficiently.

Unlike previous works, in this paper, we find an effective method to adaptively find the suitable rank layer-by-layer through the SVD of pre-trained matrix during initialization. We also propose a MoE-like method, which leverages a larger number of parameters for fine-tuning with a minimal parameter. These methods only require little computation time during initialization and retain all the advantages of LoRA, even achieving state-of-the-art performance. For the second method, only a minimal number of parameters need to be stored.

3 Correlation Analysis Between Layer Performance and Singular Values

As mentioned above, different layers exhibit varying degrees of importance and for the LoRA method, each model adapts to downstream tasks with different ranks.

To study the inter-layer different importance of LoRA, we conduct a case study experiment on LLaMA-7B (Touvron et al., 2023) based on previous work (Hu et al., 2023). We divide 32 layers of the model into four parts and fine-tune each part separately using LoRA method, testing their average accuracy on six mathematical reasoning datasets. As shown in the bar chart in Figure2, the overall performance is excellent in the lower layers and poorer in the upper layers.

Since the rank concept is related to SVD, we perform SVD on the pretrained Q and V matrices used in the classical LoRA method and analyze the singular values. Because the decomposed singular values are arranged in descending order and a small proportion of the leading values account for a large portion of the total sum, we calculate the number of singular values k needed to account for various proportion thresholds τ. (specific details on obtaining k can be referenced to algorithm 1.) As shown in the line chart in Figure 2, under all different proportion choices, the value of k decreases initially as

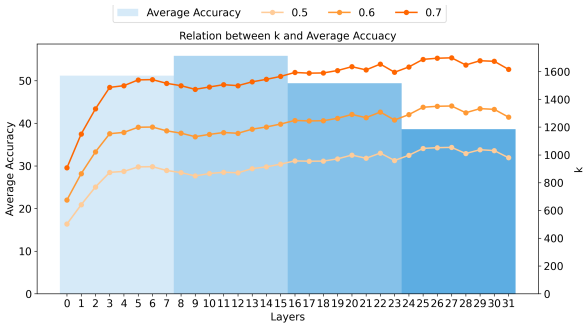

Figure 2: The impact of different layers on the average accuracy of mathematical reasoning tasks and the k of different threshold τ (mean value obtained from Q and V matrix SVD.)

the model goes from lower layers to higher layers and then increases, which is exactly opposite to the trend of performance change across layers.

We believe that this is because, to achieve similar effects, the lower layers require a lower 'intrinsic rank' while the upper layers require a higher one. This corresponds to our calculated k values. Therefore, allocating the same rank to all layers leads to shortcomings in certain layers, thus affecting the overall efficiency of the model, and it is necessary to allocate ranks to each layer according to the corresponding k values to avoid the bottleneck effect. Based on this, we design an improved method and conduct tests to compare the effects between layers, which will be presented bellow.

4 Method

4.1 Motivation

Based on the above findings, we define the number of singular values that represent a certain proportion of the total sum as k to reflect the intrinsic rank. Specifically, we use proportion threshold τ which can reflect the layer's importance instead of rank to set the hyperparameters conceptually similar to (Schotthöfer et al., 2022), and add a new truncated singular value matrix parallel to the original matrix. Through this method, we can adaptively determine the intrinsic rank of each layer during initialization.

4.2 SARA

The LoRA(Hu et al., 2021) method is based on the assumption that changes in the matrix during fine-tuning have a low 'intrinsic rank'. It involves adding a dimensionality reduction matrix $A \in \mathbf{R}^{d\times r}$ and a dimensionality expansion matrix $B \in \mathbf{R}^{r\times d}$ with a fixed scaling λ parallel to the original weight matrix$W_0 \in \mathbf{R}^{d\times d}$, using these as

the only trainable matrices. The calculation formula is as follows:

$$h = x(W_0 + \Delta W) = x(W_0 + \lambda \underline{AB}) \quad (1)$$

SVD decomposes a matrix into three parts, we represent it with the formula shown in Equation 2.

$$W = U\Lambda V \approx U_k \Lambda_k V_k \quad (2)$$

The $U \in \mathbf{R}^{d\times d}$ and $V \in \mathbf{R}^{d\times d}$ are the left and right singular value matrices, respectively. Matrix $\Lambda \in \mathbf{R}^{d\times d}$ is called the singular diagonal value matrix with non-negative singular values on the diagonal, arranged in descending order. A small proportion of the leading values accounts for a large portion of the total sum of the singular values. Therefore, a truncated singular value matrix is commonly used to approximate and reduce the original matrix.

In this way, $U_k = U[:, :\ k]$, $\Lambda_k = \Lambda_k[:\ k, :\ k]$, $V_k = V_k[:\ k, :]$, where $k < d$ needs to be determined in advance.

Thus, in our methods, we use a randomly initialized truncated singular value matrix to represent the part of the original matrix that needs to change during fine-tuning, adding it parallel to the original matrix. It further explore the 'intrinsic rank' from the pre-trained weights of the original matrix using SVD during initialization. The calculation formula is shown as follows:

$$h = x(W_0 + \underline{U_k \Lambda_k V_k}) \quad (3)$$

where the underlined part represents the trainable truncated singular value matrix, and the calculation of k is shown in the following algorithm 1:

Algorithm 1 Calculate k Value

Require: $W_{pretrain} \in R^{d\times d}$, $threshold\tau \in (0, 1)$

$U, \Lambda, V \leftarrow \text{SVD}(W_{pretrain})$
$total \leftarrow \sum \Lambda$
$target \leftarrow \tau \times total$
$cumulative \leftarrow 0, k \leftarrow 0$
while $cumulative < \tau$ **do**
  $cumulative \leftarrow cumulative + \Lambda(k, k)$
  $k \leftarrow k + 1$
end while
return(k)

Since the singular value matrix is a diagonal vector, we only need to store a one-dimensional vector, which allows us to reduce parameter storage.

The magnitude of the singular values can indicate the significance of the data. Therefore, we remove the scaling part λ in the original LoRA method, as our singular values effectively act as more fine-grained, learnable scaling factors. For ΔW in SARA, each element is expressed as shown in Equation 4:

$$\Delta W_{ij} \approx \sum_{r=1}^{k} u_{ir} S_r v_{rj} \quad (4)$$

Here, u_{ir} and v_{rj} represent $U_k(i, r)$ and $V_k(r, j)$, respectively; S_r represents $\Lambda_k(r, r)$.

4.3 Mo-SARA

In singular values, the larger singular values correspond to the main directions of variation in the data, while the smaller singular values can be regarded as noise or less important variations. Based on this, we believe that for different downstream tasks, it is sufficient to only adjust the singular values under the same eigenvector mappings.

Additionally, inspired by MoE (Jacobs et al., 1991), we believe multiple singular value diagonal matrices can be trained in parallel and selected through a routing mechanism to learn different tasks. Therefore, we explore an extreme improvement method for the trainable parameters, called Mixture-of-SARA(Mo-SARA). In this method, we keep the left and right singular vectors of the computed truncated singular value matrix unchanged and randomly initialize multiple singular value diagonal matrices for learning. To accelerate convergence referencing (Hu et al., 2021), where matrix B is initialized to 0, we also add a diagonal matrix $v \in \mathbf{R}^{d\times d}$ initialized to zero after the truncated singular value matrix. The formula for this method is shown in Equation 5:

$$h = xW_0 + \sum \underline{g_i} \odot (xU_k \underline{\Lambda_{k_i}} V_k \underline{v}) \quad (5)$$

where the gate g=[g_1,g_2...g_m] with values in the range (0,1) is computed as follows:

$$g = softmax(xU_k \underline{(W_{g_1} W_{g_2})}) \quad (6)$$

Here, we use the value of the input $x \in \mathbf{R}^{l\times d}$ with a length of l and dimension d after passing through the left singular matrix U_k as the input, and generate token-level gating $g \in \mathbf{R}^{l\times m}$ through an MLP layer composed of two gating matrices $W_{g_1} \in \mathbf{R}^{k\times 1}$ and $W_{g_2} \in \mathbf{R}^{1\times m}$. (We use two one-dimensional linear layers to minimize the number of parameters while achieving effective results.)

Method	Params(%)	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
				LLaMA-7B				
Prefix	1.2E-1	42.50	23.53	58.23	60.00	66.67	15.91	44.47
Adapter	2.9	53.50	23.53	74.68	86.36	75.49	20.08	55.61
Parallel	2.9	**63.00**	25.49	75.95	86.36	83.33	23.11	59.54
LoRA	7.8E-2	58.50	23.53	75.95	**92.73**	**88.24**	24.24	60.53
PiSSA	7.8E-2	58.00	19.61	**82.28**	85.45	87.25	**28.7**	60.23
Mo-SARA	**8.5E-3**	55.00	23.53	70.89	90.91	87.25	26.14	58.95
SARA	7.1E-2	60.00	**35.29**	79.75	89.09	84.31	24.62	**62.18**
				LLaMA-13B				
Prefix	9.4E-2	58.00	29.41	72.15	78.18	82.35	22.73	57.14
Adapter	2.4	55.00	31.37	73.42	78.18	69.61	17.05	54.10
Parallel	2.4	69.00	17.65	81.01	93.64	86.27	27.27	62.47
LoRA	6.3E-2	66.00	21.57	82.28	**95.45**	**89.22**	**36.74**	65.21
PiSSA	6.9E-2	65.50	**33.33**	**83.54**	92.73	88.24	33.71	66.18
Mo-SARA	**6.9E-3**	66.50	25.49	82.28	**95.45**	**89.22**	34.09	65.50
SARA	6.3E-2	**71.50**	27.45	81.01	93.64	88.24	36.36	**66.37**
				GPT-J-6B				
Prefix	1.1E-1	41.50	9.80	**67.09**	75.45	71.57	9.85	45.88
Adapter	1.9	43.00	13.73	56.96	76.36	64.71	9.85	44.10
Parallel	2.8	42.50	19.61	56.96	78.18	66.67	**12.88**	46.13
LoRA	7.6E-2	47.00	5.88	65.82	72.73	76.47	11.36	46.54
PiSSA	7.6E-2	46.50	25.49	**67.09**	73.64	74.51	12.12	49.89
Mo-SARA	**8.6E-3**	45.50	15.69	64.56	**82.73**	**78.43**	11.74	49.77
SARA	7.0E-2	**50.50**	**27.45**	65.82	79.09	74.51	12.50	**51.65**

Table 2: The results on six different mathematical reasoning datasets. The answer is the accuracy of calculations obtained using the zero-shot learning method on LLaMA-7B/13B, and GPT-J presented in the table.(**bold**: the best score; underline: the second best)

This method only requires the storage of one-dimensional parameters as it can expand to a diagonal singular value matrix only during computation. Even with multiple parallel sets, it still requires a few parameters to store, and each set can leverage the singular values to move the entire truncated singular value matrix, obtaining better efficiency.

5 Experiment

In this section, (1) we compare our methods with the PEFT methods as well as the latest LoRA-like methods, especially PiSSA (Meng et al., 2024), which is similar to us, across a wide range of tasks, including mathematical reasoning, commonsense inference, and E2E tasks, covering a total of 15 datasets. (2) Subsequently, we validate our method's ability to address the issue of inconsistent layer importance mentioned above. (3) We then conduct ablation experiments to discuss the effect of each component of our methods. (4) Next, we examine the parameter sensitivity of our methods and the impact of the number of parallel heads on Mo-SARA. (5) Finally, we show the routing learned by the Mo-SARA across various tasks, demonstrating the effectiveness of this mechanism.

In order to ensure the accuracy of the performance for other methods as much as possible, we compare different methods across various datasets. The detailed hyperparameter and experimental settings for all experiments in this section can be found in Appendix A.

5.1 Mathematical Reasoning

We compare our methods with five PEFT methods, including LoRA (Hu et al., 2021), Prefix (Li and Liang, 2021), Adapter(Houlsby et al., 2019), Parallel Adapter(Parallel) (Patel et al., 2021), and PiSSA (Meng et al., 2024), using three LLMs: LLaMA-7B/13B (Touvron et al., 2023), and GPT-J (Wang and Komatsuzaki, 2021), across six mathematical reasoning sub-tasks which are (1) the SVAMP (Patel et al., 2021), (2) the AQuA (Ling et al., 2017) dataset, (3) the AddSub (Hosseini et al., 2014) dataset, (4) the MultiArith (Roy and Roth, 2016) dataset, (5) the SingleEQ (Koncel-Kedziorski et al., 2015) dataset, and (6) the GSM8K (Cobbe et al., 2021) dataset. We largely follow the open-source work (Hu et al., 2023) in terms of experiments and hyperparameter settings, combining the six tasks to create a unified training dataset and testing accuracy on each task separately. To ensure a fair comparison, we adjust the threshold for k in our method during initialization to achieve a similar number of trainable parameters. The table below lists the proportion of trainable parameters to the total parameters for each method.

Table 2 shows that our SARA method significantly outperforms various baseline methods across

Method	Params(%)	ARC-c	ARC-e	Boolq	WinoG	PIQA	SIQA	OBQA	HellaS	Avg.
ChatGPT	-	79.9	89.8	73.1	66.1	85.4	68.5	74.8	78.5	77.0
					LLaMA-7B					
Prefix	1.1E-1	54.0	72.9	64.3	72.1	76.8	73.9	60.6	42.1	64.6
Adapter	9.9E-1	57.1	74.5	63.0	75.7	79.2	76.3	72.4	67.9	70.8
Parallel	3.5	57.3	73.7	67.9	78.9	76.4	78.8	75.2	69.8	72.2
LoRA	8.3E-1	61.3	77.8	68.9	78.8	80.7	77.4	74.8	78.1	74.7
PiSSA	8.5E-1	62.4	77.0	68.1	78.2	79.2	76.0	76.2	81.5	74.8
DoRA	8.4E-1	**66.2**	**81.9**	69.7	81.0	83.4	78.6	79.2	**87.2**	78.4
Mo-SARA	**8.5E-3**	54.5	74.5	62.8	71.8	76.0	73.8	65.8	50.3	66.2
SARA	8.3E-1	65.8	81.6	**70.9**	**82.6**	**83.6**	**78.8**	**81.4**	82.9	**78.5**
					LLaMA-13B					
Prefix	3.1E-2	62.9	79.5	65.3	68.6	75.4	72.1	68.0	55.2	68.4
Adapter	8.0E-1	67.3	82.5	71.8	82.4	83	79.2	81.8	88.1	79.5
Parallel	2.9	**71.2**	**84.2**	72.5	84.1	**84.9**	79.8	82.4	92.1	81.4
LoRA	6.7E-1	68.3	82.8	72.1	83.5	80.5	**83.7**	82.4	90.5	80.5
PiSSA	6.7E-1	66.0	81.5	70.3	81.4	83.7	79.2	81.0	90.4	79.2
DoRA	6.8E-1	69.6	**84.2**	72.4	84.2	**84.9**	81.5	82.8	**92.4**	81.5
Mo-SARA	**6.9E-3**	61.6	78.7	67.9	76.9	80.2	76.3	72.6	76.4	73.8
SARA	6.8E-1	69.8	84.1	**73.2**	**84.9**	83.9	80.6	**84.6**	92.2	**81.7**

Table 3: The results on 8 commonsense inference datasets, with ChatGPT and baseline results taken from (Hu et al., 2023), the DoRA method results sourced from (Liu et al., 2024).(**bold**: the best score; underline: the second best)

a wide range of models, achieving up to an 11% improvement over the LoRA method. Additionally, our Mo-SARA method achieves remarkable results with an order of magnitude fewer trainable parameters, even surpassing all baselines on the LLaMA-13B and GPT-J models.

5.2 Commonsense Inference

For commonsense reasoning, which includes eight downstream tasks as follows:(1) the ARC-c and (2) the ARC-e are the Challenge Set and Easy Set of ARC (Clark et al., 2018), (3) the Boolq (Clark et al., 2019), (4) the WinoGrande (Sakaguchi et al., 2021), (5) the PIQA (Bisk et al., 2020), (6) the SIQA (Sap et al., 2019), (7) the OBQA (Mihaylov et al., 2018), and (8) the HellaSwag (Zellers et al., 2019). We conduct experiments on the LLaMA-7B/13B (Touvron et al., 2023) models to extend the comparison with DoRA (Liu et al., 2024) and the results obtained with GPT-3.5-turbo API through zero-shot CoT (Wei et al., 2022). We also largely follow this work (Hu et al., 2023).

The results in Table 3 show that SARA achieves better results across a variety of models and datasets, with up to a 5% improvement over the LoRA method. Our Mo-SARA method, despite inherently using fewer training parameters, achieves comparable results on this task with almost two orders of magnitude fewer parameters, even surpassing the performance of the prefix method.

5.3 E2E Benchmark

To further validate the performance of our methods through broader comparisons, we also conduct experiments on E2E (Novikova et al., 2017). We follow the experimental setup from (Hu et al., 2021) and use GPT-2 Medium (Radford et al., 2019) model. In addition to LoRA, we compare new variants of the LoRA method, including Adalora (Zhang et al., 2023), Dylora (Valipour et al., 2022), and Vera (Kopiczko et al., 2023). For VeRA method, we use all the experimental settings mentioned in the paper (Kopiczko et al., 2023).

The experimental results are shown in Table 4. It can be seen that our SARA method achieves better results with fewer trainable parameters. In particular, our Mo-SARA method outperforms VeRA (Kopiczko et al., 2023) with fewer parameters.

5.4 Improvement of SARA across Layers

We use the SARA method, dividing 32 layers of LLaMA-7B into four parts for separate fine-tuning to verify our method's effectiveness in allocating ranks using singular values, addressing the issue of poorer results caused by inconsistent importance across layers. As shown in the Figure 3, our method consistently outperforms in each fine-tuning part, reducing the variance among layer results and addressing the problem posed in section 3.

5.5 Ablation Study

To analyze the impact of each component of SARA, we set up two groups of ablation experiments. These experiments verify whether it is necessary to initialize the up-projection matrix V to zero as in the original LoRA method and whether it is necessary to add the singular value diagonal matrix Λ. We conduct experiments using LLaMA-7B on

Method	Params	BLEU	NIST	METEOR	ROUGE-L	CIDEr
FT^1	354.92M	68.2	8.62	46.2	71.0	2.47
$Adpt^{L1}$	0.37M	66.3	8.41	45.0	69.8	2.40
$Adpt^{L1}$	11.09M	68.9	8.71	46.1	71.3	2.47
$Adpt^{H1}$	11.09M	67.3	8.50	46.0	70.7	2.44
$DyLoRA^2$	0.39M	69.2	8.75	46.3	70.8	2.46
$AdaLoRA^3$	0.38M	68.2	8.58	44.1	70.7	2.35
$LoRA^1$	0.35M	**70.4**	**8.85**	**46.8**	71.8	2.53
VeRA	0.098M	69.1	8.71	46.3	70.8	2.43
Mo-SARA	**0.094M**	69.4	8.77	46.4	71.1	2.48
SARA	0.33M	**70.4**	8.84	46.7	**72.3**	**2.55**

Table 4: The results on the E2E dataset, with the results for ([1],[2],[3]) taken from previous work. [1](Hu et al., 2021),[2](Zhang et al., 2023),[3](Valipour et al., 2023)

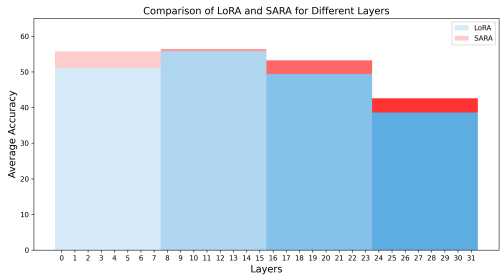

Figure 3: Average accuracy of SARA and LoRA methods across layers in mathematical reasoning tasks.

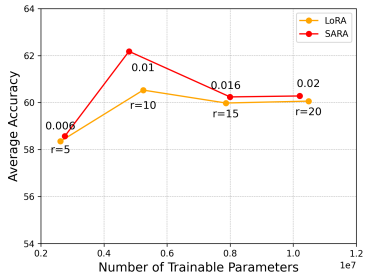

Figure 4: Average accuracy of the SARA and LoRA methods on mathematical reasoning tasks with different trainable parameters. The thresholds τ for determining k in the SARA method [0.006, 0.01, 0.016, 0.02] and the r values used to adjust the parameter count in the LoRA method [5, 10, 15, 20] are indicated in the figure.

mathematical reasoning tasks, as shown in the table 5. It can be seen that our approach of directly adding the truncated singular value matrix next to the original matrix yields better results and ddding singular value diagonal matrix almost does not increase the parameter count.

We also study the structure of adding a new truncated singular value matrix parallel to the original matrix behaves without our rank adaptation method as shown in Table 5, it shows the effectiveness of methods assigning different ranks to different layers.

Additionally, we conduct a set of experiments on the scaling value λ of the original LoRA to show that the original LoRA is also sensitive to the λ and our method of replacing scaling with singular values Λ to some extent addresses this issue. The results can be seen in Appendix C.

For Mo-SARA, we try to omit the diagonal matrix v (for a fair comparison of parameter quantities, we parallel 10 heads to increase the number of trainable parameters.), which is added after the singular value matrix for fast convergence, and also placing v in front of the truncated singular value matrix. The experimental results are shown in Table 5. It can be seen that, regardless of its position, a diagonal matrix for fast convergence plays a significant role. At the same time, even when only fine-tuning the singular value part, it still achieves decent results with a small parameter count, proving the effect of this component for fine-tuning, consistent with our hypothesis.

5.6 Robustness of the SARA Method

We conduct experiments using LLaMA-7B on mathematical tasks to compare the trends of SARA and LoRA under different trainable parameter sizes. The experimental results are shown in Figure 4. It can be seen that our method outperforms the LoRA method under all trainable parameter sizes and exhibits similar trends to the LoRA method. This demonstrates that our approach of assigning different ranks to different layers during initialization offers greater advantages.

5.7 Analysis Under Parameter Limits

To explore methods for further reducing trainable parameters, we conduct experiments on the Mo-SARA using LLaMA-7B with mathematical reasoning tasks.

Threshold τ: We design four sets of experiments without parallel structure. The threshold τ for determining k is set incrementally to 0.1, 0.3, 0.5, and 0.7. The parameter counts and results are shown in Figure 5, displaying a trend of initial increase followed by a gradual decrease. Thus, in

Method	Params(%)	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
SARA	7.1E-2	60.00	**35.29**	**79.75**	89.09	84.31	24.62	**62.18**
w/o Λ	7.1E-2	47.50	17.65	65.82	80.00	72.55	12.88	49.40
V=0	7.1E-2	58.00	17.65	74.68	90.00	**86.27**	**26.52**	58.85
w/o Λ,V=0	7.1E-2	60.00	23.53	77.22	88.18	80.39	21.21	58.42
w/o rank adapt	7.8E-2	**61.50**	17.65	77.22	**90.91**	78.43	24.24	58.32
Mo-SARA	**8.5E-3**	55.00	23.53	70.89	90.91	**87.25**	**26.14**	58.95
w/o v	9.0E-3	49.00	25.49	69.62	80.00	75.49	18.18	52.88
v in front	**8.5E-3**	57.00	21.57	73.42	**91.82**	86.27	23.48	58.93

Table 5: Ablation of SARA and Mo-SARA methods on the mathematical reasoning tasks with LLaMA-7B.

our experiments, we use 0.5 as the threshold for determining k in the Mo-SARA method.

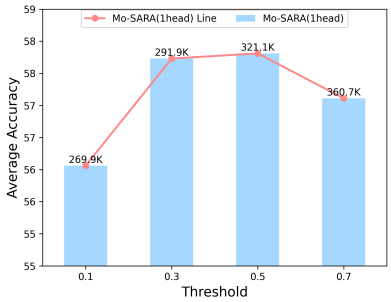

Figure 5: Average accuracy of Mo-SARA (1 head) on mathematical reasoning tasks under different thresholds τ, the bar chart displays the trainable parameters above.

Parallel Heads: We further explore the choice of parallel heads for the parallel structure, using soft routing to control 3, 5, 7, and 9 groups of parallel singular values and compare the results with that without parallel structure. As shown in Figure 6, the experimental results demonstrate a stable increase in performance as parallel heads increases, gradually approaching the results of the original LoRA method with nearly ten times the parameter count. Considering the balance between parameter count and performance, we adopt a structure with 5 parallel groups in main experiments.

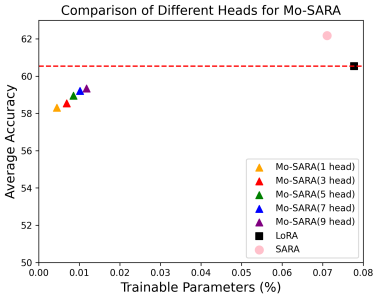

Figure 6: Average accuracy of Mo-SARA on mathematical reasoning tasks with different numbers of parallel heads, compared to SARA and LoRA methods.

5.8 Analysis of Routing Effects

To explore the effect of using mixture parallel structure in the Mo-SARA method, we employ the model trained on LLaMA-7B to extract the first question across various test tasks. The routing results of the first model pass are averaged across 'batch' and 'length' dimensions to obtain the routing's heatmap. Figure7 illustrates the routing results of the Mo-SARA method applied alongside the Q-matrix in mathematical reasoning and commonsense inference tasks. It is observed that for different tasks, the routing mechanism learns different allocation strategies, assigning different weights to each set of singular values, and each of them also learns the tasks it excels at. This indicates the role of the routing in assisting the Mo-SARA method in parallelizing and leveraging the entire singular value matrix for fine-tuning.

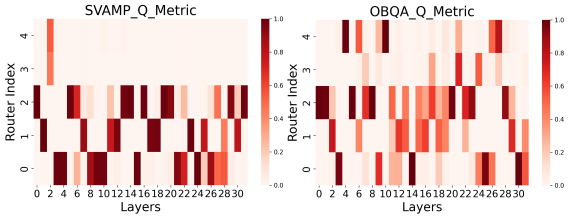

Figure 7: The heatmap of routing generated by the model trained with the Mo-SARA on mathematical and commonsense inference tasks through test tasks.

6 Conclution

In this work, we analyze the relationship between the SVD results of pre-trained model parameters and provide a new perspective for addressing the varying importance across layers. During the exploration, we propose an effective method, SARA, which can adaptively find the most suitable rank for each layer during initialization. We further introduce the Mo-SARA, which only fine-tunes the routing mechanism and the mixture of singular values, significantly reducing the trainable parameters. Various experiments on 15 datasets demonstrate our methods' higher performance while retaining the advantages of the LoRA method, advancing the field of PEFT by improving performance and largely reducing trainable parameters.

7 Limitation

Although our method retains the advantages of the LoRA method, allowing the additional parameter parts to be directly loaded alongside the original matrix without extra inference overhead, there is still a small time cost during training initialization. In the future, we will investigate methods to accelerate SVD decomposition to further speed up our model's training process. Meanwhile, our proposed Mo-SARA method adopts a mechanism similar to MoE(Jacobs et al., 1991), using a token-level soft routing approach for the gating mechanism, which selects all experts and performs a weighted sum based on the gating. Although we have not conducted extensive research on the choice of gating methods, we have already achieved excellent results as presented. In the future, we will study more MoE methods, to further explore the potential of PEFT methods with minimal parameter sizes.

8 Ethic Statement

The main purpose of this paper is to explore effective fine-tuning methods in low-resource scenarios. By using SVD, we investigate the relationship between pre-trained matrices and the performance of different layers in the model, and propose two efficient fine-tuning methods that significantly reduces the number of trainable parameters. All the models and datasets we used are open source, so we believe that the work in this paper does not pose any potential threats.

References

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

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. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar. Association for Computational Linguistics.

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

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

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2023. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language*

Processing (Volume 1: Long Papers), pages 4582–4597, Online. Association for Computational Linguistics.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

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*.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Steffen Schotthöfer, Emanuele Zangrando, Jonas Kusch, Gianluca Ceruti, and Francesco Tudisco. 2022. Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations. *Advances in Neural Information Processing Systems*, 35:20051–20063.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.

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

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

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287, Dubrovnik, Croatia. Association for Computational Linguistics.

Ben Wang and Aran Komatsuzaki. 2021. Gpt-j-6b: A 6 billion parameter autoregressive language model.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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*.

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

A Experimental Details

Data Usage: The datasets used in this paper come from the open-source work of previous research papers(Hu et al., 2023, 2021). For the mathematical reasoning tasks, all six datasets are combined by randomly selecting 80% of each, resulting in a total

of 3260 data points for training. Testing is then performed on the remaining data for each dataset. For commonsense inference tasks, 170k version of this work(Hu et al., 2023) are used for training, amalgamating the training datasets from all 8 subtasks to create this final training dataset, and testing is conducted on their individual testing dataset for each task. For the tasks above, during training and testing, a prompt is added to the data: 'Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.' For the E2E dataset, we directly adopte the training and testing datasets used in this work(Hu et al., 2021).

Hyperparameter Settings: In addition to the hyperparameters mentioned in the text experiments, all other experimental hyperparameters are consistent with those of the main experiment. The experimental hyperparameters of the main experiments for mathematical reasoning, commonsense inference, and E2E tasks are shown in Tables 6, 7and 8, respectively. The hyperparameters for most baseline experiments are based on references from (Hu et al., 2023) and 2(Hu et al., 2021), along with their provided open-source code.

All of our methods and PiSSA (Meng et al., 2024) are consistent with the original LoRA method (Hu et al., 2021), with the added matrices being parallel to the Q and V matrices. The random initialization mentioned in our method follows the Kaiming uniform approach.

Model Usage: In this paper, we utilize four models: LLaMA-7B/13B (Touvron et al., 2023), GPTJ-6B (Wang and Komatsuzaki, 2021), and GPT-2(Radford et al., 2019). All training and testing experiments are conducted using a single Nvidia A40, Nvidia RTX4090 or NVIDIA L20.

B Relationship between Layers and k under Different Thresholds.

We follow the method described in section 3 to calculate the k-values obtained from matrix SVD decomposition under different thresholds ranging from 0.1 to 0.9, observing the impact as the number of layers changes. The results for the Q and V matrices are shown in Figures 8, respectively. All k-values show a trend of initially decreasing around the eighth layer and then increasing as the model's depth increases, which is the opposite of the model performance trend with layer variation, consistent with what we mentioned in section 3.

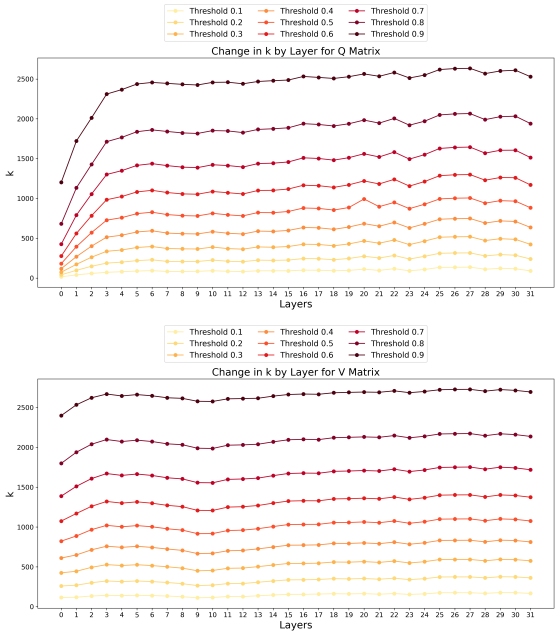

Figure 8: Average accuracy of the LoRA method on mathematical reasoning tasks at different λ scaling ratios compared to the SARA method.

C Analysis of the LoRA Method under Different λ Hyperparameters.

We modify the λ values in the LoRA method into four sets and conduct experiments using LLaMA-7B on the mathematical reasoning tasks. The experimental results are shown in the figure9. The original LoRA method is also sensitive to the λ hyperparameter values, yielding different results under the four different settings, all of which are lower than those obtained by our SARA method. This indicates that the LoRA method requires validation to find the optimal λ values for different tasks, while our approach, which replaces scaling with singular values, partially addresses this issue for adding singular values allows for a more fine-grained determination of the appropriate scaling factor

D Heatmaps of routing across layers for various test tasks using the Mo-SARA method.

The experiments for obtaining this heatmaps is consistent with that described in Section 5.8 of the paper.

The results from the following figures, Figure 10 and Figure 11 show that mathematical reason-

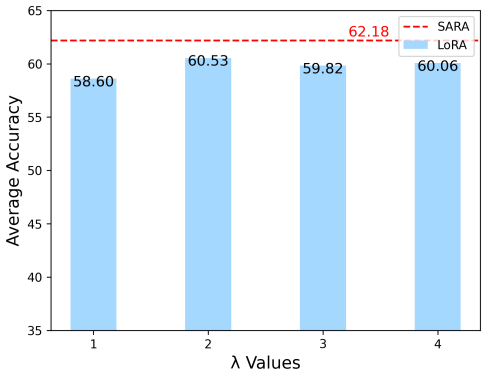

Figure 9: Average accuracy of the LoRA method on mathematical reasoning tasks at different λ values compared to the SARA method.

ing tasks and commonsense inference tasks exhibit similar routing distributions respectively, and for each layer, there is typically a predominant routing value. This indicates that different sets of singular values play similar roles across different test sets for models trained on the same training set, with each layer being dominated by a specific set of singular values.

E Supplementary Results for Each Dataset.

Specific results of the experimental supplements on each dataset are presented in the following table 9, 10,11, 12, 13 as shown.

F Scientific Artifacts

The datasets we use include the mathematical reasoning dataset SVAMP (Patel et al., 2021), AQuA (Ling et al., 2017), AddSub (Hosseini et al., 2014), MultiArith (Roy and Roth, 2016), the SingleEQ (Koncel-Kedziorski et al., 2015), GSM8K (Cobbe et al., 2021), and the commonsense inference dataset ARC (Clark et al., 2018), Boolq (Clark et al., 2019), WinoGrande (Sakaguchi et al., 2021), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), and OBQA (Mihaylov et al., 2018). The pre-trained models we utilize are LLaMA-7B/13B (Touvron et al., 2023), and GPT-J-6B (Wang and Komatsuzaki, 2021), as well as E2E task(Novikova et al., 2017). All the aforementioned datasets and models are open-source, and our work is solely for scientific research purposes, aligning with their original intent.

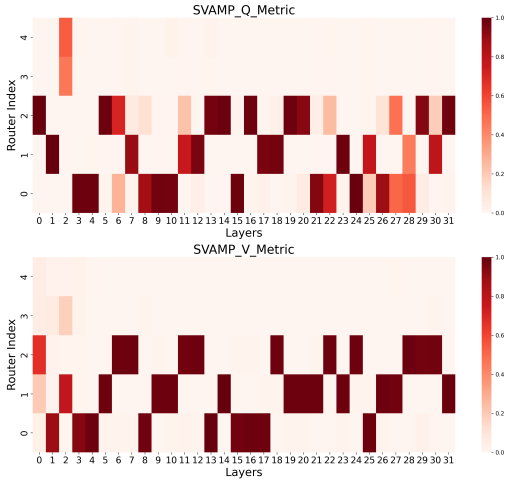

Figure 10: SVAMP Heatmaps.

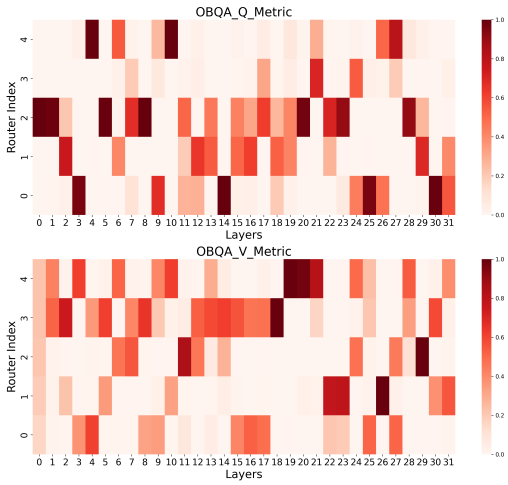

Figure 11: OBQA Heatmaps.

Hyperparameters	Prefix	LoRA	Adapter	Parallel	PiSSA	SARA	Mo-SARA
LLaMA-7B							
Rank r	-	10	-	-	10	-	-
λ	-	2	-	-	1	-	-
Virtual Tokens	30	-	-	-	-	-	-
Bottleneck Size	-	-	256	256	-	-	-
Thresholdτ	-	-	-	-	-	0.01	0.5
Parallel Heads	-	-	-	-	-	-	5
Dropout	0.05						
Optimizer	AdamW						
LR	3e-2	3e-4	3e-4	3e-4	3e-4	3e-3	3e-2
LR Scheduler	Linear						
Batch size	16						
Warmup Steps	100						
Epochs	3						
Training Seed	42						
LLaMA-13B							
Rank r	-	10	-	-	11	-	-
λ	-	2	-	-	1	-	-
Virtual Tokens	30	-	-	-	-	-	-
Bottleneck Size	-	-	256	256	-	-	-
Threshold τ	-	-	-	-	-	0.009	0.5
Parallel Heads	-	-	-	-	-	-	5
Dropout	0.05						
Optimizer	AdamW						
LR	3e-2	3e-4	3e-4	3e-4	3e-4	1e-2	3e-2
LR Scheduler	Linear						
Batch size	16						
Warmup Steps	100						
Epochs	3						
Training Seed	42						
GPT-J-6B							
Rank r	-	10	-	-	10	-	-
λ	-	2	-	-	1	-	-
Virtual Tokens	30	-	-	-	-	-	-
Bottleneck Size	-	-	256	256	-	-	-
Threshold τ	-	-	-	-	-	0.009	0.5
Parallel Heads	-	-	-	-	-	-	5
Dropout	0.05						
Optimizer	AdamW						
LR	3e-2	3e-4	3e-4	3e-4	3e-4	3e-3	3e-2
LR Scheduler	Linear						
Batch size	16						
Warmup Steps	100						
Epochs	3						
Training Seed	42						

Table 6: Hyperparameters for Mathematical Reasoning Tasks

Hyperparameters	LLaMA-7B			LLaMA-13B		
	PiSSA	**SARA**	**Mo-SARA**	**PiSSA**	**SARA**	**Mo-SARA**
Rank r	11	-	-	11	-	-
λ	1	-	-	1	-	-
Threshold τ	-	0.09	0.8	-	0.075	0.5
Parallel Heads	-	-	5	-	-	5
Dropout	0.05					
Optimizer	AdamW					
LR	3e-4	1e-3	3e-2	3e-4	1e-3	3e-2
LR Scheduler	Linear					
Batch size	16					
Warmup Steps	100					
Epochs	3					

Table 7: Hyperparameters for Commensense Inference Tasks

Hyperparameters	VeRA	SARA	Mo-SARA
Threshold τ	-	0.012	0.5
Parallel Heads	-	-	3
Optimizer	AdamW		
LR	1e-1	8e-3	7e-2
LR Scheduler	Linear		
Batch size	16		
Weight Decay	0.01		
Lable Smooth	0.1		
Rank	1024	-	-
LoRA α	1024	-	-
Warmup Steps	500		
Epochs	5		
Training Seed	314		

Table 8: Hyperparameters for E2E Task

Method	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
LoRA(0-7)	48.50	11.76	73.42	74.55	79.41	19.32	51.16
LoRA(8-15)	49.50	25.49	69.62	84.55	85.29	20.45	55.82
LoRA(16-23)	40.50	25.49	69.62	70.00	76.47	14.39	49.41
LoRA(24-31)	30.50	25.49	62.03	42.73	61.76	9.09	38.60
SARA(0-7)	56.00	29.41	73.42	71.82	82.35	21.59	55.77
SARA(8-15)	54.00	27.45	74.68	77.27	82.35	22.73	56.41
SARA(16-23)	43.50	31.37	74.68	82.73	73.53	13.64	53.24
SARA(24-31)	37.00	15.69	68.35	64.55	63.73	6.06	42.56

Table 9: Supplement to the average accuracy of SARA and LoRA methods across different layers in mathematical reason- ing tasks(Figure 3).

Method	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
LoRA(r=5)	51.50	23.53	73.42	90.91	87.25	23.48	58.35
LoRA(r=10)	58.50	23.53	75.95	92.73	88.24	24.24	60.53
LoRA(r=15)	60.00	17.65	78.48	93.64	86.27	23.86	59.98
LoRA(r=20)	58.50	19.61	79.75	89.09	87.25	26.14	60.06
SARA(0.006)	55.00	19.61	74.68	85.45	88.24	28.41	58.57
SARA(0.01)	60.00	35.29	79.75	89.09	84.31	24.62	62.18
SARA(0.016)	61.50	23.53	78.48	89.09	82.35	26.52	60.24
SARA(0.02)	59.50	25.49	82.28	85.45	84.31	24.62	60.28

Table 10: Supplement to the average accuracy of the SARA and LoRA methods on mathematical reasoning tasks with different trainable parameter counts.(Figure 4)

Method	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
Threshold=0.1	51.50	27.45	69.62	84.55	82.35	23.86	56.56
Threshold=0.3	55.00	25.49	77.22	85.45	82.35	23.86	58.23
Threshold=0.5	56.00	23.53	73.42	89.09	84.31	23.48	58.31
Threshold=0.7	56.50	15.69	73.42	90.91	85.29	23.86	57.61

Table 11: Supplement to the average accuracy of Mo-SARA (1 head) on mathematical reasoning tasks under different thresholds.(Figure5)

Method	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
Mo-SARA(1 head)	56.00	23.53	73.42	89.09	84.31	23.48	58.31
Mo-SARA(3 head)	54.50	21.57	75.95	89.09	85.29	23.86	58.38
Mo-SARA(5 head)	55.00	23.53	70.89	90.91	87.25	26.14	58.95
Mo-SARA(7 head)	55.50	23.53	75.95	90.00	85.29	25.00	59.21
Mo-SARA(9 head)	53.00	25.49	78.48	88.18	86.27	24.62	59.34

Table 12: Supplement to the average accuracy of Mo-SARA on mathematical reasoning tasks with different numbers of parallel heads, compared to SARA and LoRA methods.(Figure 6)

Method	SVAMP	AQuA	AddSub	MultiArith	SingleEQ	GSM8K	Avg.
λ**=1**	52.50	23.53	74.68	90.91	87.25	22.73	58.60
λ**=2**	58.50	23.53	75.95	92.73	88.24	24.24	60.53
λ**=3**	58.00	19.61	74.68	93.64	87.25	25.76	59.82
λ**=4**	58.00	21.57	74.68	93.64	88.24	24.24	60.06

Table 13: Supplement to the average accuracy of the LoRA method on mathematical reasoning tasks at different λ values(Figure 9)