SIBO: A Simple Booster for Parameter-Efficient Fine-Tuning

Zhihao Wen[1*] **Jie Zhang**[2*] **Yuan Fang**[1†]
[1]Singapore Management University
[2]National University of Singapore
zhwen.2019@smu.edu.sg, jiezhang_jz@u.nus.edu, yfang@smu.edu.sg

Abstract

Fine-tuning all parameters of large language models (LLMs) necessitates substantial computational power and extended time. Latest advancements in parameter-efficient fine-tuning (PEFT) techniques, such as Adapter tuning and LoRA, allow for adjustments to only a minor fraction of the parameters of these LLMs. Concurrently, it has been noted that the issue of over-smoothing diminishes the effectiveness of these Transformer-based LLMs, resulting in suboptimal performances in downstream tasks. In this paper, we present SIBO, which is a SImple BOoster to enhance PEFT, by injecting an *initial residual*. SIBO is straightforward and readily extensible to a range of state-of-the-art PEFT techniques to alleviate over-smoothing and enhance performance. Extensive experiments on 22 benchmark datasets demonstrate that SIBO significantly enhances the performance of various strong baselines, achieving up to 15.7% and 23.5% improvement over existing PEFT methods on the arithmetic and commonsense reasoning tasks, respectively.

1 Introduction

Many Transformer-based large language models (LLMs) exhibit significant depth, *e.g.*, BERT-large (Devlin et al., 2019) has 24 layers, LLaMA-7B (Touvron et al., 2023) has 32 layers, and LLaMA-65B has 80 layers. Yet, this depth presents a challenge (Zhou et al., 2021; Gong et al., 2021): Deep Transformers tend to encounter the *over-smoothing* problem. This issue, as detailed by Brunner et al. (2019), manifests in the deeper layers of Transformers, where token representations increasingly converge toward uniformity. The over-smoothness not only impedes the scalability of Transformer training, particularly in terms of depth, but also limits the efficacy of scaling up the model size. Consequently, expanding the model often results in marginal enhancements or, in some cases, reduced accuracy (Xue et al., 2023).

Meanwhile, a significant drawback of full-model fine-tuning for LLMs is that it requires updating all the parameters of the original model. While this constitutes a relatively minor limitation for models like BERT-large (Devlin et al., 2019) or RoBERTa-large (Liu et al., 2019), it escalates into a major obstacle for larger models such as LLaMA (Touvron et al., 2023), which contain billions of trainable parameters. Many approaches (Houlsby et al., 2019; Hu et al., 2022; Li and Liang, 2021; Lester et al., 2021) have been explored to address this issue by updating only a subset of the parameters or lightweight external modules tailored for new tasks. Such strategies require storing and loading a relatively small number of task-specific parameters alongside the pre-trained model for each task. These compelling alternatives to full-model fine-tuning are called *parameter-efficient fine-tuning* (PEFT) (Houlsby et al., 2019), which significantly enhances the feasibility of deploying LLMs.

Although some approaches have been proposed to deal with the over-smoothing problem, such as adding specifically designed regularization to avoid "uniform tokens" (Zhou et al., 2021; Gong et al., 2021) and fusing the representations from all layers (Shi et al., 2022), no PEFT method has yet been proposed to alleviate the over-smoothing issue. In the era of LLMs, when internal modifications to models are infeasible for most use cases, addressing the over-smoothing issue through PEFT techniques becomes critical.

Challenge and present work. Given that existing solutions to over-smoothing involve changes to the model architecture and are hence not parameter-efficient, the question arises: How can over-smoothing be effectively addressed for PEFT techniques? Two primary factors may contribute to the over-smoothing problem: 1) *redundancy* within the

[*]Co-first authors with equal contribution. † Corresponding author.

model's encoding layers, and 2) a *suboptimal training process* that hinders the effective optimization of the deeper layers. To address the first issue, a straightforward and logical solution is to reduce the number of layers in the encoder. However, this approach can result in a decline in performance (Chen et al., 2023). To address the second issue, previous approaches (Gong et al., 2021; Zhou et al., 2021; Shi et al., 2022) are not parameter-efficient, thereby limiting their application to LLMs.

To devise a flexible yet simple plug-and-play framework for alleviating over-smoothing with existing PEFT techniques, our idea boils down to injecting an initial residual into the PEFT input. This initial residual connection ensures that the final representation of each token preserves at least a minimum portion of the input layer's features, aiming to reduce the uniformity of the final token representations. We name the novel framework SIBO, a SImple BOoster to enhance PEFT techniques designed for LLMs, most notably Adapter (Houlsby et al., 2019) and LoRA (Hu et al., 2022).

Empirically, on the arithmetic reasoning task, SIBO outperforms Adapter and LoRA by up to 15.7% and 13.6%, respectively. On the commonsense reasoning task, the improvement is up to 7.6% over Adapter, and 23.5% over LoRA.

2 Preliminaries

In the following, we present a summary of two popular lines of PEFT techniques: adapters and reparametrization-based methods.

Adapters. Adapters fall into two distinct categories: parallel and serial adapters. Parallel adapters (He et al., 2021) integrate additional learnable modules alongside various layers of the core model. In contrast, series adapters (Houlsby et al., 2019) insert these modules sequentially between specific layers, *e.g.*, adding fully connected networks after both the attention and feed forward layers in the Transformer model. In this work, we focus on the classical serial adapter, which has the following general formulation:

$$\mathbf{h} \leftarrow \mathbf{h} + f(\mathbf{h}\mathbf{W}_{\text{down}})\mathbf{W}_{\text{up}}, \tag{1}$$

where $\mathbf{h} \in \mathbb{R}^{1\times d}$ represents the output of the preceding layer, after which the adapter is inserted. Consequently, $\mathbf{h}$ serves as the input to the adapter. It first undergoes a down-projection to a lower dimension r via $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d\times r}$, followed by an up-projection back to its original dimension d via $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times d}$. The function $f(\cdot)$ represents a nonlinear function.

Reparameterization-based methods. These methods are designed to modify network weights through a low-rank strategy. This technique effectively reduces the number of tunable parameters without compromising performance. For example, Low-Rank Adaptation (LoRA) approximates the update $\Delta\mathbf{W}$ to a pre-trained weight matrix $\mathbf{W} \in \mathbb{R}^{d\times d}$ through a low-rank decomposition:

$$\mathbf{h} \leftarrow \mathbf{h}(\mathbf{W} + s \cdot \mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}), \tag{2}$$

where $\mathbf{h} \in \mathbb{R}^{d}$ is the output of the preceding layer, and $\mathbf{W} \in \mathbb{R}^{d\times d}$ is a pre-trained weight matrix, *e.g.*, for multilayer perceptron (MLP) or attention layers. The matrices $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d\times r}$ and $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times d}$ are lower-rank matrices to approximate the update, *i.e.*, $\Delta\mathbf{W} \approx \mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$. Here, $r \ll d$ serves as a crucial hyperparameter for LoRA, while the scalar $s \geq 1$ is an adjustable hyperparameter.

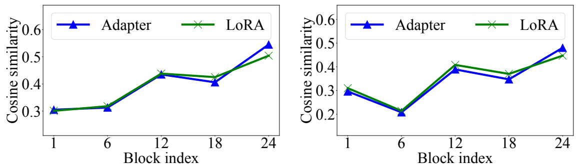
3 Methodology

In this section, we first analyze the over-smoothing issue in PEFT techniques, and subsequently present our proposed framework, SIBO.

3.1 Over-smoothing in PEFT

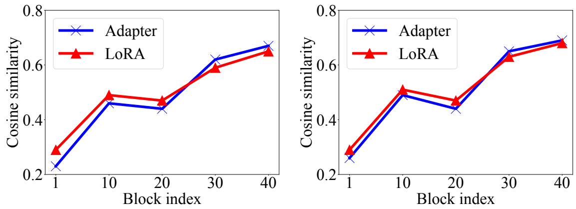
Originating from graph neural networks, the term *over-smoothing* denotes a decline in performance attributed to the increasing homogeneity of node representations (Li et al., 2018; Xu et al., 2018; Huang et al., 2020), stemming from the repetitive use of the same adjacency matrix in successive aggregation layers. Shi et al. (2022) have since identified an over-smoothing phenomenon in language models as well, wherein distinct tokens in an input sentence exhibit increasingly similar representations as more layers are stacked, diminishing the effectiveness of deep Transformer models.

While several strategies have been proposed to mitigate over-smoothing (Zhou et al., 2021; Gong et al., 2021; Shi et al., 2022), they are not designed for PEFT techniques, making them less practical for LLMs. In particular, we also observe over-smoothing in widely adopted PEFT techniques including adapters and LoRA, especially with deep layers, through quantitative analysis. In our analysis, over-smoothing can be detected by assessing the similarity among tokens in the same sentence, known as *token-wise cosine similarity*. Given a sentence consisting of m tokens, represented by



(a) CoLA (b) STS-B

Figure 1: Over-smoothing in PEFT. The results are the averaged token-wise similarity of sentences in the test sets of the corpora in the GLUE benchmark (Wang et al., 2018), with BERT-large as the backbone.



(a) MAWPS (b) SVAMP

Figure 2: Over-smoothing in PEFT. The results are the averaged token-wise similarity of sentences in the test sets of MAWPS (Koncel-Kedziorski et al., 2016) and SVAMP (Patel et al., 2021), with LLaMA (13B) as the backbone.

$(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m)$, its token-wise cosine similarity is computed as

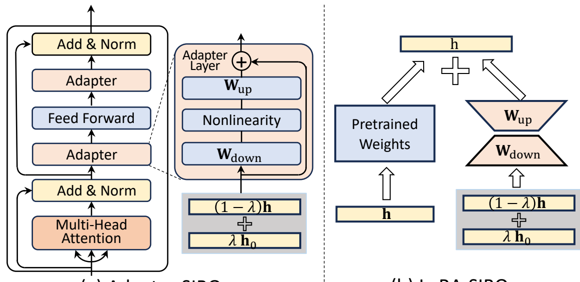
$$\frac{1}{m(m-1)} \sum_{i \neq j} \frac{\mathbf{h}_i^\top \mathbf{h}_j}{\|\mathbf{h}_i\|_2 \|\mathbf{h}_j\|_2}, \tag{3}$$

where $\|\cdot\|_2$ is the Euclidean norm. As shown in Figs. 1 and 2, with both adepters and LoRA, an increase in token-wise similarity is noted consistently as the layer depth in the backbone language model increases. Hence, the issue of over-smoothing also persists in pre-trained language models that have undergone adaptation via PEFT techniques. Therefore, it is imperative to devise a general framework that eases over-smoothing for PEFT methods, while retaining their efficiency.

3.2 Initial residual integration

To achieve a universal plug-and-play enhancement for PEFT, we start with the input to the PEFT module, and inject an initial residual into the input at each layer of the pre-trained model.

Let the initial token representation serving as input to the pre-trained model be denoted by $\mathbf{h}_0 \in \mathbb{R}^d$. Integrating an initial residual from $\mathbf{h}_0$ guarantees that the final representation of each token preserves at least a λ portion of the information



(a) Adapter-SIBO (b) LoRA-SIBO

Figure 3: Proposed framework SIBO, applying to two popular PEFT methods: (a) Adapter, and (b) LoRA.

from the input layer. Here, $0 < \lambda < 1$ is a crucial factor when multiple layers are involved. Practically, we treat λ as a hyperparameter, and setting it to a reasonable value, such as 0.2, ensures that the final token representation incorporates a substantial part of the input token feature, thereby reducing over-smoothness throughout the layers. We present a theoretical analysis in Appendix A. In the following, we illustrate how our proposed framework, SIBO, can be applied to Adapter and LoRA, two most popular PEFT techniques.

Adapter-SIBO. Implementing the initial residual injection for adapters is straightforward. As illustrated in Fig. 3(a), SIBO adds the initial token representation $\mathbf{h}_0$ to a hidden state $\mathbf{h}$ at the entry point of the adapter (*i.e.*, output from the preceding layer and input to the adapter), within each Transformer layer. This process is executed through a basic vector addition operation as follows.

$$\begin{aligned} \mathbf{h} \leftarrow & \tilde{\mathbf{h}} + f(\tilde{\mathbf{h}}\mathbf{W}_{\text{down}})\mathbf{W}_{\text{up}} \\ & \text{s.t. } \tilde{\mathbf{h}} = (1-\lambda)\mathbf{h} + \lambda\mathbf{h}_0, \end{aligned} \tag{4}$$

where $0 < \lambda < 1$ is a hyper-parameter used to control the strength of the initial residual.

LoRA-SIBO. In each LoRA module at every Transformer layer, the input to its update, $\Delta\mathbf{W}$, is solely the hidden state $\mathbf{h}$ from the preceding layer, with $\Delta\mathbf{W}$ being approximated by low-rank matrices. In LoRA-SIBO, we introduce a modification to the input to $\Delta\mathbf{W}$, which becomes a combination of $\mathbf{h}$ and $\mathbf{h}_0$, as follows.

$$\begin{aligned} \mathbf{h} \leftarrow & \mathbf{h}\mathbf{W} + s \cdot \tilde{\mathbf{h}}\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}} \\ & \text{s.t. } \tilde{\mathbf{h}} = (1-\lambda)\mathbf{h} + \lambda\mathbf{h}_0. \end{aligned} \tag{5}$$

4 Experiments

In this section, we perform extensive experiments across various benchmarks and language models.

4.1 Datasets

Our study encompasses a thorough empirical examination of 22 benchmark datasets, categorized into three distinct problem areas as follows.

Arithmetic reasoning. (1) GSM8K (Cobbe et al., 2021): linguistically varied grade-school math word problems created by skilled problem writers. (2) AQuA (Ling et al., 2017): algebraic word problems with natural language explanations. (3) MAWPS (Koncel-Kedziorski et al., 2016): a variety of arithmetic and algebra word problems of different complexities. (4) SVAMP (Patel et al., 2021): arithmetic word problems aimed at students up to the 4th grade, derived by making minor modifications to an existing problem set. PEFT techniques adopt a supervised fine-tuning (SFT) setting (Ouyang et al., 2022), where the supervision is derived from Math10K (Hu et al., 2023), which comprises the training sets of GSM8K, AQuA, and MAWPS. The pre-trained model is fine-tuned on the examples in Math10K to replicate their styles and characteristics.

Commonsense reasoning. (1) BoolQ (Clark et al., 2019): Yes/no questions, originating from natural, unrestricted environments. (2) PIQA (Bisk et al., 2020): Questions requiring physical commonsense for resolving two possible solutions. (3) SIQA (Sap et al., 2019): Questions focusing on the understanding the social implications of human actions. (4) HellaSwag: Commonsense natural language inference questions with various endings to complete a given context. (5) WinoGrande (Sakaguchi et al., 2021): A fill-in-the-blank task with binary choices, demanding commonsense reasoning to select the appropriate option. (6) ARC-c and (7) ARC-e (Clark et al., 2018): The Challenge and Easy sets of the ARC dataset, featuring genuine grade-school level science questions in multiple-choice format. (8) OBQA: Questions that necessitate multi-step reasoning, additional common and commonsense knowledge, and comprehensive text understanding. To perform SFT, we employ a training set named Commonsense170K (Hu et al., 2023), which is tailored for enhancing commonsense reasoning capabilities. It includes the training sets from the above eight commonsense reasoning datasets.

A summary of the datasets on arithmetic and commonsense reasoning is presented in Table 1.

Dataset	Domain	# train	# test	Answer
GSM8K	Math	8.8K	1,319	Number
AQuA	Math	100K	254	Option
MAWPS	Math	1.9k	238	Number
SVAMP	Math	-	1,000	Number
BoolQ	CS	9.4K	3,270	Yes/No
PIQA	CS	16.1K	1,838	Option
SIQA	CS	33.4K	1,954	Option
HellaSwag	CS	39.9K	10,042	Option
WinoGrande	CS	63.2K	1,267	Option
ARC-e	CS	1.1K	2,376	Option
ARC-c	CS	2.3K	1,172	Option
OBQA	CS	5.0K	500	Option

Table 1: Datasets on arithmetic reasoning (Math) or commonsense reasoning (CS).

GLUE. The General Language Understanding Evaluation Benchmark (Wang et al., 2018) encompasses eight corpora for various natural language understanding tasks: CoLA, SST-2, MRPC, STS-B, QQP, MNLI, QNLI, and RTE.

4.2 Implementations

Arithmetic and commonsense reasoning. We use LLaMA (7B, 13B) (Touvron et al., 2023) and GPT-J (6B) (Wang and Komatsuzaki, 2021) as the foundational models, which are designed for natural language generation tasks. We choose Adapter (Houlsby et al., 2019) and LoRA (Hu et al., 2022) as baselines, and follow previous work (Hu et al., 2023) for the experimental setup and hyperparameters. In particular, for Adapter, we integrate it into the feed-forward layers with a bottleneck size of 256; for LoRA, we incorporate it into both the multi-head attention and feed-forward layers with rank 32. For Adapter-SIBO and LoRA-SIBO, we inject the initial residual into the modules at the feed-forward layers only, and choose $\lambda \in \{0.1, 0.2, 0.3\}$ empirically while retaining other settings in the vanilla Adapter and LoRA. More details on the experimental setup can be found in Appendix D.

GLUE. We use BERT-large (Devlin et al., 2019) as the backbone. While larger models have recently surpassed BERT on the GLUE benchmark, BERT continues to be favored for its efficiency. Moreover, it is relatively easy to perform full-model fine-tuning (FT) on BERT, enabling a direct comparison between FT and PEFT techniques. For Adapter, we apply the typical setting (Houlsby et al., 2019) where adapter layers are added after the multi-head attention and feed-forward layers; for LoRA, we follow previous work (Hu et al., 2022) and apply to weights $\mathbf{W}_q$ and $\mathbf{W}_v$ with rank 8. For Adapter-SIBO, we inject the initial residual to the adapter modules after the self-attention layers; for LoRA-

PLM	PEFT method	# Params. tuned	GSM8K	AQuA	MAWPS	SVAMP	Overall	Improv.
GPT-3.5 (175B)*	–	–	56.4	38.9	87.4	69.9	63.2	–
GPT-J (6B)	Adapter*	112M	14.3	20.5	62.2	38.1	33.8	–
	Adapter-SIBO	112M	19.0	18.9	72.7	45.9	39.1	15.7%
	LoRA*	35M	17.4	21.3	70.2	41.0	37.5	–
	LoRA-SIBO	35M	22.4	20.5	77.7	49.7	42.6	13.6%
LLaMA (7B)	Adapter*	128M	33.3	15.0	77.7	52.3	44.6	–
	Adapter-SIBO	128M	33.1	18.9	80.3	48.0	45.1	1.1%
	LoRA*	40M	37.5	18.9	79.0	52.1	46.9	–
	LoRA-SIBO	40M	37.8	18.5	82.8	50.7	47.5	1.3%
LLaMA (13B)	Adapter*	200M	44.0	22.0	78.6	50.8	48.9	–
	Adapter-SIBO	200M	43.2	22.4	82.4	52.9	50.2	2.7%
	LoRA*	62.5M	47.5	18.5	83.6	54.6	51.1	–
	LoRA-SIBO	62.5M	47.0	20.5	84.0	57.6	52.3	2.3%

Table 2: Performance of LLMs with different PEFT methods on arithmetic reasoning, using GPT-3.5 with zero-shot CoT as a reference point. * indicates results from prior work by Hu et al. (2023), where the exact same experimental setup and evaluation protocols are adopted. Improvement is calculated relative to the counterpart without SIBO.

Method	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Overall	Improv.
Adapter*	62.1	63.5	72.3	30.6	68.0	63.9	48.1	63.8	59.0	–
Adapter-SIBO	62.2	73.0	73.0	48.7	67.8	65.5	51.9	65.6	63.5	7.6%
LoRA*	62.4	68.6	49.5	43.1	57.3	43.4	31.0	46.6	50.2	–
LoRA-SIBO	63.9	70.3	71.0	47.8	67.2	63.3	48.7	63.8	62.0	23.5%

Table 3: Performance of GPT-J (6B) with different PEFT methods on commonsense reasoning. * indicates results from prior work (Hu et al., 2023), where the exact same experimental setup and evaluation protocols are adopted.

SIBO, we inject the initial residual to all LoRA modules. For both SIBO approaches, we choose $\lambda \in \{0.1, 0.2, \ldots, 0.7\}$ empirically while following previous work (Houlsby et al., 2019; Hu et al., 2022) to set other hyperparameters.

4.3 Performance comparison

We evaluate the performance of SIBO in comparison to baselines across the three problem areas.

Arithmetic reasoning. We compare the performance of Adapter and LoRA with or without SIBO, by performing PEFT on the pre-trained LLaMA and GPT-J models using the Math10K dataset. We then test the fine-tuned models across the test set of the four math reasoning datasets. As a standard reference (Hu et al., 2023), we further compare to the GPT-3.5 model (text-Davinci-003 version), which employs zero-shot Chain of Thought (CoT) (Kojima et al., 2022).

As reported in Table 2, the 175B-parameter GPT-3.5 model demonstrates superior accuracy over other LLMs. Despite this, LoRA-SIBO applied on LLaMA (13B) has reached a performance level comparable to that of GPT-3.5 with only a small gap. Compared to the counterparts without SIBO, SIBO has achieved notable improvements: 2.3%–2.7% on LLaMA (13B), and 1.1%–1.3% on LLaMA (7B). The smaller improvements on the 7B model can be attributed to the less pronounced over-smoothing issue in smaller models with fewer layers, indirectly underscoring the necessity to address over-smoothing in deeper models. Meanwhile, SIBO achieves up to 15.7% improvement on the weaker GPT-J, significantly reducing the gap from LLaMA (7B).

Moreover, we observe enhancements by SIBO in both in- and out-of-distribution scenarios. The dataset utilized for fine-tuning, Math10K, encompasses the training sets from GSM8K, AQuA, and MAWPS, excluding SVAMP. It can be observed that SIBO not only enhances the performance of PEFT methods on the first three datasets in an in-distribution setting, but also extends the improvements to SVAMP, an out-of-distribution scenario, demonstrating the robustness and generalizability of our methodology.

Commonsense reasoning. Next, we investigate the performance of SIBO for commonsense reasoning tasks. Table 3 presents a comparative analysis of the PEFT methods applied to GPT-J (6B). It is evident that SIBO consistently and significantly enhances the performance of Adapter and LoRA across eight diverse corpora/tasks, with average improvements ranging between 7.6% and 23.5%.

GLUE. Lastly, we present the results on the GLUE benchmark in Table 4, using BERT-large as the backbone model. SIBO consistently outperforms the vanilla PEFT methods across eight diverse

Method	# Params. tuned	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Overall
FT*	340.0M	62.8	94.1	91.9	89.8	87.6	86.5	93.5	71.8	84.8
Adapter	6.0M	$62.1_{\pm1.1}$	$93.7_{\pm0.2}$	$90.4_{\pm0.5}$	$90.2_{\pm0.2}$	$88.3_{\pm0.3}$	$85.9_{\pm0.1}$	$92.2_{\pm0.2}$	$71.5_{\pm2.4}$	84.3
Adapter-SIBO	6.0M	$63.1_{\pm1.2}$	$94.6_{\pm0.2}$	$90.9_{\pm0.1}$	$90.2_{\pm0.1}$	$88.3_{\pm0.2}$	$86.0_{\pm0.1}$	$92.4_{\pm0.3}$	$73.2_{\pm1.1}$	84.8
LoRA	0.8M	$60.1_{\pm1.0}$	$93.6_{\pm0.3}$	$90.3_{\pm0.3}$	$89.6_{\pm0.1}$	$87.8_{\pm0.1}$	$85.5_{\pm0.1}$	$92.1_{\pm0.3}$	$71.1_{\pm0.8}$	83.8
LoRA-SIBO	0.8M	$61.6_{\pm0.8}$	$93.8_{\pm0.2}$	$90.8_{\pm0.1}$	$89.9_{\pm0.1}$	$87.7_{\pm0.2}$	$85.6_{\pm0.2}$	$92.2_{\pm0.2}$	$71.8_{\pm1.8}$	84.2

Table 4: Performance of BERT-large with different PEFT methods on the GLUE benchmark. * indicates results from prior work (Zaken et al., 2022), where the exact same experimental setup and evaluation protocols are adopted. We report mean (and standard deviation) of the performance over three different runs.

datasets/tasks. Notably, the effectiveness of the Adapter-SIBO even matches that of full-model fine-tuning (FT). More experimental results using an alternative pre-trained model, RoBERTa-large, are in Appendix B.

4.4 Analyses

In this section, we first analyze the optimal placement of the initial residual for Adapter and LoRA. Following that, we examine the effect of the sole hyperparameter λ we introduced. Then, we explore the overhead incurred by SIBO. Finally, we visualize the role of the initial residual in mitigating the over-smoothing issue, and present a case study. In these studies, we employ BERT-large as the backbone on the CoLA and STS-B datasets.

Placement. In this section, we investigate the placement of initial residual for PEFT modules.

For Adapter, each Transformer layer employs two adapter modules, positioned respectively after the attention layer (ATT) and the feed-forward layer (FFN). The question arises: which position is more suitable for the injection of the initial residual? As shown in Table 5, injecting the initial residual solely at the ATT position achieves almost identical performance to that at the FFN position. However, injecting initial residuals at both ATT and FFN results in a slight decline in performance. This suggests that injecting the initial residual once per Transformer layer is sufficient, as excessive injections can introduce noises.

For LoRA, each module involves two types of parameters: frozen pre-trained weights and learnable low-rank matrices. We explore whether it is necessary to inject the initial residual into both or solely into the low-rank matrices. Table 6 reveals that injecting the initial residual only into the learnable low-rank matrices yields better results. A potential reason is that frozen weights do not integrate well with the layer's hidden state and the initial residual.

Impact of λ. SIBO only introduces one new hyperparameter, λ, which balances the trade-off between

Placement	CoLA	STS-B	Average
ATT	$63.1_{\pm1.2}$	$90.2_{\pm0.2}$	76.7
FFN	$63.3_{\pm0.6}$	$90.1_{\pm0.1}$	76.7
Both	$61.8_{\pm1.5}$	$90.0_{\pm0.1}$	75.9

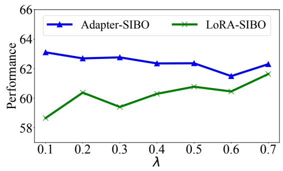
Table 5: Initial residual placement for Adapter.

‘ Placement	CoLA	STS-B	Average
Low-rank matrices	$61.6_{\pm0.8}$	$89.9_{\pm0.1}$	75.8
+ Pre-trained weights	$61.0_{\pm0.8}$	$89.7_{\pm0.3}$	75.1

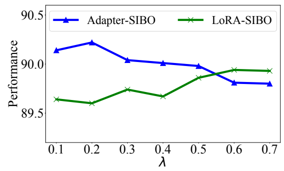
Table 6: Initial residual placement for LoRA.

the hidden state and the initial residual. The selection of λ also guarantees the minimum portion of input features preserved in the final token representation, directly mitigating the over-smoothing issue. Hence, we investigate the optimal value for λ, varying it between 0.1 and 0.7. As illustrated in Fig. 4, for Adapter, a lower λ value, such as 0.2, is generally more effective. While it is crucial to ensure that the final representation of each token maintains a minimum portion of λ from the input layer across multiple stacked layers, this proportion should not be excessively large to avoid compromising the learning capacity of Adapter. For LoRA, the hidden state is fed to both the pre-trained weights and the low-rank matrices, implying that the effect of λ ratio is naturally "halved". In other words, a λ value of 0.6 for LoRA is roughly equivalent to a λ value of 0.3 for Adapter. Therefore, the optimal value of λ in LoRA is larger than in Adapter, occurring around 0.6–0.7.

Despite the effort to select λ, our approach remains pragmatic and resource-conscious. For smaller models like BERT, extensive tuning over a large range of values for λ is feasible. In the context of larger models, such as LLaMA (13B), a smaller range of values $\lambda \in \{0.1, 0.2, 0.3\}$ have been considered in our tuning, which still yields significant improvements across various tasks, as shown in Tables 7 and 8. Notably, many non-optimal values of λ could still result in significant performance gains, underscoring the efficacy and robustness of



(a) CoLA



(b) STS-B

Figure 4: Impact of initial residual portion λ.

Methods	GSM8k	AQuA	MAWPS	SVAMP	Avg.
Adapter	**44.0**	22.0	78.6	50.8	48.9
SIBO ($\lambda = 0.1$)	42.7	17.3	**83.6**	**55.9**	49.9
SIBO ($\lambda = 0.2$)	42.3	20.1	81.9	55.3	49.9
SIBO ($\lambda = 0.3$)	43.2	**22.4**	82.4	52.9	**50.2**

Table 7: Impact of λ, with LLaMA (13B) as the backbone, and adapter as the PEFT method.

Methods	GSM8k	AQuA	MAWPS	SVAMP	Avg.
LoRA	47.5	18.5	83.6	54.6	51.1
SIBO ($\lambda = 0.1$)	47.0	20.5	**84.0**	57.6	**52.3**
SIBO ($\lambda = 0.2$)	46.6	19.3	**84.0**	**57.8**	51.9
SIBO ($\lambda = 0.3$)	**47.8**	**21.3**	83.2	52.9	51.3

Table 8: Impact of λ, with LLaMA (13B) as the backbone, and LoRA as the PEFT method.

Methods	# Params.	FLOPs	CoLA Time (s)	STS-B Time (s)
Adapter	6.0M	6,291,456	$108.3_{\pm 0.9}$	$82.7_{\pm 0.6}$
Adapter-SIBO	6.0M	6,389,760	$110.0_{\pm 1.0}$	$80.7_{\pm 1.5}$
LoRA	0.8M	835,584	$86.3_{\pm 0.6}$	$55.7_{\pm 0.5}$
LoRA-SIBO	0.8M	884,736	$90.0_{\pm 1.0}$	$52.7_{\pm 7.5}$

Table 9: Complexity analysis. Time includes fine-tuning one epoch and then testing, averaged over three runs.

SIBO without extensive hyperparameter tuning.

Complexity. SIBO is remarkably efficient, involving only an additional summation operation with the initial residual vector at each Transformer layer, without introducing any extra parameter. To demonstrate its efficiency, we compare the number of floating point operations (FLOPs) and the wall-clock time for fine-tuning and testing. As shown in Table 9, the overhead of summing the initial residual vector only marginally increases the FLOPs. Moreover, the wall-clock time is almost identical to that of the vanilla PEFT methods, which does not include the initial residual, highlighting the simplicity and efficiency of SIBO.

Visualizations of over-smoothing. The thesis of the work is to employ the initial residual to alleviate over-smoothing. To examine whether SIBO effectively reduces over-smoothing, we conduct experiments comparing the token-wise cosine similarity, as defined in Eq. 3, in the last five layers of the language model after applying PEFT methods with or without SIBO. As observed in Figs. 5 and 6, the token-wise similarity generally decreases when SIBO is applied alongside Adapter and LoRA. In essence, SIBO has lessened the degree of over-smoothing, leading to better task performance.

Qualitative case study. Finally, we supplement our quantitative findings with qualitative analysis in a case study. Table 10 presents a question sampled from SVAMP, showcasing responses from ChatGPT 3.5, as well as LLaMA (13B) using various PEFT methods. While ChatGPT is generally robust, it is not infallible and has provided an incorrect answer in this instance. Similarly, the answers generated by Adapter and LoRA were found to be erroneous, mainly due to their initial confusion between the related concepts of `cracker` and `snack`.

In contrast, LoRA-SIBO shows an improvement by correctly distinguishing between the two concepts at least in the initial phase of reasoning. Adapter-SIBO outperforms all by maintaining clarity on the two concepts throughout, leading to the correct answer. This improvement underscores the benefit of addressing over-smoothness, which enhances word discrimination and thus provides a clearer distinction between closely related concepts such as `cracker` and `snack`, improving overall performance. Moreover, the responses from Adapter-SIBO and LoRA-SIBO appear more detailed and of higher quality than those from the vanilla Adapter and LoRA. Further qualitative examples are presented in Appendix C.

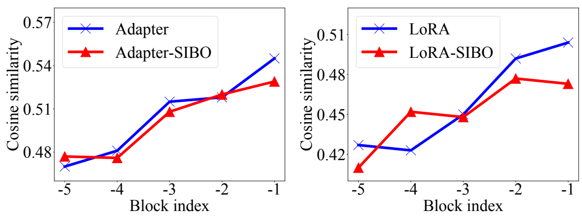
5 Related work

We focus on the discussion of parameter-efficient fine-tuning (PEFT) of pre-trained language models. There exists three main categories of methods, including prompt-based learning, adapters, and reparametrization methods. However, none of the existing approaches is designed to alleviate the over-smoothing issue.

Prompt-based learning extends the identification of the ideal discrete (hard) prompt into the optimization of a continuous (soft) prompt instead. Lester et al. (2021) have introduced the idea of prompt tuning, which involves attaching a trainable tensor as a prefix to the input embeddings. Similarly, Li and Liang (2021) have developed an independent method known as prefix tuning, which integrates soft prompts into the hidden states across all layers. Another technique, Intrinsic Prompt Tun-

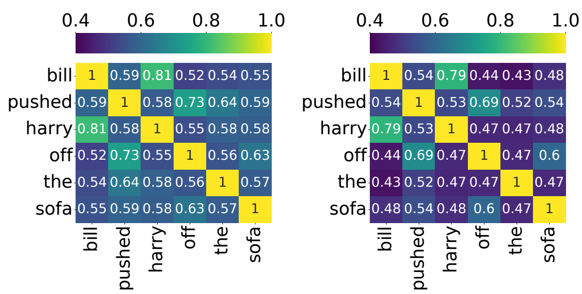
Question	Matthew gave equal numbers of crackers and cakes to his 4 friends. If he had 32 crackers and 98 cakes initially. How many crackers did each person eat?
ChatGPT*	To calculate the number of crackers each person ate, we need to find the number of crackers that each friend received initially and subtract it from the total number of crackers Matthew had. Here are the steps: 1. Divide the total number of crackers by the number of friends (4) to find the number of crackers each friend received initially: 32 / 4 = 8 crackers per friend. 2. Since we want to find the number of crackers each person ate, we need to subtract the initial number of crackers per friend from the total number of crackers: 32 - 8 = 24 crackers.
LLaMA (13B) + Adapter*	1. Divide the total number of crackers by the number of friends: 32 / 4 = 8 2. Divide the total number of cakes by the number of friends: 98 / 4 = 24 3. Add the number of crackers and cakes each friend received: 8 + 24 = 32 4. Therefore, each person ate 32 crackers and 24 cakes. Answer: Each person ate 32 crackers.
LLaMA (13B) + Adapter-SIBO	1. Find the total number of crackers and cakes: 32 + 98 = 130 2. Divide the total number of crackers by the number of friends: 32 ÷ 4 = 8 3. Divide the total number of cakes by the number of friends: 98 ÷ 4 = 22 4. Since Matthew gave equal numbers of crackers and cakes, each friend ate the same number of crackers and cakes. 5. Therefore, each person ate 8 crackers and 22 cakes. **Therefore, each person ate 8 crackers.**
LLaMA (13B) + LoRA*	1. Divide the total number of crackers by the number of friends: 32 / 4 = 8 2. Divide the total number of cakes by the number of friends: 98 / 4 = 24 3. Add the number of crackers and cakes each person received: 8 + 24 = 32
LLaMA (13B) + LoRA-SIBO	1. Find the total number of friends Matthew gave crackers and cakes to: 4 friends 2. Find the total number of crackers and cakes Matthew gave away: 32 crackers + 98 cakes = 130 snacks 3. Divide the total number of snacks by the number of friends: 130 snacks ÷ 4 friends = 32.5 snacks per person 4. Round down to the nearest whole number since Matthew can't give away a fraction of a cracker: 32 crackers per person Therefore, each person ate 32 crackers.

Table 10: An example question from SVAMP, with the ground-truth answer **8**. We show the output of ChatGPT and various PEFT methods on LLaMA (13B). * indicates results from prior work (Hu et al., 2023).



(a) Adapter (b) LoRA

Figure 5: Token-wise similarity in last five layers computed from PEFT methods, with and without SIBO.



(a) LoRA (b) LoRA-SIBO

Figure 6: Heatmap of token-wise similarity in the last layer computed from LoRA and LoRA-SIBO, on a sentence randomly sampled from the test set of CoLA.

ing (Qin et al., 2021), utilizes an autoencoder to both compress and decompress the soft prompt at the cost of limiting the sequence length.

Adapters exist in parallel and serial forms. Parallel adapters (He et al., 2021) integrate additional learnable modules alongside various layers of the core model. A different strategy, termed Ladder Side-Tuning (Sung et al., 2022), focuses on developing a streamlined auxiliary network akin to a ladder. This auxiliary network receives intermediate activations from the main network via direct shortcut pathways, referred to as ladders. In contrast, serial adapters insert these modules sequentially between specific layers. Houlsby et al. (2019) add fully connected networks after both the attention and feed-forward layers in the Transformer model. Pfeiffer et al. (2020) have demonstrated that inserting an adapter only after the self-attention layer can yield performance comparable to using two adapters per transformer block, whereas AdaMix Wang et al. (2022) employs multiple serial adapters in a mixture-of-experts approach. To further reduce computational complexity while preserving performance, Compacter (Karimi Mahabadi et al., 2021) leverages the Kronecker product, low-rank matrices, and parameter sharing across layers for adapter weight generation.

Finally, reparametrization-based methods are designed to modify network weights through a low-rank approximation. This technique effectively minimizes the number of trainable parameters without compromising the representational capacity of high-dimensional matrices. The work on Intrinsic SAID (Aghajanyan et al., 2021) examines the essential dimensionality of fine-tuning within a low-rank framework. On the other hand, LoRA (Hu et al., 2022) models its update to a pre-trained weight matrix through a low-rank decomposition. Building on this, Edalati et al. (2022) enhance the matrix decomposition feature of LoRA by incorporating the Kronecker product into their method.

6 Conclusion

We present a novel framework SIBO, a SImple BOoster to enhance parameter-efficient fine-tuning (PEFT) techniques for large pre-trained language models. Our core idea revolves around mitigating the over-smoothing issue, which is achieved by injecting an *initial residual* into various PEFT modules at specific positions within the pre-trained models. SIBO is straightforward and readily extensible to various state-of-the-art PEFT methods including Adapter and LoRA. Extensive experiments on 22 benchmark datasets across three problem areas demonstrate that SIBO effectively mitigate over-smoothing and significantly improves the performance of existing PEFT techniques.

7 Limitations

Our method is straightforward and effective, yet it has one limitation: selecting the optimal value for the hyperparameter λ requires time and computational resources. This cost is manageable given that we only introduced one new hyperparameter, especially for medium-sized models. However, it may become prohibitive for very large models. A viable solution is to transform this hyperparameter into a continuous learnable parameter, allowing the model to autonomously determine the optimal weight for the initial residual.

Acknowledgements

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funds (Grant No. A20H6b0151). The authors wish to thank Dr. Lei Wang from Singapore Management University and Mr. Zhiqiang Hu from the Singapore University of Technology and Design for their valuable support of this work.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Annual Meeting of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 7319–7328.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, pages 7432–7439.

Gino Brunner, Yang Liu, Damian Pascual, Oliver Richter, Massimiliano Ciaramita, and Roger Wattenhofer. 2019. On identifiability in transformers. In *International Conference on Learning Representations (ICLR)*.

Nuo Chen, Linjun Shou, Ming Gong, Jian Pei, Bowen Cao, Jianhui Chang, Daxin Jiang, and Jia Li. 2023. Alleviating over-smoothing for unsupervised sentence representation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3552–3566.

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 *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2924–2936.

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*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186.

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

Chengyue Gong, Dilin Wang, Meng Li, Vikas Chandra, and Qiang Liu. 2021. Vision transformers with patch diversification. *arXiv preprint arXiv:2104.12753*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations (ICLR)*.

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 (ICML)*, pages 2790–2799.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.

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*.

Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. 2020. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*.

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

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 (NeurIPS)*, 35:22199–22213.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1152–1157.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3045–3059.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI conference on Artificial Intelligence*, pages 3538–3545.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 4582–4597.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:27730–27744.

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

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673.

Yujia Qin, Xiaozhi Wang, Yusheng Su, Yankai Lin, Ning Ding, Jing Yi, Weize Chen, Zhiyuan Liu, Juanzi Li, Lei Hou, et al. 2021. Exploring universal intrinsic task subspace via prompt tuning. *arXiv preprint arXiv:2110.07867*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An adversarial Winograd Schema Challenge at scale. *Communications of the ACM (CACM)*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social iqa: Commonsense reasoning about social interactions. In *Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473.

Han Shi, Jiahui Gao, Hang Xu, Xiaodan Liang, Zhenguo Li, Lingpeng Kong, Stephen Lee, and James T Kwok.

2022. Revisiting over-smoothing in bert from the perspective of graph. In *International Conference on Learning Represenations (ICLR)*.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:12991–13005.

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*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*.

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

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. AdaMix: Mixture-of-adapter for parameter-efficient tuning of large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5744–5760.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning (ICML)*, pages 5453–5462.

Fuzhao Xue, Jianghai Chen, Aixin Sun, Xiaozhe Ren, Zangwei Zheng, Xiaoxin He, Yongming Chen, Xin Jiang, and Yang You. 2023. A study on transformer configuration and training objective. In *International Conference on Machine Learning (ICML)*, pages 38913–38925.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–9.

Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. 2021. DeepViT: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*.

Appendices

A Theoretical analysis

To theoretically analyze that SIBO preserves a portion of the original token information and mitigates the tendency of the token vectors becoming similar across Transformer layers, let us delve into the mathematical details.

Theorem: *Preservation of token uniqueness in Transformer layers via initial residual connections.*

Given: Assume the following conditions. (1) A Transformer model with L layers, each performing a self-attention mechanism followed by a position-wise feedforward network. (2) An initial token representation $\mathbf{h}_0 \in \mathbb{R}^d$ for any token in the input sequence. (3) An initial residual connection that injects $\mathbf{h}_0$ at each layer of the Transformer, modulated by a parameter $\lambda \in (0, 1]$, ensuring that each layer's output includes at least λ proportion of $\mathbf{h}_0$.

Claim: *For any layer l, $1 \leq l \leq L$, the output representation $\mathbf{h}_l \in \mathbb{R}^d$ of any token satisfies the following condition*:

$$\mathbf{h}_l = \lambda \mathbf{h}_0 + (1 - \lambda)\mathbf{F}_l(\mathbf{h}_0, \mathbf{H}_{l-1}),$$

where $\mathbf{H}_{l-1} \in \mathbb{R}^{n \times d}$ represents the matrix of token vectors at layer $l - 1$ and $\mathbf{F}_l$ denotes the transformation function of layer l, which includes self-attention and feed-forward network operations.

Proof Sketch:

1. **Base Case**: For $l = 1$, the claim trivially holds by the definition of the initial residual connection.

2. **Inductive Step**: Assume the claim holds for layer $l - 1$. Then, for layer l, by the properties of linear transformations in self-attention and feedforward networks, along with the Fourier Transform's linearity, we can represent $\mathbf{F}_l$ as a combination of these operations applied to $\mathbf{h}_0$ and the residual information from $\mathbf{H}_{l-1}$.

   Since the self-attention mechanism aggregates information across tokens modulated by attention weights and the feedforward network applies position-wise transformations, the output of layer l can be represented as a linear combination of the input and transformations applied up to that layer, weighted by λ and $1 - \lambda$ respectively.

3. **Fourier perspective**: The Fourier transform of the token representations facilitates the analysis of how frequency components are preserved or altered through layers. The modulation by λ ensures that a minimum portion of the original frequency spectrum of $\mathbf{h}_0$ is preserved in each layer's output, mitigating the over-smoothing effect observed as L increases.

4. **Conclusion**: By induction, we conclude that each layer's output preserves at least a λ portion of the initial token representation $\mathbf{h}_0$, in addition to contributions from the transformations applied within the Transformer network.

Implications. This theorem demonstrates that through the use of initial residual connections modulated by λ, it is possible to quantitatively ensure that each token's representation in a Transformer model retains a significant portion of its original unique information, thereby reducing the tendency of token vectors to become overly similar across layers. This approach offers a theoretical foundation for mitigating the over-smoothing problem in deep Transformer models, supporting the preservation of information diversity and richness in token representations through the network's depth.

B Using RoBERTa as backbone on GLUE

In the GLUE benchmark, to investigate the generalization ability and robustness of our model, we have also experimented with another popular backbone, RoBERTa. As shown in Table 11, our approach consistently enhances the performance of the PEFT method across different backbones, demonstrating the robustness of our method.

C Qualitative results

Regarding the qualitative results, we present not only outcomes related to a randomly selected question from SVAMP but also results for several questions randomly chosen from MAWPS. These findings are detailed in Tables 12, 13, and 14.

D Environment and settings

Our experimental environment utilizes servers equipped with A40 GPUs and AMD EPYC 7543 CPUs, running PyTorch version 2.0.0. Both our training and inference processes are conducted on a single GPU.

Detailed hyperparameter settings are shown in Tables 15, 16, 17, 18, and 19.

Method	# Param.	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Overall
FT*	355.0M	68.0	96.4	90.9	92.4	92.2	90.2	94.7	86.6	88.9
Adapter*	6.0M	66.5	96.2	88.7	91.0	92.1	89.9	94.7	83.4	87.8
Adapter-SIBO	6.0M	$67.1_{\pm 0.7}$	$96.6_{\pm 0.1}$	$90.8_{\pm 0.1}$	$92.1_{\pm 0.1}$	$91.9_{\pm 1.4}$	$89.9_{\pm 0.1}$	$94.8_{\pm 0.1}$	$84.8_{\pm 0.4}$	88.5
LoRA*	0.8M	68.2	96.2	90.2	92.3	91.6	90.6	94.8	85.2	88.6
LoRA-SIBO	0.8M	$68.5_{\pm 1.6}$	$96.4_{\pm 0.1}$	$90.4_{\pm 0.6}$	$92.4_{\pm 0.2}$	$91.6_{\pm 0.2}$	$90.5_{\pm 0.3}$	$95.0_{\pm 0.2}$	$85.2_{\pm 0.3}$	88.7

Table 11: Performance of RoBERTa-large with different PEFT methods on the GLUE benchmark. * indicates results from prior work (Hu et al., 2022), where the exact same experimental setup and evaluation protocols are adopted. We report mean (and standard deviation) of the performance over three different runs.

Question	`There are 7 crayons in the drawer. Mary took 3 crayons out of the drawer. How many crayons are there now?`
ChatGPT*	`To calculate the number of crayons remaining in the drawer after Mary took some out, follow these steps: Subtract the number of crayons Mary took from the total number of crayons in the drawer: 7 - 3 = 4 crayons.` `Therefore, there are now 4 crayons remaining in the drawer.`

LLaMA (13B) + Adapter*	`1. Start with the initial number of crayons in the drawer: 7`
	`2. Subtract the number of crayons Mary took out: 3`
	`3. Perform the subtraction: 7 - 3 = 4`
	`4. Therefore, there are now 4 crayons in the drawer.`
	`Answer: There are now 4 crayons in the drawer.`

LLaMA (13B) + Adapter-SIBO	`1. Start with the initial number of crayons in the drawer: 7`
	`2. Subtract the number of crayons Mary took out: 3`
	`3. Perform the subtraction: 7 - 3 = 4`
	`Therefore, there are now 4 crayons in the drawer. The answer in Arabic numerals is 4.`

LLaMA (13B) + LoRA*	`1. Start with the initial number of crayons: 7`
	`2. Subtract the number of crayons Mary took out: 7 - 3 = 4`
	`3. The final answer is 4 crayons.`
	`Therefore, there are now 4 crayons in the drawer.`

LLaMA (13B) + LoRA-SIBO	`1. Start with the initial number of crayons in the drawer: 7`
	`2. Subtract the number of crayons Mary took out: 3`
	`3. Perform the subtraction: 7 - 3 = 4`
	`4. The final answer is 4 crayons.`
	`Therefore, there are now 4 crayons in the drawer.`

Table 12: An example question from MAWPS, with the ground-truth answer **4**. We show the output of ChatGPT and various PEFT methods on LLaMA (13B). * indicates results from prior work (Hu et al., 2023).

Question	`At the schools book fair Sam bought 13 adventure books and 17 mystery books. If 15 of the books were used, how many new books did he buy?`

ChatGPT*	`To calculate the number of mosaic tiles needed to cover two-thirds of Mr. Boarden's 36 square foot bathroom, follow these steps:`
	`Find two-thirds of 36: Multiply 36 by 2/3.`
	`(36 * 2/3) = 24 square feet`
	`Determine the number of mosaic tiles needed per square foot: Given that Mr. Boarden needs 24 mosaic tiles per square foot, multiply the result from step 1 by 24.`
	`(24 * 24) = 576 mosaic tiles`
	`Therefore, Mr. Boarden would need 576 mosaic tiles to cover two-thirds of his 36 square foot bathroom.`

LLaMA (13B) + Adapter*	`1. Find the total number of books Sam bought: 13 + 17 = 30`
	`2. Subtract the number of used books from the total number of books: 30 - 15 = 15`
	`3. Therefore, Sam bought 15 new books.`
	`Answer: 15`

LLaMA (13B) + Adapter-SIBO	`1. Find the total number of books Sam bought: 13 + 17 = 30`
	`2. Subtract the number of used books from the total: 30 - 15 = 15`
	`3. Therefore, Sam bought 15 new books.`
	`Answer: 15 new books.`

LLaMA (13B) + LoRA*	`1. Find the total number of books Sam bought: 13 + 17 = 30`
	`2. Subtract the number of used books from the total number of books: 30 - 15 = 15`
	`3. Therefore, Sam bought 15 new books.`
	`Answer: 15`

LLaMA (13B) + LoRA-SIBO	`1. Find the total number of books Sam bought:`
	`13 + 17 = 30`
	`2. Subtract the number of used books from the total number of books:`
	`30 - 15 = 15`
	`Therefore, Sam bought 15 new books.`

Table 13: An example question from MAWPS, with the ground-truth answer **15**. We show the output of ChatGPT and various PEFT methods on LLaMA (13B). * indicates results from prior work (Hu et al., 2023).

Question	Oceanside Bike Rental Shop charges 17 dollars plus 7 dollars an hour for renting a bike. Tom paid 80 dollars to rent a bike. How many hours did he pay to have the bike checked out?
ChatGPT*	To calculate the number of hours Tom paid to have the bike checked out, follow these steps: 1. Subtract the base charge from the total amount Tom paid: 80 - 17 = 63 dollars. 2. Divide the remaining amount by the hourly rate to find the number of hours: 63 / 7 = 9 hours. Therefore, Tom paid to have the bike checked out for 9 hours. Note: It's important to note that the result represents the whole number of hours. If there were any partial hours, they would not be reflected in the calculation.
LLaMA (13B) + Adapter*	1. Find the total cost of renting the bike: $17 + $7/hour = $24/hour $80 ÷ $24/hour = 3.33 hours 3. Round up to the nearest whole number since you can't rent a bike for a fraction of an hour: 3 hours Therefore, Tom paid to have the bike checked out for 3 hours.
LLaMA (13B) + Adapter-SIBO	1. Subtract the initial charge of $17 from the total cost of $80: $80 - $17 = $63 2. Divide the remaining cost by the hourly rate of $7: $63 ÷ $7 = 9 Therefore, Tom paid to have the bike checked out for 9 hours. The answer in Arabic numerals is 9.
LLaMA (13B) + LoRA*	1. Let x be the number of hours Tom paid to have the bike checked out. 2. The cost of renting the bike for x hours is 7x dollars. 3. The total cost of renting the bike for x hours is 17 + 7x dollars. 4. We know that Tom paid 80 dollars to rent the bike, so we can set up an equation: 17 + 7x = 80 5. Solving for x, we can subtract 17 from both sides: 7x = 63 6. Dividing both sides by 7, we get x = 9 Therefore, Tom paid to have the bike checked out for 9 hours.
LLaMA (13B) + LoRA-SIBO	1. Start with the total amount Tom paid for renting the bike: 80 dollars 2. Subtract the initial charge of 17 dollars: 80 - 17 = 63 dollars 3. Divide the remaining amount by the hourly rate of 7 dollars: 63 ÷ 7 = 9 4. The answer is 9, which represents the number of hours Tom paid to have the bike checked out. Therefore, Tom paid to have the bike checked out for 9 hours.

Table 14: An example question from MAWPS, with the ground-truth answer **9**. We show the output of ChatGPT and various PEFT methods on LLaMA (13B). * indicates results from prior work (Hu et al., 2023).

Method	Dataset	GSM8K	AQuA	MAWPS	SVAMP
	Optimizer	AdamW			
	Warmup Ratio	0.06			
	LR Schedule	Linear			
Adapter	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	Bottleneck r	256			
	Max Seq. Len.	256			
Adapter-SIBO	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	Bottleneck r	256			
	Max Seq. Len.	256			
	λ	0.1			
LoRA	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	LoRA Config.	$r_q = r_k = r_v = 32$			
	LoRA α	64			
	Max Seq. Len.	256			
LoRA-SIBO	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	LoRA Config.	$r_q = r_k = r_v = 32$			
	LoRA α	64			
	Max Seq. Len.	256			
	λ	0.1			

Table 15: Hyperparameters for the arithmetic reasoning experiments, using GPT-J (6B) as the pre-trained model.

Method	Dataset	GSM8K	AQuA	MAWPS	SVAMP
	Optimizer	AdamW			
	Warmup Ratio	0.06			
	LR Schedule	Linear			
Adapter	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	Bottleneck r	256			
	Max Seq. Len.	256			
Adapter-SIBO	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	Bottleneck r	256			
	Max Seq. Len.	256			
	λ	0.1			
LoRA	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	LoRA Config.	$r_q = r_k = r_v = 32$			
	LoRA α	64			
	Max Seq. Len.	256			
LoRA-SIBO	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	LoRA Config.	$r_q = r_k = r_v = 32$			
	LoRA α	64			
	Max Seq. Len.	256			
	λ	0.2			

Table 16: Hyperparameters for the arithmetic reasoning experiments, using LLaMA (7B) as the pre-trained model.

Method	Dataset	GSM8K	AQuA	MAWPS	SVAMP
	Optimizer	AdamW			
	Warmup Ratio	0.06			
	LR Schedule	Linear			
Adapter	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	Bottleneck r	256			
	Max Seq. Len.	256			
Adapter-SIBO	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	Bottleneck r	256			
	Max Seq. Len.	256			
	λ	0.3			
LoRA	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	LoRA Config.	$r_q = r_k = r_v = 32$			
	LoRA α	64			
	Max Seq. Len.	256			
LoRA-SIBO	Batch Size	16			
	Micro batch size	4			
	# Epochs	3			
	Learning Rate	3e-4			
	LoRA Config.	$r_q = r_k = r_v = 32$			
	LoRA α	64			
	Max Seq. Len.	256			
	λ	0.1			

Table 17: Hyperparameters for the arithmetic reasoning experiments, using LLaMA (13B) as the pre-trained model.

Method	Dataset	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
	Optimizer	AdamW							
	Warmup Ratio	0.06							
	LR Schedule	Linear							
Adapter	Batch Size	16							
	Micro batch size	4							
	# Epochs	3							
	Learning Rate	3e-4							
	Bottleneck r	256							
	Max Seq.Len.	256							
Adapter-SIBO	Batch Size	16							
	Micro batch size	4							
	# Epochs	3							
	Learning Rate	3e-4							
	Bottleneck r	256							
	Max Seq.Len.	256							
	λ	0.1							
LoRA	Batch Size	16							
	Micro batch size	4							
	# Epochs	3							
	Learning Rate	3e-4							
	LoRA Config.	$r_q = r_k = r_v = 32$							
	LoRA α	64							
	Max Seq.Len.	256							
LoRA-SIBO	Batch Size	16							
	Micro batch size	4							
	# Epochs	3							
	Learning Rate	3e-4							
	LoRA Config.	$r_q = r_k = r_v = 32$							
	LoRA α	64							
	Max Seq.Len.	256							
	λ	0.3							

Table 18: Hyperparameters for the commonsense reasoning experiments, using GPT-J (6B) as the pre-trained model.

Method	Dataset	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE
	Optimizer	AdamW							
	Warmup Ratio	0.06							
	LR Schedule	Linear							
Adapter	Batch Size	32							
	# Epochs	20	10	20	10	20	10	10	20
	Learning Rate	2e-4	4e-4	3e-4	2e-4	3e-4	3e-4	2e-4	4e-4
	Bottleneck r	64							
	Max Seq. Len.	128	128	128	128	128	128	128	64
Adapter-SIBO	Batch Size	32							
	# Epochs	20	10	20	10	20	10	10	20
	Learning Rate	2e-4	4e-4	3e-4	2e-4	3e-4	3e-4	2e-4	4e-4
	Bottleneck r	64							
	Max Seq.Len.	128							
	λ	0.1	0.1	0.1	0.3	0.1	0.2	0.2	0.3
LoRA	Batch Size	32							
	# Epochs	20	10	20	10	20	10	10	20
	Learning Rate	2e-4	4e-4	3e-4	2e-4	3e-4	3e-4	2e-4	4e-4
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	16							
	Max Seq. Len.	128							
LoRA-SIBO	Batch Size	32							
	# Epochs	20	10	20	10	20	10	10	20
	Learning Rate	2e-4	4e-4	3e-4	2e-4	3e-4	3e-4	2e-4	4e-4
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	16							
	Max Seq. Len.	128							
	λ	0.7	0.4	0.5	0.6	0.1	0.1	0.1	0.3

Table 19: Hyperparameters for the GLUE benchmark experiments, using BERT-large as the pre-trained model.