SSMLoRA: Enhancing Low-Rank Adaptation with State Space Model

Anonymous ACL submission

Abstract

Fine-tuning is a key approach for adapting language models to specific downstream tasks, but updating all model parameters becomes impractical as model sizes increase. Parameter-Efficient Fine-Tuning (PEFT) methods, such as Low-Rank Adaptation (LoRA), address this challenge by introducing additional adaptation parameters into pre-trained weight matrices. However, LoRA's performance varies across different insertion points within the model, highlighting potential parameter inefficiency due to unnecessary insertions. To this end, we propose SSMLoRA (**S**tate **S**pace **M**odel **L**ow-**R**ank **A**daptation), an extension of LoRA that incorporates a State Space Model (SSM) to interconnect low-rank matrices. SSMLoRA ensures that performance is maintained even with sparser insertions. SSMLoRA allows the model to not only map inputs to a low-rank space for better feature extraction but also leverage the computations from the previous low-rank space. Our method achieves comparable performance to LoRA on the General Language Understanding Evaluation (GLUE) benchmark while using only half the parameters. Additionally, due to its structure, SSMLoRA shows promise in handling tasks with longer input sequences.

1 Introduction

Fine-tuning (Park and Lee, 2021) is a technique aimed at enhancing model performance by adjusting the parameters of a pretrained model on specific task data. This approach effectively leverages the knowledge accumulated through large-scale training, accelerating the model's adaptation to new tasks and improving results. However, fine-tuning requires substantial computational resources and is prone to overfitting when applied to small datasets. These limitations have prompted researchers to explore more efficient parameter adjustment strategies.

To address the challenges of resource consumption and overfitting in fine-tuning, Parameter-Efficient Fine-Tuning (PEFT) (Houlsby et al., 2019) techniques, which are a set of methods designed to optimize parameter updates, have been proposed. PEFT updates only a small subset of the model's parameters while keeping the majority of pretrained parameters unchanged. LoRA (Hu et al., 2022), one of the most commonly used PEFT techniques, introduces low-rank matrices to adjust specific weights in the pretrained model instead of updating all parameters comprehensively. This allows the model to retain most of its structure while adapting to downstream tasks, particularly for transformer-based deep language models. Following the successful application of LoRA, various extensions have emerged to further enhance its adaptability and efficiency. These methods aim to improve the model's capability to perform well in resource-constrained environments.

Despite its success, LoRA and its variants still face certain challenges. SoRA (Ding et al., 2023) has indicated that LoRA may lead to unnecessary parameter overhead. More efficient methods, such as pruning certain low-rank matrices and using smaller scaling factors to reduce parameter usage, were employed. Moreover, compared to full fine-tuning, parameter-efficient methods tend to have limitations in capturing information, exacerbating the inherent difficulty that transformer-based models face when processing long texts. This leads to the observation that, for many tasks, the number of parameters used by LoRA is excessive, while for tasks involving long-text processing in large language models, it is insufficient.

To address this issue, we propose State Space Model Low-Rank Adaptation (SSMLoRA), a novel technique that builds on the state-space equation framework to insert sparse low-rank matrices. We introduce interval layers and apply our technique selectively to the query and value matrices within the attention mechanism, as this design significantly reduces parameter overhead. Additionally,

we incorporate a State Space Model to connect low-rank matrices inserted into the same type of neural network layers. State Space Model, which excels at handling long-text tasks, is based on state transitions but differs from traditional RNNs by enabling parallel computation through FFT-based transformations of the state transitions. This enhancement facilitates parallel training in our approach. Experimental results demonstrate that our technique performs well when applied to both linear and convolutional layers. Compared to LoRA, SSMLoRA achieves superior performance across a wide range of tasks while requiring only half of the parameters. Furthermore, it excels in long-text and certain other specific tasks. We believe that this represents a new direction in the exploration of LoRA-derived techniques, adopting a more holistic and macro-level perspective.

2 Related Works

2.1 PEFT

Parameter-Efficient Fine-Tuning (PEFT) (Houlsby et al., 2019) aims to reduce the computational and storage costs associated with fine-tuning large language models (LLMs). By tuning only a small subset of additional parameters while freezing the majority of pretrained model parameters, PEFT effectively mitigates the catastrophic forgetting issue often encountered in full fine-tuning. Moreover, PEFT outperforms traditional fine-tuning methods in low-data environments, demonstrating superior generalization capability. This technique is not limited to natural language processing tasks but extends to computer vision and audio domains, significantly enhancing the model's flexibility and adaptability. LoRA, as one of the most widely adopted PEFT techniques, adjusts specific pretrained model weights by introducing low-rank matrices, rather than updating all parameters, making it well-suited to downstream tasks. LoRA has shown exceptional adaptability and efficiency in large language models, allowing them to swiftly adapt to new tasks while preserving most of their original structure. The success of LoRA has sparked numerous further studies in the field.

AdaLoRA (Zhang et al., 2023), for instance, introduces orthogonal regularization to ensure that the low-rank projection matrices comply with Singular Value Decomposition (SVD), thus avoiding the reliance on incremental updates, albeit at the cost of increased computational complexity. QLoRA (Dettmers et al., 2023) improves computational efficiency and reduces resource consumption through dynamic quantization and advanced strategies, though it may potentially impact model accuracy. DoRA (Liu et al., 2024) decomposes pretrained weights into magnitude and direction components, utilizing LoRA for directional updates, reducing trainable parameters and enhancing fine-tuning performance, though its complexity and dependence on data quality may limit its effectiveness. Vera (Kopiczko et al., 2024), on the other hand, reduces parameter overhead by sharing low-rank matrices and learning small-scale vectors, while maintaining performance, though the model's complexity and hyperparameter tuning requirements might hinder overall effectiveness.

2.2 State Space Model

State Space Model (SSM) are mathematical models used to describe dynamic systems, comprising an input sequence $x(t)$, a latent state representation, and an output sequence $y(t)$. The primary objective of an SSM is to predict future states based on the current input and previous states. In recent years, researchers have made significant progress in applying SSMs to neural network architectures. Compared to traditional Recurrent Neural Networks (RNNs), SSM-based models are capable of resolving the issue of sequential processing bottlenecks, thus significantly improving training efficiency. Models based on SSMs, such as HIPPO (Hierarchy of Integrators with Projection and Partial Orthogonalization (Gu et al., 2020), have been shown to capture complex temporal dependencies more effectively. S4 (Structured State Space Sequence Model) (Gu et al., 2022) leveraged SSM architecture to outperform transformers and other architectures in handling ultra-long sequences. Mamba (Gu and Dao, 2023), an extension of SSM, introduced a selection mechanism that dynamically adjusts the model based on input, thereby improving adaptability. However, despite these advantages, the generalization capabilities of SSMs across various domains remain to be fully validated. To address this, we propose incorporating SSM into fine-tuning methodologies. By improving the existing LoRA (Low-Rank Adaptation) technique, we aim to enhance fine-tuning performance on specific tasks while preserving the Transformer architecture. This leads to our proposed method, SSMLoRA, which combines modified SSM equa-

tions with LoRA. Compared to traditional LoRA and its variants, SSMLoRA strengthens the connections between inserted low-rank matrices, improving data matching capabilities and achieving comparable or superior performance across most datasets.

2.3 Sparsification Methods

Sparsification methods aim to reduce computational complexity and memory requirements by decreasing the number of model parameters while maintaining performance as much as possible. These techniques include weight pruning, structured sparsity, dynamic sparse training, and sparse regularization. Wen et al. (2016) proposed Structured Sparsity Learning (SSL), a technique capable of learning efficient and compact structures from large deep neural networks, significantly enhancing model speed and performance. Anwar et al. (2015) optimized convolutional neural networks through structured pruning, effectively reducing overall memory demand. Xie et al. (2018) proposed IGCV2, which applied interleaved structured sparsity to convolutional layers. These studies demonstrate the effectiveness and generalizability of employing sparser structures. In our work, we aim to improve the parameter utilization of low-rank matrices while significantly reducing memory requirements and improving computational efficiency without sacrificing performance. This provides vital support for subsequent model optimization and deployment.

3 SSMLoRA

3.1 Time Module

In the original LoRA technique, the core concept involves introducing two new projection matrices, W_a and W_b. Let W_0 represent the weight matrix of the pre-trained model at a given layer, and let the input vector x have a dimensionality of d. After scaling, the input vector, denoted as x_{new}, is mapped to a lower-dimensional space with a rank r. The projection matrices W_a and W_b, with dimensions $d \times r$ and $r \times d$ respectively, project x to a lower dimension and then back to its original dimension, ensuring that the output dimensionality matches the input. Throughout this process, the parameters of the pre-trained model remain frozen. As a result, compared to training the original W_0, the adaptation to downstream tasks is achieved by training only the small matrices W_a and W_b, significantly reducing computational costs. The model is able to fit downstream tasks within the low-rank space. Therefore, after introducing these two matrices, the model's output can be expressed as:

$$y = x \times W_0 + x \times W_a \times W_b \tag{1}$$

In our method, we introduce a new module called the Time Module, represented by each orange square in the dashed box in Figure 1. Each Time Module contains the two projection matrices W_a and W_b from the original LoRA technique, as well as two additional $r \times r$ matrices, W_c and W_d, whose functions will be described later. In each Time Module, we receive the input x along with the time state h_{t-1} calculated by the previous module. The Time Module computes the output to adjust the behavior of the pre-trained model and updates the state to h_t, which is then passed to the next Time Module. Our modules can be inserted into various neural network layers, such as linear layers, convolutional layers, and multi-head attention mechanisms, and operate independently of the pre-trained model.

3.2 TimeAxis Update

In SSMLoRA, the core innovation, similar to LoRA, lies in the integration of the Time Module with specific pre-trained layers. Before discussing how the Time Module propagates forward, it is essential to understand how it retrieves the state computed by the previous Time Module and updates it. In the context of State Space Model, the matrix W_c within the Time Module is commonly referred to as the state matrix, while W_d is known as the control matrix. These matrices are used to combine the previous state and update the current state in the time cycle, which is particularly effective for extracting information from long sequences. Building on this concept, we follow the principles of the S4 model, utilizing these two matrices to update the state from the previous time step h_{t-1} to the current state h_t, and propagate it forward. Specifically, the process is as follows:

First, we compute the derivative of h_{t-1} at the current position using the following formula, which is based on the classical State Space Model (SSM) equations, producing the derivative of the current state:

$$h'_t = h_t \times W_c + x_{\text{new}} \times W_d \tag{2}$$

This equation corresponds to the first addition operation in the Time Module structure shown in

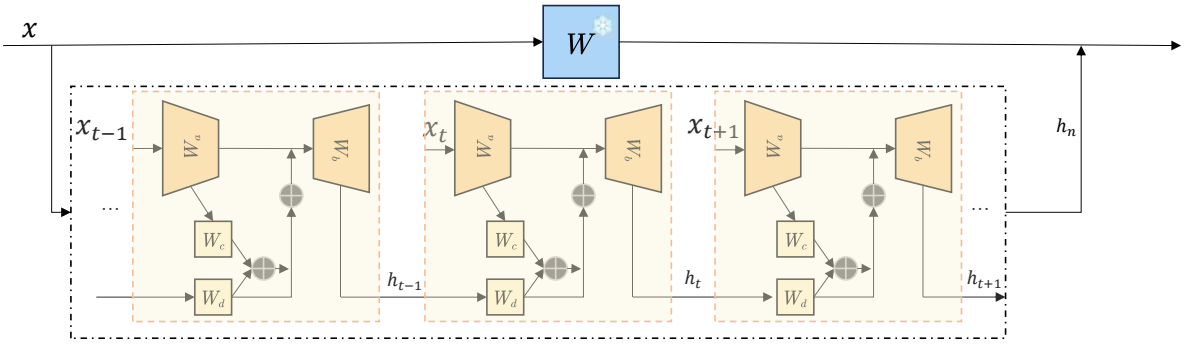

Figure 1: This image illustrates how our technique is applied to the original pre-trained model. In the lower part of the image, the dashed box contains three orange blocks, each representing a Time module. W refers to a layer from the original pre-trained matrix, which could be a linear layer, convolutional layer, or query, key, and value matrices from the self-attention mechanism. When SSMLoRA fine-tuning is applied to a specific type of neural network layer, these layers are connected along a time axis. Each layer is assigned a position on this axis and linked to a corresponding Time module that adjusts its output. Here, t denotes the position in the time axis, and n represents the maximum sequence length.

Figure 1. Next, unlike the S4 or SSM methods, we use a Taylor expansion to calculate the increment for the next time step, which corresponds to the second addition operation in Figure 1:

$$h_{t+1} = h'_t + h_t \tag{3}$$

Thus, the complete computation process can be expressed as follows:

$$h_{t+1} = h_t \times W_c + x_{\text{new}} \times W_d + h_t \tag{4}$$

It is important to note that for the Time Module at the beginning of the time axis, h_{t-1} is initialized to a zero vector of the same dimensionality as x_{new}. This initialization helps maintain numerical stability and ensures that the model is not adversely affected during the early stages of training.

Using this method, we compute the current time state h_t based on the previous state h_{t-1}. Our approach, which approximates the current time value using a Taylor expansion at time step t, is motivated by several considerations. First, we can discretize the state directly, transforming the continuous function into a discrete representation corresponding to the spatial position of the Time Module. This avoids the need for discretizing W_c and W_d, as done in S4, thus reducing computational overhead. Discretization ensures that the model retains both precision and stability when processing sequential data, and by appropriately controlling the time steps, the model's adaptability to long sequences is enhanced. Moreover, compared to the original SSM or S4 models, we reduce parameter usage by directly utilizing the current time value h_{t+1} to adjust the output of the pre-trained model. Furthermore, leveraging the S4 model, we can apply FFT-based matrix transformations, enabling our method to overcome the parallelization limitations typically encountered in RNNs.

In our method, W_c and W_d are new parameters that participate in training, while h_t is detached from the computational graph and does not participate in training. Therefore, the state update is controlled solely by W_c, W_d, and the input x.

3.3 Tuned Model Forward

As discussed earlier, our proposed SSMLoRA technique introduces a state transition mechanism based on the State Space Model (SSM) architecture, specifically by adding two matrices. The detailed architecture is shown in Figure 1, where we illustrate the structure of the Time Module, which forms a timeline based on the SSM model, represented by the dashed section on the right side of the figure. At each Time Module, the current state h_t is retrieved, followed by the computation of the output y and the next state h_{t+1}. This mechanism is introduced into the pre-trained model to dynamically adjust the feature representation of the input x, enhancing the model's performance on tasks that are highly dependent on temporal information.

First, we clarify the origin of the variable x_{new}, used throughout the previous sections, which is obtained by projecting the input x into a low-rank matrix:

$$x_{\text{new}} = x \times W_a$$

The dimensionality of the projected low-rank matrix x_{new} becomes r, which matches the dimensionality of the current state value h_{t+1} calculated through the process described in Section 3.2, allowing direct computation. However, for numerical stability, we normalize h_{t+1} before performing further calculations, as outlined in the following equations:

$$\text{min_val} = \min(h_{t+1}) \tag{5}$$

$$\text{max_val} = \max(h_{t+1}) \tag{6}$$

$$h_{t+1}^{\text{norm}} = \frac{h_{t+1} - \text{min_val}}{\text{max_val} - \text{min_val} + \epsilon} \tag{7}$$

Next, before restoring x to its original dimensionality, we adjust x using the normalized h_{t+1}. Therefore, the output of the Time Module is given by:

$$y = x_{\text{new}} + h_{t+1}^{\text{norm}} \times W_b$$

In summary, the output of the original pretrained weight matrix can be expanded as follows, with our technique introducing additional adjustments using fewer parameters and operations than LoRA:

$$y = x \times W_0 + (x \times W_a + h_{t+1}^{\text{norm}}) \times W_b$$

Since each Time Module is influenced by the preceding Time Module along the timeline, we design the Time Modules to be placed on different timelines depending on the type of neural network layer they are inserted into, preventing mutual interference. For instance, in the self-attention module of the model, if a timeline is set, separate timelines will be established for the query, key, and value matrices to prevent temporal misalignment. Therefore, the example shown in Figure 1 demonstrates the insertion into a single neural network layer, rather than a complete example across the entire model.

3.4 Sparse Insertion

As emphasized earlier, our method applies a sparsification approach to the attention mechanism. Research on LoRA has shown that, for large language models based on the transformer architecture, the best results are achieved when both the query and value matrices are adjusted simultaneously. Therefore, in this section, we adopt a more efficient strategy by applying the Time module only to the query and value matrices in the attention mechanism.

We further found that sparsifying the full structure does not lead to any loss of accuracy. Specifically, in each encoder's attention module, SSMLoRA operates on only one of the query or value matrices at a time, meaning one matrix is "activated" for adjustment. We recommend alternating sparse insertions in a regular pattern, which is extremely easy to implement. As illustrated in Figure 1, if W represents the query matrix of the l-th layer's attention mechanism, the query matrices in layers $l+1$ and $l-1$ are excluded from the time axis, while the value matrix follows a separate time axis.

For other non-multihead attention components, our method essentially replaces LoRA with SSMLoRA, maintaining a compact parameter configuration. This approach, as demonstrated in Section 4.1, outperforms other fine-tuning methods, achieving better results with less than 80% of the parameters used by LoRA.

3.5 Initialization Strategy

In the SSMLoRA model, we introduce four key matrices W_a, W_b, W_c, and W_d to achieve state transitions and feature mapping. To ensure stability and effectiveness during the initialization phase, we adopt a differentiated initialization strategy. Furthermore, although the time vector h_t primarily serves as a transmitted quantity rather than an independently trained parameter, its initialization method still warrants in-depth exploration.

Specifically, for matrices W_a and W_b, we draw inspiration from the initialization method of the LoRA fine-tuning technique. The initialization of W_a employs scaled Gaussian-distributed random values, effectively introducing moderate randomness to avoid the potential slow convergence problem associated with zero initialization. In contrast, W_b is initialized as a zero matrix to ensure that the overall output remains unaffected in the initial stage while allowing the model to gradually learn effective mapping relationships during the training process.

For matrices W_c, W_d, and the time vector h_t, we uniformly adopt a zero initialization strategy. As shown in Equation (8), we primarily optimize the output of the LoRA technique by adjusting the bias term. This approach aims to minimize the introduction of additional noise in the early stages of training, subsequently learning the weight distribution of temporal states progressively. Consequently,

SSMLoRA can be viewed as a sparsified variant of LoRA in the initial training phase, which may result in slightly lower initial performance compared to standard LoRA. However, as training progresses, the model can learn the intrinsic connections between low-rank matrices across different layers, thereby achieving steady performance improvements and ultimately surpassing traditional LoRA techniques.

In the experiments presented in Section 4, we demonstrate that SSMLoRA outperforms LoRA in tasks involving high-density short text matching, question answering, and long-text sequence processing. These results provide compelling evidence for the effectiveness of our proposed initialization strategy and fine-tuning technique.

4 Experiments

To validate the effectiveness of SSMLoRA, we designed and conducted a series of comprehensive experiments. Our experiments focused on two main objectives:

(1) Comprehensive evaluation as a parameter-efficient method: We compared SSMLoRA with full parameter fine-tuning and other existing parameter-efficient fine-tuning techniques across various tasks and scenarios. The goal of these experiments was to assess SSMLoRA's performance in different types of tasks and applications, verifying its potential as a widely applicable fine-tuning method. We also identified specific areas where this new technique excels, providing insights into the underlying factors contributing to its performance advantages.

(2) Systematic comparison with LoRA: A key aspect of SSMLoRA is the sparse insertion of low-rank matrices, and we aimed to highlight the impact of this feature. Given that SSMLoRA's trainable parameters—particularly in comparison to LoRA—are largely influenced by the rank (r), we tested various values of r. We paid special attention to the performance of SSMLoRA when using the most sparse insertion configuration and compared it with LoRA under conditions of extremely low parameter count. This comparison was designed to evaluate the parameter efficiency of SSMLoRA.

We employed multiple benchmark models in our experiments, including RoBERTa-base, RoBERTa-large (Liu et al., 2019), DeBERTaV3-base (He et al., 2021), and GPT-2 (Radford et al., 2019). The experiments were conducted on the GLUE (Wang et al., 2019) and SUPERGLUE (Wang et al., 2020) benchmarks, and we compared SSMLoRA with other fine-tuning methods such as LoRA and full fine-tuning. The results strongly support the effectiveness of our proposed approach.

4.1 Experimental Settings

We first conducted a controlled experiment on the RoBERTa-base model, comparing fine-tuning with full parameter updates, BitFit, LoRA, and SSMLoRA across the GLUE benchmark tasks, including CoLA (Warstadt et al., 2018), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), STS-B (Cer et al., 2017), QQP, MNLI (Williams et al., 2018), QNLI, and RTE (Dagan et al., 2005). All experiments were performed on an NVIDIA RTX 3090 GPU, and the results are presented in Table 1.

In our setup, all four techniques were trained with learning rates ranging from 1e-7 to 1e-4, and the best result from each method, across all learning rates, was reported for each dataset. Both LoRA and SSMLoRA were configured with a rank $r = 8$ and applied to the model's self-attention mechanism, fully connected layers, and classifier.

For SSMLoRA, sparse insertion was used in the self-attention module, where we applied alternating sparse insertions to the Q and V matrices, while maintaining dense insertions in other layers. Notably, the trainable parameters introduced by SSMLoRA were less than 80% of those introduced by LoRA. Our technique demonstrated superior performance across multiple datasets, achieving the best results among the four methods on MRPC and QNLI, and surpassing LoRA in more than half of the datasets. These findings suggest that SSMLoRA performs well in NLP tasks, particularly excelling in sentence-pair tasks, as evidenced by subsequent experiments. However, it exhibits some limitations in terms of cross-domain generalization.

Compared to LoRA, our method utilizes fewer parameters, resulting in significant advantages in memory usage, allowing training in more resource-constrained environments. In terms of total training time, SSMLoRA also shows competitive performance.

From these results, we conclude that SSMLoRA can effectively replace LoRA as a parameter-efficient fine-tuning method. Moreover, as an innovative approach leveraging state-space equations, SSMLoRA demonstrates particularly strong perfor-

Method (Trainable params)		CoLA MCC	SST-2 Acc	MRPC F1/Acc	STS-B P/S Corr	QQP F1/Acc	MNLI-m Acc	MNLI-mm Acc	QNLI Acc	RTE Acc
Fine-tune	(124M)	63.24	94.61	89.73/85.74	91.00	88.46/91.35	86.27	86.31	90.13	76.53
BitFit	(102K)	60.27	93.46	89.05/85.22	90.91	85.61/88.95	83.98	84.38	91.34	83.39
LoRA	(1.3M)	61.64	94.50	89.54/86.03	90.37	86.95/90.24	84.94	85.50	90.76	72.20
SSMLoRA	(1.0M)	61.76	94.15	91.99/88.73	90.46	87.14/90.52	85.73	85.16	91.91	76.17

Table 1: Performance comparison of four methods based on the RoBERTa-base model across various tasks

mance on certain datasets.

4.2 Advantages Exploration

Method	DeBERTaV3-base	
	SQuAD	NarrativeQA
Fine-tune	85.08/69.23	23.05
BitFit	55.58/39.28	21.29
LoRA	84.63/69.27	24.41
SSMLoRA	84.54/69.37	24.10

Table 2: Performance comparison of methods on DeBERTaV3-base for SQuAD and NarrativeQA tasks

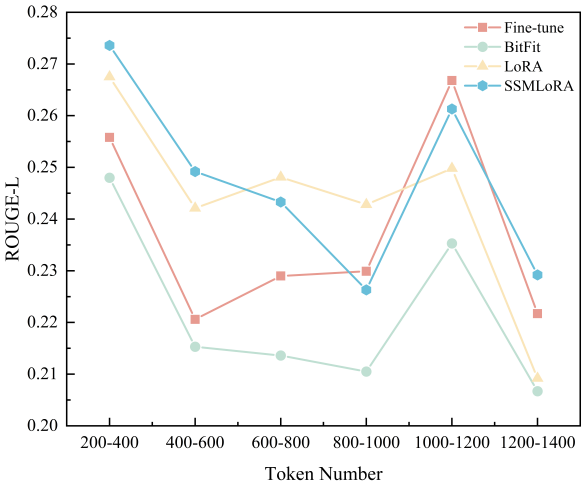

Figure 2: This figure shows the results of the DeBERTaV3-base model fine-tuned with four different techniques on the NarrativeQA dataset, evaluated on test data across various length intervals.

Based on the characteristics of the state-space model (SSM), we aimed to explore the potential of our technique in logical reasoning and handling long-text data. To comprehensively assess the advantages of our approach, we conducted additional evaluations on a range of datasets, with results showing consistently strong performance.

We selected two datasets, SQuAD (Rajpurkar et al., 2016) and NarrativeQA (Kočiský et al., 2017), for evaluation. SQuAD is a question-answering dataset based on Wikipedia content, requiring the model to answer questions from large volumes of text, with performance measured by F1 and Exact Match (EM) scores. NarrativeQA, on the other hand, requires the model to understand the underlying narrative rather than performing simple content matching, with performance evaluated using the ROUGE-L metric. We fine-tuned the DeBERTaV3-base model using four different techniques on both the SQuAD and NarrativeQA datasets, and the final results are presented in Table 2.

In short-text matching tasks, SSMLoRA demonstrated a superior ability to accurately match results, excelling in scenarios requiring precise answers. For the long-text datasets, SSMLoRA also performed well. These findings suggest that despite reducing the number of parameters, SSMLoRA significantly enhances the model's logical reasoning abilities, and its capability to handle long-text tasks remains uncompromised due to the structural advantages of the state-space model.

We further explored the performance of SSMLoRA on data of varying lengths within the NarrativeQA tasks. The test sets of both datasets were divided into intervals based on the preprocessed data length, and the fine-tuned models were evaluated on these intervals. For the NarrativeQA dataset, the detailed results are shown in Figure 2, where the data was divided into six intervals, each containing 200 tokens. Our technique achieved the highest ROUGE-L scores among the four methods in short-text scenarios. When the token length exceeded 1000, SSMLoRA showed a marked improvement over LoRA, demonstrating exceptional performance.

4.3 Sparsification Study of SSMLoRA

In this section, we explore the comparison between SSMLoRA with sparse insertion and LoRA, and present the experimental results. Unlike previous sections, we focus on investigating the performance of SSMLoRA using purely sparse insertion, applied to fewer types of neural network layers, to examine the differences in performance compared to LoRA fine-tuning. Given that the newly intro-

duced matrices are influenced by the rank r, we conducted experiments with five different values of r: 1, 2, 4, 8, and 16.

We selected two models, RoBERTa-large and GPT-2, for our experiments, as their attention modules consist of commonly used linear and convolutional layers, respectively. We aimed to verify SSMLoRA's effectiveness in both linear and convolutional layers. The experiments were first conducted on the GLUE benchmark, and the results are shown in Tables 5 and 6. As can be observed, SSMLoRA achieved nearly identical performance to LoRA while using only half the parameters, and in most tasks, it even outperformed LoRA. In these experiments, SSMLoRA was sparsely inserted into the pre-trained models. Inspired by the findings of LoRA, for the RoBERTa-large model, we applied SSMLoRA only to the query and value matrices of the self-attention mechanism, as modifying both matrices simultaneously has been shown to yield the best training results. Regarding the choice of sparse insertion strategy, we aimed to design a simple and easy-to-implement solution. For RoBERTa-large, we adopted an alternating sparse insertion approach, while for GPT-2, since the self-attention mechanism is encapsulated within a complete convolutional layer, SSMLoRA was applied directly to the entire output, using a skip-insertion strategy that skips one layer each time. The experimental results confirm the effectiveness of our design, yielding strong performance across the majority of tasks.

To further evaluate our method, we extended the experiments to the SuperGLUE benchmark, which includes datasets such as CB (De Marneffe et al., 2019), COPA (Roemmele et al., 2011), MultiRC (Khashabi et al., 2018), RTE (Dagan et al., 2005), WiC (Pilehvar and Camacho-Collados, 2019), WSC (), BoolQ (Clark et al., 2019), and ReCoRD (Zhang et al., 2018). This benchmark poses a greater challenge, particularly for SSMLoRA, which operates with only half the number of parameters compared to LoRA. Nevertheless, by applying the same design strategy, SSMLoRA continued to demonstrate strong performance, as shown in Table 5.

In conclusion, SSMLoRA achieves comparable performance to LoRA across most datasets while using only half the parameters, and it exhibits outstanding performance on certain datasets.

4.4 Train Time

Due to the recursive nature of our approach, which behaves similarly to RNNs when FFT is not applied to transform the time axis into a more computation-friendly format, we further explored the training time required to ensure the efficiency of our method. This section complements the findings from Section 4.1 by providing a more detailed analysis. We recorded both the total time taken to reach convergence using early stopping techniques and the average time per epoch. All experiments were conducted on an NVIDIA RTX 3090 GPU, and the results are presented in Tables 8 and 9.

As observed in datasets with smaller sizes, such as CoLA, MRPC, and QNLI, SSMLoRA tends to require a longer total training time and a greater per-epoch duration. However, with larger datasets, it becomes evident that SSMLoRA exhibits significant improvements in both total training time and per-epoch efficiency compared to more basic techniques like fine-tuning and BitFit. When comparing with LoRA, SSMLoRA requires a longer total training time on five datasets and a longer per-epoch time on four datasets. Coupled with the results from Section 4.1, this indicates that SSMLoRA adopts a more gradual learning approach to capture data features, potentially requiring more training epochs but ultimately converging to superior results.

5 Conclusion

In this work, we propose a novel approach for parameter-efficient fine-tuning of large pre-trained language models, termed State Space Model Low-Rank Adaptation (SSMLoRA). While the insertion of low-rank matrices has been a common strategy, it often leads to unnecessary parameter overhead, particularly when applied uniformly across all neural network layers. To address this, we introduce an improved state space equation that strategically connects sparsely inserted low-rank matrices, significantly reducing the number of parameters while enhancing parameter efficiency. Our method is simple yet effective, and given the demonstrated potential of State Space Model, it offers a holistic improvement to LoRA techniques. Furthermore, our approach can be seamlessly integrated with existing optimization strategies, further expanding its applicability in large-scale model fine-tuning.

Limitations

Despite the promising results achieved by SSMLoRA, our research still has some limitations. The core of SSMLoRA lies in adjusting the input mapped to a lower-dimensional space, which allows the input to better adapt to various downstream tasks. However, this also means that the state h_t must match the dimensionality of the input mapped to the low-rank space. For instance, when the batch sizes used in training and testing differ, although h_t can be adjusted to accommodate the new dimensions, this inevitably results in a performance drop. Therefore, further research is needed to develop better solutions to mitigate performance degradation caused by dimensionality mismatches.

References

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2015. Structured pruning of deep convolutional neural networks.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

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.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. 2020. Hippo: Recurrent memory with optimal polynomial projections. *arXiv preprint arXiv:2008.07669*.

Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention.

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.

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*.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface:a challenge set for reading comprehension over multiple sentences. In *Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL)*.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. 2024. Vera: Vector-based random matrix adaptation.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2017. The narrativeqa reading comprehension challenge.

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*.

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.

Seongmin Park and Jihwa Lee. 2021. Finetuning pre-trained transformers into variational autoencoders. In *Proceedings of the Second Workshop on Insights from Negative Results in NLP*, pages 29–35, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI spring symposium series*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2020. Superglue: A stickier benchmark for general-purpose language understanding systems.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Guotian Xie, Jingdong Wang, Ting Zhang, Jianhuang Lai, Richang Hong, and Guo-Jun Qi. 2018. Igcv2: Interleaved structured sparse convolutional neural networks.

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

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension.

A Experimental Details

This section contains seven tables that supplement the experimental results described in Section 4. Except for Table 6, which was conducted on an NVIDIA RTX A6000 due to dataset constraints, all other tables present data collected using a single NVIDIA RTX 3090 GPU.

Tables 3 present the performance of models fine-tuned using Fine-tune, BitFit, LoRA, and SSMLoRA, evaluated on data of varying lengths. Table 3 is divided into two parts: the first part shows the performance of the four fine-tuning techniques as the number of tokens in the data increases, and the second part provides the data volume and the number of samples used in each token-length interval. For intervals with a smaller sample size, all data was used for testing, while for larger intervals, 40% of the data was randomly sampled. The results demonstrate that SSMLoRA shows clear advantages with both shorter and longer texts. We attribute this phenomenon to two key reasons: for shorter texts, the sparse design of SSMLoRA helps prevent overfitting and improves the model's ability to understand the text; for longer texts, other fine-tuning techniques struggle to maintain performance, while SSMLoRA benefits from the advantages of the State Space Model, leading to superior results.

Tables 5 and 6 use the RoBERTa-large and GPT-2 models, respectively, to compare the performance of SSMLoRA and LoRA as the rank r changes. Table 7 further explores these techniques using RoBERTa-large to validate their performance on the SUPERGLUE benchmark. In these experiments, the rank r was set to 1, 2, 4, 8, and 16. The application of these techniques has been discussed in previous sections. From the results, we can conclude that although SSMLoRA uses less than 60% of the trainable parameters compared to LoRA, it achieves comparable performance on most datasets. This leads to an exciting conclusion: SSMLoRA successfully reduces the number of parameters while maintaining the envisioned performance.

Tables 8 and 9 provide insights into the total training time and the average epoch training time for the four fine-tuning methods. These results

show that SSMLoRA achieves faster training times on larger datasets, though training time increases for smaller datasets. We attribute this phenomenon to the fixed overhead required by SSMLoRA to establish the State Space Model. Once this initial setup is complete, the model can train at a much faster rate. However, for smaller datasets, this fixed cost is harder to offset, which explains the increase in training time.

Method	200-400	400-600	600-800	800-1000	1000-1200	1200-1400
Fine-tune	0.2558	0.2206	0.2290	0.2299	0.2668	0.2217
BitFit	0.2480	0.2153	0.2136	0.2105	0.2353	0.2067
LoRA	0.2675	0.2421	0.2481	0.2428	0.2498	0.2092
SSMLoRA	0.2736	0.2492	0.2433	0.2263	0.2613	0.2292
Samples in interval						
Total	1704	1888	2334	3236	1285	110
Evaluated	681	755	933	1294	514	44

Table 3: The performance of the DeBERTaV3-base model fine-tuned with four different techniques on the NarrativeQA test set, evaluated across varying input sequence lengths.

Method	#Params	SST-2	RTE	CoLA	MRPC	QNLI	WNLI	QQP	MNLI Matched	MNLI Mismatched
LoRA $r = 1$	98.3K	94.95	81.23	64.19	87.73/89.22	92.92	56.34	86.23/89.37	89.11	88.77
SSMLoRA $r = 1$	49.2K	95.14	81.23	64.11	92.15/88.97	93.17	56.34	85.37/89.01	88.37	87.92
LoRA $r = 2$	196K	93.35	82.67	62.68	87.68/89.46	93.2	56.34	86.71/89.95	88.81	88.41
SSMLoRA $r = 2$	98.5K	95.76	82.67	63.92	92.39/89.46	93.2	56.34	85.83/89.30	88.68	88.88
LoRA $r = 4$	393K	93.58	85.56	61.62	87.12/88.73	93.45	54.93	86.80/89.98	89.302	88.87
SSMLoRA $r = 4$	197K	95.76	85.56	63.79	93.81/91.42	92.24	66.20	86.55/89.85	88.77	88.86
LoRA $r = 8$	786K	94.72	82.67	65.17	87.63/89.46	93.32	56.34	87.05/90.19	88.54	88.51
SSMLoRA $r = 8$	396K	95.64	82.67	65.03	89.95/92.61	92.46	69.01	79.74/85.72	88.75	88.72
LoRA $r = 16$	1.6M	95.07	83.03	62.88	88.71/90.20	93.4	69.01	87.09/90.34	88.24	88.69
SSMLoRA $r = 16$	798K	95.30	83.03	65.33	92.93/90.20	93.43	56.34	81.98/86.82	88.91	88.36

Table 4: Comparison of LoRA and SSMLoRA methods on the GLUE dataset (Rank r=1,2,4,8,16)

Method	#Params	SST-2	RTE	CoLA	MRPC	QNLI	WNLI	QQP	MNLI Matched	MNLI Mismatched
LoRA $r = 1$	37.6K	86.01	57.4	5.3	78.43	81.38	56.34	80.77/85.17	73.46	73.53
SSMLoRA $r = 1$	18.4K	84.86	64.62	5.54	78.19	81.51	57.75	85.37/89.01	75.11	75.32
LoRA $r = 2$	75.3K	85.78	59.21	2.56	76.72	82.23	49.3	80.89/85.15	76.24	76.36
SSMLoRA $r = 2$	36.9K	86.24	61.37	5.17	72.79	82.59	50.7	85.84/89.30	75.51	75.24
LoRA $r = 4$	150K	86.81	59.21	2.56	75.98	81.27	53.52	81.80/86.15	74.73	75.31
SSMLoRA $r = 4$	73.9K	86.93	58.48	6	76.72	82.23	40.85	86.56/89.86	76.28	76.61
LoRA $r = 8$	301K	86.58	61.01	6.63	78.68	82.08	56.34	81.99/86.01	77.21	77.52
SSMLoRA $r = 8$	148K	87.04	62.45	4.43	77.94	83.12	56.34	81.15/85.14	77.04	77.57
LoRA $r = 16$	602k	87.50	58.84	8.16	76.23	82.70	47.89	82.89/87.27	77.34	78.40
SSMLoRA $r = 16$	297k	86.58	59.57	1.81	76.47	83.95	43.66	82.0/86.2	78.11	78.50

Table 5: Comparison of LoRA and SSMLoRA methods on the GLUE dataset (Rank r=1,2,4,8,16), model: GPT-2

Method	#Params	WSC	WiC	RTE	BoolQ	COPA	CB (F1/Acc)	ReCoRD (F1/Acc)	MultiRC (F1a/EM)
LoRA $r = 1$	98.3K	64.42	71.63	81.95	80.98	91.0	47.34/67.86	23.30/57.16	78.45/21.95
SSMLoRA $r = 1$	49.2K	63.46	69.44	81.59	77.77	86.0	46.21/66.07	23.35/60.88	77.77/20.79
LoRA $r = 2$	196K	63.46	70.69	82.31	81.25	94.0	81.60/82.14	23.75/58.29	79.30/25.25
SSMLoRA $r = 2$	98.5K	63.46	69.44	81.59	79.69	89.0	48.73/69.64	24.39/56.81	78.19/21.95
LoRA $r = 4$	393K	63.46	69.44	84.48	62.17	88.0	53.15/73.21	24.74/56.65	79.28/23.93
SSMLoRA $r = 4$	197K	64.42	71.00	83.75	82.08	90.0	48.68/69.64	24.82/56.07	78.33/19.64
LoRA $r = 8$	786K	63.46	71.00	81.95	62.17	84.0	78.44/75.00	27.01/55.79	76.11/20.30
SSMLoRA $r = 8$	396K	63.46	72.88	83.75	80.76	92.0	58.87/83.93	24.05/57.27	79.94/25.08
LoRA $r = 16$	1.6M	63.46	63.64	83.03	62.17	94.0	88.18/81.07	25.30/54.93	80.08/23.43
SSMLoRA $r = 16$	798K	63.46	71.63	84.12	62.17	91.0	46.15/66.07	23.83/57.70	78.22/20.79

Table 6: Comparison of LoRA and SSMLoRA methods on the SuperGLUE dataset (Rank r=1,2,4,8,16), including MultiRC

Method (Trainable params)		CoLA time	SST-2 time	MRPC time	STS-B time	QQP time	MNLI time	QNLI time	RTE time
Fine-tune	(124M)	4.26min	28.56min	3.51min	5.1min	335.25min	191.7min	46.00min	1.46min
BitFit	(102K)	2.23min	24.82min	1.37min	5.03min	511.64min	360.37min	142.8min	3.79min
LoRA	(1.3M)	5.27min	34.1min	2.45min	3.47min	191.39min	229.41min	34.69min	1.99min
SSMLoRA	(1.0M)	8.01min	23.89min	3.68min	3.86min	192.15min	223.55min	72.30min	1.85min

Table 7: Performance comparison of four methods based on the RoBERTa-base model across various tasks

Method (Trainable params)		CoLA time	SST-2 time	MRPC time	STS-B time	QQP time	MNLI time	QNLI time	RTE time
Fine-tune	(124M)	0.47min	4.08min	0.39min	0.34min	37.25min	47.93min	9.2min	0.16min
BitFit	(102K)	0.32min	3.17min	0.15min	0.63min	36.55min	40.04min	8.93min	0.27min
LoRA	(1.3M)	0.48min	6.82min	0.22min	0.35min	19.14min	45.88min	5.78min	0.17min
SSMLoRA	(1.0M)	0.62min	5.97min	0.26min	0.30min	16.01min	44.71min	12.05min	0.20min

Table 8: Performance comparison of four methods based on the RoBERTa-base model across various tasks