Chain of LoRA: Efficient Fine-tuning of Language Models via Residual Learning

Anonymous ACL submission

Abstract

Fine-tuning is the primary methodology for tailoring pre-trained large language models to specific tasks. As the model's scale and the diversity of tasks expand, parameter-efficient fine-tuning methods are of paramount importance. One of the most widely used family of methods is low-rank adaptation (LoRA) and its variants. LoRA encodes weight update as the product of two low-rank matrices. Despite its advantages, LoRA falls short of full-parameter fine-tuning in terms of generalization error for certain tasks.

We introduce Chain of LoRA (COLA), an iterative optimization framework inspired by the Frank-Wolfe algorithm, to bridge the gap between LoRA and full parameter fine-tuning, without incurring additional computational costs or memory overheads. COLA employs a residual learning procedure where it merges learned LoRA modules into the pre-trained language model parameters and re-initialize optimization for new born LoRA modules. We provide theoretical convergence guarantees as well as empirical results to validate the effectiveness of our algorithm. Across various models (OPT and Llama-2) and 11 benchmarking tasks, we demonstrate that COLA can consistently outperform LoRA without additional computational or memory costs.

1 Introduction

Pre-trained language models have become instrumental in natural language processing, demonstrating remarkable performance across various fields. Large language model fine-tuning is a process for adapting pre-trained models to specific tasks, allowing for improved performance on various real-world applications, such as machine translation and code analysis (Lewis et al., 2019; Wang et al., 2021; Qin et al., 2023). Despite the notable benefits of full parameter fine-tuning, the computational expenses and memory requirements it entails present significant challenges, particularly in light of the ever-growing size of large language models.

For this reason, parameter efficient fine-tuning (PEFT) methods have received significant attention (Pfeiffer et al., 2020; He et al., 2021). Instead of adjusting all the parameters of the model, PEFT involves fewer adjustments to the original model parameters to specialize its knowledge for a particular application (Houlsby et al., 2019; Lester et al., 2021). One of the most widely used paradigms in parameter efficient fine-tuning is Low-Rank Adaptation (LoRA) (Hu et al., 2021). LoRA focuses on modifying only a small, low-rank portion of the model's weights. This is achieved by adding low-rank matrices to the weights of the model during training. The advantage of LoRA is that it significantly reduces the computational burden and time required for fine-tuning, making it more efficient and scalable, especially for very large models. Despite the significant computational advantage of LoRA, it is inferior to full parameter fine-tuning in terms of generalization error.

In this paper we investigate whether the generalization error gap between LoRA and full parameter fine-tuning can be reduced albeit preserving the computational efficiency. We do this by learning a higher rank augmentation of the LLM weights by the method of residual learning. The high-rank augmentation is composed of several low-rank structures. Namely, we use an iterative procedure to learn a low-rank addition to the existing approximation, thereby increasing its rank. Hence, we call the procedure "Chain of LoRA", or COLA for short.

This residual learning method is inspired by the Frank-Wolfe algorithm as applied to matrix completion, which augments an existing completion by a rank one addition. Over many iterations, this residual learning procedure can be shown to produce an accurate higher rank completion.

Our contributions

- We present an iterative optimization framework, COLA, for parameter-efficient fine-tuning. COLA is based on the Frank Wolfe method from mathematical optimization, and we formalize this relationship.

- We demonstrate the effectiveness of COLA via extensive experiments across datasets and models. COLA consistently outperforms LoRA in terms of generalization error with no additional cost of computation. For example, fine-tuning OPT-1.3B with COLA brings a relative 6.47% test accuracy gain to LoRA on WSC. Llama2-7B experiments show up to 4.36% relative test score improvement.

- We provide theoretical analyses of the iterative learning framework employed in our proposed method, demonstrating the convergence to stationary points in the setting of smooth nonconvex optimization.

2 Related Work

Conventional full-parameter fine-tuning becomes computationally impractical as both model size and the number of downstream tasks increase. In response to this challenge, recent advancements in parameter-efficient finetuning methods suggest modifying only a small portion of parameters while maintaining the majority of pre-trained model parameters unchanged.

Adapter based methods Within this domain, a line of research known as adapter-based approach involves inserting compact adapter modules between transformer layers. Throughout the fine-tuning process, only the newly introduced lightweight adapters are trained, while the pre-trained model remains frozen and shared across tasks, thus significantly enhancing the practicality and efficiency of adapting large models to diverse tasks. Houlsby et al. (2019) propose a new bottleneck adapter module and position it twice within each transformer layer (Vaswani et al., 2017). The adapter employs a bottleneck architecture, incorporating a skip connection to effectively constrain the number of parameters involved in the module design. Variant adapter architecture and placements are proposed in concurrent work (Bapna and Firat, 2019; Stickland and Murray, 2019). Building upon the success of adapter-based approaches for single-task adaptation, subsequent studies extend the adapter-based architecture to the realm of multi-task learning scenarios (Mahabadi et al., 2021). AdapterFusion proposes a two-stage learning framework where task-specific adapters are learned and then later combined in a separate knowledge composition step (Pfeiffer et al., 2020).

Prefix tuning methods Alternative research investigates the incorporation of tunable parameters into both the input and hidden layers, as explored by Li and Liang (2021). These lightweight task-specific vectors, commonly referred to as the prefix, offer a notable reduction in the memory load required for storing task-specific models. Additionally, they outperform full fine-tuning, particularly in scenarios with limited data availability. Efficient prompt tuning further simplifies prefix tuning by concatenating a trainable tensor ("soft prompt") with the model's input embeddings (Lester et al., 2021). These "soft prompts" are learned through backpropagation to perform downstream tasks.

LoRA and its variants The most closely related work to ours is LoRA (Hu et al., 2021), which introduces trainable low-rank matrices to approximate weight updates during fine-tuning. Building on LoRA, numerous recent studies have explored its variants from different perspectives. QLoRA (Dettmers et al., 2023) further leverages 4-bit quantization to effectively and efficiently fine-tune LLMs. To enhance parameter efficiency, Tied-LoRA, introduced by Renduchintala et al. (2023), incorporates weight tying and selective training. Chen et al. (2023) propose LongLoRA to extend the context sizes of LLMs with limited computation cost. MultiLoRA (Wang et al., 2023) is designed specifically for better multi-task adaptation. Concurrently, Sheng et al. (2023) introduce S-LoRA, offering a framework that enhances the scalable serving of multiple LoRA adapters. Lialin et al. (2023) [1] explores pre-training with multiple stages of low-rank matrices to facilitate efficiency.

Optimization for fine-tuning LLMs has special challenges, notably memory constraints. For this reason, zero-order optimization methods were proposed (Malladi et al., 2023).

3 Our Method

In this section we describe our method for fine-tuning. It is divided into two parts, in the first we present necessary background for our exposition, and the second gives details of COLA.

[1] this is independent and concurrent work to our paper.

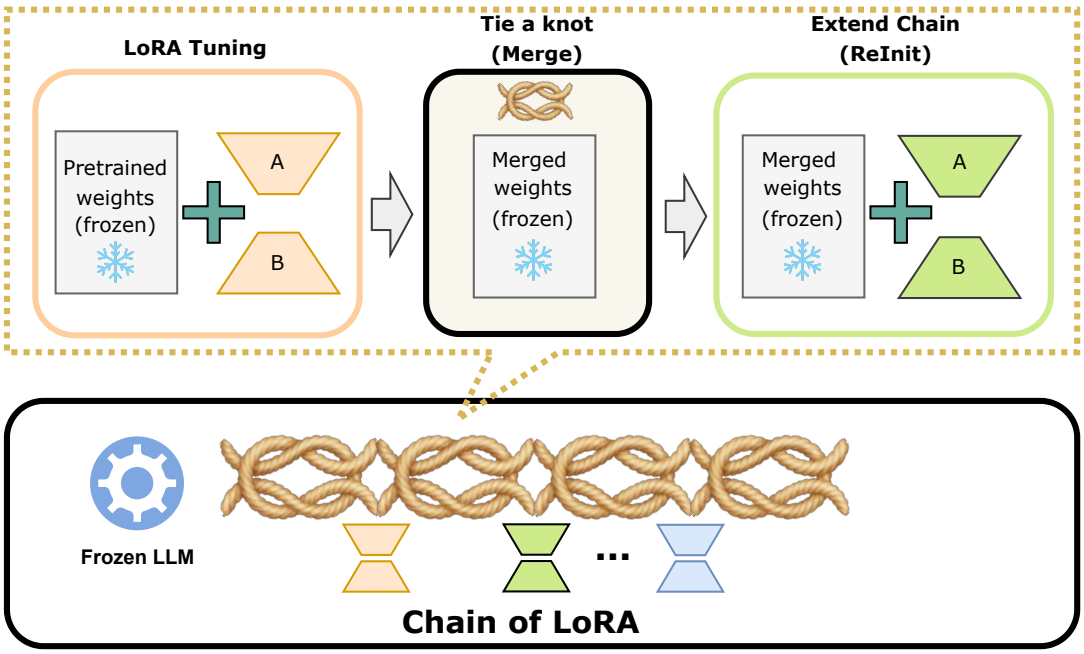

Figure 1: An illustration of Chain of LoRA. Our approach starts with a frozen LLM, and learns a sequence of low-rank matrices to approximate a high-rank augmentation to perform task adaptation. As shown in the dashed line box, each residual learning procedure consists of three steps: (1) LoRA Tuning, (2) Tie a knot, and (3) Extend the chain. In step 1, low-rank LoRA modules are fine-tuned, In step 2, the learned LoRA weights are merged into the frozen model. In step 3, a new LoRA module is instantiated and the optimizer state is reset. These three steps are repeated in this residual learning paradigm.

3.1 Preliminaries

Low Rank Adaptation (LoRA) LoRA (Hu et al., 2021) aims to improve the efficiency of fine-tuning large language models by training much smaller low-rank decomposition matrices of certain weights. It hypothesizes a low "intrinsic rank" of weight updates at task adaptation and injects trainable low-rank decomposition matrices into each layer of the Transformer architecture. Consider a weight matrix W_{frozen} from the pre-trained model, the weight update ΔW for task adaptation is represented with a low-rank decomposition BA. The forward pass with LoRA is as follows:

$$W_{frozen}x + \Delta W x = W_{frozen}x + \mathbf{BA}x,$$

where $W_{frozen}, \Delta W \in \mathbb{R}^{d\times k}$, $A \in \mathbb{R}^{r\times k}$, $B \in \mathbb{R}^{d\times r}$ and $r \ll min(d, k)$. During training, W_{frozen} is frozen and only B, A are optimized. At deployment, the learned low-rank matrices can merge with the frozen pre-trained model weights.

Frank-Wolfe The Frank-Wolfe method, also known as the conditional gradient method, is an optimization algorithm for solving constrained convex, and more recently nonconvex, optimization problems. The key feature of the Frank-Wolfe method is how it handles the constraints. Instead of projecting onto the constraint set via projections, it uses a linear optimization oracle. Iteratively, the method finds a linear approximation of the objective function within the feasible region and moves towards the minimizer of this approximation.

The Frank-Wolfe algorithm is particularly suited for problems in which linear optimization is easier than Euclidean projections. For this reason, "projection free" methods were considered in the machine learning community (Hazan, 2008; Jaggi, 2013; Hazan and Kale, 2012; Garber and Hazan, 2016). More recently nonconvex optimization was considered using the Frank Wolfe method in Lacoste-Julien (2016); Reddi et al. (2016).

3.2 Chain of LoRA

In this section we give the details of our optimization framework. The key idea of our method is to form a chain (sequence) of LoRAs and iteratively learn the low-rank adaptation LoRA modules. As illustrated in Figure 1, our method is comprised of three stages: **Tune LoRA, Tie a knot, Extend the chain**. We first introduce notations, followed by an explanation of the three stages in the workflow. We

also provide the detailed step-by-step procedure in Algorithm 1.

Algorithm 1 Chain of LoRA (COLA)

Input: frozen pre-trained weights W, chain knots $\{\tau_1, \ldots, \tau_m\}$, fine-tuning dataset $\mathcal{D}$, training objective $\mathcal{L}$, total training iterations T.
Initialize LoRA parameters to A_0, B_0
for $t = 1, \ldots, T$ **do**
  Sample minibatch $\mathcal{B}_t \subset \mathcal{D}$
  if $t \in \{\tau_1, \ldots, \tau_m\}$ **then**
    Tie knot: Merge LoRA to backbone weights $W = W + B_t A_t$
    Extend chain: Re-initialize LoRA parameters $A_t = A_0, B_t = B_0$ and optimizer states
  end if
  forward pass with LoRA
  backward pass and update LoRA parameters

$$(A_t, B_t) = (A_{t-1}, B_{t-1}) - \eta_t * \hat{\nabla}_{A,B} \mathcal{L}(W)$$

end for

For a pre-trained LLM weight matrix $W_{pretrained} \in \mathbb{R}^{d \times k}$, we denote the weights update occurred during fine-tuning as ΔW. Ideal adaptation yields the optimal weights $W^\star$ tailored for the given task and the corresponding optimal weight update $\Delta W^\star$, as shown below.

$$W^\star = W_{pretrained} + \Delta W^\star$$

In COLA, we propose to approximate $\Delta W^\star$ with a chain (basically a sequence) of low-rank matrix decompositions $\{(A_1, B_1), \ldots, (A_M, B_M)\}$, where $A_i \in \mathbb{R}^{r_i \times k}$, $B_i \in \mathbb{R}^{d \times r_i}$ and $r_i \ll min(d, k)$ for $1 \leq i \leq M$. Each low-rank tuple (A_i, B_i) is obtained by optimizing

$$\arg\min_{B_i A_i} \mathcal{L}(W_{pretrained} + \sum_{j=1}^{i} B_j A_j),$$

where $\mathcal{L}$ is the task-specific objective function. ***COLA follows an iterative residual learning paradigm.*** Fine-tuning each (A_i, B_i) can be viewed as learning the residual of $\Delta W^\star - \sum_{j=1}^{i-1} B_j A_j$, which is an easier optimization problem compared to learning $\Delta W^\star$ from scratch. We hypothesize that $\sum_{i=1}^{M} B_i A_i$ approximates $\Delta W^\star$ better than a single LoRA update BA, and we design a chaining framework to achieve this with less computation compared to the baseline LoRA.

COLA forms a chain of LoRAs by iteratively tuning, merging, and extending LoRA modules, as depicted in Figure 1. We denote the length of the chain in COLA as the number of residual LoRA modules optimized. For COLA with a chain length of M, the three sub-steps in Figure 1 are repeated M times. Below we describe the three sub-steps in detail.

Tune LoRA In this step, we perform standard LoRA tuning, i.e., learning only the A and B matrices and leaving all other model parameters untouched. At initialization of COLA, this step learns LoRA modules (A_1, B_1) on top of the frozen pre-trained LLM weights $W_{pretrained}$. After the initial phase of COLA, the LoRA modules (A_i, B_i) are fine-tuned on top of fixed model weights incorporated with previously learned LoRAs' weights. The fixed model weights at the i-th iteration of COLA is $W_{pretrained} + \sum_{j=1}^{i-1} B_j A_j$.

Tie a knot After the current LoRA modules (A_i, B_i) are trained, we merge them into the previously frozen LLM weights and we refer to this step as "tie a knot". This way, we incorporate the weight update, approximated by $B_i A_i$, into the frozen model weights. The resulting frozen model weights becomes $W_{pretrained} + \sum_{j=1}^{i} B_j A_j$. This allows learning only the residual information $\Delta W^\star - \sum_{j=1}^{i} B_j A_j$ for the next iteration. Additionally, merging the LoRA modules into the frozen LLM helps reduce memory burden under limited resource scenarios. Instead of storing a list of LoRA modules introduced in the COLA, merging them to the frozen model weights in a running fashion helps keep the GPU memory consumption the same as training LoRA only once.

Extend the chain We extend the COLA chain by re-initializing a new set of LoRA module (A_{i+1}, B_{i+1}) to learn the residual weights update needed to adapt the LLM to certain task. In this step, the newly introduced A_{i+1} adopts Gaussian initialization and B_{i+1} is initialized to zero, following Hu et al. (2021). Additionally, we reset all of the optimizer states, including but not limited to the parameters to be optimized and the gradient history.

4 Convergence of COLA and the Non-convex Frank-Wolfe method

The COLA algorithm described in Figure 1 is motivated by and closely related to the Frank Wolfe

algorithm (Frank et al., 1956). To see this, notice that COLA is an iterative algorithm whose iterations are succinctly described by the equation

$$W \leftarrow W + \arg\min_{BA} \mathcal{L}(W + BA).$$

Taking the linear Taylor approximation we can write

$$\mathcal{L}(W + BA) \approx L(W) + \nabla\mathcal{L}(W) \times BA,$$

and thus, a constrained minimization over a set $\mathcal{K} \subseteq \mathcal{R}^d$ can be seen to be approximately

$$\arg\min_{BA\in\mathcal{K}} \mathcal{L}(W+BA) \approx \arg\min_{BA\in\mathcal{K}} \nabla\mathcal{L}(W)\times BA.$$

This is reminiscent of the Frank-Wolfe algorithm, which was historically developed in the context of linear programming. Below we analyze a variant of the Frank Wolfe algorithm for stochastic non-convex smooth optimization. The algorithm pseudo-code is given in Algorithm 2, and it is written in COLA notations as an application to fine tuning of LLM. The stochasticity is captured in equation (1), where it is assumed that the direction of the gradient is approximated up to ε using a stochastic gradient method.

Algorithm 2 Idealized COLA

Input: step sizes $\{\eta_t \in (0,1],\ t \in [T]\}$, initial $W_1 \in \mathcal{K}$.
for $t = 1$ to T **do**
Approximate via stochastic optimization

$$\mathbf{V}_t \in_\varepsilon \arg\min_{W\in\mathcal{K}} \left\{W^\top \nabla\mathcal{L}(W_t)\right\} \quad (1)$$

$W_{t+1} \leftarrow W_t + \eta_t(\mathbf{V}_t - W_t)$.
end for

Specifically, we assume that COLA performs gradient updates such that after every epoch we have that

$$\mathbf{V}_t^\top \nabla\mathcal{L}(W_t) \le \arg\min_{W\in\mathcal{K}} \left\{W^\top \nabla\mathcal{L}(W_t)\right\} + \varepsilon.$$

Notice that we have replaced the low rank matrices A, B with a single matrix W. This deviates from the exact specification of COLA, but can be justified according to the following intuition. Linear optimization over the trace norm ball results in a rank one solution, as shown in the context of the Frank Wolfe method in Hazan (2008); Allen-Zhu et al. (2017). In COLA, we perform non-convex optimization over A, B directly, and their rank can be larger than one.

Below we give an analysis of this algorithm which incorporates the stochastic approximation of the iterates A_t, B_t. Henceforth, let $h_t = \mathcal{L}(W_t) - \mathcal{L}(W^*)$, and

$$g_t \triangleq \left\{\max_{\mathbf{V}\in\mathcal{K}} \nabla\mathcal{L}(W_t)^\top(\mathbf{V} - W_t)\right\}.$$

The latter quantity is a metric of convergence in non-convex optimization, which is sometimes called the Frank-Wolfe gap. Notice that g_t is zero if and only if the projected gradient of $\mathcal{L}$ at W_t is zero.

The following theorem establishes that Algorithm 2 guarantees average duality gap approaching zero for stochastic smooth non-convex optimization, as long as the distribution shift is bounded sublinearly with time.

Theorem 4.1. *Algorithm 2 applied to a sequence of stochastic gradients of β-smooth non-convex functions that are bounded in $\mathcal{K}$ by M, with step sizes $\eta_t = \frac{\sqrt{M}}{D\sqrt{\beta T}}$ attains the following convergence guarantee*

$$\frac{1}{T}\sum_{t=1}^{T} g_t \le \frac{2\sqrt{M\beta}D}{\sqrt{T}} + \varepsilon$$

Proof. We denote $\nabla_t = \nabla\mathcal{L}(W_t)$. For any set of step sizes, we have

$$\begin{aligned}
h_{t+1} &= \mathcal{L}(W_{t+1}) - \mathcal{L}(W^\star) \\
&= \mathcal{L}(W_t + \eta_t(\mathbf{V}_t - W_t)) - \mathcal{L}(W^\star) \\
&\le \mathcal{L}(W_t) - \mathcal{L}(W^\star) + \eta_t(\mathbf{V}_t - W_t)^\top\nabla_t \\
&\quad + \eta_t^2\tfrac{\beta}{2}\|\mathbf{V}_t - W_t\|^2 \qquad \text{smoothness} \\
&\le \mathcal{L}(W_t) - \mathcal{L}(W^\star) + \eta_t(\mathbf{V}_t - W_t)^\top\nabla_t \\
&\quad + \eta_t^2\tfrac{\beta}{2}D^2 \\
&\le h_t + \eta_t(g_t + \varepsilon) + \eta_t^2\tfrac{\beta D^2}{2}. \quad \mathbf{V}_t \text{ choice.}
\end{aligned}$$

Here we denoted by D the diameter of the set $\mathcal{K}$. We reached the equation $g_t + \varepsilon \le \frac{h_t - h_{t+1}}{\eta_t} + \eta_t\frac{\beta D^2}{2}$. Summing up over all iterations and normalizing we get ,

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T} g_t + \varepsilon &\le \tfrac{h_0 - h_T}{\eta T} + \eta\beta D^2 \\
&\le \tfrac{M}{\eta T} + \eta\beta D^2 \\
&\le \tfrac{2\sqrt{M\beta}D}{\sqrt{T}},
\end{aligned}$$

which implies the Theorem. □

Task	SST-2	WSC	CB	WIC	BoolQ	MultiRC	RTE	DROP	SQuAD	COPA	ReCoRD
LoRA	93.16	56.53	75.35	63.47	70.70	68.94	72.49	30.89	83.23	75.80	70.80
COLA (ours)	**93.32**	**60.19**	**76.42**	**64.26**	**72.08**	**70.63**	**74.15**	**31.49**	**83.56**	**76.80**	**71.02**
relative gains	0.17%	6.47%	1.42%	1.24%	1.95%	2.45%	2.29%	1.94%	0.39%	1.31%	0.31%
Finetune	93.33	60.00	72.50	62.73	68.44	70.36	71.62	31.34	83.07	77.50	72.14

Table 1: Experiments on OPT-1.3B with 1,000 test examples over various tasks. Task performance is reported after averaging over five random seeds. COLA consistently outperforms LoRA across all tasks.

5 Experimental Setup

In this section, we initially outline the tasks and models, followed by an introduction to the methods under comparison in our study. Finally, we provide details on the implementation.

5.1 Models and Tasks

Models We experiment with OPT-1.3B (Zhang et al., 2022) and Llama2-7B (Touvron et al., 2023). We use the pre-trained checkpoints from HuggingFace for both models.

Datasets We evaluate the effectiveness of our method and compare it with the LoRA baseline on task adaptation across classification, multiple-choice, and generation tasks. Following the benchmark selection in Malladi et al. (2023), we use SST-2, WSC, CB, WIC, BoolQ, MultiRC, and RTE for classification tasks. For multiple-choice tasks, we evaluate on COPA and ReCoRD. For generation tasks, we use DROP and SQuAD.

Methods Compared We compare COLA with LoRA and full parameter fine-tuning. For Llama2-7B experiments, we also add in-context learning (ICL) and 0-shot performance.

5.2 Implementation Details

We implemented our method with the PyTorch and Transformers library (Wolf et al., 2020). All experiments are carried out on NVIDIA A100 (80G) GPU.

For comprehensive experimental details, including information on the dataset, hyperparameters and LoRA implementations, please refer to Appendix A.1.

6 Results and analysis

6.1 Main Results

We report the test performance of our method and baseline across various tasks in this section. The experiment results on OPT-1.3B are detailed in Table 1, and the results for Llama2-7B are provided in Table 2. Notably, our method consistently outperforms LoRA on all datasets under the same training budget and inference cost, showcasing its superior performance.

Specifically, for OPT-1.3B experiments, COLA brings a performance boost to LoRA by 3.66 (relative improvement of 6.47%), 1.38 (relative improvement of 1.95%), 1.66 (relative improvement of 2.29 %) on tasks WSC, BoolQ and RTE, respectively. For Llama2-7B experiments, COLA boosts the test score on WSC from 57.30 to 59.80, which corresponds to a 2.5 gain and 4.36% relative improvement.

In our reported results, as detailed in Table 1 and Table 2, we maintain consistency by setting the rank of all injected modules in the sequence to 8, aligning with the baseline LoRA setup. Additionally, we use an equal training epoch budget for different methods and thus ensuring the same training computation cost, as explained in Appendix A.5.

6.2 Ablation Study

Different number of LoRAs in the chain As described in Section 3.2, COLA consists of repeated iterations of LoRA tuning and merging. We denote the length of COLA as the number of LoRAs learned and merged in the fine-tuning process. To investigate the effect of the chain length of COLA on task adaptation performance, we further conduct experiments by varying the length of COLA. Specifically, we studied chain length of 1, 2, 3 and present the findings in Table 3 and Figure 2. In Figure 2, test score refers to test accuracy for tasks WSC, CB, Copa, MultiRC and RTE. For SQuAD, test score specifically denotes the f1 score.

Here, chain length of 1 corresponds to the baseline LoRA fine-tuning. All experiments are conducted with a total of 5 training epochs. For example, in COLA experiments with chain length of 2,

Task	WSC	CB	RTE	Copa	SQuAD
LoRA	57.30	91.78	85.70	84.59	90.66
COLA (ours)	**59.80**	**93.21**	**86.21**	**85.60**	**90.76**
relative improvement	4.36%	1.56%	0.59%	1.19%	0.11%
Finetune	62.30	90.35	86.35	86.40	91.19
ICL	62.50	82.14	72.56	91.00	86.81
0-shot	36.53	32.14	62.09	79.00	55.84

Table 2: Experiments on Llama2-7B with 1,000 test examples over various tasks. Task performance is reported after averaging over five random seeds. COLA consistently outperforms LoRA across all tasks.

	WSC	SQuAD	MultiRC
length = 1	56.53 (± 7.67)	83.23 (± 1.42)	68.94 (± 3.06)
length = 2	59.81 (± 4.10)	83.28 (± 1.35)	69.44 (± 1.55)
length = 3	**60.19 (± 3.77)**	**83.55 (± 0.80)**	**70.63 (± 2.12)**
	CB	**Copa**	**RTE**
length = 1	75.35 (± 4.84)	75.8 (± 2.40)	72.49 (± 2.39)
length = 2	**76.78 (± 6.38)**	76.8 (± 1.83)	72.63 (± 1.46)
length = 3	76.42 (± 4.97)	**76.8 (± 1.60)**	**74.15 (± 1.36)**

Table 3: Evaluation of COLA with varying chain length. Test score across tasks is reported using 5 random seeds and is presented in the "average (± standard deviation)" format. The highest average performance for each task is highlighted in bold.

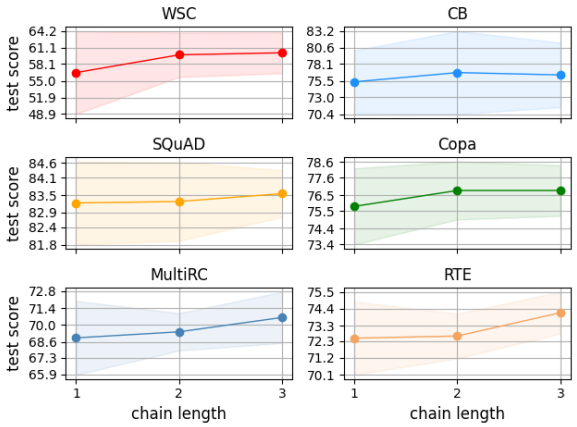

Figure 2: Test performance of COLA with varying chain length across tasks. Results are reported after averaging five different seeds and the shaded area corresponds to standard deviation. The general trend is that the test accuracy increases with the chain length.

the first LoRA training phase lasts from epoch 1 to epoch 3. After the first LoRA module merges with the pre-trained LLM weights and optimizer states reinitialize, the second LoRA starts from epoch 4 to epoch 5, which in total uses the same 5 total training epochs. All experiments results are reported over five random seeds.

As shown in Figure 2, there is a growing trend of test accuracy as the chain length increases across tasks. This is consistent with our hypothesis that residual learning of LoRA modules will lead to a better approximation of the optimal weight update to the fixed pre-trained LLM for task adaptation. For a majority of tasks, COLA is more robust in terms of generalization error compared to baseline LoRA, as shown by COLA's smaller standard deviations.

Different base optimizer We conduct experiments on the COLA framework with different base optimizers to show its effectiveness. We consider swapping the default AdamW optimizer with the SGD and AdaGrad (Duchi et al., 2011) optimizer. This ablation study is conducted on OPT-1.3B following the same experiment setup in Appendix A.1.

We report the average test score over five random seeds in Table 5. For both SGD and AdaGrad as the base optimizer, COLA outperforms the baseline LoRA across tasks, demonstrating the robustness of our framework.

For fine-tuning with SGD, the learning rate grid we searched is $\{2 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 2 \times 10^{-2}, 5 \times 10^{-2}\}$. For Adagrad, we search learning rate from $\{1 \times 10^{-3}, 8 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}\}$.

Rank step-down Since COLA operates on a residual learning paradigm, we are interested in exploring whether the residual weight updates can be learned with even lower ranks. This could potentially reduce the number of learnable parameters and computational costs while maintaining satisfactory test performance. Therefore, instead of using a chain of LoRAs with a fixed rank of eight, as described in Section 6.1, we conduct further studies

Methods	CB		WSC		WIC	
	test score	train FLOPs saved	test score	train FLOPs saved	test score	train FLOPs saved
LoRA	75.35	-	56.53	-	63.47	-
COLA (8, 8)	**76.78**	-	**59.81**	-	63.51	
COLA (8, 6)	76.43	3.60×10^{11}	58.26	4.28×10^{10}	63.85	5.21×10^{11}
COLA (8, 4)	75.35	7.20×10^{11}	57.30	8.56×10^{10}	**64.04**	1.04×10^{12}
COLA (8, 2)	76.07	1.08×10^{12}	57.30	1.28×10^{11}	63.19	1.56×10^{12}

Table 4: COLA rank step-down experiments. Test scores and train FLOPs saved compared to LoRA are reported. Method COLA (r_1, r_2) indicates that the first iteration learns LoRAs with rank r_1, and the second iteration learns LoRAs with rank r_2. All numbers are reported over five random seeds. COLA (8,8) uses the same amount of training FLOPs as the baseline, as denoted by "-".

		WSC	CB	WIC	Copa	SQuAD
SGD	LoRA	52.31	69.29	58.97	76.60	82.19
	COLA (ours)	**55.0**	**70.71**	**60.40**	**77.40**	**82.63**
AdaGrad	LoRA	56.73	69.29	63.42	76.60	83.09
	COLA (ours)	**61.92**	**73.21**	**64.23**	**76.60**	**83.24**

Table 5: Experiments of COLA with different base optimizers: SGD and AdaGrad.

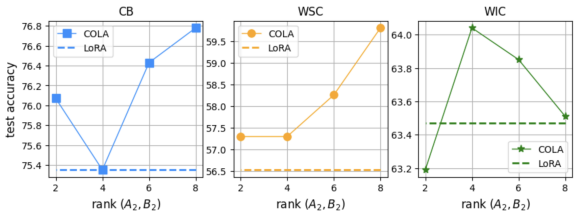

Figure 3: COLA with rank step-down. Experiments are conducted with COLA of length 2 where (A_1, B_1) has a fixed rank of 8, and (A_2, B_2) rank is as shown in the figure.

on lowering the rank.

Here, we consider a simple setting of COLA with length of two. We fix the rank to 8 for the first three epochs and set the rank for the remaining epochs to either 2, 4, 6, or 8. We show the results in Figure 3 and report the test performance in Table 4.

Figure 3 shows that COLA with rank step-down outperforms LoRA with a fixed rank of 8 for all tasks (with the exception of one data point–WIC with rank 2). Thus COLA with rank step-down offers both superior generalization ability over standard LoRA and lower computational cost. In addition, our results indicate that the optimal rank to use for COLA is task-dependent. The CB and WSC tasks both benefit from higher rank LoRA modules in the second learning phase. The WIC task, on the other hand, surprisingly shows maximal test accuracy at a rank of 4 for (A_2, B_2).

Computation comparison Table 4 provides a detailed comparison of the training computation cost between COLA of different rank step-down configurations and the baseline. We also include discussion on computational cost in Appendix A.5.

The training FLOPs are obtained from the HuggingFace trainer state, and are reported as the aggregate over five random seeds. The baseline LoRA uses a fixed rank of 8 throughout training, while COLA starts with rank 8 and continues with different ranks in the residual learning phase. As expected, stepping down the rank in the chain results in higher FLOPs savings. Overall, COLA offers lower generalization error with less compute.

Additional study We conduct additional ablation studies on COLA, exploring low-resource and large-scale settings, as well as the effects of augmentation ranks. For detailed results, please see Appendix A.2, A.3 and A.4.

7 Conclusions and future work

In this work, we introduce Chain of LoRA (COLA) for efficient fine-tuning of large language models. The idea is to use an iterative low rank residual learning procedure to approximate the optimal weight update needed for task adaptation. Experimental results show that COLA consistently outperforms LoRA albeit using the same, or less, computational resources.

Future work may explore automating the selection of hyperparamters involved in the optimization procedure such as the location to extend the COLA chain and the learning rate schedule. For example, one direction is to use the convergence behavior of the loss to determine where and whether to introduce additional LoRAs.

Limitations

Due to the fine-tuning cost, we conduct experiments and evaluation on subsets of training, validation and test data. For ablation study, we evaluate on representative tasks. Future work could involve investigating a wider variety of tasks.

References

Zeyuan Allen-Zhu, Elad Hazan, Wei Hu, and Yuanzhi Li. 2017. Linear convergence of a frank-wolfe type algorithm over trace-norm balls. *Advances in neural information processing systems*, 30.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*.

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

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Marguerite Frank, Philip Wolfe, et al. 1956. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110.

Dan Garber and Elad Hazan. 2016. A linearly convergent variant of the conditional gradient algorithm under strong convexity, with applications to online and stochastic optimization. *SIAM Journal on Optimization*, 26(3):1493–1528.

Elad Hazan. 2008. Sparse approximate solutions to semidefinite programs. In *LATIN*, pages 306–316.

Elad Hazan and Satyen Kale. 2012. Projection-free online learning. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

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

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

Martin Jaggi. 2013. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *ICML*.

Simon Lacoste-Julien. 2016. Convergence rate of frank-wolfe for non-convex objectives. *arXiv preprint arXiv:1607.00345*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. 2023. Stack more layers differently: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *In 7th International Conference on Learning Representations (ICLR)*.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *arXiv preprint arXiv:2305.17333*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Chengwei Qin, Wenhan Xia, Fangkai Jiao, and Shafiq Joty. 2023. Improving in-context learning via bidirectional alignment. *arXiv preprint arXiv:2312.17055*.

Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. 2016. Stochastic frank-wolfe methods for nonconvex optimization. In *2016 54th annual Allerton conference on communication, control, and computing (Allerton)*, pages 1244–1251. IEEE.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 2023. Tied-lora: Enhacing parameter efficiency of lora with weight tying. *arXiv preprint arXiv:2311.09578*.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*.

Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5986–5995. PMLR.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. 2023. Multilora: Democratizing lora for better multi-task learning. *arXiv preprint arXiv:2311.11501*.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

A Appendix

A.1 Implementation Details

We adopt the experimental setup outlined in Malladi et al. (2023), where we randomly select 1000 examples for training, 500 for validation, and another 1000 for testing across each dataset under consideration. In COLA training, we use AdamW (Loshchilov and Hutter, 2019) as the default base optimizer and train for a total of 5 epochs. For a fair comparison, we keep the total epoch number consistent with our baseline. A linear learning rate schedule is applied with the initial learning rate selected from $\{1 \times 10^{-3}, 8 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}\}$ for both COLA and LoRA experiments. The batch size is set to 8 for OPT-1.3B experiments and 4 for Llama2-7B experiments. For the Llama2-7B full parameter fine-tuning baseline, we report the best results by searching the learning rate from $\{1 \times 10^{-7}, 5 \times 10^{-7}, 8 \times 10^{-7}, 1 \times 10^{-6}, 5 \times 10^{-6}, 8 \times 10^{-6}, 1 \times 10^{-5}, 5 \times 10^{-5}, 8 \times 10^{-5}\}$. For full parameter fine-tuning of OPT-1.3B, the learning rate grid is set to $\{1 \times 10^{-6}, 5 \times 10^{-6}, 8 \times 10^{-6}, 1 \times 10^{-5}, 5 \times 10^{-5}, 8 \times 10^{-5}\}$. The reported results represent the best score after hyperparameter grid-search for all experiments, conducted over five random seeds.

In implementing LoRA, we adhere to the practice outlined in Hu et al. (2021), introducing trainable linear low-rank modules to both query and value projections within all self-attention layers. While some research has explored the application of LoRA to all projection matrices or all weight matrices, the specific choice of where to apply LoRA is not a pivotal aspect of our work (Zhang et al., 2023). For OPT experiments, we incorporate bias into the injected LoRA modules, aligning with the approach taken in Mahabadi et al. (2021). Conversely, in Llama-2 experiments, we deliberately disable bias in LoRA to ensure module key matching with the pre-trained checkpoint "meta-llama/Llama-2-7b-hf." We set the rank of LoRA (denoted as "r") to 8 and α to 16, where the ratio α/r is employed to scale the weight updates.

A.2 Performance of COLA in low data setting

In the preceding sections, we report performance of COLA with the experiment setup detailed in Appendix A.1, where we use 1000 training samples, 500 validation samples, and 1000 test samples across each dataset under consideration. LoRA is extremely effective in low-resource scenarios as

Model	Task										
	SST2	WSC	CB	WIC	BoolQ	MultiRC	RTE	SQuAD	DROP	Copa	ReCord
100 data											
LoRA	89.80	61.60	72.76	58.80	61.60	59.00	59.20	78.99	27.99	75.60	70.00
COLA	**90.00**	**63.20**	**73.21**	**60.40**	**64.19**	**60.60**	**63.40**	**79.63**	**29.01**	**76.19**	**70.20**
Relative gains	0.22%	2.60%	0.62%	2.72%	4.20%	2.71%	7.09%	0.81%	3.64%	0.78%	0.28%
50 data											
LoRA	87.60	60.80	70.00	60.00	64.80	56.80	59.20	78.03	22.32	76.40	68.80
COLA	**89.60**	**61.60**	**70.80**	**62.40**	**67.60**	**61.20**	**62.80**	**78.59**	**24.10**	**78.00**	**70.00**
Relative gains	2.28%	1.31%	1.14%	4.00%	4.32%	7.75%	6.08%	0.72%	7.97%	2.09%	1.74%

Table 6: Performance comparison of LoRA and COLA under low-resource setting

	WIC	SST2	BoolQ
LoRA	68.18	94.56	77.02
COLA	**69.53**	**94.89**	**77.66**

Table 7: Performance Comparison of LoRA and COLA with 5000 training samples.

well. Consequently, we extend our investigations of COLA under low-data conditions to assess its effectiveness.

Specifically, we consider low-resource settings with 100 training samples and 50 training samples. We report the test results of OPT-1.3B in Table 6. For experiments with 100 training samples, we randomly select 100 test samples for evaluation. For experiments on 50 training samples, we randomly select 50 test samples.

The test results in Table 6 conclusively demonstrate that COLA consistently surpasses the baseline performance under low-resource settings. For training with 100 samples, COLA achieves up to 7.09% test score improvement on top of LoRA. For training with 50 samples, COLA achieves up to 7.97% test score relative gains compared to LoRA.

A.3 Performance of COLA in large scale setting

To evaluate the scalability and efficacy of COLA in larger-scale tasks with substantial fine-tuning datasets, we conduct experiments involving 5000 training samples while maintaining the experimental setup as previously outlined.

Notably, among the 11 benchmark tasks considered in this research, only seven tasks possess training sets exceeding the 5000-sample threshold. We experiment on three representative tasks and present the average test scores with five random seeds for COLA and LoRA in Table 7. As the results show, COLA consistently outperforms the baseline in larger-scale fine-tuning setting with 5000 training samples.

	Tasks				
	SST2	WSC	WIC	RTE	SQuAD
rank=16					
LoRA	93.07	55.96	62.31	72.70	83.22
COLA	**93.53**	**57.88**	**62.60**	**73.72**	**83.57**
rank=32					
LoRA	93.14	57.50	62.94	72.49	83.18
COLA	**93.56**	**61.15**	**63.73**	**74.51**	**83.57**
rank=64					
LoRA	92.95	54.80	62.28	**75.66**	82.97
COLA	**93.62**	**63.08**	**64.23**	75.60	**83.44**

Table 8: Experiments with varying LoRA ranks. Test score across tasks is reported using 5 random seeds. The highest average performance for each task is highlighted in bold.

A.4 Effects of varying augmentation ranks

The main results reported so far are obtained with the augmentation rank set to 8. This choice of rank is primarily guided by empirical performance considerations and aligns with the hyper-parameter selection utilized in (Malladi et al., 2023). (Hu et al., 2021) systematically explored the impact of varying the augmentation matrix's rank used in LoRA.

In this context, we focus on evaluating our framework's performance under different rank settings. To this end, we conduct three additional sets of experiments with ranks set to 16, 32, and 64, as demonstrated in Table 8. We present the baseline

LoRA results corresponding to each rank across various tasks. For COLA's performance under each rank, we report the test scores of the three-stage COLA configuration, wherein all stages employ a consistent rank.

From Table 8, we observe consistent findings akin to those reported in (Hu et al., 2021), indicating that varying rank of LoRA offers competitive performance with the optimal rank being task-dependent. Nonetheless, COLA shows superiority to LoRA across tasks and ranks.

A.5 Training and Inference cost of COLA

The training cost of COLA is determined by the rank of the LoRA modules used to form the chain. The training computation for COLA is the same as LoRA when the rank is the same. In COLA, progressively lowering the rank of the LoRA modules may be an effective strategy to approximate optimal residual weight updates for specific tasks and lower the overall training cost. We explore this direction in our experiment section. At inference, all of the learned B_jA_j can be integrated into the original model weights. Since $W_{\text{pretrained}}$ has the same shape as B_jA_j, the final integrated model weight has the same number of parameters as the original pre-trained LLM. Therefore, no latency overhead is introduced during inference.