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

**Wenhan Xia** [*1] **Chengwei Qin** [*2] **Elad Hazan** [1]

## 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. Fine-tuning these models for specific tasks enhances their performance in downstream applications (Lewis et al., 2019; Wang et al., 2021; Qin et al., 2023). However, full parameter fine-tuning is computationally expensive and demanding, especially given the increasing size of large language models.

For this reason, parameter-efficient fine-tuning (PEFT) methods, such as Low-Rank Adaptation (LoRA), have gained popularity (Houlsby et al., 2019; Lester et al., 2021; Hu et al., 2021). LoRA updates only a small portion of the model's weights by adding trainable low-rank matrices to the model, reducing computational costs and time. However, it is inferior to full parameter fine-tuning in terms of generalization error.

This paper explores reducing the generalization gap between LoRA and full fine-tuning while maintaining the computational efficiency. We propose "Chain of LoRA" (COLA), which learns a higher rank augmentation of the LLM weights by the method of residual learning. Inspired by the Frank-Wolfe algorithm for matrix completion, COLA iteratively adds low-rank structures to improve the model's generalization performance. In this paper, we formalize the relationship between COLA and the Frank-Wolfe method from mathematical optimization. We provide theoretical analyses and demonstrate empirical effectivess of COLA via extensive experiments across datasets and models.

## 2. Related Work

Full-parameter fine-tuning faces practical challenges for increasingly larger models and tasks. Recent advancements in parameter-efficient fine-tuning address this by modifying only a small portion of parameters.

**Adapter based methods:** Adapter-based approaches introduce compact modules between transformer layers and train only these lightweight adapters while keeping the pre-trained model frozen (Houlsby et al., 2019; Bapna & Firat, 2019; Mahabadi et al., 2021).

**Prefix tuning methods:** Alternative research explores adding tunable lightweight parameters, called prefixes, to the input and hidden layers (Li & Liang, 2021). Efficient prompt tuning simplifies this by adding a trainable tensor ("soft prompt") to input embeddings (Lester et al., 2021).

[*]Equal contribution [1]Department of Computer Science, Princeton University [2]School of Computer Science and Engineering, Nanyang Technological University. Correspondence to: Wenhan Xia <wxia@princeton.edu>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. 

**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. Followup work explores variants quantization, context sizes extension, and multi-task adaptation (Dettmers et al., 2023; Chen et al., 2023; Wang et al., 2023). Lialin et al. (2023) [1] explores pre-training with multiple low rank matrices.

## 3. Our Method

### 3.1. Preliminary: Low Rank Adaptation (LoRA)

Given a pre-trained weight matrix $W_{frozen}$, LoRA (Hu et al., 2021) approximates the weight update $\Delta W$ for task adaptation with trainable low-rank decomposition matrices $BA$. The forward pass is: $W_{frozen}x + \Delta W x = W_{frozen}x + \mathbf{BA}x$. Here $W_{frozen}, \Delta W \in \mathbb{R}^{d\times k}$, $A \in \mathbb{R}^{r\times k}$, $B \in \mathbb{R}^{d\times r}$ and $r \ll min(d,k)$. During training, $W_{frozen}$ is frozen and only $B$, $A$ are optimized. At deployment, the learned $BA$ merge with the pre-trained weights.

### 3.2. Chain of LoRA

The key idea of our method is to form a chain of LoRAs and iteratively learn the low-rank adaptation LoRA modules. As illustrated in Figure 1, our method involves three steps: **Tune LoRA, Tie a knot, Extend the chain**. We present the detailed procedure in Algorithm 2 in Appendix A.1.

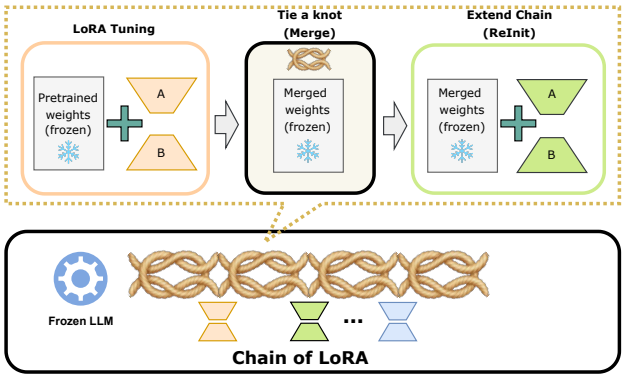


Figure 1: Chain of LoRA learns a sequence of low-rank matrices to approximate a high-rank augmentation in three steps: (1) LoRA Tuning, (2) Tie a Knot - merge LoRA weights into the model, and (3) Extend the Chain - add a new LoRA module and reset the optimizer. These steps are repeated in the residual learning paradigm.

For a pre-trained LLM weight matrix $W_{pretrained} \in \mathbb{R}^{d\times k}$, we denote the weights update during fine-tuning as $\Delta W$. Ideal task adaptation yields the optimal final weights $W^\star$ and the optimal weight update $\Delta W^\star$, as in $W^\star = W_{pretrained} + \Delta W^\star$.

In COLA, we propose to approximate $\Delta W^\star$ with a chain/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 $1 \le i \le M$. Each low-rank tuple $(A_i, B_i)$ is obtained by optimizing $\arg\min_{B_iA_i} \mathcal{L}(W_{pretrained} + \sum_{j=1}^{i} B_jA_j)$, 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 $\Delta W^\star - \sum_{j=1}^{i-1} B_jA_j$. We hypothesize that $\sum_{i=1}^{M} B_iA_i$ approximates $\Delta W^\star$ better than a single LoRA update $BA$, and we design a framework to achieve this with less computation compared to the baseline LoRA. Below, we describe in detail the three steps involved in Figure 1.

**Tune LoRA**: This step entails standard LoRA tuning, where $(A_i, B_i)$ are fine-tuned on fixed model weights at the i-th iteration. The fixed weights integrate previously learned LoRA weights, represented as $W_{pretrained} + \sum_{j=1}^{i-1} B_jA_j$.

**Tie a knot**: After the LoRA modules $(A_i, B_i)$ are learned, we merge them with the frozen LLM weights to incorporate the weight update into the frozen model. This allows learning only the residual information $\Delta W^\star - \sum_{j=1}^{i} B_jA_j$ for the next iteration.

**Extend the chain**: We extend the chain by initializing new LoRA modules $(A_{i+1}, B_{i+1})$ to learn residual weight updates needed for task adaptation. We reset all optimizer states, including parameters and gradient history.

## 4. Convergence of COLA and the Nonconvex Frank-Wolfe method

The COLA algorithm described in Algorithm 2 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 $W \leftarrow W + \arg\min_{BA} \mathcal{L}(W + BA)$.

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. Below we analyze a variant of the Frank Wolfe algorithm, presented in Algorithm 1, for stochastic non-convex smooth optimization. The stochasticity is captured in equation (1), where it is

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

Table 1: Results on OPT-1.3B with 1,000 test examples across diverse tasks. Test score is reported over five random seeds.

| 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 |

assumed that the direction of the gradient is approximated up to $\varepsilon$ using a stochastic gradient method.

**Algorithm 1** 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

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

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

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

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

Notice that we have replaced the low rank matrices $A, B$ with a single matrix $W$. This can be justified by 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 convergence metric 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 1 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 1 applied to a sequence of stochastic gradients of $\beta$-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*

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

Table 2: Results on Llama2-7B with 1,000 test examples over five random seeds.

| 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 |

The proof is provided in Appendix A.2

## 5. Experimental Results

### 5.1. Experimental Setup

**Models:**We experiment with OPT-1.3B (Zhang et al., 2022) and Llama2-7B (Touvron et al., 2023).

**Datasets:** We follow the benchmark selection in Malladi et al. (2023). We consider classification, multiple-choice and generation tasks.

**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.

Implementation details are provided in Appendix A.3.

### 5.2. 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.

For our reported results in Table 1 and Table 2, we maintain

Table 3: 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 $r_2$. COLA (8,8) uses the same amount of training FLOPs as the baseline, as denoted by "-".

| | CB | | WSC | | WIC | |
|---|---|---|---|---|---|---|
| Methods | 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 | $3.60\times10^{11}$ | 58.26 | $4.28\times10^{10}$ | 63.85 | $5.21\times10^{11}$ |
| COLA (8, 4) | 75.35 | $7.20\times10^{11}$ | 57.30 | $8.56\times10^{10}$ | **64.04** | $1.04\times10^{12}$ |
| COLA (8, 2) | 76.07 | $1.08\times10^{12}$ | 57.30 | $1.28\times10^{11}$ | 63.19 | $1.56\times10^{12}$ |

consistency by setting the rank of all injected modules in the sequence to 8, aligning with the baseline LoRA setup. We set the fine-tuning epochs to 5 for all methods.

### 5.3. Ablation Study

**Chain length**: We denote the length of COLA as the number of LoRAs involved in the fine-tuning process. To investigate the effect of the chain length of COLA on task adaptation performance, we conduct experiments by varying the length of COLA. Specifically, we studied chain length of 1, 2, 3 and present the findings in Figure 2.

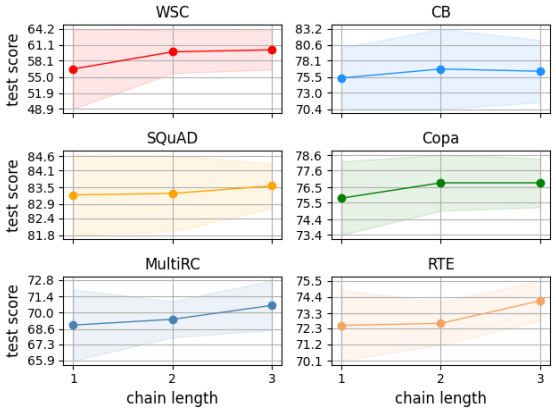


Figure 2: Performance of COLA with varying chain length.

Figure 2 shows a growing trend of test score with increasing chain length, which supports our hypothesis that residual learning of LoRA modules better approximates the optimal weight update for task adaptation.

**Rank step-down**: Instead of using a chain of LoRAs with a fixed rank, we conduct further studies on lowering the rank progressively. 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. Table 3 shows the test performance and training cost between COLA of different rank step-down configurations and the baseline.

Table 4: Experiments of COLA with different base optimizers: SGD and AdaGrad.

| | | 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** |

Here, COLA starts with rank 8 and continues with lower ranks in the residual learning phase. As expected, stepping down the rank in the chain results in higher FLOPs savings. Some rank step-down configurations show superior test scores over the baseline albeit using less training FLOPs. Overall, COLA offers lower generalization error with less compute.

**Different base optimizer**: We conduct ablation study with different base optimizers to show COLA's effectiveness. We consider swapping the default AdamW optimizer with the SGD and AdaGrad (Duchi et al., 2011). We experiment with OPT-1.3B following the experiment setup in Appendix A.3 and report test scores over five random seeds in Table 4. For both SGD and AdaGrad as the base optimizer, COLA outperforms the baseline LoRA across tasks, demonstrating the robustness of our framework.

## 6. 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 hyperparameters 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 conver-

gence behavior of the loss to determine where and whether to introduce additional LoRAs.

## References


Allen-Zhu, Z., Hazan, E., Hu, W., and Li, Y. Linear convergence of a frank-wolfe type algorithm over trace-norm balls. *Advances in neural information processing systems*, 30, 2017.

Bapna, A. and Firat, O. Simple, scalable adaptation for neural machine translation. In Inui, K., Jiang, J., Ng, V., and Wan, X. (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1538–1548, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1165. URL https://aclanthology.org/D19-1165.

Chen, Y., Qian, S., Tang, H., Lai, X., Liu, Z., Han, S., and Jia, J. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

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

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

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

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Li, X. L. and Liang, P. 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, August 2021.

Lialin, V., Shivagunde, N., Muckatira, S., and Rumshisky, A. Stack more layers differently: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

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

Mahabadi, R. K., Ruder, S., Dehghani, M., and Henderson, J. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *arXiv preprint arXiv:2305.17333*, 2023.

Qin, C., Xia, W., Jiao, F., and Joty, S. Improving in-context learning via bidirectional alignment. *arXiv preprint arXiv:2312.17055*, 2023.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wang, Y., Wang, W., Joty, S., and Hoi, S. C. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

Wang, Y., Lin, Y., Zeng, X., and Zhang, G. Multilora: Democratizing lora for better multi-task learning. *arXiv preprint arXiv:2311.11501*, 2023.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

# A. Appendix

## A.1. COLA algorithm

**Algorithm 2** 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 params 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**

## A.2. Proof of Theorem 4.1

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

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

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

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

which implies the Theorem. □

## A.3. 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.

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 & 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 $\{1\times10^{-3}, 8\times10^{-4}, 5\times10^{-4}, 1\times10^{-4}, 5\times10^{-5}\}$ 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 $\{1\times10^{-6}, 5\times10^{-6}, 8\times10^{-6}, 1\times10^{-5}, 5\times10^{-5}, 8\times10^{-5}\}$. 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 $\alpha$ to 16, where the ratio $\alpha/r$ is employed to scale the weight updates.