Mitigating Forgetting in Low Rank Adaptation

Joanna Sliwa [1] **Frank Schneider** [1] **Philipp Hennig** [1] **José Miguel Hernández-Lobato** [2]

Abstract

Parameter-efficient finetuning (PEFT) enables quick adaptation of large pre-trained language models to different downstream applications. However, this process often leads to catastrophic forgetting of the model's original domain knowledge. We address this issue with **LALoRA**, a weight-space regularization method that applies a **L**aplace **A**pproximation to **Lo**w-**R**ank **A**daptation. We estimate how confident the model is in each parameter and constrain updates in high-confidence directions. This preserves original knowledge while still allowing efficient target domain learning. We showcase the improved learning-forgetting trade-off compared to existing baseline methods and discuss different approximations of the loss landscape curvature, through which we estimate the parameters' uncertainty.

1. Introduction

The ability to finetune large pre-trained foundation models is essential for many downstream applications. However, full finetuning i.e. updating all parameters of a large model is often prohibitively expensive. For example, Hu et al. (2021) observed that finetuning GPT-3 175B requires about 1.2 TB of VRAM, and is therefore often impractical. To address this, they proposed Low-Rank Adaptation (LoRA), which reduces the number of trainable parameters by restricting training to low-rank adapter layers. For a transformer layer with input dimension D_{in} and output dimension D_{out}, and an adapter rank $r \ll \min(D_{\text{out}}, D_{\text{in}})$, LoRA cuts the trainable parameters per layer from $D_{\text{in}} \cdot D_{\text{out}}$ down to $r \cdot (D_{\text{in}} + D_{\text{out}})$. In practice, one often trains only about 0.01% of the original parameters.

As Biderman et al. (2024) observed, compared to full finetuning, LoRA better preserves pre-training knowledge (i.e., suffers less forgetting), although some drop in source domain accuracy still occurs. This raises a key question: how can we maintain as much of the original knowledge as possible, while still achieving good downstream performance? To improve this learning-forgetting trade-off, different approaches have been proposed, assuming no direct access to pre-training data: MIGU (Du et al., 2024) constrains updates to parameters determined by activation patterns, while other methods update only the minor or principal directions of the weight matrices (Wang et al., 2025; Meng et al., 2025). However, additional methods are needed to further reduce the forgetting rate.

Building on this line of work, we aim to tackle the problem of source domain forgetting while maintaining strong learning capabilities in the target domain. To achieve that, we utilize the uncertainty estimates to identify which weights are important for good source domain performance. Motivated by previous work for continual learning (Kirkpatrick et al., 2017; Ritter et al., 2018), we assume that parameters with low uncertainty are essential for solving the source domain problem and therefore shouldn't be changed. In contrast, parameters with high uncertainty have little impact on the source domain task and can be more freely adjusted to learn the target data. During finetuning, we will use the uncertainty information measure as a regularizer to discourage updates to critical parameters, thereby mitigating forgetting.

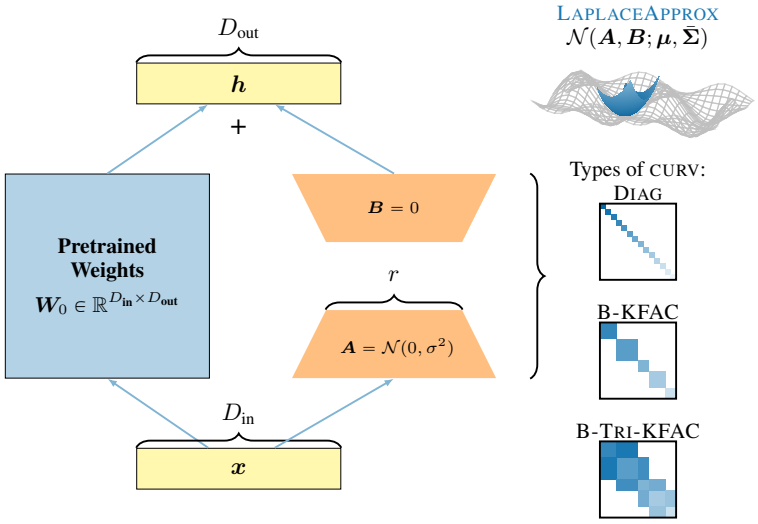
Contributions: We leverage Laplace approximation on the LoRA weights (LALoRA) to mitigate forgetting during finetuning. We introduce a regularizer that allows for a trade-off between learning and forgetting, controlled by the strength of regularization λ. We compare the performance to other methods and discuss variants of loss curvature approximations.

2. Background

Notation: We consider supervised learning with a target dataset $\mathbb{D}_{\text{T}} = \{\boldsymbol{b}_i = (\boldsymbol{x}_i, \boldsymbol{y}_i) \,|\, i = 1, \ldots, N\}$ containing training inputs $\boldsymbol{x}_i$ and outputs $\boldsymbol{y}_i$. The source domain consists of batches, coming from different sub-datasets which form the source dataset $\mathbb{D}_{\text{S}} = \{\mathbb{D}_{\text{S1}}, \ldots, \mathbb{D}_{\text{S}n_{\mathbb{D}_S}}\}$.The goal is to find parameters $\boldsymbol{W} \in \mathbb{R}^D$ of a model $f_{\boldsymbol{W}}$ that minimize a given *regularized* loss $\mathcal{L}_{\text{reg}}$.

[1]Tübingen AI Center, University of Tübingen, Germany [2]University of Cambridge, UK. Correspondence to: Joanna Sliwa <joanna.sliwa@uni-tuebingen.de>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.



Obtaining parameter uncertainty via LA

```
// Loop over source data batches
for b_s in D_S do
    // Compute loss curvature
    μ, Σ_s^{-1} = LAPLACEAPPROX(b_s, CURV)
    Σ̄^{-1} = accumulate(Σ_s^{-1})
Return μ, Σ̄^{-1}
```

Fine-tuning using regularized loss

```
// Loop over fine-tuning batches
for b_t in D_T do
    // Compute regularizer
    r = LAPLACEREG(W, b_t, μ, Σ̄^{-1})
    L_reg = −log p(b_t|W) + λ r(Wμ, Σ̄^{-1})
    // Finetune on regularized loss
    W = minimize(L_reg)
```

Figure 1: **Laplace regularizer maintains learning while preventing forgetting of the source domain.** The figure is organized as follows: the left panel illustrates the LoRA setup; the center panel visualizes the Laplace approximation, highlighting alternative curvature approximations used for estimating which weights are important for good source domain performance; and the right panel presents the LALoRA algorithm.

LoRA: Low Rank Adaptation of large (language) models (LoRA) (Hu et al., 2021) derives from the notion that weight updates during finetuning have a low intrinsic rank. Therefore, for the pre-trained weight $\boldsymbol{W}_0 \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$ the additive updates $\Delta\boldsymbol{W}$ have a low-rank decomposition $\Delta\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$ where $\boldsymbol{B} \in \mathbb{R}^{D_{\text{in}} \times r}$ and $\boldsymbol{A} \in \mathbb{R}^{r \times D_{\text{out}}}$. The training concentrates around the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ while $\boldsymbol{W}_0$ is kept frozen. The forward pass is given as

$$\boldsymbol{h} = \boldsymbol{W}_0\boldsymbol{x} + \Delta\boldsymbol{W}\boldsymbol{x}. \tag{1}$$

The matrix $\boldsymbol{A}$ is initialized with a random Gaussian noise and $\boldsymbol{B}$ with zeros. There have been many studies on how LoRA performs when compared to full finetuning (Biderman et al., 2024; Dettmers et al., 2023; Ghosh et al., 2024; Zhao et al., 2024a; Ivison et al., 2023; Zhuo et al., 2024).

Laplace approximations (LA): The Bayesian posterior over the model's parameters, $p(\boldsymbol{W} \mid \mathbb{D}_S)$, describes the belief over the values of each parameter and thus reflects (un)certainty about each parameter's value. It, therefore, identifies which parameters still offer flexibility to learn a new application. The Laplace approximation (e.g. MacKay, 1992; Daxberger et al., 2022) provides a local Gaussian approximation to this typically intractable posterior. It stems from a second-order Taylor expansion of the loss around the maximum a posteriori (MAP) estimate of the parameters, i.e. the trained $\boldsymbol{\mu}$, as $\mathcal{L}(\boldsymbol{W}, \mathbb{D}_S) \approx \mathcal{L}(\boldsymbol{\mu}, \mathbb{D}_S) + \frac{1}{2}(\boldsymbol{W} - \boldsymbol{\mu})^\top \bar{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\mu})(\boldsymbol{W} - \boldsymbol{\mu})$, where $\bar{\boldsymbol{\Sigma}}^{-1} = \nabla^2_{\boldsymbol{W}} \log p(\mathbb{D}_S, \boldsymbol{W}) \in \mathbb{R}^{D \times D}$ is the Hessian of the loss with respect to the parameters. This results in a Gaussian distribution $p(\boldsymbol{W} \mid \mathbb{D}_S) \approx \mathcal{N}(\boldsymbol{W}; \boldsymbol{\mu}, \bar{\boldsymbol{\Sigma}}(\boldsymbol{\mu}))$ called the Laplace approximation. Treating this approximation as a prior or a weight-space regularizer r transforms the loss for the target domain to $\mathcal{L}_{\text{reg}}(\boldsymbol{W}, \mathbb{D}_T) = -\log p(\mathbb{D}_T \mid \boldsymbol{W}) + \lambda r(\boldsymbol{W}, \boldsymbol{\mu}, \bar{\boldsymbol{\Sigma}}) = \log p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{W}) + \frac{\lambda}{2}(\boldsymbol{W} - \boldsymbol{\mu})^\top \bar{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\mu})(\boldsymbol{W} - \boldsymbol{\mu})$, with $\log p(\mathbb{D}_T \mid \boldsymbol{W})$ as the log-likelihood and λ as the regularization strength. We favor parameters near the source domain solution $\boldsymbol{\mu}$, allowing variation in low-curvature directions but preserving those whose change would sharply raise pre-training loss.

3. Methodology

The methodology consists of two stages described below.

STAGE 1 (Figure 1, right panel, blue box): To create a regularizer that captures which weights are crucial for the source domain performance, firstly, we need to compute the Laplace approximation on the source datasets. For that we need access to at least one mini-batch of source data or a representation of it, $\mathbb{D}_S$. Since, the pre-trained weights $\boldsymbol{W}_0$ remain frozen and fitting the Gaussian to all the weights of the model $\boldsymbol{W}$ would be infeasible, we do so only for the LoRA weights of chosen modules i.e. for $\Delta\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$. This results in much lower computation cost than fitting LA on the full network. The Laplace approximation is defined by

$$p(\Delta\boldsymbol{W} \mid \mathbb{D}_S) \approx \mathcal{N}(\Delta\boldsymbol{W}; \boldsymbol{\mu}, \bar{\boldsymbol{\Sigma}}) = \mathcal{N}\left(\boldsymbol{A}, \boldsymbol{B}; \begin{bmatrix}\boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B\end{bmatrix}, \bar{\boldsymbol{\Sigma}}\right). \tag{2}$$

The mean $\boldsymbol{\mu}$ is equal to the mode of the pre-trained weights—

for LoRA weights it is the initialization setting i.e. $\boldsymbol{\mu}_{\boldsymbol{A}}$ is Gaussian noise and $\boldsymbol{\mu}_{\boldsymbol{B}} = 0$. The precision matrix $\bar{\boldsymbol{\Sigma}}^{-1}$ is a sum of precisions of all source sub-datasets $\bar{\boldsymbol{\Sigma}}^{-1} = \Sigma_s^{n_{\mathbb{D}_S}} \boldsymbol{\Sigma}_s^{-1}$ and the choice of how to compute $\boldsymbol{\Sigma}_s^{-1}$ is discussed below. Both $\boldsymbol{\mu}$ and $\bar{\boldsymbol{\Sigma}}^{-1}$ are stored, now we have all necessary to construct a regularizer during the finetuning.

STAGE 2 (Figure 1, right panel, red box): After creating the regularizer, we can now finetune on the target data $\mathbb{D}_T$ with a regularized loss $\mathcal{L}_{\text{reg}}(\boldsymbol{W}, \boldsymbol{b}_t)$, defined as,

$$\mathcal{L}_{\text{reg}} = -\log p(\boldsymbol{b}_t|\boldsymbol{W}) + \lambda\, \boldsymbol{r}(\boldsymbol{W}, \boldsymbol{\mu}, \bar{\boldsymbol{\Sigma}}^{-1}) = \quad (3)$$

$$= -\log p(\boldsymbol{b}_t|\boldsymbol{W}) + \quad (4)$$

$$+ \text{vec}(\Delta\boldsymbol{W} - \boldsymbol{\mu})^\top \bar{\boldsymbol{\Sigma}}^{-1} \text{vec}(\Delta\boldsymbol{W} - \boldsymbol{\mu}).$$

where $-\log p(\boldsymbol{b}_t|\boldsymbol{W})$ is the negative log-likelihood and $\boldsymbol{r}$ is the regularizer scaled by λ. Minimizing the regularized loss (i) learns the target domain by maximizing the log-likelihood and (ii) mitigates forgetting of the source domain via minimizing the regularizer.

Curvature approximations. The challenge now is to efficiently compute the precision matrices Σ_s^{-1}. Firstly, we can simplify the precision $\boldsymbol{\Sigma}_s^{-1}$ to a diagonal

$$\boldsymbol{D} = \left(\frac{\partial \log p(\boldsymbol{y}\,|\,\boldsymbol{x}, \boldsymbol{W})}{\partial \boldsymbol{W}} \right)^2 \in \mathbb{R}^{r(D_{\text{in}} + D_{\text{out}}) \times 1}. \quad (5)$$

This DIAG approach results in the following regularizer $\boldsymbol{r} = \text{vec}(\boldsymbol{A} - \boldsymbol{\mu})^\top \boldsymbol{D} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu})$. The method is computationally efficient because it needs calculating only the gradient, which is then squared. However, it discards some information—specifically, the interactions among weights within a layer and across layers. In the Appendix C, we describe different methods based on Kronecker Factored approximation (KFAC), B-KFAC and B-TRI-KFAC, to address this issue. We showcase theoretically, how we could expand the regularizer to capture uncertainty information between weights in the same layer, i.e., within layer $\boldsymbol{A}$ and within layer $\boldsymbol{B}$, as well as, additionally between layers $\boldsymbol{A}$ and $\boldsymbol{B}$ for each adapter module $\Delta\boldsymbol{W}$.

4. Related Work

Forgetting the source domain in LoRA: Biderman et al. (2024) compare LoRA with full finetuning and measure performance on both the source (to quantify forgetting) and the target (to assess new learning). Their results show that LoRA retains noticeably more of the source-domain knowledge but gains less on the target domain. We adapt their source/target domain setup for our own experiments. In contrast, Shuttleworth et al. (2024) match LoRA and full finetuning on their final accuracy and show via SVD that LoRA introduces large singular directions orthogonal to the pre-trained weights, increasing forgetting. We further study this line of research inspecting updating only the uncertain directions of the pre-trained model. Du et al. (2024) try to mitigate forgetting in LoRA by using the differences in magnitude distribution of the L^1-normalized output in linear layers. They only update the parameters with large values in L^1-normalized magnitude based on a threshold $1-t$. In comparison, our method computes a metric based on the backward pass and not forward.

Laplace approximation with LoRA: Yang et al. (2024) apply a post-hoc Laplace approximation to the LoRA weights and report improved calibration of the finetuned model. The benefits, however, appear only with a Kronecker-factored approximation of the loss curvature, a diagonal approximation offers no gains. By contrast, our method incorporates the Laplace approximation *during* training as a regularizer, not after and pursues a different goal of source domain forgetting. We also extend the curvature approximation from block-diagonal to a block tri-diagonal form.

Initialization of LoRA weights: The default initialization sets $\boldsymbol{A}$ to Gaussian noise and $\boldsymbol{B} = 0$ (Hu et al., 2021). PiSSA (Meng et al., 2025) initializes $\boldsymbol{A}$ and $\boldsymbol{B}$ with the *top–r* singular values and vectors of the pre-trained matrix $\boldsymbol{W}_0$ (via SVD) and freezes the remainder, claiming faster convergence and higher accuracy. However, we anticipate that PiSSA will struggle with forgetting, because it updates the very top singular vectors that encode most of the source-domain information. In contrast, MiLoRA (Wang et al., 2025) updates just the *minor–r* singular values and vectors, initializing $\boldsymbol{A}$ and $\boldsymbol{B}$ in the orthogonal subspace to preserve pre-trained knowledge while adapting to downstream tasks.

5. Experiments

Our aim is to finetune the model on mathematical data so that it matches vanilla LoRA's performance on the target task while surpassing LoRA on the source domain, i.e., exhibiting *less* forgetting, as measured on commonsense reasoning datasets. We empirically test whether a simple diagonal LALoRA regularizer can achieve that. Specifically, we analyse the optimal choice of the regularization strength. Finally, we compare our method to other methods.

Datatsets and model: For the source pre-trained knowledge $\mathbb{D}_S$ we evaluate the model's commonsense reasoning performance (Biderman et al., 2024; Du et al., 2024) on HellaSwag ($\mathbb{D}_{S1}$, Zellers et al., 2019), WinoGrande ($\mathbb{D}_{S2}$, Sakaguchi et al., 2019) and ARC Challenge ($\mathbb{D}_{S3}$, Clark et al., 2018). We perform instruction finetuning for math knowledge on GSM-8k ($\mathbb{D}_T$, Cobbe et al., 2021). We employ LlaMA-3.2-3B with LoRA of rank $r = 32$.

Forgetting vs learning: We examine if the Laplace regularizer mitigates forgetting of the source domain. We compare a model trained with an unregularized loss i.e. LALoRA with $\lambda = 0$ (Baseline) and one with the addition of the diag-

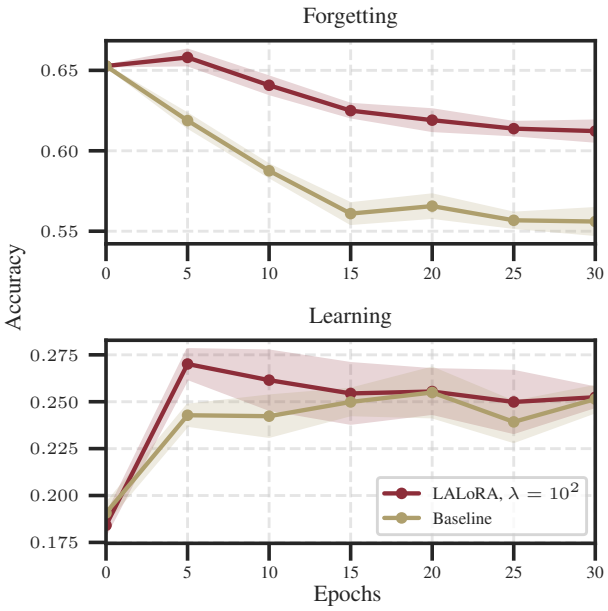


Figure 2: **Diagonal LALoRA regularization reduces forgetting while maintaining good learning ability.** The figure shows accuracy over the course of finetuning: (*top*) average source domain accuracy (forgetting) and (*bottom*) target dataset accuracy (learning) and corresponding standard deviation. Each setting corresponds to three random seeds.

onal prior. Figure 2 shows LALoRA minimizes the source domain forgetting by 5.6% meanwhile it keeps finetuning performance at a similar level.

Strength of regularization Next, we inspect the trade-off between forgetting and learning. We sweep a set of regularization strengths, $\lambda \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$, where higher regularization value stronger penalize deviating from the source domain optimal parameters. We plot each run's source- versus target-accuracy pair as a Pareto curve in Figure 3. We observe that the regularization strength enables us to adjust this trade-off, and the hyperparameter λ should be selected based on how much weight we assign to source versus target performance.

Comparison to other methods Next, we compare LALoRA to other methods i.e. MIGU, MiLoRA and PiSSA. We re-implement every method from its open-source code. For MIGU, Table 1 reports final forgetting and learning average accuracies with their standard deviations; the same trade-off is visualised in Figure 3. We can notice that LALoRA compared to MIGU provides better trade-off. Additionally, PiSSA and MiLoRA results are presented in Appendix E, however their suboptimal performance, potentially due to shortcomings in our implementation or configuration, warrants further investigation.

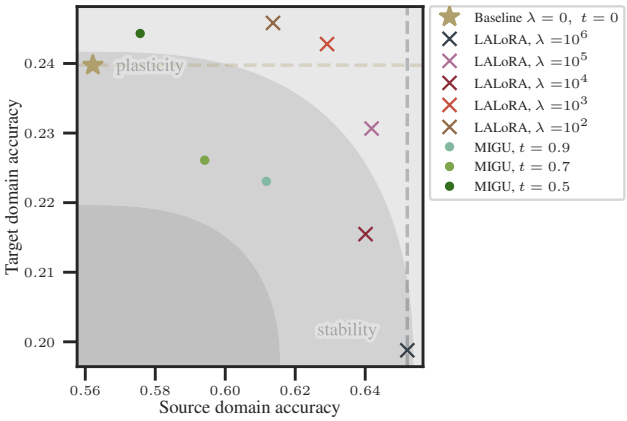


Figure 3: **Laplace regularization leads to improved learning-forgetting trade-off.** The figure shows on x-axis average source domain accuracy (forgetting) and on y-axis target dataset accuracy (learning). The final epoch accuracy for one random seed is plotted for different values of λ and methods.

Table 1: **Comparing our regularization to other methods**. We report the final *average* accuracy ($\pm$ one standard deviation across 3 seeds) on source and target domain for math dataset.

	Source domain	**Target domain**
LA-LoRA DIAG	0.612 ± 0.006	0.252 ± 0.005
MIGU, $t = 0.7$	0.597 ± 0.010	0.215 ± 0.010
Baseline *(no regularizer)*	0.556 ± 0.008	0.251 ± 0.007

We show further results on how the trade-off evolves across epochs and how forgetting varies by dataset in Appendix E.

6. Conclusion

Summary: We propose a method that mitigates forgetting of the source domain, commonsense reasoning, for efficient finetuning on math data using Low-Rank Adaptation. We notice that a simple diagonal regularizer can reliably reduce forgetting and observe improved trade-off compared other baselines. Additionally, we discuss different curvature approximation variants needed to construct the weight-regularizer using Laplace approximation.

Limitations: The method requires at least one batch representing source data for LA computation, as well as, storing the additional $\boldsymbol{\mu}$ and $\bar{\boldsymbol{\Sigma}}$ throughout the whole finetuning. The choice of λ requires tuning.

Future directions: We will empirically inspect the proposed curvature approximation variants, as well as, using more than a single random mini-batch for LA.

References

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P. Cunningham. Lora learns less and forgets less, 2024. URL https://arxiv.org/abs/2405.09673.

Wenyu Du, Shuang Cheng, Tongxu Luo, Zihan Qiu, Zeyu Huang, Ka Chun Cheung, Reynold Cheng, and Jie Fu. Unlocking continual learning abilities in language models, 2024. URL https://arxiv.org/abs/2406.17245.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. Milora: Harnessing minor singular components for parameter-efficient llm finetuning, 2025. URL https://arxiv.org/abs/2406.09044.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models, 2025. URL https://arxiv.org/abs/2404.02948.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13): 3521–3526, March 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting, 2018. URL https://arxiv.org/abs/1805.07810.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023. URL https://arxiv.org/abs/2305.14314.

Sreyan Ghosh, Chandra Kiran Reddy Evuru, Sonal Kumar, Ramaneswaran S, Deepali Aneja, Zeyu Jin, Ramani Duraiswami, and Dinesh Manocha. A closer look at the limitations of instruction tuning, 2024. URL https://arxiv.org/abs/2402.05119.

Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. Lora land: 310 fine-tuned llms that rival gpt-4, a technical report, 2024a. URL https://arxiv.org/abs/2405.00732.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. Camels in a changing climate: Enhancing lm adaptation with tulu 2, 2023. URL https://arxiv.org/abs/2311.10702.

Terry Yue Zhuo, Armel Zebaze, Nitchakarn Suppattarachai, Leandro von Werra, Harm de Vries, Qian Liu, and Niklas Muennighoff. Astraios: Parameter-efficient instruction tuning code large language models, 2024. URL https://arxiv.org/abs/2401.00788.

David John Cameron MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992. URL https://api.semanticscholar.org/CorpusID:16543854.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux – effortless bayesian deep learning, 2022. URL https://arxiv.org/abs/2106.14806.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. Lora vs full fine-tuning: An illusion of equivalence, 2024. URL https://arxiv.org/abs/2410.21228.

Adam X. Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. Bayesian low-rank adaptation for large language models, 2024. URL https://arxiv.org/abs/2308.13111.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert,

Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023. URL https://arxiv.org/abs/2312.12148.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019. URL https://arxiv.org/abs/1902.00751.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature, 2020. URL https://arxiv.org/abs/1503.05671.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. Continual learning for large language models: A survey, 2024. URL https://arxiv.org/abs/2402.01364.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners, 2022. URL https://arxiv.org/abs/2205.12393.

Yifan Wang, Yafei Liu, Chufan Shi, Haoling Li, Chen Chen, Haonan Lu, and Yujiu Yang. Inscl: A data-efficient continual learning paradigm for fine-tuning large language models with instructions, 2024a. URL https://arxiv.org/abs/2403.11435.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. Demix layers: Disentangling domains for modular language modeling, 2021. URL https://arxiv.org/abs/2108.05036.

Yujia Qin, Jiajie Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Elle: Efficient lifelong pre-training for emerging data, 2022. URL https://arxiv.org/abs/2203.06311.

Junhao Zheng, Shengjie Qiu, and Qianli Ma. Learn or recall? revisiting incremental learning with pre-trained language models, 2024. URL https://arxiv.org/abs/2312.07887.

Didi Zhu, Zhongyi Sun, Zexi Li, Tao Shen, Ke Yan, Shouhong Ding, Kun Kuang, and Chao Wu. Model tailor: Mitigating catastrophic forgetting in multi-modal large language models, 2024. URL https://arxiv.org/abs/2402.12048.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning, 2023a. URL https://arxiv.org/abs/2310.14152.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning, 2023b. URL https://arxiv.org/abs/2310.14152.

Tao Li, Zhengbao He, Yujun Li, Yasheng Wang, Lifeng Shang, and Xiaolin Huang. Flat-lora: Low-rank adaption over a flat loss landscape, 2024. URL https://arxiv.org/abs/2409.14396.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization, 2022. URL https://arxiv.org/abs/2207.00112.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models, 2024. URL https://arxiv.org/abs/2312.05821.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression, 2024b. URL https://arxiv.org/abs/2403.07378.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation, 2024. URL https://arxiv.org/abs/2402.09353.

Martin Wistuba, Prabhu Teja Sivaprasad, Lukas Balles, and Giovanni Zappella. Continual learning with low rank adaptation, 2023. URL https://arxiv.org/abs/2311.17601.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. Loramoe: Alleviate world knowledge forgetting in large language models via moe-style plugin, 2024. URL https://arxiv.org/abs/2312.09979.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024b. URL https://arxiv.org/abs/2403.03507.

Appendix

A. Background

Full-finetuning: The problem of finetuning has been thoroughly studied where a model is adapted to a specific task. Full finetuning is inefficient because *every* parameter in the model must be updated. For instance, Xu et al. (2023) estimate that fully finetuning Falcon-180B would require about 5.1 TB of GPU memory. A practical alternative is to train only a subset of layers. Parameter-Efficient FineTuning (PEFT, Houlsby et al., 2019) offers several such techniques—additive, partial, re-parameterized, hybrid, and unified finetuning, that greatly reduce the number of trainable parameters.

Transformers: (Vaswani et al., 2023) proposed a compilation of encoder-decoder modules with attention mechanism. For an input $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ where n is the length of the sequence and d is the hidden dimension. We transform the input via query, key and value vectors given as $\boldsymbol{K}, \boldsymbol{Q}, \boldsymbol{V}$:

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_k + \boldsymbol{b}_k, \boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_q + \boldsymbol{b}_q, \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_v + \boldsymbol{b}_v \tag{6}$$

Kronecker-factored Approximate Curvature (K-FAC): Martens and Grosse (2020) propose an approximation to the loss curvature. They represent the Hessian as a Fisher Information Matrix,

$$F = \mathbb{E}\left[\left(\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W})}{\partial \boldsymbol{W}}\right)\left(\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W})}{\partial \boldsymbol{W}}\right)^\top\right], \tag{7}$$

where they factorize each layer's block into the Kronecker product of two much smaller matrices $\boldsymbol{A}_{l-1,m-1} \otimes \boldsymbol{G}_{l,m}$. We define the input as $\boldsymbol{a}_0 = \boldsymbol{x}$ which is passed through $1, \ldots, L$ layers, this leads to an output $\boldsymbol{h}_L$. The pre-activations are defined as $\boldsymbol{s}_l = \boldsymbol{W}_l \boldsymbol{a}_{l-1} \in \mathbb{R}^{D_l}$ and the activations as $\boldsymbol{a}_l = f_l(\boldsymbol{s}_l) \in \mathbb{R}^{D_{l-1}}$. The Hessian blocks of the layers l, m can be written as

$$\boldsymbol{\Sigma}_{l,m}^{-1} = \mathbb{E}\left[\left(\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W}_l)}{\partial \boldsymbol{W}_l}\right)\left(\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W}_m)}{\partial \boldsymbol{W}_m}\right)^\top\right], \tag{8}$$

with $\boldsymbol{W}_l \in \mathbb{R}^{(D_{\text{in}} \times D_{\text{out}})}$. Utilizing the K-FAC approximation, this simplifies to $\boldsymbol{\Sigma}_{l,m}^{-1} = \boldsymbol{A}_{l-1,m-1} \otimes \boldsymbol{G}_{l,m}$ with $\boldsymbol{A}_{l-1,m-1} = \mathbb{E}[\boldsymbol{a}_{l-1}\boldsymbol{a}_{m-1}^\top] \in \mathbb{R}^{D_{l-1} \times D_{m-1}}$ and

$$\boldsymbol{G}_{l,m} = \mathbb{E}[\boldsymbol{g}_l \boldsymbol{g}_m^\top] = \frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{W}) \partial \log p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{W})}{\partial \boldsymbol{s}_l \partial \boldsymbol{s}_l} \tag{9}$$

$$\in \mathbb{R}^{D_l \times D_m}. \tag{10}$$

B. Related Work

Continual finetuning: The desirable feature of LLMs is to equip the models with new knowledge or skills i.e. continual learning. The review (Wu et al., 2024) introduces categorization: continual pre-training, instruction tuning and alignment. The first expands the models understanding of language, the second improves responses to the specific commands, and last, ensures the model is abiding by ethical norms. There has been a lot of work done on continual learning via rehearsal, architecture based methods (Scialom et al., 2022; Wang et al., 2024a; Gururangan et al., 2021; Qin et al., 2022) and more importantly for this work, parameter-based (Zheng et al., 2024; Zhu et al., 2024) or gradient-based (Wang et al., 2023a). Wang et al. (2023b) propose O-LoRA for mitigating catastrophic forgetting via learning tasks in orthogonal vector subspaces. They use instruction tuning. Their main reasoning is that gradient subspaces of previous tasks are represented by LoRA parameters. Li et al. (2024) proposes a low-rank adaptation in flat regions of the full paramater space, for which they use random weight perturbation.

Model low-rank decomposition: Biderman et al. (2024) show via an SVD that full finetuning doesn't change the spectrum significantly. The following papers deal with the full model low-rank decomposition to capture as much information and the least performance degradation. Fisher-Weighted SVD (FWSVD) (Hsu et al., 2022) assigns importance scores via Fischer Information Matrix, they observe that singular values' magnitude doesn't directly correspond to performance drop therefore the smallest values may still be needed. (Yuan et al., 2024) propose Activation-aware Singular Value Decomposition (ASVD) that is a training-free method which manages activation outliers via scaling the weights accordingly. Wang et al. (2024b) propose direct mapping between singular values and model compression loss by ensuring that each channel is independent of each other.

Other approaches: DoRA (Liu et al., 2024) decomposes W_0 into magnitude and direction components. The method uses only the directional updates during finetuning and scales the magnitude.

Forgetting the source domain in LoRA: Wistuba et al. (2023) use LoRA to train a dedicated expert model for each new incoming dataset. They use a k cluster based method to infer which LoRA module to use for which task. By contrast, LoRAMoE Dou et al. (2024) introduces a mixture of experts architecture in which multiple low-rank adapters work together, dynamically weighted by a router network.

In contrast to LoRA, GaLore (Zhao et al., 2024b) leverages low-rank structure of *gradients*. The authors project the gradient matrix into a low rank updates, this results in

substantial memory cost reduction with regard to full fine-tuning, mainly of optimizer states. They notice a slight improvement on GLUE tasks compared to LoRA.

C. Extended methodology

Curvature approximations. Since, DIAG discards some information—specifically, the interactions among weights within a layer and across layers, we propose B-K-FAC which looks at intra-layer interactions and B-TRI-K-FAC at inter-layer between $\boldsymbol{A}$ and $\boldsymbol{B}$ for a given $\Delta \boldsymbol{W}_l$. B-K-FAC uses a block-diagonal K-FAC approximation for individual weight matrices i.e. $\boldsymbol{\Sigma}_{l,l}^{-1}$ given by:

$$\boldsymbol{\Sigma}_{\Delta \boldsymbol{W}_l}^{-1} = \begin{bmatrix} \boldsymbol{A}_{0,0} \otimes \boldsymbol{G}_{1,1} & 0 \\ 0 & \boldsymbol{A}_{1,1} \otimes \boldsymbol{G}_{2,2} \end{bmatrix}. \tag{11}$$

The regularizer resulting from such an approximation is defined as:

$$\begin{aligned} \boldsymbol{r} &= \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top \text{vec}(\boldsymbol{G}_{1,1}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})\boldsymbol{A}_{0,0}^\top) + \\ &+ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \text{vec}(\boldsymbol{G}_{2,2}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})\boldsymbol{A}_{1,1}^\top). \end{aligned}$$

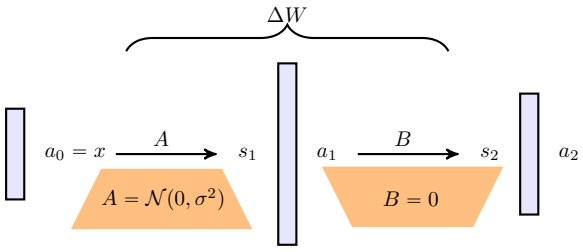
Additionally, unlike the block-diagonal K-FAC used previously (Ritter et al., 2018; Yang et al., 2024), we treat the $\boldsymbol{A}$ and $\boldsymbol{B}$ layers jointly as one block, capturing inter-layer as well as intra-layer interactions i.e. $\boldsymbol{\Sigma}_{\boldsymbol{A},\boldsymbol{A}}^{-1}, \boldsymbol{\Sigma}_{\boldsymbol{A},\boldsymbol{B}}^{-1}, \boldsymbol{\Sigma}_{\boldsymbol{B},\boldsymbol{A}}^{-1}, \boldsymbol{\Sigma}_{\boldsymbol{B},\boldsymbol{B}}^{-1}$. This leads to a block tri-diagonal K-FAC approximation B-TRI-K-FAC for consecutive layers:

$$\boldsymbol{\Sigma}_{\Delta \boldsymbol{W}_l}^{-1} = \begin{bmatrix} \boldsymbol{A}_{0,0} \otimes \boldsymbol{G}_{1,1} & \boldsymbol{A}_{0,1} \otimes \boldsymbol{G}_{1,2} \\ \boldsymbol{A}_{1,0} \otimes \boldsymbol{G}_{2,1} & \boldsymbol{A}_{1,1} \otimes \boldsymbol{G}_{2,2} \end{bmatrix}. \tag{12}$$

The regularizer for this approximation is efficiently computed with matrix vector products as follows:

$$\begin{aligned} \boldsymbol{r} &= \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top \text{vec}(\boldsymbol{G}_{1,1}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})\boldsymbol{A}_{0,0}^\top) \\ &+ 2\text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top \text{vec}(\boldsymbol{G}_{1,2}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})\boldsymbol{A}_{0,1}^\top) \\ &+ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \text{vec}(\boldsymbol{G}_{2,2}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})\boldsymbol{A}_{1,1}^\top). \end{aligned}$$

Below, we delve into more details. We present a LoRA module for one layer.



The precision for one layer can be approximated as:

$$\boldsymbol{\Sigma}_{\Delta W_l}^{-1} = \begin{bmatrix} \mathbf{A}_{0,0} \otimes \mathbf{G}_{1,1} & \mathbf{A}_{0,1} \otimes \mathbf{G}_{1,2} \\ \mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1} & \mathbf{A}_{1,1} \otimes \mathbf{G}_{2,2} \end{bmatrix}. \tag{13}$$

with shapes

$$\mathbf{A}_{0,0} \otimes \mathbf{G}_{1,1} \in \mathbb{R}^{(D_{\text{in}} \times D_{\text{in}}) \times (r \times r) = (D_{\text{in}} r \times D_{\text{in}} r)} \tag{14}$$

$$\mathbf{A}_{0,1} \otimes \mathbf{G}_{1,2} \in \mathbb{R}^{(D_{\text{in}} \times r) \times (r \times D_{\text{out}}) = (D_{\text{in}} r \times r D_{\text{out}})} \tag{15}$$

$$\mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1} \in \mathbb{R}^{(r \times D_{\text{in}}) \times (D_{\text{out}} \times r) = (D_{\text{out}} r \times r D_{\text{in}})} \tag{16}$$

$$\mathbf{A}_{1,1} \otimes \mathbf{G}_{2,2} \in \mathbb{R}^{(r \times r) \times (D_{\text{out}} \times D_{\text{out}}) = (r D_{\text{out}} \times r D_{\text{out}})} \tag{17}$$

The Lalplace approximation for these low-rank weights can be represented as:

$$p(\Delta \boldsymbol{W}_l | \mathbb{D}_{\text{S}}) \sim \mathcal{N}(\boldsymbol{A}, \boldsymbol{B}; \boldsymbol{\mu}, \boldsymbol{\Sigma}_{\Delta \boldsymbol{W}_l}) \tag{18}$$

$$p(\Delta \boldsymbol{W}_l | \mathbb{D}_{\text{S}}) \sim \exp\Bigg(-\frac{1}{2} \begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top & \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \end{bmatrix} \tag{19}$$

$$\boldsymbol{\Sigma}_{\Delta W_l}^{-1} \begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}}) \\ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}}) \end{bmatrix} \Bigg)$$

$$p(\Delta \boldsymbol{W}_l | \mathbb{D}_{\text{S}}) \sim \exp\Bigg(-\frac{1}{2} \begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top & \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \end{bmatrix} \tag{20}$$

$$\begin{bmatrix} \mathbf{A}_{0,0} \otimes \mathbf{G}_{1,1} & \mathbf{A}_{0,1} \otimes \mathbf{G}_{1,2} \\ \mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1} & \mathbf{A}_{1,1} \otimes \mathbf{G}_{2,2} \end{bmatrix} \begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}}) \\ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}}) \end{bmatrix} \Bigg)$$

We take into account $(\boldsymbol{Q} \otimes \boldsymbol{U})\text{vec}\boldsymbol{V} = \text{vec}(\boldsymbol{U}\boldsymbol{V}\boldsymbol{Q}^\top)$, which leads to $(\mathbf{A}_{l-1,m-1} \otimes \mathbf{G}_{l,m})\text{vec}(W_{l \to m} - W^*) = \text{vec}(\mathbf{G}_{l,m}(W_{l \to m} - W^*)\mathbf{A}_{l-1,m-1})^\top$. We utilize it for the regularizer below

$$\begin{aligned} &\begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \end{bmatrix} \\ &\begin{bmatrix} \mathbf{A}_{0,0} \otimes \mathbf{G}_{1,1} & \mathbf{A}_{0,1} \otimes \mathbf{G}_{1,2} \\ \mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1} & \mathbf{A}_{1,1} \otimes \mathbf{G}_{2,2} \end{bmatrix} \begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}}) \\ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}}) \end{bmatrix}, \end{aligned} \tag{21}$$

where $\text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}}) \in \mathbb{R}^{(D_{\text{in}} r \times 1)}$ and $\text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}}) \in \mathbb{R}^{(r D_{\text{out}} \times 1}$. This leads to the following result:

$$\begin{aligned} &\begin{bmatrix} \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top & \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \end{bmatrix} \\ &\begin{bmatrix} (\mathbf{A}_{0,0} \otimes \mathbf{G}_{1,1})\text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}}) + \\ + (\mathbf{A}_{0,1} \otimes \mathbf{G}_{1,2})\text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}}) \\ \\ (\mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1})\text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}}) + \\ + (\mathbf{A}_{1,1} \otimes \mathbf{G}_{2,2})\text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}}) \end{bmatrix} \end{aligned} \tag{22}$$

$$\begin{aligned} &= \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top \text{vec}\big(\mathbf{G}_{1,1}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})\mathbf{A}_{0,0}^\top\big) \\ &(1 \times D_{\text{in}} r) \cdot ((r \times r)\,(r \times D_{\text{in}})\,(D_{\text{in}} \times D_{\text{in}})) \\ &+ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \text{vec}\big(\mathbf{G}_{2,1}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})\mathbf{A}_{1,0}^\top\big) \\ &(1 \times D_{\text{out}} r) \cdot ((D_{\text{out}} \times r)\,(r \times D_{\text{in}})\,(D_{\text{in}} \times r)) \\ &+ \text{vec}(\boldsymbol{A} - \boldsymbol{\mu}_{\boldsymbol{A}})^\top \text{vec}\big(\mathbf{G}_{1,2}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})\mathbf{A}_{0,1}^\top\big) \\ &(1 \times D_{\text{in}} r) \cdot ((r \times D_{\text{out}})\,(D_{\text{out}} \times r)\,(r \times D_{\text{in}})) \\ &+ \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})^\top \text{vec}\big(\mathbf{G}_{2,2}(\boldsymbol{B} - \boldsymbol{\mu}_{\boldsymbol{B}})\mathbf{A}_{1,1}^\top\big) \\ &(1 \times D_{\text{out}} r) \cdot ((D_{\text{out}} \times D_{\text{out}})\,(D_{\text{out}} \times r)\,(r \times r)) \end{aligned}$$

The cost of this operation is:

- compute: $\mathcal{O}(D_{\text{in}}^2 r + D_{\text{out}}^2 r + r^2(D_{\text{in}} + D_{\text{out}}))$
- memory: $\mathcal{O}(D_{\text{in}}^2 + D_{\text{out}}^2 + r^2)$

We can simplify the cross layer terms since $A_{ji} = (\boldsymbol{A}_{ij})^\top$.

$$\begin{aligned}
&\text{vec}(\boldsymbol{A} - \boldsymbol{\mu_A})^\top (\mathbf{A}_{0,1} \otimes \mathbf{G}_{1,2}) \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_B) = \qquad (23)\\
&= \text{vec}(\boldsymbol{A} - \boldsymbol{\mu_A})^\top (\mathbf{A}_{1,0}^\top \otimes \mathbf{G}_{2,1}^\top) \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_B) =\\
&= \text{vec}(\boldsymbol{A} - \boldsymbol{\mu_A})^\top (\mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1})^\top \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_B) =\\
&= \left(\text{vec}(\boldsymbol{A} - \boldsymbol{\mu_A})(\mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1}) \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_B)^\top\right)^\top =\\
&= \left(\text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_B)^\top (\mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1}) \text{vec}(\boldsymbol{A} - \boldsymbol{\mu_A})\right)^\top =\\
&= \text{vec}(\boldsymbol{B} - \boldsymbol{\mu}_B)^\top (\mathbf{A}_{1,0} \otimes \mathbf{G}_{2,1}) \text{vec}(\boldsymbol{A} - \boldsymbol{\mu_A})
\end{aligned}$$

Importantly, since $\boldsymbol{W}$ is a linear layer and there is no activation between layers $\boldsymbol{A}$ and $\boldsymbol{B}$, the pre-activationas and activations are the same, i.e., $\boldsymbol{s}_1 = \boldsymbol{a}_1$ and $\boldsymbol{s}_2 = \boldsymbol{a}_2$.

This results in $\mathbf{A}_{l,m} = \mathbb{E}[s_l s_m^\top] \in \mathbb{R}^{d_l \times d_m}$, with

$$\mathbf{G}_{l,m} = \mathbb{E}[g_l g_m^\top] = \frac{\delta^2 \log p(\boldsymbol{y}|\boldsymbol{x}, \theta)}{\delta s_l \delta s_l} \in \mathbb{R}^{d_l \times d_m}$$

The number of stored blocks is $3L$.

D. Experimental setup

The strength of regularization λ was chosen via runs with validation dataset split. The probed values of λ were $\{10^2, 10^3, 10^4, 10^5, 10^6\}$. The results for each value were as follows $\{\lambda = 10^6 : 0.4142, \lambda = 10^5 : 0.4140, \lambda = 10^4 : 0.4036, \lambda = 10^3 : 0.3964, \lambda = 10^2 : 0.4182\}$. We chose the best average accuracy for forgetting and learning trade-off which resulted to $\lambda = 10^2$ for diagonal approximation. We then, ran three seeds for optimal λ and baseline. For MIGU, we chose $t = 0.7$ as the value for Table 1, since that value was recommended by the authors of the method. Other arguments are shown in the Table 2.

E. More experimental results

We report the results for MiLoRA and PiSSA in Table 3. We note though that these results require further inspection.

Table 2: **Experimental setup**. The table presents the values for the arguments used in the training across experiments.

Argument	
dataset name	gsm8k
per device train batch size	1
per device evaluate batch size	4
learning rate	5e-4
number of epochs	30
sequence length	512
causal generation length	100
seeds	42
evaluation frequency	5000
LoRA rank	32
LoRA α	32
LoRA dropout	0.1

Table 3: **Other methods performance**. We report the final *average* accuracy ($\pm$ one standard deviation across 3 seeds) on source and target domain for math dataset.

	Source domain	**Target domain**
PiSSA	0.513 ± 0.002	0.118 ± 0.014
MiLoRA	0.536 ± 0.007	0.246 ± 0.006

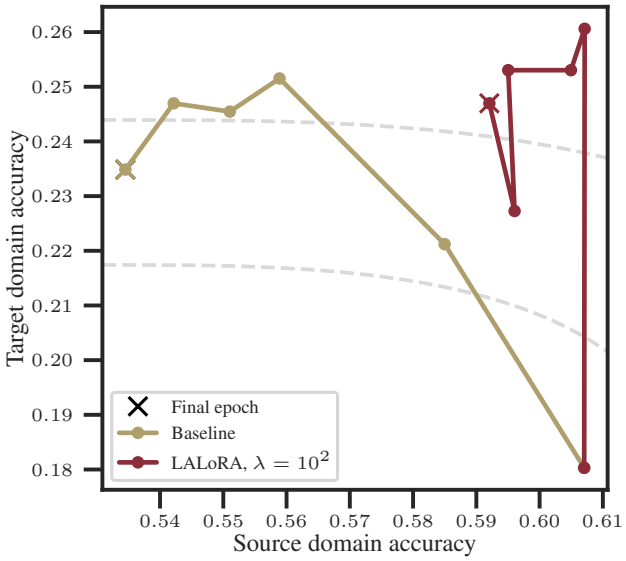


Figure 4: **Laplace regularization** DIAG **leads to improved learning-forgetting trade-off across finetuning.** The figure shows accuracy over the course of finetuning: on x-axis average source domain accuracy (forgetting) and on y-axis target dataset accuracy (learning). The final accuracy for one random seed is plotted with a $\times$. Each dot corresponds to a result 5 epochs later. We notice that the regularized approach leads to better learning-forgetting trade-off across the training.

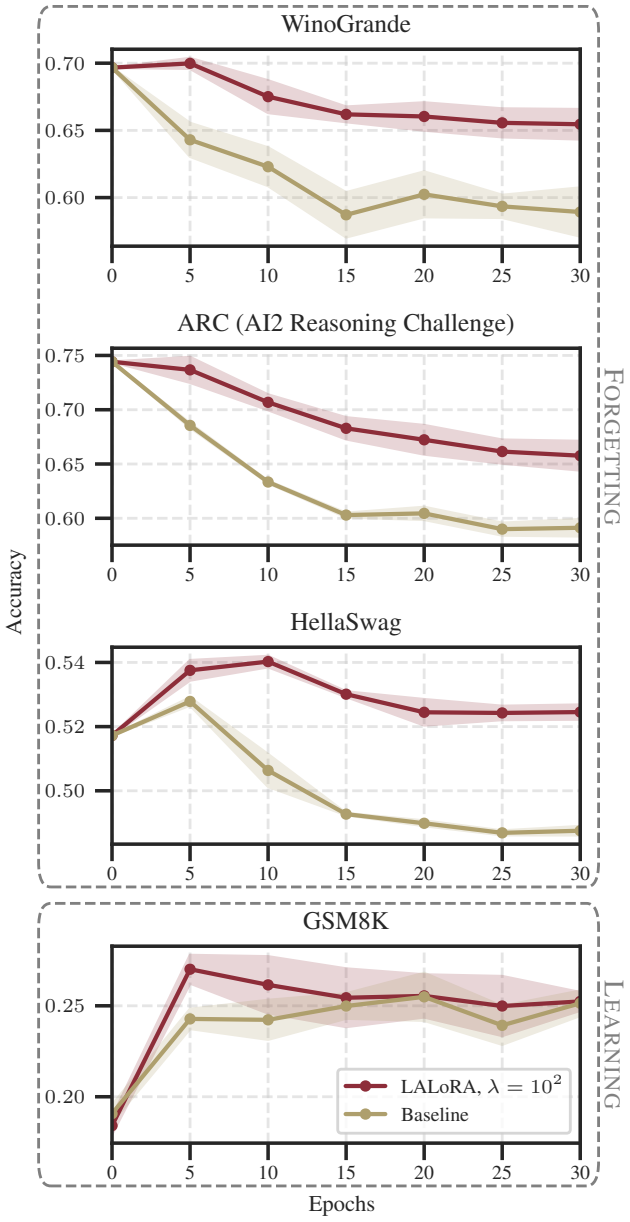


Figure 5: **Diagonal regularization reduces forgetting across all datasets.** The figure shows accuracy over the course of finetuning: average source domain accuracy (forgetting) and target dataset accuracy and corresponding standard deviation. Each setting corresponds to three random seeds. Results are displayed top-to-bottom for the three source domain datasets: WinoGrande, ARC, and HellaSwag, and, on the far bottom, for the target math dataset GSM8K. We notice less forgetting for the regularized approach compared to the baseline, and similar learning capabilities.

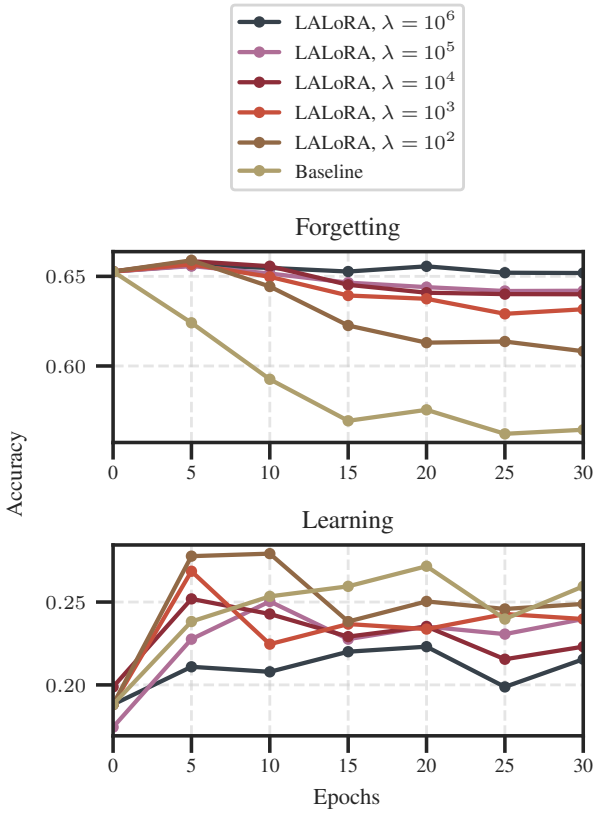


Figure 6: **A stronger diagonal regularizer mitigates source domain forgetting more.** The figure shows accuracy over the course of finetuning: (*top*) average source domain accuracy (forgetting), (*bottom*) target dataset accuracy. Each setting corresponds to a single random seed. As the regularization strength λ increases, forgetting declines, but the model's ability to learn the new task is reduced.