Dynamic Low-Rank Module Allocation

Jesse Brouwers [1] Zsombor Fülöp [1] Miklós Hamar [1] Matey Krastev [1]

Abstract

Efficient adaptation of large-scale Transformer models has become a pressing issue in recent years and gave rise to the field of Parameter-Efficient Fine-Tuning (PEFT) methods. In this work, we investigate the training dynamics of Low Rank Adaptation (LoRA)-based approaches. We propose an extension, DynaLoRA, which aims to further reduce the memory demand of model adaptation, by only optimizing a subset of the trainable parameters (selected based on their importance) at any point in time during training. We explore different resource allocation methods by proposing different importance heuristics and allocation schedules. Our results show, that we can further reduce the effective number of trainable parameters by 33%, without any significant performance drop compared to LoRA.

1. Introduction

Foundation models have shown impressive abilities in a wide range of tasks even within completely zero-shot settings. Nevertheless, adapting them to downstream tasks remains crucial for optimizing performance. However, performing traditional full or partial fine-tuning has become challenging as the models have substantially increased in size, thereby imposing significant memory requirements. For example, the LLaMA 65 billion parameter model (Touvron et al., 2023) requires more than 780 GB of GPU memory when adapting using traditional full fine-tuning (Dettmers et al., 2024). As the current largest models are on the scale of hundreds of billions or even trillions of parameters, full-finetuning has become infeasible for most researchers and practitioners.

This challenge gave rise to parameter-efficient fine-tuning (PEFT) methods, which attempt to adapt large-scale models by optimizing over a small fraction of their parameters. Most remarkably, LoRA (Hu et al., 2021) optimizes over the low-rank decomposition of the learned weight matrices, while keeping the pre-trained model weights unchanged. Through this method, LoRA achieves comparable fine-tuning performance to full fine-tuning, while maintaining the number of learnable parameters orders of magnitude smaller than the total parameter count of the pre-trained model.

Inspired by the success of LoRA, parameter-efficient fine-tuning has become a rapidly evolving field, with numerous extensions and improvements being proposed. These methods all attempt to further optimize the fine-tuning process, by proposing novel ways of reducing the number of learnable parameters (Kopiczko et al., 2023), applying quantization to the learned weights (Dettmers et al., 2024), proposing different initialization methods for the learnable low-rank decomposition matrices (Zhang et al., 2023), dynamically adjusting the rank of different modules during adaptation (Liu et al., 2024), or optimizing the memory demand of the gradient updates (Zhao et al., 2024). Even though all these methods show remarkable performance, they mostly address the question, *how to reduce* the number of trainable parameters and disregard the question *where should* the reduction happen.

In our work, we aim to address the latter question by proposing different methods to *dynamically* allocate resources to certain adapter modules, while saving computation on other parameters, deemed *less important* for achieving some given down-stream task. To this end, we propose DynaLoRA, a novel adaptation strategy that dynamically allocates a fixed computational resource budget at the module level. By evaluating module importance at various points during training, we aim to approximate the loss-landscape and dynamically allocate resources so that at any point in time, only the most important modules are updated. We perform a thorough evaluation of DynaLoRA's capabilities and limitations, hopefully providing valuable insights for future research in the field.

Our main contributions can be summarised as follows:

1. We propose DynaLoRA, a novel parameter-efficient fine-tuning extension to LoRA[1] that dynamically allocates resources to the most important modules at a given stage in training, further reducing the training requirements for fine-tuning large models while maintaining comparable performance. DynaLoRA is orthogonal to existing approaches for PEFT and pro-

[1] here we use LoRA to abbreviate LoRA and other LoRA-like PEFT methods.

vides an effective way to reduce training requirements.

2. We evaluate the capabilities of DynaLoRA against state-of-the-art PEFT methods by fine-tuning RoBERTa-base on several GLUE benchmarks.

3. We provide extensive qualitative analysis of the module selection procedure through ablation studies and visualizations, aiming to enhance the understanding of the inner workings of DynaLoRA.

2. Related Works

The Low-Rank Adaptation (LoRA) method (Hu et al., 2021) represents a pivotal advancement in the fine-tuning process of large pre-trained language models. By employing a low-rank decomposition to learn updates for weight matrices within dense layers, LoRA significantly reduces the computational overhead associated with fine-tuning. This approach not only achieves performance comparable to full fine-tuning but also drastically lowers the hardware requirements by minimizing the number of parameters involved in gradient calculations. Since its release, LoRA has become the de-facto baseline for comparing different PEFT methods. Our work also largely relies on its implementation in the Huggingface PEFT library.

Building on this foundational work, Kopiczko et al. (2023) developed VeRA, a modification to LoRA that further minimizes the number of trainable parameters. VeRA introduces trainable one-dimensional scaling vectors that adjust frozen low-rank random projections. These projections are shared across all modules and can be deterministically initialised at any time. They demonstrate that effective training can be achieved with even fewer parameters, maintaining performance levels similar to LoRA. Simultaneously, Dettmers et al. (2024) showed the LoRA methodology to be compatible with quantized weights by introducing QLoRA, further reducing memory requirements and lowering the hardware barriers even more.

Another line of research has focused on enhancing efficiency through smarter allocation of resources. AdaLoRA, proposed by Zhang et al. (2023), uses a dynamic rank adjustment strategy based on singular value decomposition (SVD). This method incrementally prunes less critical components of the learned SVD matrices until reaching a pre-set parameter budget, effectively reallocating resources to enhance performance in critical modules. Conversely, SoRA (Ding et al., 2023) enhances the dynamic capability of LoRA by introducing a gating unit that controls rank sparsity, allowing the removal of zero-valued ranks during inference to optimize resource usage. ALoRA (Liu et al., 2024) proposes dynamically adjusting the intrinsic rank of different modules during adaptation, reallocating ranks from less necessary modules and, in contrast with other methods, reallocates this saved budget to modules requiring more representational power. Collectively, these methods enhance representational power by allowing for dynamic determination of representational needs per module, rather than fixing the rank beforehand as in the original LoRA method. By flexibly allocating higher ranks to more important modules and lower ranks to less critical ones, they underscore the varying levels of importance across different modules.

As opposed to methods which primarily focus on feature-based rank pruning, Zhou et al. (2024) introduce LoRA-Drop, which determines module importance based on the norm of the forward activations. After initial training on a subset of data, the method freezes the learned LoRA parameters to assess layer importance, enabling the pruning of less critical layers. This strategy enhances parameter efficiency and achieves performance that closely matches the original LoRA method, illustrating an alternative approach to rank-based pruning.

Previous studies highlight the adaptability of the LoRA method and the potential for significant efficiency gains through the effective allocation of computational resources. However, no prior research has explored improving efficiency by dynamically activating and deactivating LoRA adapters in its entirety during training. Therefore we propose DynaLoRA, a novel adaptation strategy aimed to address this gap.

3. Methodology

In the following section, we introduce some formal notation for our method, as well as the key building blocks of DynaLoRA. First, we will outline how module importance is determined, after which we will discuss the different module selection strategies. Subsequently, we will discuss the scheduling strategies, and finally, we will define the two different forms of optimization considered in this work.

Based on the notation of Hu et al. (2021), given some pre-trained model $\mathcal{M}$, composed of M modules, for each module $m \in \mathcal{M}$ we characterize the inference on the fine-tuned model simply as

$$h = (W_{\mathcal{M}}^{(m)} + \Delta W^{(m)})x,$$

where $W_{\mathcal{M}}^{(m)} \in \mathbb{R}^{(d\times d)}$ is the pre-trained weight matrix, $\Delta W^{(m)} \in \mathbb{R}^{(d\times d)}$ is what is learned during fine-tuning, and $x, h \in \mathbb{R}^d$ are the d dimensional input and output representations of the data we are performing inference over.

Depending on the PEFT method, $\Delta W^{(m)}$ may take different forms. In case of LoRA, $\Delta W^{(m)}$ is a low-rank decomposition $\Delta W^{(m)} = BA$, with $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times d}$ for some $r \ll d$ denoting the rank of the decomposition. In the case of VeRA, $\Delta W^{(m)} = \Lambda_b B \Lambda_d A$, with $\Lambda_b, \Lambda_d \in \mathbb{R}^d$ being scaling vectors. As we will show in the rest of this section, DynaLoRA is agnostic to how the $\Delta W^{(m)}$ matrices are expressed and instead focuses on their relative *importance* during training.

3.1. Importance of Delta Layers

The main research question we aim to investigate is whether it is possible to dynamically allocate resources to the most important modules during training by refraining from gradient computation for less important ones. Throughout this report, we refer to certain modules as 'active' or 'inactive', i.e. 'receives gradient updates' or not, respectively.

To that end, we investigate several potential indicators for module importance during training, specifically the output of linear layer application, and the backward gradients of the given linear layer's adapter weight matrices. These are alternatively referred to in the text as forward activations and backwards activations, respectively.

Formally, we define the activation score as a scalar value computed from the output representation h, denoted as $||h|| = ||(W_{\mathcal{M}}^{(m)} + \Delta W^{(m)})x|| \in \mathbb{R}$. To build comparable aggregate importance scores, we use an aggregation function ν that takes in the output representation and computes a scalar score. For example, we can use the L1 or L2 norms of the output vectors. During training, at each step t, we compute the importance of each module m as

$$y_t^{(m)} = \sum_{x \in \mathcal{D}_t} ||(W_{\mathcal{M}}^{(m)} + \Delta W^{(m)})x||_p \qquad (1)$$

where $\mathcal{D}_t$ denotes the set of training samples encountered up to time t and $p \in \{1, 2\}$. We opt to use the L2 norm throughout the rest of our study. We then normalize these activations to obtain the importance of each module m as

$$\hat{y}_t^{(m)} = \frac{y_t^{(m)}}{\sum_{k \in M} y_t^{(k)}} \in [0, 1]. \qquad (2)$$

In practice, to save computational resources, we can omit this calculation and take $y^{(m)}$ as the importance of module m. It is important to note that the above characterization of importance is in line with Zhou et al. (2024), who use the Frobenius norm of the forward activations up to some $t > 0$ steps and then select the most important transformer-blocks for further training and share a single adapter weight matrix across the modules in all other transformer-blocks. We extend upon this idea by allowing certain modules to be activated and deactivated dynamically during training and performing selection over individual modules instead of transformer blocks. We hypothesize that, in the process of training, certain modules may (1) converge and no longer learn as effectively, making less frequently activated adapters more beneficial targets for training; or (2) become more important for achieving downstream tasks.

In the rest of this section, we describe *how* modules are selected in this dynamic setting, and *at which points* during training. Finally, we discuss possible optimization strategies which ensure stable training and convergence.

3.2. Module Selection

To ensure that the most important modules are updated, we select which modules to update during fine-tuning based on the importance of each module $\hat{y}_t^{(m)}$. We explore four different selection methods.

Top-K: Simply select the top-k modules with the highest importance scores $y_t^{(m)}$ at certain points during training.

Threshold: Analogously to the work of Zhou et al. (2024), in this method we define some threshold $p \in [0, 1]$ and select the first $n \geq 1$ most important modules, such that $\sum_n y_t^{(n)} \geq p$.

Stochastic: As shown in Figure **??**, there may be some modules with smaller activations in the early steps of training that become more relevant as training progresses. Consequently, the two methods described above do not guarantee stable learning. To incentivise exploration, we select each module m with probability $\hat{y}_m$.

Discounted Stochastic: Finally, during our experiments we observed that, based on their role in the architecture of the model, some modules tend to have much higher forward activations than others. In the case of the RoBERTa model (Liu et al., 2019), *dense* layers tend to have much higher activations than the linear modules in the attention blocks. As a result, these modules are more likely to be selected in all of the above methods. To address this, we introduce a counter for each module, $\mathcal{C}_t^{(m)}$, which is incremented each time a module receives a gradient update. Then, we discount the importance of each module by the number of times it was selected for training by computing

$$\tilde{y}_t^{(m)} = \frac{\hat{y}_t^{(m)}}{\mathcal{C}^{(m)}} \qquad (3)$$

and perform categorical sampling as before. This method further incentivises the exploration of modules with smaller activations and we expect it to yield more balanced training dynamics.

3.3. Module Allocation Schedule

Finally, we discuss how often we should update our selection of modules. Intuitively, if we reallocate resources too often, we may not be able to train the selected modules effectively, as the model will not be able to learn new information consistently. On the other hand, if we reallocate resources too infrequently, we may miss out on the opportunity to adapt to the changing importance of different modules. In line with that, we explore the following scheduling strategies:

1. **Reallocate once schedule**: We select the modules at the beginning of training and once more after some specified number of steps α. This method is most similar to Zhou et al. (2024), who select the most important transformer-blocks after some specified number of training epochs.

2. **Periodic schedule**: We select the modules after every n steps during training. This method allows for more frequent reallocation of resources and may lead to better performance in the long run.

3.4. Gradient optimization

The above-described methodology gives us a way to dynamically select which modules to update during training. To save computational resources, during stochastic optimization, we disable the gradient updates for the modules that are not selected. Here we note that this approach may lead to instability in the gradient updates, especially in the case of more advanced optimization algorithms such as Adam (Kingma & Ba, 2014) or AdamW (Loshchilov & Hutter, 2018), which rely on the moving averages of the gradients. Moreover, the learning rate schedules may also need to be adjusted to account for the fact that some modules are not updated during training. To address this, we propose the following strategies:

1. **Global optimization**: We initialize a global optimizer and learning rate scheduler and update only the current selection of modules at each step. This method disregards the fact that some modules are not updated during training and may lead to instability in the gradient updates.

2. **Local optimization**: We initialize a separate optimizer and learning rate scheduler for each module and update only the selected modules at each step. We expect this method to lead to a more stable training process, at a slight additional memory overhead, due to having to keep $|M|$ separate optimizer states in memory.

In our experiments, we performed extensive hyperparameter search to find the most optimal settings for the above methods and we provide the results in Section 5.

4. Experiments and Results

To evaluate the performance of our method, we conduct a series of experiments on a selected subset of the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), in line with previous work by Hu et al. (2021), Kopiczko et al. (2023), and many others. Due to the need for extensive hyperparameter search coupled with time and computational constraints, we only experiment with RoBERTa-base (Liu et al., 2019). However, to underline the effectiveness of our approach, it would be necessary to fine-tune larger models such as DeBERTa XXL (He et al., 2021) and GPT-2 (Radford et al., 2019), since we hypothesise to see more significant memory gains when DynaLoRA is used on larger models.

4.1. Experimental Setup

As outlined in Section 3, we perform a variety of ablation studies aimed at finding the most optimal settings for DynaLoRA. Those vary across the following categories:

- **PEFT method**: A Parameter Efficient Fine-Tuning method that serves as a backbone for the adaptation process. In our experiments, we only consider LoRA (Hu et al., 2021) and VeRA (Kopiczko et al., 2023), but the method is agnostic to most PEFT methods.

- **Importance Measure**: An aggregate score for importance. In this paper, we only consider the L2 norms of the forward activations and layer gradients, as introduced in Section 3.1.

- **Allocator**: A function determining which modules should be trained based on the selected measure of importance, as introduced in Section 3.2.

- **Scheduler**: At which points during training we should update the selection of modules, as described in Section 3.3.

- **Optimizer**: Global or local, as described in Section 3.4.

4.2. Hardware Requirements

In our experiments, we train and test all models on a single HPC node with an NVIDIA A100 (40 GB) GPU. Additionally, we provide checkpoints for our fine-tuned models and experimental setup to ensure reproducibility.

4.3. Baseline Comparison

Comparison to baseline LoRA and VeRA as well as full-fine tuning and BitFit (Zaken et al., 2021) is reported in Table 1. We also implemented dynamic reallocation on VeRA which

Method	# Active Trainable	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Average
FT*	125M	94.8	90.2	63.6	92.8	78.7	91.2	85.2
BitFit*	0.1M	93.7	**92.7**	62.0	91.8	81.5	90.8	85.4
VeRA*	0.04M	$94.6_{\pm.1}$	$89.5_{\pm.5}$	$\mathbf{65.6_{\pm.8}}$	$91.8_{\pm.2}$	$78.7_{\pm.7}$	$90.7_{\pm.2}$	85.1
LoRA*	0.3M	$\mathbf{95.1_{\pm.2}}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3_{\pm.3}}$	$\mathbf{86.6_{\pm.7}}$	$\mathbf{91.5_{\pm.2}}$	**86.6**
DynaLoRA /QV/16/F (R)	**0.2M**	93.6	88.0	62.8	92.0	72.9	90.4	84.6
DynaLoRA /QV/16/F	**0.2M**	93.6	88.0	61.3	92.5	74.4	90.2	83.4
DynaLoRA /QV/16/B	**0.2M**	94.1	86.8	0.0	92.0	79.1	90.5	73.8
DynaVeRA /QV/16/F	**19.0K**	94.8	87.3	65.1	92.4	76.5	90.6	84.5

Table 1: Results on the GLUE benchmark for the RoBERTa-base model. Results derived from the original papers of the specific methods are indicated with asterisk. Each DynaLoRA and DynaVeRA variant were fine-tuned with periodic scheduler and periodicity 50, discounted stochastic allocator targeting only query and key modules (based on the observations of the original LoRA paper by Hu et al.) from the transformer blocks, precisely 16 of the total 24. "F" means using forward activations as importance measure, "B" backward activation. We distinguish "R" to mean random selection of modules without importance measure consideration. All other hyperparameters used for DynaLoRA and DynaVeRA are the same as the ones reported in the original papers (Hu et al., 2021; Kopiczko et al., 2023).

we named DynaVeRA. DynaLoRA and DynaVeRA are almost as performant as their baseline counterparts despite actively training 67% of the parameters at any time step. Initially, DynaLoRA and DynaVeRA are initialized with the same number of trainable parameters as their underlying adapter method (LoRA or VeRA). However, they do not train all of them during fine-tuning at each given step. A fixed number of modules (here 16) are activated, the rest are frozen, dynamically changing which are activated at certain steps. This way, not all gradient are calculated which ultimately leads to less memory requirement.

Despite of our initial theoretical discussion on importance scores calculated from forward and backward activations, we observed that random selection of modules is equally performant. In Section 5.1 we provide a more thorough analysis of the different allocation strategies. Interestingly, when fine-tuning on CoLA with DynaLoRA with backward importance scores, the model gradually becomes worse and worse, eventually reaching Matthews correlation score of 0. We rerun the experiment several times and consistently observed the same behaviour. Understanding why this happens requires more in-depth analysis of the dataset which we leave for future work.

4.4. Training Dynamics

We study how computational aspects of training are affected by DynaLoRA's inclusion. More specifically, we look at validation set loss (Figure 1) and peak memory consumption (Figure 2).

We note that the evaluation loss grows with each LoRA variant while the evaluation metrics also grow. We are unsure about the underlying reasons but note that with both DynaLoRA variants, i.e. using either forward activations or backward gradients as importance scores, the evaluation loss grows less intensely than with normal LoRA training. We hypothesize that the model is overfitting on the dataset, as CoLA contains relatively few samples, and periodically deactivating some modules acts as a way of regularization.

Furthermore, through DynaLoRA, the number of active adapter modules at any given time step is decreased, thus achieving non-negligible improvements in memory consumption. As seen in Figure 2, DynaLoRA is able to conserve memory at minimal drop in model performance. We expect this trend to be even more apparent when DynaLoRA is applied on larger models, leading to significant reduction in memory demands during fine-tuning.

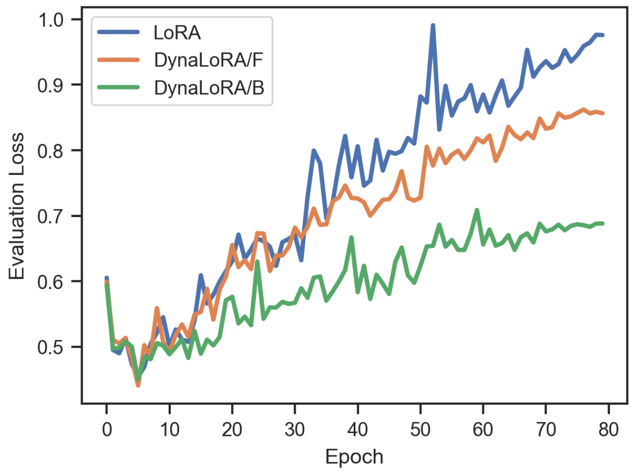

Figure 1: Model loss estimated on validation set (Task: CoLA, target modules: query and value).

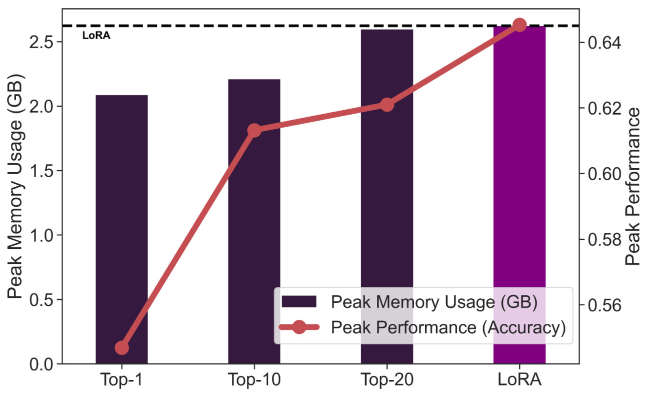

Figure 2: Peak Memory compared to Performance on CoLA task for varying number of active modules at once in RoBERTa-base. Bars indicate memory usage, while the line plot indicates downstream performance.

5. Ablation Study

In this section, we provide the results of the ablation studies performed. First, the results of the experiments regarding the allocation strategy are covered. Subsequently, the experiments that explore the scheduling policy are outlined. Finally, experimental results on the target modules included for selection are provided, along with a qualitative analysis of the model's module allocation behavior.

5.1. Allocation Strategy

We focus on different strategies for selecting the most appropriate modules to keep active in training throughout a given period. We present our results in Figure 3.

We discover that, across the selected variety of tasks, random allocation achieves very similar results to the discounted stochastic allocator with importance score calculated from forward or backward activations. We hypothesize that this may be attributed to the number of active modules being large enough to improve on the task regardless of the importance scores in the allocation strategy. To understand better the significance of different importance scores, we ran experiments using very low number (1-8) of active modules. See Figure 3

Our initial results on the CoLA task indicate that the importance scores calculated from gradient norms may be a strong indicator for module importance. However, our additional testing across other tasks does not support this – instead, we observe a very high variance in the target accuracy metrics.

5.2. Scheduling Policy

Briefly, we examine the various effects of allocation schedules and their impact on the performance of the model. The results are presented in Figure 4. We note a significant trend where too frequent re-allocation periods can negatively affect the model's training. We hypothesize the high frequency of updates inhibits model training by preventing it from meaningfully learning new information and updating its representations.

Beyond that, we find that there is a significant variance in the results and the effect of periodicity does not seem to manifest any apparent trends with regards to model performance.

5.3. Target Module Selection

To further examine the viability of dynamic LoRA adapter application, we fine-tune with LoRA on all linear layers, i.e. both the query, key, and value modules from the transformer blocks, as well as the MLP linear layers following the transformer blocks. We present our findings in Figure 5.

The results indicate high preference for DynaLoRA adapters at MLP linear layers of the neural network. We suspect that this trend may be attributed to the magnitude of these specific layers' forward activations in comparison to the transformer block's QKV modules. As the query and key matrix are always multiplied together, they need to have sufficiently low outputs in order to avoid exploding gradients or large parameter magnitude.

6. Conclusion and Limitations

We introduce DynaLoRA, an effective PEFT training method further reducing the costs of training adapter modules that is agnostic to the adapter method used. We perform a variety of ablation studies to determine the most effective parameter settings for training with DynaLoRA, including computed aggregate importance scores, adapter dispatching strategies, and scheduling policies.

We look forward to further experimentation with larger models, as well as on a wider variety of benchmarks and downstream tasks. We also welcome any reproductions and feasibility studies.

Further investigation will be needed for establishing the effectiveness of DynaLoRA and its variants within other applications, namely in computer vision, diffusion-based image and video generation, and so on.

References

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Ding, N., Lv, X., Wang, Q., Chen, Y., Zhou, B., Liu, Z., and Sun, M. Sparse low-rank adaptation of pre-trained lan-

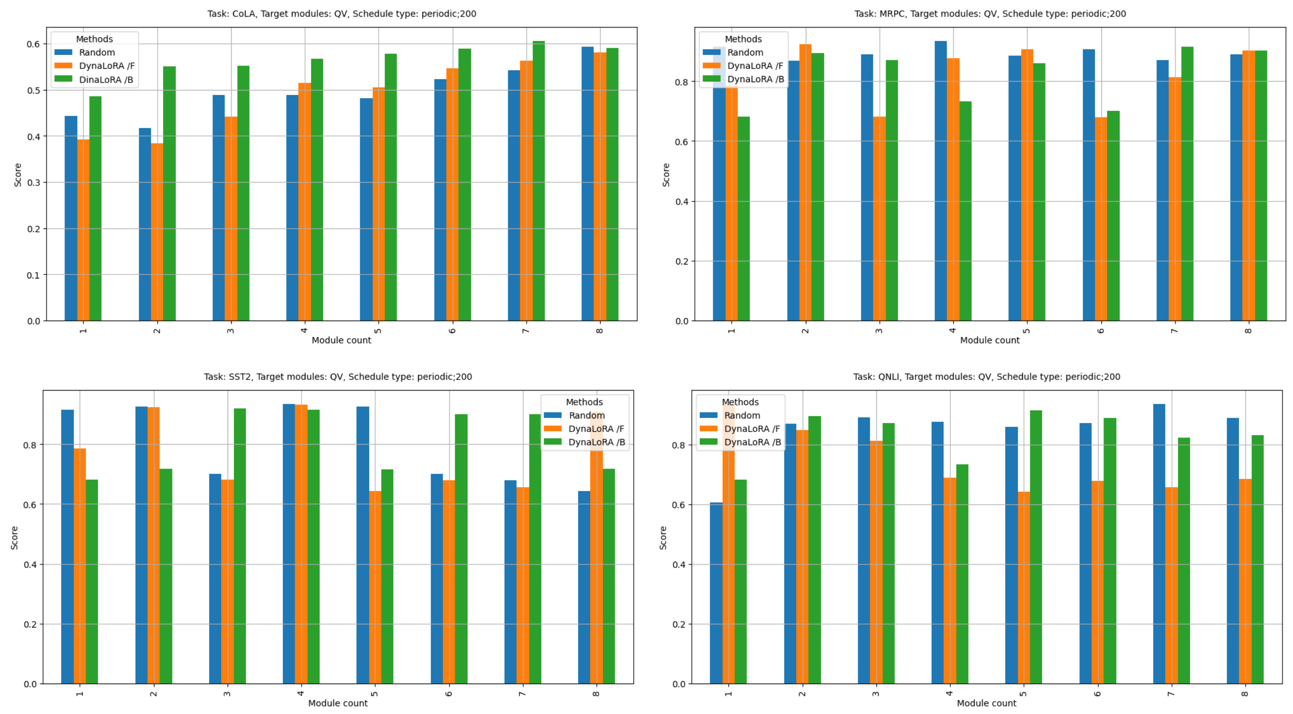

Figure 3: Three different scores for allocation strategies using very low number of active modules across selected GLUE-tasks. Random allocation (blue), discounted stochastic with importance scores from forward activations (orange), discounted stochastic with importance scores from backward activations (green). Each model was fine-tuned with periodic schedule type with periodicity of 200 steps. The modules targeted for fine-tuning were query and value modules within the transformer blocks.

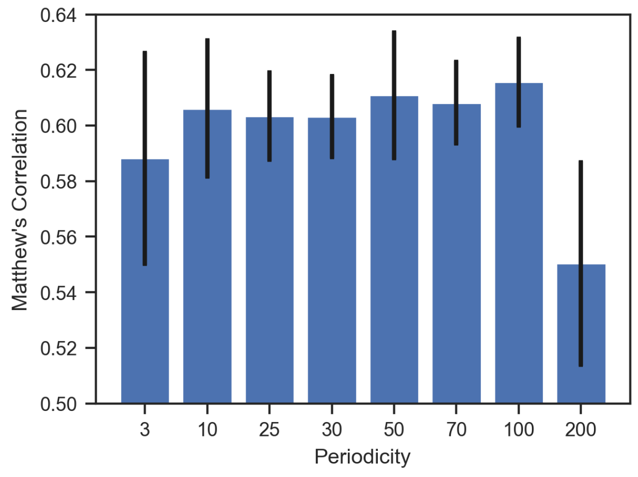

Figure 4: Effect of the periodicity of active module re-allocation. We fix the allocation strategy and only vary the period of updates.

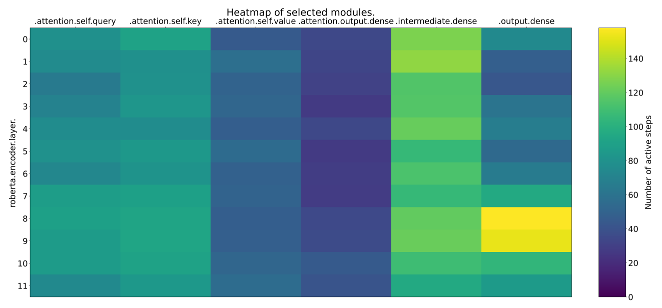

Figure 5: Heatmap of the total number of active steps a module has been active through. We only consider matrix norms of output of the forward application of the layer and allow for activation of all linear layers (including the MLP between Transformer blocks, diverging slightly from Hu et al. (2021) who only consider the query, key and value matrices).

guage models. *arXiv preprint arXiv:2311.11696*, 2023.

He, P., Liu, X., Gao, J., and Chen, W. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=XPZIaotutsD.

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.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Liu, Z., Lyn, J., Zhu, W., Tian, X., and Graham, Y. Alora: Allocating low-rank adaptation for fine-tuning large language models. *arXiv preprint arXiv:2403.16187*, 2024.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners, 2019.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Zaken, E. B., Ravfogel, S., and Goldberg, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *CoRR*, abs/2106.10199, 2021. URL https://arxiv.org/abs/2106.10199.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection, 2024.

Zhou, H., Lu, X., Xu, W., Zhu, C., and Zhao, T. Loradrop: Efficient lora parameter pruning based on output evaluation. *arXiv preprint arXiv:2402.07721*, 2024.