Different Rates for Different Weights: Decoupled Relative Learning Rate Schedules

Jan Ludziejewski
University of Warsaw
IDEAS NCBR

Jan Małaśnicki
University of Warsaw
IDEAS NCBR

Maciej Pióro
Polish Academy of Sciences
IDEAS NCBR

Michał Krutul
University of Warsaw
IDEAS NCBR

Kamil Ciebiera
University of Warsaw
IDEAS NCBR

Maciej Stefaniak
University of Warsaw
IDEAS NCBR

Jakub Krajewski
University of Warsaw
IDEAS NCBR

Piotr Sankowski
University of Warsaw
IDEAS NCBR

Marek Cygan
University of Warsaw
Nomagic

Kamil Adamczewski
IDEAS NCBR

Sebastian Jaszczur
University of Warsaw
IDEAS NCBR

Abstract

In this work, we introduce a novel approach for optimizing neural network training by adjusting learning rates across weights of different components in Transformer models. Traditional methods often apply a uniform learning rate across all network layers, potentially overlooking the unique dynamics of each part. Remarkably, our introduced Relative Learning Rate Schedules (RLRS) method accelerates the training process by up to 23%, particularly in complex models such as Mixture of Experts (MoE). Hyperparameters of RLRS can be efficiently tuned on smaller models and then extrapolated to $27\times$ larger ones. This simple and effective method results in a substantial reduction in training time and computational resources, offering a practical and scalable solution for optimizing large-scale neural networks.

1 Introduction

The learning rate is a crucial hyperparameter in Deep Learning, determining the size of the steps the optimization algorithm takes when updating model parameters during training. In the context of Transformers, which are widely used for tasks in Natural Language Processing and other areas, the learning rate significantly impacts the model's convergence and overall performance. While higher learning rates, due to larger updates to the model, may generally converge faster, the training also becomes less stable. Therefore, the learning rate needs to be chosen specifically to balance the speed and stability of the training process.

At the same time, modern Deep Learning architectures are not homogeneous, with different parts serving different purposes and behaving differently. For example, in the Transformer, the Attention block, the Feed-Forward (MLP) block, and the token Embedding have very different structures, functions, and—importantly—training dynamics. Moreover, components may behave differently depending on the training phase. For instance, in Mixture of Experts (MoE) models, the Router often stabilizes early in training, leading to deterministic routing to the Experts [17]. Additionally, weights in the Unembedding may diverge, potentially causing instabilities later in the training process [1, 21].

It is reasonable to assume that with such varied layers, their needs also vary, particularly in balancing training speed and stability. Nevertheless, we often treat all modules uniformly when setting the learning rate. A common practice, for example, is to reduce the learning rate for the whole model after

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

introducing a MoE layer [14]. By design, hyperparameters are typically tuned for the entire network, disregarding the fact that only one of the layers might be introducing issues such as instabilities. We aim to investigate the improvements made possible by loosening the implicit assumption of common hyperparameters across the model. Since each layer's training dynamics could be different, can we improve the training procedure by tailoring the learning rate schedules accordingly?

In this work, we introduce Relative Learning Rate Schedules (RLRS) method. RLRS decouples various optimization hyperparameters in Transformers and tunes them separately for different model components, including the Unembedding and Embedding layers, Attention, and, in the case of Mixture of Experts architectures, the Router and Experts. By tailoring the learning rate to the specific needs of each part, we enhance the overall performance and stability of the model.

We also propose a method for extrapolating RLRS effectively to models at least $27\times$ larger, making our approach practical. In essence, we offer a method: first, relative LRs should be tuned on a small model; later, the same relative LRs can be reused when training the model's significantly larger counterpart. Our method is easy to implement, with no additional overhead required, apart from the relatively inexpensive hyperparameter search on the small model. While tailored to our specific training setup, our relative values have proven to be robust across a range of hyperparameters, making them an excellent starting point. Additionally, we provide an analysis showing how these values, determined using automated methods, align with our intuitive understanding of Transformer training. In summary:

- We propose distinct learning rate schemes tailored for different components of a Transformer model, optimizing each part individually for improved overall performance.
- We show performance improvements of the introduced methods in standard Transformers with improvements growing in the Mixture of Experts-based model, highlighting the importance of relative learning rates for more complex models.
- We demonstrate that the hyperparameters tuned on small models extrapolate to larger models, showing that our approach generalizes effectively across different architecture sizes.

2 Decoupled Relative Learning Rates

We define a *decoupled learning rate* as a separate learning rate schedule applied to different layer types (also referred to as components, modules, or parts). Decoupled learning rate schedules allow the learning procedure to focus on different components during various phases of a model's pretraining, facilitating a more targeted and efficient optimization process.

We specify decoupled learning rates following the structure of a cosine learning rate scheduler [9], which is widely used for training Large Language Models (LLMs) [16, 5]. The cosine scheduler adjusts the learning rate over time according to a cosine function, starting with a high learning rate that gradually decays to a minimum value in a smooth, non-linear manner.

The parameters we introduce are:

- *Base LR* (η_{base}) – This is the reference learning rate for the entire model. In a typical cosine schedule, it represents the initial (or maximum) learning rate, serving as the peak value during the training cycle, following any possible warm-up period.
- *Base LR Final Fraction* (λ_{base}) – This fraction of the base learning rate determines the final learning rate at the end of the training. The Final (or Minimum) Learning Rate is the lowest learning rate value by the end of the training cycle.

For each component m of the model, we further define:

- *Relative Start LR* (λ^m_{start}) — the scaling factor of the base learning rate at the beginning of training.
- *Relative End LR* (λ^m_{end}) — the scaling factor of the final learning rate at the end of training.

Thus, the *decoupled learning rates* η^m_{start} and η^m_{end} for a component m are defined as:

$$\eta^m_{\text{start}} = \eta_{\text{base}} \times \lambda^m_{\text{start}} \qquad \eta^m_{\text{end}} = \eta_{\text{base}} \times \lambda_{\text{base}} \times \lambda^m_{\text{end}}$$

These values are adjusted for each transformer component. In this work, we distinguish following layer modules: Embedding, Attention, Final Linear Layer and additionally for dense model Feed-forward layer and for Mixture of Experts model, Expert and Router layer.

2.1 Extrapolation Method

Directly tuning relative values on Large Language Models (LLMs) may be impractical due to the high computational costs. To address this, we propose a method that fine-tunes these values on a smaller model and transfers them to a larger one. This approach significantly reduces the need for costly tuning on large models, offering substantial computational savings.

As described in Algorithm 1, our method involves conducting a local search for optimal values on smaller models. This search is based on the assumption that these relative values extrapolate effectively to larger models. It consumes only a fraction of the training time required for large models.

Algorithm 1 Relative LR Adjustment Algorithm

1: Find η_{base} for a small model.
2: For each module m, find relative values λ^m_{start} and λ^m_{end} on a small model.
3: Find base learning rate η_{base} for the large model.
4: Apply relative learning rates λ^m_{start} and λ^m_{end} from the small model.

While we do not claim that λ^m_{begin} and λ^m_{end} values are optimal for larger models, they are easy to use and yield substantial improvements, as shown in the next section. We leave the investigation of optimal extrapolation as future work.

3 Results

A smaller model permits a broader exploration of hyperparameters. For a fine-grained search, we perform a local search (for details, see Appendix A.1) for the optimal η_{base}, as well as for each module λ^m_{start} and λ^m_{end}.

Type	LR Type	Base LR	Train Tokens	Speed-Up
$\text{MoE}_{8\times 34\text{M}}$	baseline	3×10^{-3}	1.3B	-
	relative	3×10^{-3}	1.3B	22.8%
$\text{Dense}_{34\text{M}}$	baseline	2×10^{-3}	1.3B	-
	relative	2×10^{-3}	1.3B	17.2%

Table 1: Using RLRS results in faster model convergence.

Type	LR Type	Base LR	Train Tokens	Speed-Up
$\text{Dense}_{113\text{M}}$	baseline	1×10^{-3}	2.5B	-
	relative	1×10^{-3}	2.5B	19.0%
$\text{MoE}_{8\times 113\text{M}}$	baseline	2×10^{-3}	2.5B	-
	relative	1×10^{-3}	2.5B	19.0%
$\text{MoE}_{8\times 113\text{M}}$	baseline	1×10^{-3}	14B	-
(overtrained)	relative	1×10^{-3}	14B	14.6%
$\text{Dense}_{906\text{M}}$	baseline	5×10^{-4}	20B	-
	relative	5×10^{-4}	20B	7.7%
$\text{MoE}_{8\times 906\text{M}}$	baseline	2×10^{-4}	20B	-
	relative	2×10^{-4}	20B	13.6%

Table 2: Gains from extrapolating relative learning rates to larger models.

Subsequently, we extrapolate the relative values to large models, with the largest one having 906M active parameters (and 5.67B total parameters). The models are trained both in a compute-optimal and overtrained setting. Remarkably, our gains largely hold. As shown in Table 2, applying the relative rates from smaller to larger models results in nearly a fifth reduction in training time for both MoE and dense models.

4 Analysis

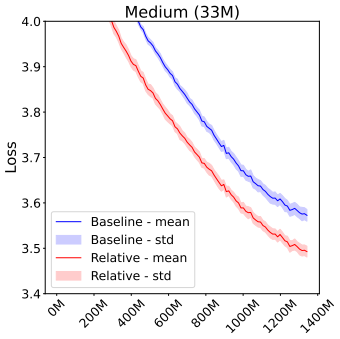

Figure 1: Training loss for Baseline versus RLRS for $\text{MoE}_{8\times 33\text{M}}$.

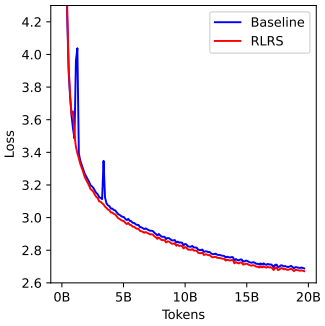

Figure 2: Stabilizing training with RLRS vs. baseline LR for $\text{MoE}_{8\times 906\text{M}}$.

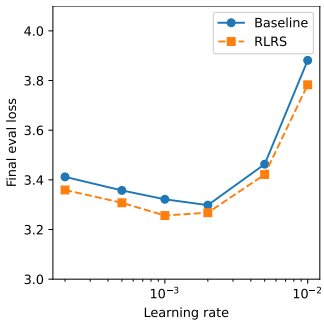

Figure 3: Loss of RLRS and baseline for different η_{base} on $\text{MoE}_{8\times 113\text{M}}$ models. RLRS results in better loss across a range of learning rates.

In this section, we present the numerical results and trends for relative learning rates and analyze them in relation to each layer module. Although the values were obtained experimentally, they are notably interpretable and align well to address the existing issues of each component.

For the **Embedding** layer, the relative learning rate λ_{start} starts high at 5 and decays to 0.6. This aggressive early training helps the Embedding layer stabilize quickly, as it influences the entire network. As training progresses, the learning rate is reduced to avoid drastic changes in the embeddings, allowing the rest of the model to adjust accordingly. Similarly, in the **Unembedding** layer, the rate decreases from 1 (0.6 in the case of MoE) to 0.4. The **Router**, known for its tendency to freeze early in training [17], determines which subset of the model's parameters is used for each token. Our relative rates aim to prevent this premature convergence, increasing from 0.6 to 1 during pretraining. Moreover, as the Router tends to be unstable initially, the rate starts with a coefficient less than 1. The **Experts** have their learning rates adjusted accordingly. Initially, the rate is lower (0.3) to aid stability when the Router is essentially random, and later it increases to 1.125 as the model stabilizes. For the **Attention** layers, the relative learning rate remains unchanged in MoE, making it unique in not requiring relative rates, while in dense models, it decreases from 1 to 0.2.

The overall customization of learning rates proposed in this work has several benefits. First, it leads to faster training convergence, as shown in Section 3 and Figure 1. Furthermore, training with relative rates is more stable in MoE setting. As seen in Figure 2, the baseline exhibits instabilities that are absent with the relative schedules. This is also intuitive, as MoE models are considered unstable and require lower learning rates for optimal learning. Finally as shown in Figure 3, relative learning rates consistently improve the baseline across various base learning rates, distinguishing our schedule from simply scaling the base learning rate.

5 Conclusion

We have presented a method for decoupling learning rate schedules across different Transformer components, which removes the implicit assumption of homogeneity between them, resulting in better training speed and stability. This method significantly enhances performance in both dense and Mixture of Experts (MoE) models. By tuning relative learning rates on smaller models, this approach can be used to cost-effectively achieve significant improvements in the training of order-of-magnitude larger models.

References

[1] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

[2] Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers, 2024.

[3] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

[4] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. arXiv preprint arXiv:2402.12354, 2024.

[5] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. arXiv preprint arXiv:1801.06146, 2018.

[7] 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. arXiv preprint arXiv:2106.09685, 2021.

[8] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407, 2018.

[9] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983, 2016.

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[11] Jan Ludziejewski, Jakub Krajewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts. In Proceedings of the 41st International Conference on Machine Learning (ICML), 2024.

[12] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[13] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[14] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In Proceedings of the 39th International Conference on Machine Learning, 2022.

[15] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18, pages 194–206. Springer, 2019.

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[17] Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. Openmoe: An early effort on open mixture-of-experts language models. arXiv preprint arXiv:2402.01739, 2024.

[18] Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. arXiv preprint arXiv:2006.14548, 2020.

[19] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. arXiv preprint arXiv:2203.03466, 2022.

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

[21] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. arXiv preprint arXiv:2202.08906, 2022.

A Experimental Setup

All models used in this study are decoder-only Transformers trained on the C4 dataset [13]. The GPT-2 tokenizer [12] is employed. We optimize using AdamW [10] and apply cosine decay with a linear warmup for the first 1% of training steps. For better stability, weight initialization follows a truncated normal distribution with a reduced scale, as suggested by [3]. Mixed precision training is used, with Attention and Router calculated at high precision. The models use SwiGLU activation and Token Choice routing with 8 Experts, of which 1 is activated. We use two auxiliary losses for the Router: z-loss with a weight of 0.001 [21] and load balancing with a weight of 0.01 [3]. Compute-optimal training durations are based on [5], calculated for MoE as $20\times$ the number of active parameters excluding Embedding and Unembedding, as recommended in [11]. Moreover, we provide one comparison on overtrained $\text{MoE}_{8\times 113\text{M}}$, with almost 130 token to active parameter ratio. For all extrapolations, we tune base learning rates separately for RLRS and the baseline with the precision of a grid defined by $\{1e{-}n, 2e{-}n, 5e{-}n\}$.

For both dense and MoE models, the weight decay value has been optimized to 0.1, the initialization scale to 0.15, and *Base LR Final Fraction* (λ_{base}) to 0.04 for MoE and 0.06 for dense.

Type	Active Params	Total Params	d_{model}	n_{layers}	$n_{experts}$	BS	SL
$\text{MoE}_{8\times 34\text{M}}$	33.6M	210M	512	8	8	256	512
$\text{Dense}_{34\text{M}}$	33.6M	33.6M	512	8	8	256	512
$\text{MoE}_{8\times 113\text{M}}$	113M	708M	768	12	8	256	512
$\text{Dense}_{113\text{M}}$	113M	113M	768	12	8	256	512
$\text{MoE}_{8\times 906\text{M}}$	906M	5.67B	1536	24	8	384	1024
$\text{Dense}_{906\text{M}}$	906M	906M	1536	24	8	384	1024

Table 3: Models used in this paper. BS indicates batch size, and SL indicates sequence length.

In Tables 1 and 2, we report a speedup metric that measures how much faster a training process becomes when relative rates are applied. It is calculated using $(\frac{T_{\text{base}}}{T_{\text{relative}}} - 1) \times 100\%$, where T_{base} is the number of steps performed in the standard training with a base learning rate, and T_{relative} is the number of steps incurred until the loss of the training with the relative learning rate schedule exceeds baseline loss. It is important to note that using this metric likely underestimates the improvement of our method since for relative learning rate training steps, when we compute the speedup, the cosine schedule has not yet reached its end. We perform three runs for each configuration, except for $\text{Dense}_{906\text{M}}$, due to compute limitations. For each run, we measure the loss per S steps, where S is 1% of all training steps. The speedup is calculated over the means of 3 runs. To reduce variance from random data seeds, we use 3 specified data seeds for each model type comparison.

	Embedding	Unembedding	Router	Experts	Attention
Start	5	0.6	0.6	0.3	1
End	0.6	0.4	1	1.125	1

Table 4: Relative learning rate values (λ) for MoE.

	Embedding	Unembedding	Feed-Forward	Attention
Start	5	1	1	1
End	0.6	0.4	0.6	0.2

Table 5: Relative learning rate values (λ) for dense models.

A.1 Hyperparameter Optimization - Local Search

For small models, we tune all η and λ values, as well as weight decay and initialization scale. There are various ways to optimize these hyperparameters, and we assume our method is largely invariant to the specific optimization algorithm used. While grid search might be the most straightforward approach, it requires proper initialization values and an exponential number of experiments. Instead, we opted for a simple local search algorithm described below, which ran approximately 300 experiments on $\text{MoE}_{8\times 33\text{M}}$ before converging:

Algorithm 2 Local Search

1: Iterate over the set of optimized hyperparameters (η, λ, `weight_decay`, `init_scale`).
2: For a given hyperparameter, change its value by $\{\frac{1}{5}, \frac{2}{3}, 1, \frac{3}{2}, \frac{5}{1}\}$.
3: Run experiments and select the best result as the current value.
4: If any change has been made to all hyperparameters, return to step 1.

We used the same algorithm to optimize η's, weight decay, and initialization scale for the baseline.

B Cosine Scheduler Details

The cosine scheduler adjusts the learning rate according to a cosine curve over a specified number of epochs or iterations. The objective is to gradually reduce the learning rate from an initial value to a lower final value using the cosine function. The learning rate η_t at step t is computed using the cosine function:

$$\eta_t = \eta_{\text{end}} + \frac{1}{2}(\eta_{\text{start}} - \eta_{\text{end}}) \left(1 + \cos\left(\frac{T_t}{T_{\text{start}}}\pi\right)\right)$$

Where η_{max}: The initial learning rate. - η_{end}: The minimum learning rate (often set to a small value), T_t: The current epoch or iteration, T_{start}: The total number of epochs or iterations. s

C Ablations

	Model Part	Train Tokens	Speedup Relative
0	Embedding	2.5B	30.1%
1	Head	2.5B	14.5%
2	Gating	2.5B	2.2%
3	Expert	2.5B	10.5%

Table 6: Above, we present the speed-up obtained from selecting the relative LR for all the components over the training where one component is excluded. The speed-up is adjusted in relationship to the full relative schedule. Ablation studies are performed on a small 34M MoE model.

Figure 4 also shows the importance of tuning the relative learning rates for individual modules. The study indicates the particular significance of Embedding and Unembedding. It is important to note that the improvement brought by the method comes largely from the interactions between the relative rates for all the components, rather than any specific module.

D Related and Future Work

The literature on learning rates in Machine Learning, particularly for Transformers, highlights the importance of adaptive learning rate schedules. Stochastic Weight Averaging (SWA) [8] utilizes a modified learning rate schedule that applies a decaying learning rate during the initial phase of training, followed by a constant rate for the remainder. Differentiating learning rates for different layers is also commonly discussed, as the bottom and top layers of Transformer models capture different types of information. In [15], the authors introduce layer-wise learning rate decay, applying higher learning

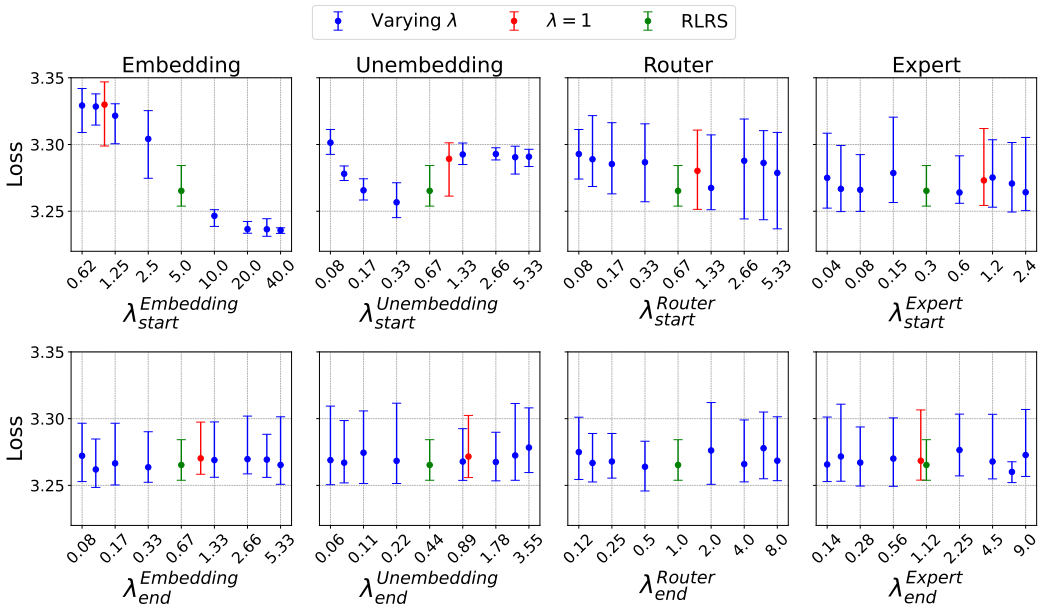

Figure 4: Varying relative learning rates for different Transformer components for both λ_{start} and λ_{end}. A given scaling factor for a component is varied, while others are fixed. The values in green are the results of the local search applied in this paper. The study shows that the selected values are consistently better than the baseline learning rate and generally outperform neighboring values, with some potential room for improvement. Note that in some instances, the training diverged, which caused the extraordinarily large bars.

rates to top layers and lower rates to bottom layers. A related concept, discriminative fine-tuning, is discussed in [6]. Additionally, [2] explores how various parameterizations and optimizers impact the learning process in large-scale models and proposes a per-layer learning rate strategy.

D.1 Combination with Tensor Programs

Our method examines the transfer of relative learning rates; however, the base learning rate must still be tuned independently for the extrapolated model. Approaches such as Tensor Programs [18, 19] propose parameterizations that facilitate the transfer of the base learning rate. By integrating these two approaches, it may be possible to achieve a zero-shot transfer of the relative learning rates.

While our methods share similarities with Tensor Programs and draw inspiration from them, our project has a distinct goal. We seek to identify implicit assumptions in the tuning process and decouple parameters to devise a scheme that allows Large Language Models (LLMs) to converge in fewer steps. Our extrapolations demonstrate that our optimization scheme is dependent on the architecture rather than the model size. This scheme is defined relative to the base learning rate, which must be tuned individually for each model size. Our method does not aim to facilitate learning rate transfer between different model sizes and is supported by experimental evidence. We do not mathematically examine the limits of parameter updates in a gradient descent step. A key difference is that our relative values change dynamically during training, and our goal is to enable the model to focus on different aspects during pretraining.

D.2 Fine-Tuning

Fine-tuning allows users to adapt pre-trained Large Language Models (LLMs) to more specialized tasks. In traditional fine-tuning, certain model components are often "frozen" (effectively setting their relative learning rates to zero) to preserve learned knowledge while adapting other parts. Our proposed method introduces a more flexible approach, serving as a continuous alternative to freezing parameters. This allows for fine-grained control over information transfer within specific components

of the model. Consequently, our method could be particularly applicable to fine-tuning scenarios and could complement existing methods that involve freezing parameters. Parameter-Efficient Fine-Tuning (PEFT) techniques, such as LoRA [7], address this by updating only a subset of parameters while freezing the rest. Our work aligns with more advanced methods like LoRA+ [4], which selects different learning rates for the adapter matrices, and AdaLoRA [20], which adapts the rank of the LoRA matrices, providing enhanced flexibility in the fine-tuning process.