Extending μP: Spectral Conditions for Feature Learning Across Optimizers

Akshita Gupta GUPTA417@PURDUE.EDU
Purdue University, West Lafayette, IN, USA

Marieme Ngom MNGOM@ANL.GOV
Sam Foreman FOREMANS@ANL.GOV
Venkatram Vishwanath VENKAT@ANL.GOV
Argonne National Laboratory, Lemont, IL, USA

Abstract

Tuning hyperparameters (HPs) for large language models (LLMs) is computationally expensive. Maximal update parameterization (μP) offers width-independent scaling rules that stabilize HPs, but prior derivations for SGD and Adam rely on tensor programs, which are difficult to extend. Building on recent work that introduced spectral conditions as an alternative to tensor programs, we propose a framework to derive μP for a broader class of optimizers, including AdamW, ADOPT, LAMB, and Sophia. We validate our derivations on NanoGPT and further provide empirical insights into depth-scaling parameterization for these optimizers.

1. Introduction

Large language models (LLMs) have achieved remarkable progress in generative AI, yet their performance and reproducibility depend on many interacting factors. A key aspect of training LLMs is the optimization routine, which can become unstable as models grow in size and complexity. To improve stability and efficiency, several modifications to existing optimizers have been proposed. For example, LAMB [17] proposes a layer-wise adaptive optimization routine to reduce the computational time required for training deep neural networks over large mini-batches, while Sophia [8] is a second-order method which uses a light-weight estimate of the diagonal of the Hessian as a preconditioner, and achieves faster convergence than Adam while being more robust to non-convex landscapes. Muon is another recent optimizer designed explicitly for scaling with model size [5, 9].

Although these recent algorithms demonstrate strong performance, the computational overhead of hyperparameter (HP) tuning poses a fundamental scalability bottleneck for training LLMs. To address this challenge, practitioners have heuristically tuned HPs on smaller models to guide the search for optimal configurations in larger models. Authors in [13, 14] formalized this approach by proposing a zero-shot HP transfer algorithm based on maximal update parameterization (μP) which stabilizes feature learning across different model layers. μP is implemented through careful scaling of weights and HPs proportional to the model width, with scaling factors tailored to the specific architecture and optimization algorithm. Under μP, feature learning is stable throughout the training process and HPs are stable across increasing model width.

While μP delivers strong results [2, 12, 14], it is tedious to implement in existing large codebases and difficult to understand in practice. To address this, authors in [15] proposed simpler spectral norm conditions on weight matrices that lead to the same width-independent and maximal feature learning properties of μP. This work focuses on using the more intuitive and tractable spectral conditions to derive μP for a wide range of optimizers. This approach also allows to unify the analysis for different layers of the model without requiring matching input-output dimensions for the hidden layers.

Our contributions are threefold: (1) we derive μP for adaptive first and second-order optimizers (AdamW, ADOPT, LAMB, Sophia) via a novel spectral-conditions approach; (2) we demonstrate zero-shot HP transfer (specifically of the optimal learning rate) across model width on a benchmark LLM (NanoGPT [6]); and (3) we provide an empirical study of zero-shot HP transfer across model depth for these optimizers.

The final results can be summarized in Fig. 1 which shows that the optimal learning rate corresponding to the minimum validation loss is stable across increasing model widths (256 to 2048) based on the proposed μP scalings for AdamW, ADOPT, LAMB and Sophia (Table 2) on the NanoGPT model [6].

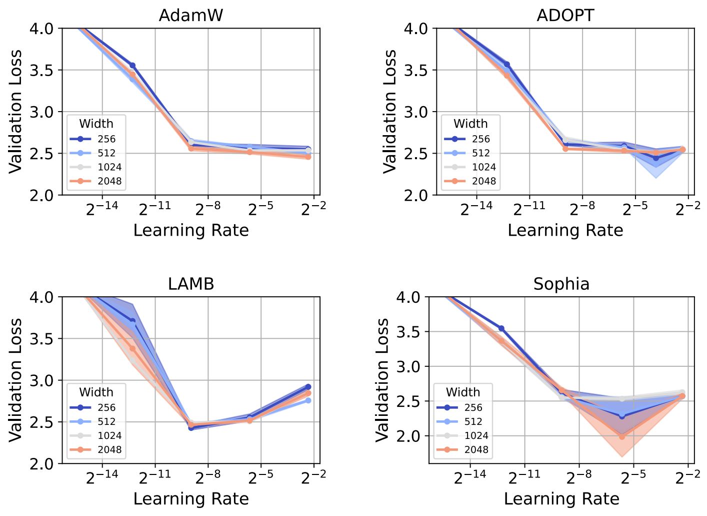

Figure 1: Mean validation loss for increasing model width and different learning rates across four optimizers: AdamW (top left), ADOPT (top right), LAMB (bottom left), and Sophia (bottom right).

2. Background

For the discussion in Sections 2 and 3 we derive μP for a linear MLP, similar to the model used in [15]. Let us consider an MLP with L layers. Let $\mathbf{x} \in \mathbb{R}^{n_0}$ denote the input vector and $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ denote the weight matrix for the $l-$th layer of the model. Then the feature vector $\mathbf{h}_l \in \mathbb{R}^{n_l}$ at layer l for the input $\mathbf{x}$ is given as

$$\mathbf{h}_l(\mathbf{x}) = \mathbf{W}_l \mathbf{h}_{l-1}(\mathbf{x}), \qquad \forall l = 1, 2, \ldots, L \tag{1}$$

where $\mathbf{h}_0(\mathbf{x}) = \mathbf{x}$. Let $\mathcal{L} = g(\mathbf{h}_L(\mathbf{x}), \mathbf{y})$ denote the loss, where $g : \mathbb{R}^{n_0} \times \mathbb{R}^{n_L} \to \mathbb{R}$ is a loss function, $\mathbf{y} \in \mathbb{R}^{n_L}$ is the target vector corresponding to the input $\mathbf{x}$ and $\mathbf{h}_L(\mathbf{x}) \in \mathbb{R}^{n_L}$ is the output vector returned by the MLP. After one step of training, the change in the weight matrices is typically a function of the history of the gradients. Let us denote this function as $\Psi(\cdot)$. The change in weights from time instant t to $t+1$ can be then written using the following update rule

$$\mathbf{W}_l^{(t+1)} = \mathbf{W}_l^{(t)} - \eta^{(t+1)} \Psi(\{\nabla_{\mathbf{W}_l^{(i)}} \mathcal{L}\}_{i=1}^{t}) \tag{2}$$

where $\eta^{(t+1)}$ is the learning rate at time instant $t+1$. We specify the forms of $\Psi(\cdot)$ for different optimizers in Table 1. To reduce cumbersome notation, we omit time indices in the remaining sections unless their inclusion is necessary for clarity. This will not affect the derivation of μP as it is sufficient to analyze a single update step of the weight update rule to determine the correct scaling laws [2, 14]. Using eqs. (1) and (2) we can write the change in the weights and feature vectors for any layer l after one training step as

$$\Delta \mathbf{W}_l = -\eta \Psi(\{\nabla_{\mathbf{W}_l}\mathcal{L}\}) \quad \text{and} \quad \Delta \mathbf{h}_l(\mathbf{x}) = \Delta \mathbf{W}_l \mathbf{h}_{l-1}(\mathbf{x}) + \Delta \mathbf{W}_l \Delta \mathbf{h}_{l-1}(\mathbf{x}) + \mathbf{W}_l \Delta \mathbf{h}_{l-1}(\mathbf{x}).$$

Optimizer	$\boldsymbol{\Psi}(\cdot)$
AdamW / ADOPT	$\dfrac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}}+\epsilon} + \lambda \mathbf{W}_l^{(t)}$
Sophia	$\text{clip}\left(\dfrac{\mathbf{m}^{(t)}}{\max\{\gamma \mathbf{h}^{(t)}, \epsilon\}}, 1\right) + \lambda \mathbf{W}_l^{(t)}$
LAMB	$\dfrac{\phi(\lvert\lvert \mathbf{W}_l^{(t)} \rvert\rvert_F)}{\lvert\lvert \mathbf{r}_l^{(t)} + \lambda \mathbf{W}_l^{(t)} \rvert\rvert_F}\left(\mathbf{r}_l^{(t)} + \lambda \mathbf{W}_l^{(t)}\right)$

Table 1: Values of $\Psi(\cdot)$ for different optimizers. Auxiliary variables are defined in Section 3 and Appendix C.

2.1. Maximal Update Parametrization (μP)

Authors in [13, 14] proposed μP to ensure that overparameterized models do not learn trivial features, or that the feature values do not blow up with increasing model width. In practice, μP is implemented via the abc-parameterization [13] which ensures that the MLP weights, their initial variance and the learning rate are appropriately scaled with respect to the width of the model. In [13], the abc-parameterization was introduced for MLPs where the hidden layers have the same width, that is, $n_{l-1} = n_l = n$ for $l = 2, \dots, L-1$. For simplicity, it was assumed that the inputs and outputs are scalars. Then, for each layer, the set of parameters $\{a_l, b_l\}_{l=1}^{L} \cup \{c\}$ comprise the abc-parameterization to

1. Initialize the weight matrix at every layer as $\mathbf{W}_l = n^{-a_l}[\mathbf{w}_l^{(i,j)}]$ where $\mathbf{w}_l^{(i,j)} \sim \mathcal{N}(0, n^{-2b_l}\sigma^2)$
2. Scale the learning rate such that $\Delta \mathbf{W}_l = -\eta\, n^{-c}\, \Psi(\{\nabla_{\mathbf{W}_l}\mathcal{L}\})$

where the scale of parameters σ and η are assumed to be independent of the model width. As emphasized in Section 1, the theoretical principles behind μP can be difficult to grasp. Recognizing these challenges, the following equivalent conditions for μP were proposed [15]

$$||\mathbf{h}_l(\mathbf{x})||_2 = \Theta(\sqrt{n_l}) \quad \text{and} \quad ||\Delta \mathbf{h}_l||_2 = \Theta(\sqrt{n_l}), \quad \text{for } l = 1, 2, \dots, L-1, \tag{C.1.}$$

where the appropriate vector and matrix norms are defined in Appendix A. While the above conditions concisely represent the requirements of μP, from eq. (1) we observe that corresponding conditions on the weight matrices are needed to derive the correct parameterization.

2.2. Spectral Conditions for Feature Learning

In [15], the authors proposed the following spectral norm conditions on the weight matrices and their one step updates

$$||\mathbf{W}_l||_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right) \quad \text{and} \quad ||\Delta\mathbf{W}_l||_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right) \quad \text{at layers} \quad l = 1, \ldots, L. \tag{C.2.}$$

In (C.2.), the condition on $||\mathbf{W}_l||_*$ is used to determine the scalings for the weights and their initial variance, while the condition on $||\Delta\mathbf{W}_l||_*$ is used to determine the scaling for the learning rate. Compared to the abc-parameterization, the condition on $||\mathbf{W}_l||_*$ determines the set of parameters $\{a_l, b_l\}$, which depends on the model architecture, while the condition on $||\Delta\mathbf{W}_l||_*$ determines the scalar c, which depends on the optimization routine. This means that deriving μP for different optimizers only changes the value of c, while the values of $\{a_l, b_l\}$ remain fixed since the underlying architecture is unchanged. Therefore, following the parameterization in [15], we set $b_l = 0$ for all $l = 1, 2, \ldots, L$ so that the initial variance of the weights is 1, and, the weight matrices are scaled by a factor of magnitude $\Theta\left(\frac{1}{\sqrt{n_{l-1}}} \min\left\{1, \sqrt{\frac{n_l}{n_{l-1}}}\right\}\right)$.

3. Deriving μP using Spectral Conditions

	Input Weights	Output Weights	Hidden Weights
Init. Var.	1 $(\frac{1}{n_{l-1}})$	1 $(\frac{1}{n_{l-1}^2})$	1 $(\frac{1}{n_{l-1}})$
Multiplier	$\frac{1}{\sqrt{n_{l-1}}}$ (1)	$\frac{1}{n_{l-1}}$ (1)	$\frac{1}{\sqrt{n_{l-1}}}$ (1)
AdamW LR	1 (1)	$\frac{1}{n_{l-1}}$ $(\frac{1}{n_{l-1}})$	$\frac{1}{n_{l-1}}$ $(\frac{1}{n_{l-1}})$
Sophia / ADOPT LR	1 $(-)$	$\frac{1}{n_{l-1}}$ $(-)$	$\frac{1}{n_{l-1}}$ $(-)$
LAMB LR	1 $(-)$	1 $(-)$	1 $(-)$

Table 2: Comparison of μP from spectral conditions (black) vs. tensor programs [14, Table 3] (red).

In [15], the authors used spectral conditions to derive μP for SGD. However, while they emphasized that conditions (C.1.) and (C.2.) are universal to any optimizer, they did not further discuss how to apply these conditions to derive μP for adaptive optimization routines. Our work highlights that although spectral conditions are easier to work with than the original tensor program approach, the analysis for each adaptive optimizer is different and requires a careful study of the order of magnitude of the coefficient terms that scale the gradients.

Therefore, we first demonstrate how to apply condition (C.2.) by deriving μP for AdamW, and corroborate our results with the μP scalings reported in the literature [14]. We then derive the proper parameterization for the ADOPT, LAMB, and Sophia optimizers (Appendix C), which have shown promising results for training LLMs. Our results are summarized in Table 2 and in Result 3.1. Note that the μP scalings derived in Table 2 can be applied in practice if the assumptions listed in Appendix B hold [15].

3.1. AdamW

Recall the update rule for AdamW [10],

$$\mathbf{W}_l^{(t+1)} = \mathbf{W}_l^{(t)} - \eta^{(t+1)}\left(\frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon} + \lambda\mathbf{W}_l^{(t)}\right) \tag{AdamW}$$

$$\text{where} \quad \hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{(1-\beta_1^t)} = \frac{1}{(1-\beta_1^t)}\left[\beta_1\mathbf{m}^{(t-1)} + (1-\beta_1)\nabla_{\mathbf{W}_l^{(t)}}\mathcal{L}\right] \quad ; \quad \mathbf{m}^{(0)} = 0$$

$$\hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{(1-\beta_2^t)} = \frac{1}{(1-\beta_2^t)}\left[\beta_2\mathbf{v}^{(t-1)} + (1-\beta_2)(\nabla_{\mathbf{W}_l^{(t)}}\mathcal{L})^2\right] \quad ; \quad \mathbf{v}^{(0)} = 0$$

From the spectral conditions in (C.2.), we need to find $c_1, c_2 \in \mathbb{R}$ such that

$$||\Delta\mathbf{W}_l||_* = \eta(n_l)^{-c_1}(n_{l-1})^{-c_2}\left\|\frac{\hat{\mathbf{m}}}{\sqrt{\hat{\mathbf{v}}}+\epsilon} + \lambda\mathbf{W}_l\right\|_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right). \tag{3}$$

Similar to previous works, we first analyze AdamW for $\beta_1 = \beta_2 = \epsilon = 0$. This reduces the above update rule to that of signSGD [1]. Additionally, since the gradient term dominates the weight decay term, we can ignore the latter because we are only concerned with an order-of-magnitude calculation. Therefore, (3) reduces to

$$||\Delta\mathbf{W}_l||_* = \eta(n_l)^{-c_1}(n_{l-1})^{-c_2}||\text{sign}(\nabla_{\mathbf{W}_l}\mathcal{L})||_* \approx \eta(n_l)^{-c_1}(n_{l-1})^{-c_2}||\text{sign}(\nabla_{\mathbf{W}_l}\mathcal{L})||_F$$

where the last equation follows because the spectral norm is almost equal to the Frobenius norm for low rank matrices (See Table 1 and discussion in [15, pg. 9]). From the definition of the Frobenius norm, we have $||\mathbf{1}_{n_l\times n_{l-1}}||_F^2 = \sum_{i=1}^{n_l}\sum_{j=i}^{n_{l-1}} 1 = n_l n_{l-1}$. This gives

$$||\Delta\mathbf{W}_l||_* = \eta(n_l)^{-c_1}(n_{l-1})^{-c_2}\Theta\left(\sqrt{n_l n_{l-1}}\right) = \Theta\left(n_l^{1/2-c_1}n_{l-1}^{1/2-c_2}\right).$$

By fixing $c_1 = 0$ and $c_2 = 1$, the spectral norm condition in (3) is satisfied. Therefore, the learning rate for AdamW should be scaled by a factor of $1/n_{l-1}$. Observe that this scaling is consistent with the μP derived using the tensor programming approach [14, Table 3], and this equivalence is highlighted in Table 2.

Scaling the Weight Decay Parameter: If the value of the weight decay parameter, λ, can not be ignored then it has to be scaled by a factor of n_{l-1}. This holds since, after scaling the learning rate, we want $||\frac{\eta}{n_{l-1}}\lambda\mathbf{W}_l||_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)$. From conditions (C.2.), $||\mathbf{W}_l||_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)$, therefore λ should be scaled to eliminate the extra $1/n_{l-1}$ factor. This is consistent with Table 1 in [3].

We use similar techniques to derive μP for ADOPT, Sophia and LAMB. The derivations are moved to Appendix C. We summarize our derivations of μP in Table 2 and formally state our result below.

Result: Under standing assumptions, for a linear MLP with L layers, if the weight matrices $\mathbf{W}_l = \sigma_l\tilde{\mathbf{W}}_l,\ l = 1, 2, \dots L$ are initialized as $\tilde{\mathbf{W}}_{i,j} \sim \mathcal{N}(0,1)$, then the spectral conditions (C.2.) are satisfied for AdamW, ADOPT and Sophia if

$$\sigma_l = \Theta\left(\frac{1}{\sqrt{n_{l-1}}}\min\left\{1, \sqrt{\frac{n_l}{n_{l-1}}}\right\}\right); \qquad \eta = \Theta\left(\frac{1}{n_{l-1}}\right),$$

and for LAMB if

$$\sigma_l = \Theta\left(\frac{1}{\sqrt{n_{l-1}}}\min\left\{1, \sqrt{\frac{n_l}{n_{l-1}}}\right\}\right); \qquad \eta = \Theta(1),$$

where $n_{l-1} = 1$ for input weights and $n_l = 1$ for output weights.

Figs. 2 and 3 illustrate the *coordinate checking* [13] validation of our derivation. We train the model over a few steps and compute the mean of the feature vectors after each step relative to their values at the first training step and increase the width of the model. The top rows show the mean values blowing up under standard parametrization (SP) while the bottom rows show stable behavior across width under our μP implementation. Coordinate check plots for LAMB and ADOPT are provided in Appendix D.

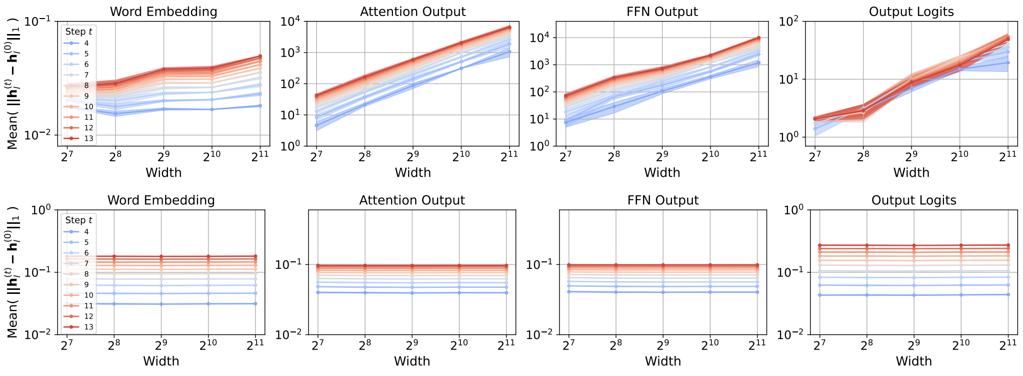

Figure 2: Coordinate check plots for AdamW under SP (top) and μP (bottom).

4. Numerical Results

We implement the derivations in Table 2 and validate our implementation on the NanoGPT codebase [6] via Fig. 1. Extensive numerical results, including training settings, hyperparameters values, depth scaling studies, and training/validation loss values for the different optimizers and model sizes can be found in Appendix D. All of our simulations were performed using four $A100$ GPUs of the Argonne Leadership Computing Facility's Polaris supercomputer [7].

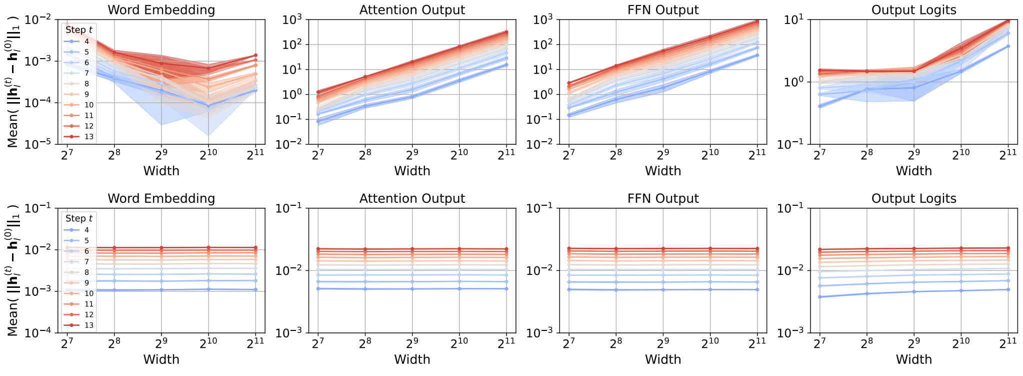

Figure 3: Coordinate check plots for Sophia under SP (top) and μP (bottom).

5. Conclusion

We derived μP for AdamW, ADOPT, Sophia, and LAMB optimizers using spectral conditions for feature learning across width and validated our results through zero-shot learning rate transfer on NanoGPT (Fig. 1). In future work, we aim to extend these derivations to second-order methods such as Shampoo [4], scale our experiments to larger models and additional hyperparameters, and, motivated by our depth-scaling simulations (Appendix D), develop a theoretical framework for depth-scaling parameterization.

References

[1] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International conference on machine learning*, pages 560–569. PMLR, 2018. URL https://doi.org/10.48550/arXiv.1802.04434.

[2] Charlie Blake, Constantin Eichenberg, Josef Dean, Lukas Balles, Luke Yuri Prince, Björn Deiseroth, Andres Felipe Cruz-Salinas, Carlo Luschi, Samuel Weinbach, and Douglas Orr. u-μp: The unit-scaled maximal update parametrization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=P7KRIiLM8T.

[3] Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025. URL https://doi.org/10.48550/arXiv.2505.01618.

[4] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018. URL https://doi.org/10.48550/arXiv.1802.09568.

[5] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. *Cited on*, page 10, 2024. URL https://kellerjordan.github.io/posts/muon/.

[6] Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

[7] Argonne Leadership Computing Facility. Polaris. https://www.alcf.anl.gov/polaris.

[8] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023. URL https://doi.org/10.48550/arXiv.2305.14342.

[9] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025. URL https://doi.org/10.48550/arXiv.2502.16982.

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. URL https://doi.org/10.48550/arXiv.1711.05101.

[11] Shohei Taniguchi, Keno Harada, Gouki Minegishi, Yuta Oshima, Seong Cheol Jeong, Go Nagahara, Tomoshi Iiyama, Masahiro Suzuki, Yusuke Iwasawa, and Yutaka Matsuo. Adopt: Modified adam can converge with any β_2 with the optimal rate. *Advances in Neural Information Processing Systems*, 37:72438–72474, 2024. URL https://doi.org/10.48550/arXiv.2411.02853.

[12] Benjamin Thérien, Charles-Étienne Joseph, Boris Knyazev, Edouard Oyallon, Irina Rish, and Eugene Belilovsky. μ lo: Compute-efficient meta-generalization of learned optimizers. In *OPT 2024: Optimization for Machine Learning*. URL https://doi.org/10.48550/arXiv.2406.00153.

[13] Greg Yang and Edward J Hu. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522*, 2020. URL https://doi.org/10.48550/arXiv.2011.14522.

[14] Greg Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34: 17084–17097, 2021. URL https://doi.org/10.48550/arXiv.2203.03466.

[15] Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023. URL https://doi.org/10.48550/arXiv.2310.17813.

[16] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite-depth neural networks. *arXiv preprint arXiv:2310.02244*, 2023. URL https://doi.org/10.48550/arXiv.2310.02244.

[17] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019. URL https://doi.org/10.48550/arXiv.1904.00962.

Acknowledgement: This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

Appendix A. Preliminaries

The l^p-norm of a vector $\mathbf{x} \in \mathbb{R}^n$, denoted as $||\mathbf{x}||_p$, is defined as $||\mathbf{x}||_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$. The spectral norm of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, denoted as $||\mathbf{A}||_*$, is defined as $||\mathbf{A}||_* := \max_{\mathbf{x}\in\mathbb{R}^n\setminus\{\mathbf{0}\}} \frac{||\mathbf{A}\mathbf{x}||_2}{||\mathbf{x}||_2}$. The Frobenius norm of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, denoted as $||\mathbf{A}||_F$, is defined as $||\mathbf{A}||_F := \sqrt{\sum_{i=1}^m \sum_{j=1}^n \mathbf{A}_{i,j}^2}$. It is a well known result that $||\mathbf{A}||_* \le ||\mathbf{A}||_F \le \sqrt{r}||\mathbf{A}||_*$, where r is the rank of matrix $\mathbf{A}$. A vector $\mathbf{x} \in \mathbb{R}^n$ is said to have $\Theta(n^\alpha)$-sized coordinates if $||\mathbf{x}||_2^2/n = \Theta(n^{2\alpha})$ as $n \to \infty$.

Appendix B. Assumptions

The derivations in Table 2 hold in practice for more complex models if the following assumptions are satisfied.

Assumption 1 *The weight updates do not cancel initial quantities.*

$$||\mathbf{W}_l + \Delta\mathbf{W}_l||_* = \Theta(||\mathbf{W}_l||_* + ||\Delta\mathbf{W}_l||_*)$$
$$||\mathbf{h}_l(\mathbf{x}) + \Delta\mathbf{h}_l(\mathbf{x})||_2 = \Theta(||\mathbf{h}_l(\mathbf{x})||_2 + ||\Delta\mathbf{h}_l(\mathbf{x})||_2).$$

Assumption 2 *If a nonlinear activation function $\phi(\cdot)$ is added to each layer of the MLP, then*

$$||\phi(\mathbf{h}_l(\mathbf{x}))||_2 = \Theta(||\mathbf{h}_l(\mathbf{x})||_2).$$

In other words, assumption 2 ensures that the order of magnitude of the inputs and outputs of an activation function are the same.

So far in our derivations, we assume that the mini-batch size is 1. In practice, if a mini-batch size of $B \in \mathbb{R}$ is used then for our calculations to hold, we need the following two assumptions. Observe that assumption 3 plays the same role as assumption 1.

Assumption 3

$$\|\Delta\mathbf{W}_l\mathbf{h}_l(\mathbf{x}_i)\|_2 = \Theta\left(\left\|\frac{1}{B}\Delta\mathbf{W}_l^{(i)}\mathbf{h}_l(\mathbf{x}_i)\right\|_2\right).$$

Assumption 4 *The batch size is independent of the width, that is $B = \Theta(1)$.*

Appendix C. Extending μP across optimizers

C.1. ADOPT

Recall that the update rule for ADOPT optimizer is the same as AdamW. The key difference lies in the sequence in which the terms $\hat{\mathbf{m}}^{(t)}$ and $\hat{\mathbf{v}}^{(t)}$ are updated [11]. From a theoretical perspective, this does not change the order of magnitude of the gradient function $\Psi(\{\nabla_{\mathbf{W}_l}\mathcal{L}\})$ from that of AdamW, and hence, the parameterization derived for AdamW also holds for ADOPT.

C.2. Sophia

Recall the update rule for Sophia [8],

$$\mathbf{W}_l^{(t+1)} = \mathbf{W}_l^{(t)} - \eta^{(t+1)} \operatorname{clip}\left(\frac{\mathbf{m}^{(t)}}{\max\{\gamma \mathbf{h}^{(t)}, \epsilon\}}, 1\right) - \eta^{(t)} \lambda \mathbf{W}_l^{(t)} \tag{Sophia}$$

where $\mathbf{h}^{(t)}$ is the estimate of the diagonal vector of the Hessian at time t. From the spectral conditions in (C.2.), we need to find $c_1, c_2 \in \mathbb{R}$ such that

$$||\Delta \mathbf{W}_l||_* = \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \left\| \operatorname{clip}\left(\frac{\mathbf{m}^{(t)}}{\max\{\gamma \mathbf{h}^{(t)}, \epsilon\}}, 1\right) - \lambda \mathbf{W}_l^{(t)} \right\|_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right).$$

For analysis, we consider $\beta_1 = \beta_2 = \epsilon = 0$, and since the weight decay term is usually very small, the above weight update simplifies to

$$\begin{aligned} ||\Delta \mathbf{W}_l||_* &= \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \left\| \operatorname{clip}\left(\frac{\nabla_{\mathbf{W}_l} \mathcal{L}}{\gamma \nabla^2_{\mathbf{W}_l} \mathcal{L}}, 1\right) \right\|_* \\ &= \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \left\| \operatorname{clip}\left(\frac{\nabla_{\mathbf{W}_l} \mathcal{L}}{\gamma |\nabla^2_{\mathbf{W}_l} \mathcal{L}|}, 1\right) \right\|_* \\ &\approx \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \left\| \operatorname{clip}\left(\frac{\nabla_{\mathbf{W}_l} \mathcal{L}}{\gamma |\nabla^2_{\mathbf{W}_l} \mathcal{L}|}, 1\right) \right\|_F \end{aligned}$$

where in the second equality we take modulus in the denominator because Sophia avoids negative diagonal terms in the Hessian (thereby avoiding convergence to a saddle point; see discussion in [8, pg. 6]). Observe that because of $\operatorname{clip}(\cdot, 1)$ the coordinate-wise weight updates, $[\Delta \mathbf{W}_l]_{i,j}$, are bounded as $|[\Delta \mathbf{W}_l]_{i,j}| \leq 1$. Therefore, we can compute an upper bound for the Frobenius norm as

$$\begin{aligned} ||\Delta \mathbf{W}_l||_* &\approx \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \left\| \operatorname{clip}\left(\frac{\nabla_{\mathbf{W}_l} \mathcal{L}}{\gamma |\nabla^2_{\mathbf{W}_l} \mathcal{L}|}, 1\right) \right\|_F \\ &\leq \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \frac{1}{\gamma} \Theta(\sqrt{n_l n_{l-1}}). \end{aligned}$$

Therefore, condition (C.2.) is satisfied by fixing $c_1 = 0$ and $c_2 = 1$, which is the same scaling used by AdamW.

Intuitively, it is easy to see why this result holds, because Sophia uses signSGD as the default method to handle negative Hessian terms (to avoid convergence to a saddle point). Additionally, when $\gamma = 1$, that is, all the elements in the weight update are clipped to 1, the upper bound holds exactly and we get the same scaling as AdamW. In practice, the authors suggest to choose γ such that $10\% - 50\%$ of the parameters are not clipped. Therefore, for each term which is not clipped, the above bound incurs an error of less than 1. However, as demonstrated in our simulations (Fig. 1), for the typical values of γ used in practice, the μP scaling derived based on the above calculation works well.

C.3. LAMB

Recall the update rule for LAMB [17]

$$\mathbf{W}_l^{(t+1)} = \mathbf{W}_l^{(t)} - \eta^{(t+1)} \frac{\phi(||\mathbf{W}_l^{(t)}||_F)}{||\mathbf{r}_l^{(t)} + \lambda \mathbf{W}_l^{(t)}||_F} \left(\mathbf{r}_l^{(t)} + \lambda \mathbf{W}_l^{(t)}\right) \tag{LAMB}$$

where

$$\mathbf{r}_l^{(t)} = \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon}.$$

The update rule for LAMB scales the gradient in each layer of the model by terms of orders $\frac{||\mathbf{W}_l||_F}{||\mathbf{r}+\lambda\mathbf{W}||_F}$. From the spectral conditions (C.2.) and because the update matrices typically have a low stable rank, we know that

$$||\mathbf{W}_l||_F \approx ||\mathbf{W}_l||_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)$$

and

$$||\mathbf{r}_l + \lambda \mathbf{W}_l||_F = \Theta\left(\sqrt{n_l n_{l-1}}\right)$$

if we ignore the weight decay term because it is dominated by the gradient term. Then,

$$\begin{aligned} ||\Delta \mathbf{W}||_* &\approx \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \Theta\left(\frac{1}{n_{l-1}}\right) ||\mathbf{r}_l^{(t)} + \lambda \mathbf{W}_l^{(t)}||_F && (4) \\ &= \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \Theta\left(\frac{1}{n_{l-1}}\right) \Theta\left(\sqrt{n_l n_{l-1}}\right) && (5) \\ &= \eta (n_l)^{-c_1} (n_{l-1})^{-c_2} \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right) && (6) \end{aligned}$$

where the second equality follows using the same analysis as for AdamW. Therefore, the update rule (LAMB) implicitly has the correct order for the weight updates. Hence, we fix $c_1 = c_2 = 0$. Intuitively, this result holds because the layerwise gradient scaling in (LAMB) causes the effective learning rate to be different for each layer.

Appendix D. Simulations

Consistent with existing literature, we verify μP for ADOPT, Sophia and LAMB optimizers by implementing the derived parameterization scheme (Table 2) in the NanoGPT codebase [6]. Although prior works have already implemented μP for AdamW, we present the results again for completeness. Table 3 lists some of the settings for our experimental setup to test μP.

We also present simulation results for depth-scaling parameterization for the above optimizers, using the implementation suggested in [3, 16]. Note that deriving proper depth-scaling parameterization for different optimizers is an ongoing work, and we only present preliminary results in this section to motivate further theoretical analysis. Table 4 lists some of the settings for our experimental setup to test depth-scaling parameterization.

The remainder of this section documents the simulation results for AdamW (Subsection D.2), ADOPT (Subsection D.3), Sophia (Subsection D.4) and LAMB (Subsection D.5) optimizers. For

Table 3: Hyperparameter values and training settings to test μP.

Architecture	NanoGPT [6]
Width	128 (scaled to 2048)
Depth	8
Number of heads	2
Total parameters	1.59 M (scaled to 403 M)
Dataset	Tiny Shakespeare
Vocab size	65
Tokens per iteration	8192
Batch size	2
Stopping criteria	Early stopping if validation loss doesnot improve in last 150 iterations
Optimizers	AdamW / Adopt / Lamb / Sophia
Hyperparameter search range	$\eta \in [2 \times 10^{-1}, 2 \times 10^{-5}]$

Table 4: Hyperparameter values and training settings to test depth-scaling parameterization.

Architecture	NanoGPT [6]
Width	256
Depth	2 (scaled to 64)
Total parameters	1.6 M (scaled to 50.56 M)
Dataset	Tiny Shakespeare
Vocab size	65
Tokens per iteration	8192
Batch size	2
Stopping criteria	Early stopping if validation loss doesnot improve in last 150 iterations
Optimizers	AdamW / Adopt / Lamb / Sophia
Hyperparameter search range	$\eta \in [2 \times 10^{-1}, 2 \times 10^{-5}]$

each optimizer we first present the coordinate check plots under standard parameterization, μP and depth-scaling parameterization. These plots serve as a quick implementation check to monitor whether the weights blow-up, diminish to zero or remain stable with increasing model size (see discussion [14, Section D.1, pg. 27]). We then provide tables and plots listing the training and validation loss for different learning rates, and increasing model width and model depth. The values in the table are the average loss values observed over multiple runs. While we do not document the standard deviations in the tables, they are highlighted in the plots. Note that since we are using an early stopping criteria, we rely more on the observations gained from the validation loss data than the training loss data.

D.1. Discussions

Overall, it is observed that the implementation of μP following Table 2 is quite stable with increasing model width. This is evident from the coordinate check plots for all the optimizers (Figs. 4, 5, 6, 7). Under standard parameterization, the top row of the coordinate check plots shows that the relative mean of the feature vectors blow-up with increasing model width. With the incorporation of μP in the codebase, the relative mean values of the feature vectors stabilize with increasing model width (middle row of coordinate check plots).

It is interesting to note that since the theoretical underpinnings for μP hold in infinite width [13], the model width has to be "large enough" for the coordinate check plots to stabilize. This is observed in the coordinate check plots for LAMB (Fig. 7) where the mean values of the feature vectors initially increase, but gradually stabilize with increasing model width. This phenomenon is also observed in Fig. 1 which demonstrate the zero-shot learning rate transfer across model width. In the minimum validation loss tables for ADOPT (Table 9) and LAMB (Table 17) the optimal value of the learning rate gradually stabilizes after a width of 256, whereas for AdamW (Table 5) and Sophia (Table 13) the optimal learning rate stabilizes after a width of 128. These inconsistencies across optimizers also suggest that introducing a "base model width" for μP scaling will introduce another HP. Therefore, we fix the value of the base model width to 1 in our implementation.

The second set of simulations empirically evaluate the performance of the depth-scaling parameterization in existing works [3, 16]. The coordinate check plots (bottom row) for depth-scaling demonstrate that the feature vectors are stable with increasing model depth. In the coordinate check plots for ADOPT and LAMB (Figs. 5 and 7) the feature vectors stabilize after a depth of 16, while for AdamW and Sophia (Figs. 4 and 6) the feature vectors are stable for shallow depths too. This phenomenon is similar to our observations for μP, because the depth-scaling parameterization is also derived for an infinite depth limit [16]. Therefore, to prevent tuning an additional "base model depth" HP, we fix its value to 1 in our simulation setup. However, the loss plots in Figs. 8 and 9 do not consistently demonstrate zero-shot learning rate transfer across increasing model depths. While the validation loss tables for AdamW (Table 7) and Sophia (Table 15) demonstrate that the optimal value of the learning rate stabilizes for deep models, the same is not observed for ADOPT (Table 11) and LAMB (Table 19), where the value of the optimal learning rate oscillates as the depth is increased. These results suggest that deriving depth-scaling parameterization for different optimizers needs a more thorough theoretical analysis. Additionally, performing simulations on a finer grid of learning rates can also give further insights into the depth-scaling behavior.

D.2. AdamW Optimizer

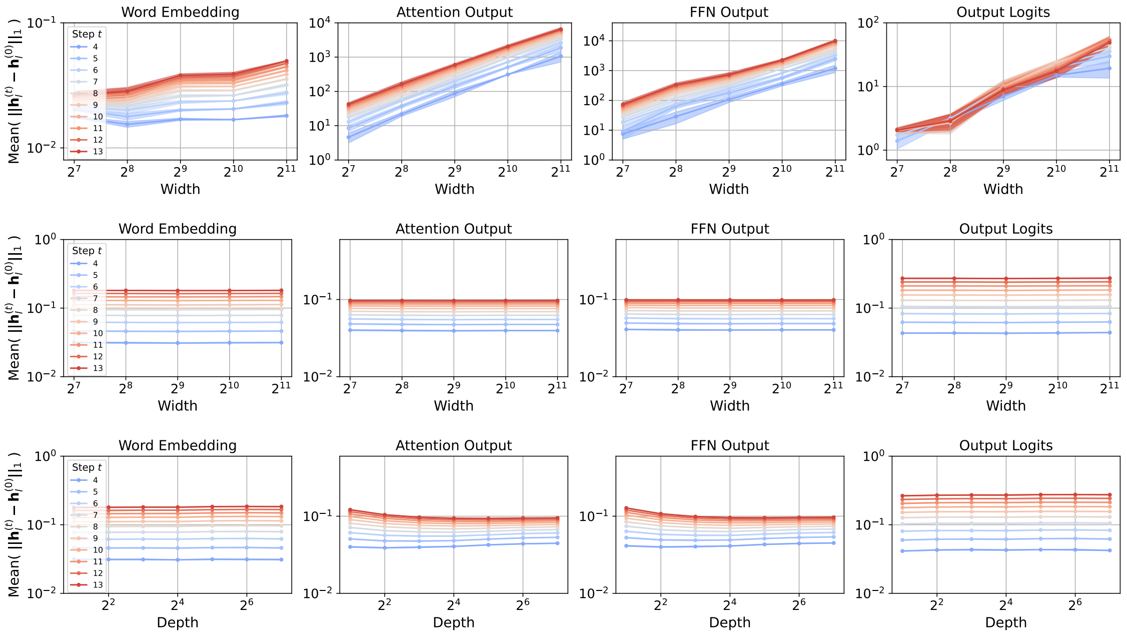

Figure 4: Coordinate check plots for AdamW under standard parameterization (top row), μP (middle row) and depth-scaling parameterization (bottom row).

Table 5: Mean validation loss for increasing model width and different learning rates for AdamW. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	2.54111195	2.54770319	2.50132585	2.53559383	2.45719266
2×10^{-2}	2.57009896	2.56583707	2.57900651	2.53385917	2.51431378
2×10^{-3}	2.63474766	2.6022807	2.64679337	2.63449661	2.55710355
2×10^{-4}	3.38827054	3.5544157	3.38896998	3.44941664	3.44561863
2×10^{-5}	4.09221347	4.08871428	4.05257797	4.08837303	4.08405908

Table 6: Mean training loss for increasing model width and different learning rates for AdamW. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	2.50418417	2.5338668	2.54295166	2.5088102	2.48891171
2×10^{-2}	2.5343496	2.54380401	2.52089596	2.53060627	2.48314587
2×10^{-3}	2.62260453	2.64226564	2.62231874	2.64035686	2.61562872
2×10^{-4}	3.44006387	3.46894224	3.35684594	3.45759169	3.43261838
2×10^{-5}	4.09007104	4.09235779	4.05840794	4.09009075	4.08594577

Table 7: Mean validation loss for increasing model depth and different learning rates for AdamW. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.53525917	2.55192765	2.53510944	2.50357556	2.51294963	2.53008548
5×10^{-2}	2.52700798	2.49422677	2.50334986	2.29428236	2.45176029	2.36860998
2×10^{-2}	2.55682977	2.52176666	2.56583563	2.30422862	2.45500112	2.5650301
2×10^{-3}	2.59745781	2.63078475	2.60228316	2.61588136	2.64065663	2.65051214
2×10^{-4}	3.41396125	3.41677833	3.55441554	3.45801504	3.43285489	3.47577778
2×10^{-5}	4.09297959	4.05970796	4.08871428	4.08113146	4.06712834	4.10902596

Table 8: Mean training loss for increasing model depth and different learning rates for AdamW. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.52620586	2.51541392	2.55663538	2.55096157	2.53933263	2.51851972
5×10^{-2}	2.53733587	2.47412181	2.48997227	2.25323725	2.4599328	2.29104145
2×10^{-2}	2.55709076	2.5514137	2.54380743	2.25292548	2.43532848	2.54148165
2×10^{-3}	2.59405073	2.61922661	2.64226492	2.62052504	2.63042879	2.59986623
2×10^{-4}	3.43114074	3.43951575	3.46894217	3.44096637	3.43087451	3.48062642
2×10^{-5}	4.09014066	4.05411609	4.09235779	4.07814837	4.06753333	4.10893885

D.3. ADOPT Optimizer

Table 9: Mean validation loss for increasing model width and different learning rates for ADOPT. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	2.55120134	2.54616404	2.54178079	2.5524296	2.54457998
7×10^{-2}	2.48560476	2.44316975	2.37087123	2.50733534	2.50883015
2×10^{-2}	2.43175697	2.58847451	2.57006375	2.54323697	2.53191725
2×10^{-3}	2.63016931	2.6073552	2.65681744	2.66118956	2.55337548
2×10^{-4}	3.528404	3.49065232	3.49065232	3.42789133	3.43255997
2×10^{-5}	4.09183598	4.08832375	4.0521698	4.08806594	4.08391444

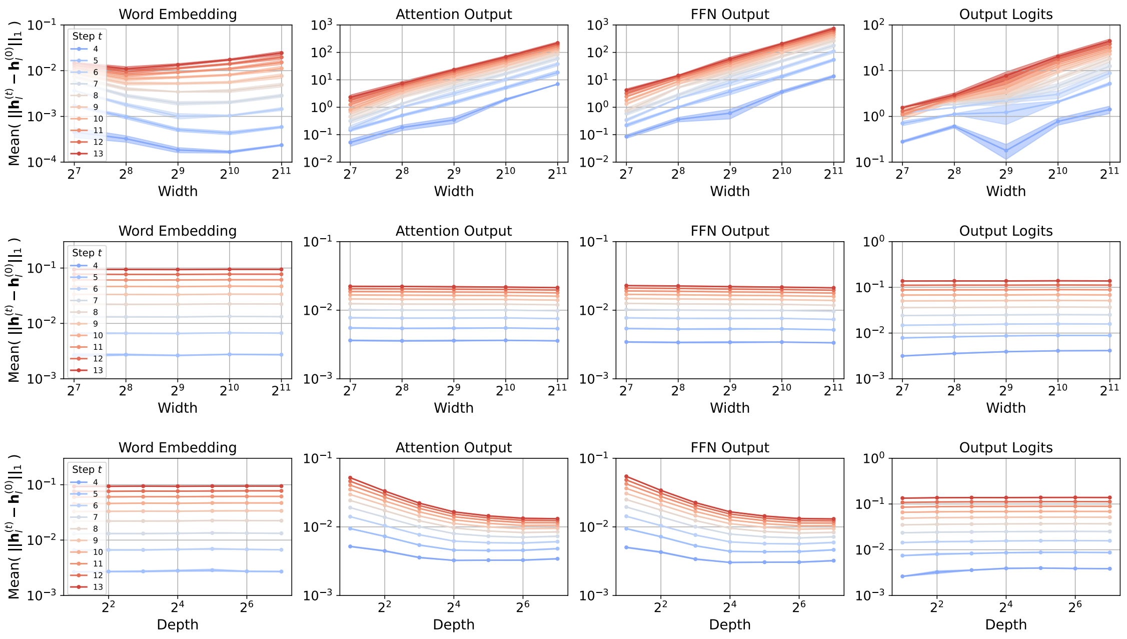

Figure 5: Coordinate check plots for ADOPT under standard parameterization (top row), μP (middle row) and depth-scaling parameterization (bottom row).

Table 10: Mean training loss for increasing model width and different learning rates for ADOPT. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	2.53555965	2.5340693	2.52383216	2.51339547	2.56196594
7×10^{-2}	2.45205251	2.44086281	2.33517623	2.49570537	2.51303013
2×10^{-2}	2.42622383	2.55501	2.48055744	2.50041199	2.49231974
2×10^{-3}	2.62170776	2.64970652	2.63305275	2.63701971	2.59599845
2×10^{-4}	3.48704513	3.46392854	3.42358224	3.36694487	3.44488136
2×10^{-5}	4.08967559	4.09197982	4.058026	4.08978923	4.08580319

Table 11: Mean validation loss for increasing model depth and different learning rates for ADOPT. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.56129368	2.51452438	2.54788987	2.51456078	2.52271922	2.55469418
9×10^{-2}	2.48695572	2.47477563	2.53124801	2.48145302	2.50687472	2.54724765
2×10^{-2}	2.56718413	2.50419029	2.58847276	2.44447954	2.54996069	2.52524622
2×10^{-3}	2.67992798	2.62949713	2.6073552	2.60433618	2.61753988	2.6286815
2×10^{-4}	3.41052596	3.46538957	3.56757394	3.47856442	3.43608022	3.56190586
2×10^{-5}	4.09267759	4.05929391	4.08832375	4.08074443	4.06675259	4.10877307

Table 12: Mean training loss for increasing model depth and different learning rates for ADOPT. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.52448209	2.50159733	2.53410482	2.5583094	2.50697446	2.5922548
9×10^{-2}	2.49610837	2.49662844	2.51774661	2.51524353	2.48542205	2.53163505
2×10^{-2}	2.53338401	2.51758933	2.55500905	2.45115765	2.50076302	2.47286979
2×10^{-3}	2.66055544	2.61120963	2.64970525	2.59826287	2.59101653	2.58349856
2×10^{-4}	3.42808644	3.46539354	3.46392854	3.4577349	3.43132528	3.49729109
2×10^{-5}	4.08983533	4.05368471	4.09197982	4.07775084	4.06715266	4.10868088

D.4. Sophia Optimizer

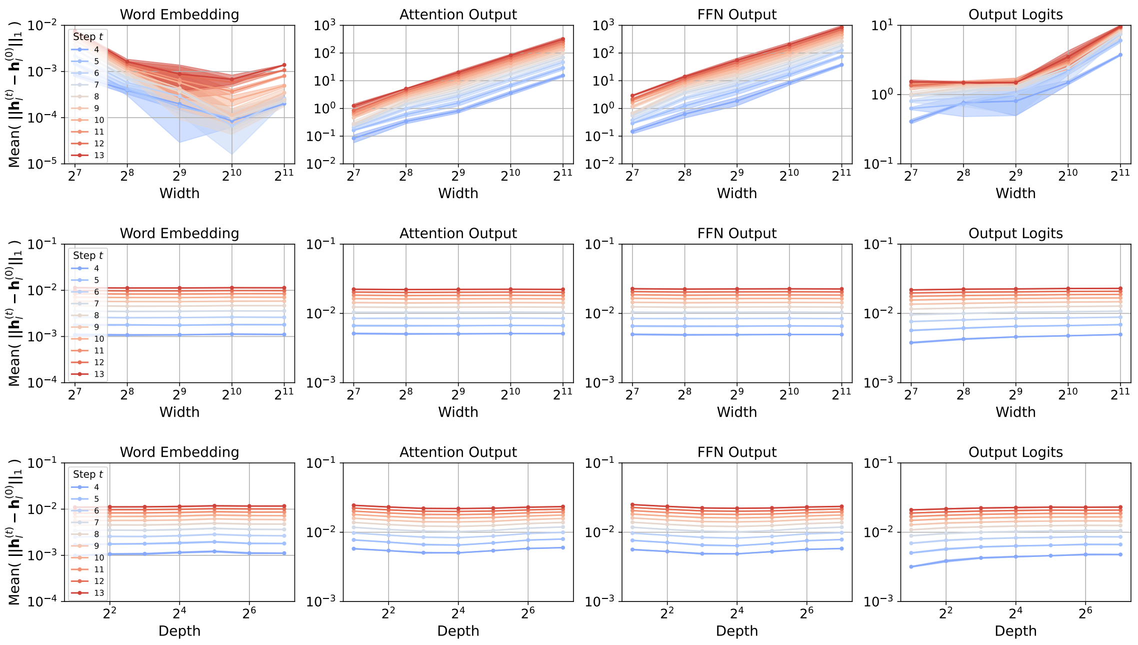

Figure 6: Coordinate check plots for Sophia optimizer under SP (top row); μP (middle row); depth scaling (bottom row).

Table 13: Mean validation loss for increasing model width and different learning rates for Sophia. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	3.0969398	2.57144117	2.56875261	2.62573036	2.57240287
2×10^{-2}	2.27450609	2.27830847	2.31632638	2.53347905	1.98427689
2×10^{-3}	2.5456597	2.61430057	2.5594302	2.54869485	2.65462987
2×10^{-4}	3.35409013	3.54614369	3.36089802	3.35862382	3.36431138
2×10^{-5}	4.08766381	4.08859126	4.06069756	4.08811712	4.08371623

Table 14: Mean training loss for increasing model width and different learning rates for Sophia. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	3.12824965	2.53182054	2.5844752	2.56465872	2.56372396
2×10^{-2}	2.1975925	2.25915702	2.20263139	2.48931464	1.89357849
2×10^{-3}	2.56100543	2.62829208	2.58921242	2.51228778	2.63180097
2×10^{-4}	3.4166019	3.4667743	3.33197419	3.31344899	3.38258688
2×10^{-5}	4.08155664	4.09204737	4.0652949	4.08984947	4.08579906

Table 15: Mean validation loss for increasing model depth and different learning rates for Sophia. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.5213503	3.01081316	3.22649105	3.34855215	3.24310446	3.12229093
2×10^{-2}	2.4717048	2.27232289	2.24736114	2.47475751	2.46061246	1.93401444
2×10^{-3}	2.54103192	2.58136233	2.61035593	2.610612	2.45068415	2.55488427
2×10^{-4}	3.40887721	3.52765425	3.54587563	3.40669481	3.33997742	3.47574107
2×10^{-5}	4.09267314	4.06576761	4.08859126	4.08140405	4.066552	4.10874732

Table 16: Mean training loss for increasing model depth and different learning rates for Sophia. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.5294884	2.98463766	3.19698143	3.31635459	3.18007644	3.11119755
2×10^{-2}	2.47805278	2.23049533	2.18775682	2.49183981	2.43022792	1.89074814
2×10^{-3}	2.50706561	2.58210929	2.62436374	2.58181063	2.48254434	2.50322294
2×10^{-4}	3.42514054	3.5769248	3.46681722	3.38244406	3.36329389	3.44658494
2×10^{-5}	4.0898249	4.06124306	4.09204753	4.07890431	4.06699435	4.10841576

D.5. LAMB Optimizer

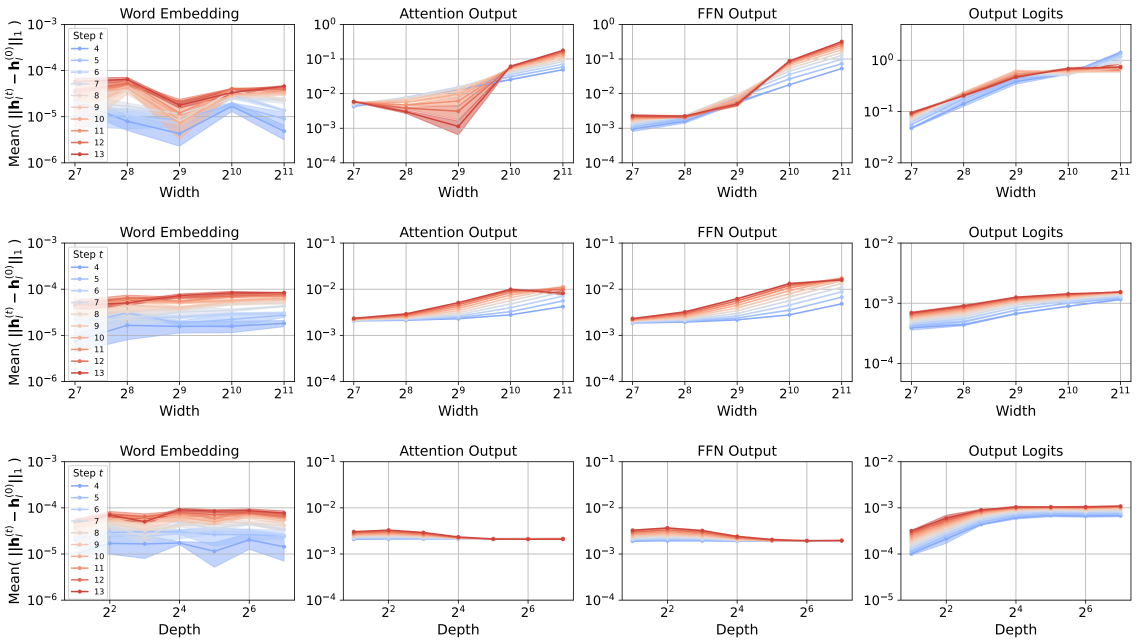

Figure 7: Coordinate check plots for LAMB optimizer under SP (top row); μP (middle row); depth scaling (bottom row).

Table 17: Mean validation loss for increasing model width and different learning rates for LAMB. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	3.3306915	2.91992474	2.75658234	2.84724092	2.84511503
2×10^{-2}	2.27427769	2.55330944	2.53250345	2.50694895	2.51612274
2×10^{-3}	2.46762419	2.42723028	2.47571055	2.49152549	2.46575729
2×10^{-4}	3.69672974	3.70961714	3.66877778	3.2370429	3.37923479
2×10^{-5}	4.16929531	4.1694754	4.1684103	4.1674579	4.16771809

Table 18: Mean training loss for different model width and different learning rates for LAMB. The minimum loss for each width is highlighted in green.

LR / Width	128	256	512	1024	2048
2×10^{-1}	3.31638861	2.88406754	2.75997527	2.79131405	2.84583108
2×10^{-2}	2.30849349	2.52181562	2.52505978	2.49564608	2.57582164
2×10^{-3}	2.54151122	2.43793472	2.4520429	2.45778489	2.45273058
2×10^{-4}	3.71788796	3.71333241	3.66877246	3.21714981	3.37149858
2×10^{-5}	4.16935237	4.1692287	4.1684432	4.16723283	4.16766739

Table 19: Mean validation loss for increasing model depth and different learning rates for LAMB. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.76534136	2.85949779	2.88115621	3.26932732	3.24093787	3.097018
2×10^{-2}	2.50858307	2.51164389	2.55355501	2.33967662	2.48308444	2.11406271
7×10^{-3}	2.45117172	2.46691815	2.50231234	2.45691435	2.48629936	2.45780365
2×10^{-3}	2.50483624	2.54284684	2.42723123	2.43291903	2.43262172	2.42000318
2×10^{-4}	3.6441706	3.79367606	3.70963343	3.57373738	3.61402575	3.42223287
2×10^{-5}	4.16981506	4.1691486	4.1694754	4.16932933	4.16817395	4.16773876

Table 20: Mean training loss for increasing model depth and different learning rates for LAMB. The minimum loss for each depth is highlighted in green.

LR / Depth	2	4	8	16	32	64
2×10^{-1}	2.75818356	2.83254337	2.89807558	3.24917714	3.13998501	3.04494317
2×10^{-2}	2.48152947	2.48629141	2.52084017	2.32515387	2.51233983	2.11694264
7×10^{-3}	2.51274339	2.50058635	2.52087537	2.48127453	2.40303373	2.41943057
2×10^{-3}	2.52206691	2.49440336	2.43791986	2.4603858	2.44611494	2.39355747
2×10^{-4}	3.63792483	3.75471322	3.71338932	3.57781116	3.61477113	3.3763895
2×10^{-5}	4.16952674	4.16912842	4.16922871	4.16896884	4.16813739	4.16744947

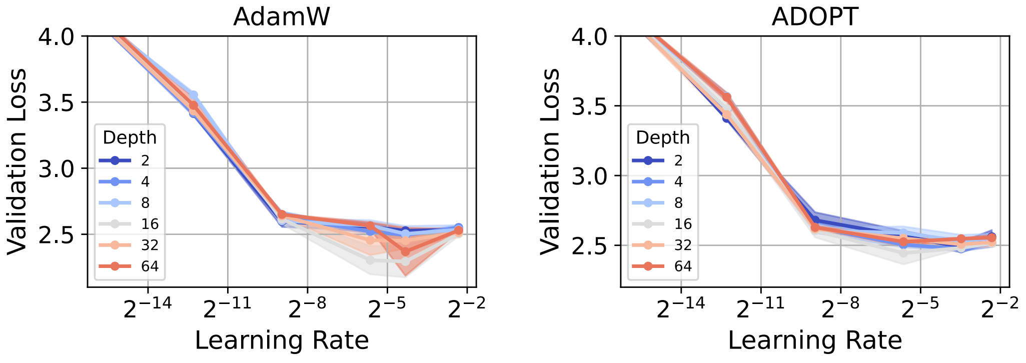

Figure 8: Mean validation loss for increasing model depth and different learning rates for AdamW (left) and ADOPT (right).

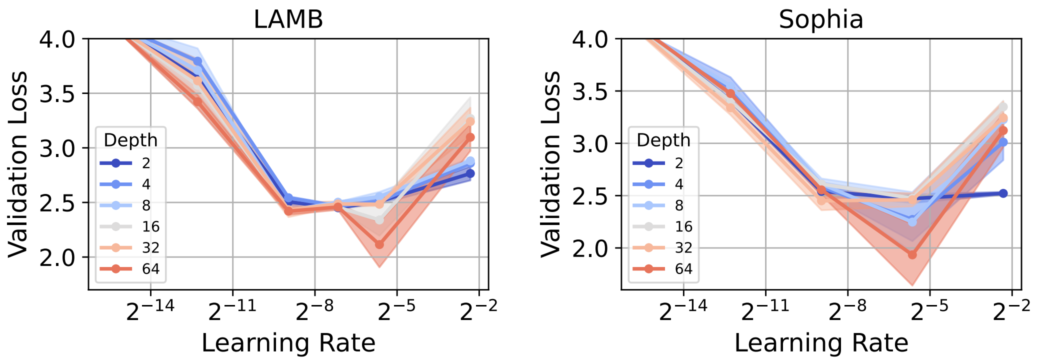

Figure 9: Mean validation loss for increasing model depth and different learning rates for LAMB (left) and Sophia (right).