Weight Decay may matter more than μP for LR Transfer in Practice

Atli Kosson[1,2†] **Jeremy Welborn**[1] **Yang Liu**[1] **Martin Jaggi**[2] **Xi Chen**[1]
[1]*Amazon FAR (Frontier AI & Robotics)*, [2]*EPFL*

Abstract

Transferring the optimal learning rate from small to large neural networks can enable efficient training at scales where hyperparameter tuning is otherwise prohibitively expensive. To this end, the Maximal Update Parameterization (μP) proposes a learning rate scaling designed to keep the update dynamics of internal representations stable across different model widths. However, the scaling rule of μP relies on strong assumptions, particularly about the alignment between the weights and updates of a layer and its inputs. We empirically show that in the practical setups where learning rate transfer is most valuable, such as LLM training, these assumptions hold only briefly at the start of training. For the remainder of training it is weight decay rather than μP that stabilizes the update dynamics of internal representations across widths, facilitating learning rate transfer. Instead, the learning rate scaling of μP acts as a form of learning rate warmup and can sometimes be replaced by one. Overall, this work fundamentally challenges prevailing beliefs about learning rate transfer and explains why μP requires the independent weight decay variant for successful transfer.

1. Introduction

The Maximal Update Parameterization (μP) has become a cornerstone of efficient training at scale, particularly for large language models (LLMs). Many open recipe LLMs employed variants of it [8, 10, 14, 17, 26] and similar learning rate (LR) scaling is likely used for other models with unknown training details. For example, llama4 uses an internal technique called MetaP, Grok-2 configuration files suggest μP use, and the GPT-4 report [1] cites a key μP paper coauthored by OpenAI staff [40].

Despite this success, several works [3, 5, 37, 39] have empirically found that learning rates only transfer well in practice when μP is combined with a specific approach to weight decay (WD), see Figure 1. We explain this phenomenon by showing how weight decay modulates the rate of feature learning, μP's main objective. Curiously, we find that independent weight decay effectively counteracts μP's LR scaling, which proves necessary since μP's key assumptions only hold very early on. This reframes the theory of LR transfer and reveals weight decay's central role in practice.

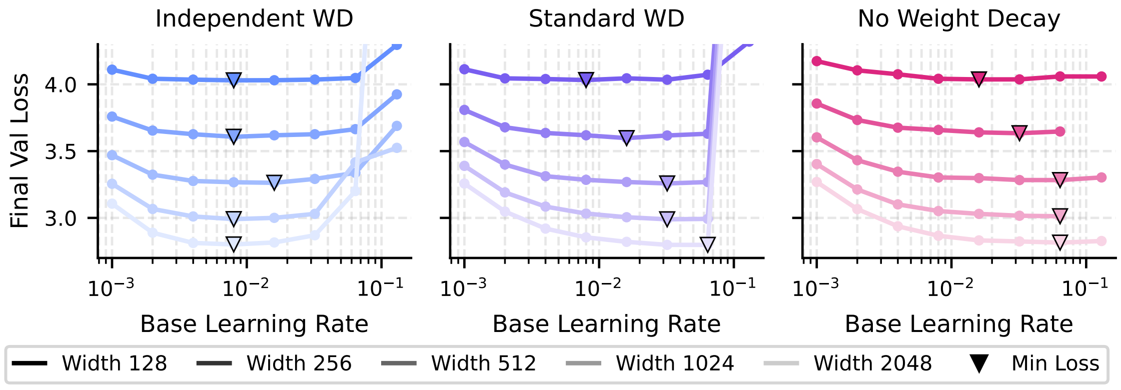

Figure 1: μP specifically requires independent weight decay for good learning rate transfer when training llama networks with AdamW. However independent weight decay scales the updates in a way that counteracts μP's LR scaling as discussed in §3. No WD is discussed in Appendix C.

† Work done while interning at Amazon FAR

2. A Common Framework for μP and Weight Decay

2.1. Single Layer Model and Representation Changes

The core μP and weight decay concepts discussed in this work can be understood from the linear transformation of a fully connected layer in isolation. For an input feature dimension C, output feature dimension K and batch size B, we define this as:

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X}, \qquad \boldsymbol{X} \in \mathbb{R}^{C\times B} \text{ (inputs)}, \quad \boldsymbol{W} \in \mathbb{R}^{K\times C} \text{ (weights)}, \quad \boldsymbol{Y} \in \mathbb{R}^{K\times B} \text{ (outputs)} \tag{1}$$

A weight update $\boldsymbol{W} \mapsto \boldsymbol{W} + \Delta\boldsymbol{W}$ causes the layer's outputs to change as $\boldsymbol{Y} \mapsto \boldsymbol{Y} + \Delta\boldsymbol{Y}$ for the same input $\boldsymbol{X}$. Note that $\|\Delta\boldsymbol{Y}\|$ is the *local* representation change from this layer alone, the *total* change would include potential changes $\Delta\boldsymbol{X}$ in the inputs arising from the updates of earlier layers. Local changes can be related to total changes [42], but we will focus on local ones for simplicity.

2.2. Alignment Relates Weight Magnitudes to Representation Sizes

Central to this work and μP in general is connecting the size of the *representation changes* $\|\Delta\boldsymbol{Y}\|$ to the size of weight updates $\|\Delta\boldsymbol{W}\|$, where $\|\cdot\|$ always denotes Frobenius norms. It is easy for an optimizer to give a specific weight update size $\|\Delta\boldsymbol{W}\|$, Adam [19] does this approximately and sign-based optimizers like Lion [6] can do this perfectly. However it is ultimately the size of the representation changes $\|\Delta\boldsymbol{Y}\|$ that determine the impact of an update. A weight update has no immediate impact without changing $\boldsymbol{Y}$, but alas, controlling $\|\Delta\boldsymbol{Y}\|$ is much harder than $\|\Delta\boldsymbol{W}\|$.

We describe the relationship between $\|\Delta\boldsymbol{Y}\|$ and $\|\Delta\boldsymbol{W}\|$ via the *update alignment*, defined as:

$$\alpha_{\Delta\boldsymbol{W}} := \frac{\|\Delta\boldsymbol{Y}\|}{\|\Delta\boldsymbol{W}\|\|\boldsymbol{X}\|} = \sqrt{\sum_{k,b} \frac{\|\Delta\boldsymbol{w}_k\|^2}{\|\Delta\boldsymbol{W}\|^2}\frac{\|\boldsymbol{x}_b\|^2}{\|\boldsymbol{X}\|^2}\cos^2\angle(\Delta\boldsymbol{w}_k, \boldsymbol{x}_b)} \in [0,1] \tag{2}$$

This is a weighted root-mean-square average of the cosine similarities between input samples $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_B]$ and rows in the weight update $\Delta\boldsymbol{W} = [\Delta\boldsymbol{w}_1, \ldots, \Delta\boldsymbol{w}_K]^\top$. Note that a high update alignment $\alpha_{\Delta\boldsymbol{W}}$ means that $\Delta\boldsymbol{w}_k$ and $\boldsymbol{x}_b$ tend to point in similar directions and results in larger representation changes $\|\Delta\boldsymbol{Y}\|$ for a given update size $\|\Delta\boldsymbol{W}\|$. We similarly define the *weight alignment* as $\alpha_{\boldsymbol{W}} := \frac{\|\boldsymbol{Y}\|}{\|\boldsymbol{W}\|\|\boldsymbol{X}\|}$ which relates the weight magnitude $\|\boldsymbol{W}\|$ to representation size $\|\boldsymbol{Y}\|$.

2.3. Weight Decay Modulates the Relative Size of Weight Updates

Weight decay is widely thought to mainly function as regularizer despite a long line of work that argues this is not the case in deep learning [7, 9, 21, 23–25, 34, 36, 44]. Instead it primarily acts as a secondary learning rate hyperparameter that modulates that size of relative weight updates $\|\Delta\boldsymbol{W}\|/\|\boldsymbol{W}\|$. Kosson et al. [21] describes these effects in AdamW [27] in detail[1]. Weight decay will cause the weight norms to converge to a fixed *equilibrium* value $\|\boldsymbol{W}\| \approx \sqrt{KC \cdot \eta/\lambda}$ for learning rate η, weight decay λ and matrix size $K \times C$. At this value the shrinking effect of weight decay and growth effect of gradient updates balance out. With AdamW's normalization of the weight updates, this results in $\|\Delta\boldsymbol{W}\|/\|\boldsymbol{W}\| \propto \sqrt{\eta\lambda}$. Crucially, the learning rate η and weight decay λ affect the relative updates in equilibrium the same way, only their product $\eta\lambda$ matters. In summary:

$$\|\boldsymbol{W}\| \propto \sqrt{KC\eta/\lambda}, \qquad \frac{\|\Delta\boldsymbol{W}\|}{\|\boldsymbol{W}\|} \propto \sqrt{\eta\lambda} \qquad \text{(equilibrium effect of weight decay)} \tag{3}$$

1. We will use the hyperparameter configuration of the PyTorch AdamW variant for the discussion, see Algorithm 1.

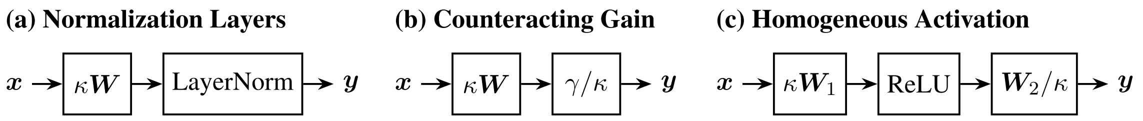

Figure 2: Neural networks are commonly invariant to certain types of rescaling transformations. Examples where applying an in-place scaling factor $\kappa > 0$ does not affect the current block output $\boldsymbol{y}$, making relative update sizes like $\|\Delta \boldsymbol{W}\|/\|\boldsymbol{W}\|$ more informative than absolute ones like $\|\Delta \boldsymbol{W}\|$.

Due to the structure of neural networks, relative changes are often more meaningful than absolute ones [4, 20, 21]. A given relative update size $\|\Delta \boldsymbol{W}\|/\|\boldsymbol{W}\|$ has the same impact across a number of equivalent parameter states obtained via transformations like those shown in Figure 2, unlike a fixed absolute update size $\|\Delta \boldsymbol{W}\|$. A similar argument extends to the representations, a given change $\Delta \boldsymbol{Y}$ typically has a greater impact when $\|\boldsymbol{Y}\|$ is small than when it is large. Based on this and the connection to weight decay, we focus on relative changes rather than absolute ones, deviating from existing work on μP. Note that at the start of training these approaches are equivalent since $\|\boldsymbol{W}\|$ and $\|\boldsymbol{Y}\|$ are roughly determined by the initialization, but later in training they can differ significantly.

2.4. Formulating μP in Terms of Relative Updates

The primary design goal of μP is to ensure stable and non-trivial feature learning at any width. Yang et al. [42] formulate this as $\|\boldsymbol{Y}\|_{\text{RMS}} := \|\boldsymbol{Y}\|_F/\sqrt{KB} = \Theta(1)$ and $\|\Delta \boldsymbol{Y}\|_{\text{RMS}} = \Theta(1)$, i.e. the representations and their changes neither explode nor vanish as the input dimension C grows.[2] In our relative framework we want to keep $\|\Delta \boldsymbol{Y}\|/\|\boldsymbol{Y}\|$ roughly constant across widths for a given learning rate. The hope is that a fixed relationship between learning rates and representation changes will result in a transfer of the optimal learning rate, but this is just a heuristic without guarantees. We can express the relative representation change in terms of the relative weight update as follows[3]:

$$\frac{\|\Delta \boldsymbol{Y}\|}{\|\boldsymbol{Y}\|} = \frac{\alpha_{\Delta \boldsymbol{W}}}{\alpha_{\boldsymbol{W}}} \frac{\|\Delta \boldsymbol{W}\|}{\|\boldsymbol{W}\|} \tag{4}$$

where we call $\alpha_{\Delta \boldsymbol{W}}/\alpha_{\boldsymbol{W}}$ the *alignment ratio*. To stabilize the relative representation changes $\|\Delta \boldsymbol{Y}\|/\|\boldsymbol{Y}\|$ across different widths C, μP must perfectly counter changes in the alignment ratio by scaling the weight updates appropriately (via the learning rate). This requires knowing how both the update alignment $\alpha_{\Delta \boldsymbol{W}}$ and weight alignment $\alpha_{\boldsymbol{W}}$ behave as a function of the network width C.

At the start of training the weights are randomly initialized. We can show that for a zero-mean and finite variance IID vector $\boldsymbol{w} \in \mathbb{R}^C$, independent of the finite inputs $\boldsymbol{x} \in \mathbb{R}^C$, we have $\mathbb{E}[\langle \boldsymbol{w}, \boldsymbol{x}\rangle^2] = \mathbb{E}[\|\boldsymbol{w}\|^2]\mathbb{E}[\|\boldsymbol{x}\|^2]/C$ resulting in $\alpha_{\boldsymbol{W}} \approx 1/\sqrt{C}$. This is analogous to the computation used to derive variance preserving initialization schemes. It also tells us that $\|\boldsymbol{W}\| \approx \sqrt{K}$ at the start of training, which notably does not depend on the input dimension C.

The update alignment is slightly more complicated since the update $\Delta \boldsymbol{W}$ is a function of the inputs $\boldsymbol{X}$ rather than independent, causing correlations that tend to increase the update alignment. Specifically the weight gradient is an outer product of the inputs $\boldsymbol{X}$ and the gradients with respect to $\boldsymbol{Y}$. For SGD at batch size one, this makes the update a scaled version of the input, giving perfect alignment $\alpha_{\Delta \boldsymbol{W}} = 1$ regardless of C. Based on this we get the core **μP alignment assumptions**:

$$\alpha_{\Delta \boldsymbol{W}} = \Theta(1), \qquad \alpha_{\boldsymbol{W}} = \Theta(1/\sqrt{C}), \qquad \frac{\alpha_{\Delta \boldsymbol{W}}}{\alpha_{\boldsymbol{W}}} = \Theta(\sqrt{C}) \tag{5}$$

2. Technically it is the total representation change across training that matters, but Yang et al. [42] show this can follow from bounded local representation changes with assumptions about the spectral norms of weights and updates.
3. We assume matrices are non-zero which is typically the case throughout training, barring special initialization schemes.

With this we can scale the learning rate to counteract changes in the alignment ratio in Equation 4. When the network width is multiplied $C \mapsto mC$, we get the following **μP-prescribed scaling**:[4]

$$\frac{\|\Delta \boldsymbol{W}\|}{\|\boldsymbol{W}\|} \propto \left(\frac{\alpha_{\Delta \boldsymbol{W}}}{\alpha_{\boldsymbol{W}}}\right)^{-1} \propto \frac{1}{\sqrt{m}} \text{ (relative update size)} \quad \Longrightarrow \quad \eta = \eta_{\text{base}}/m \text{ (Adam LR)} \tag{6}$$

The η scaling follows from the relative one based on $\|\Delta \boldsymbol{W}\| \propto \eta\sqrt{KC}$ for Adam and $\|\boldsymbol{W}\| \approx \sqrt{K}$. The optimal *base learning rate* η_{base} is what we aim to transfer across widths, tuning it at a small scale to save compute. Besides scaling the learning rate of hidden layers, μP involves special handling of the final output layer and attention normalization. Everett et al. [13] shows these are not necessary for learning rate transfer, combining standard parameterization with layer-wise scaling of the learning rate works equally well or better in practice. We adopt their simpler version in this work although we refer to the scaling as μP-prescribed or simply μP for brevity. See Appendix G.2 for details.

3. μP Requires Independent Weight Decay – A Surprising Contradiction

There are two common formulations of AdamW. In the original work by Loshchilov and Hutter [27], the rate of weight decay was determined by λ alone, i.e. at each step the weights are scaled by $1-\lambda$. However in the default implementation in PyTorch the weights are instead shrunk by $1-\eta\lambda$. These are equivalent since we can scale λ to get one from the other, but will behave differently when η is modified by μP. In this work we base all discussion on the PyTorch variant but consider the following two approaches for weight decay as μP scales the learning rate $\eta \mapsto \eta/m$:

$$(\eta, \lambda) \mapsto (\eta/m, \lambda) \qquad \text{(standard scaling)} \tag{7}$$

$$(\eta, \lambda) \mapsto (\eta/m, m\lambda) \qquad \text{(independent scaling)} \tag{8}$$

Note that independent scaling keeps the product $\eta\lambda$ constant, meaning the relative updates in equilibrium are not scaled at all (see Equation 3). This fundamentally differs from μP's prescribed scaling in Equation 6, meaning that **independent weight decay scaling contradicts μP.** On the other hand standard weight decay scaling results in equilibrium behavior that follows the relative update scaling prescribed by μP. This is surprising because multiple works have empirically found that independent weight decay gives better learning rate transfer [3, 5, 37, 39]. We replicate this finding for llama [33] training on DCLM [22] next token prediction in Figure 1. This suggest that **the μP-theoretical scaling of the relative update size does not work well in practice.**

4. Weight Decay Eventually Determines the Size of Relative Feature Updates – not μP

In Figure 3 we measure how well the different scaling approaches achieve our goal of stabilizing relative representation changes across widths. We find that independent scaling does this very well compared to standard scaling. Combined with Figure 1, this suggest that **maintaining relative representation changes is an effective goal for learning rate transfer, but μP relies on weight decay to achieve this later in training**. Figure 4 shows that **μP's alignment assumptions only hold very briefly at the start of training**. Rather than varying with the width as in Equation 5, the alignment ratio is width-independent for most of training. In this case the relative weight update should be kept constant as achieved via independent weight decay, explaining why it is needed. For the majority of training it is thus weight decay that governs the size of the relative feature updates, stabilizing their size across widths by directly counteracting the effects of μP's learning rate scaling.

4. This relative update scaling is consistent with prior works, see Appendix B.3 of [40] or Adafactor in Table 1 of [13].

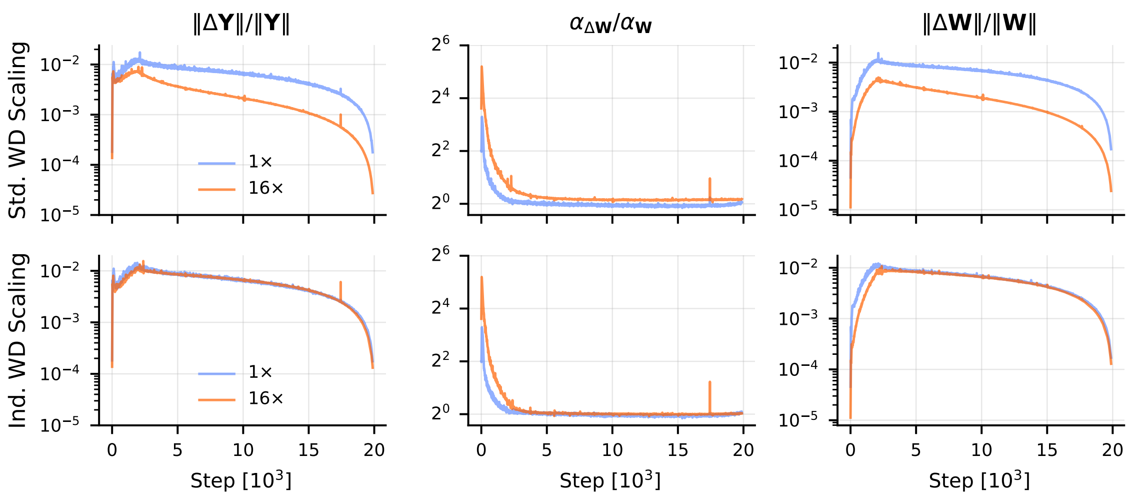

Figure 3: Standard μP scaling fails to maintain relative representation changes across widths. Comparing the relative representation change (RRC) for μP with standard weight decay scaling (first row) and independent weight decay scaling (second row) for llama training with a 10% linear warmup followed by linear decay. Plots shown for a single representative layer with $C = 128$ at $1\times$ width, see G.3.2 for details. **Left:** Standard scaling does not preserve the RRC for the same learning rate unlike independent scaling. Recall that the RRC is the product of the alignment ratio and the relative weight update (Equation 4). **Middle:** The alignment ratio is similar for both widths and approximately 1. **Right:** To maintain the RRC across widths the relative weight update size should then be kept constant as achieved by independent weight decay rather than scaled as μP prescribes.

5. Why do the Alignment Assumptions of μP Break Down in Practice?

5.1. Update Alignment

We experimentally observed that the update alignment varies with the width C, contrary to μP's assumptions. In Appendix E we show that **the update alignment assumptions of μP can break down when the batch size B is large compared to the input dimension C.** This is frequently the case in practice but does not happen in the infinite width setting μP was originally meant to target.

For the conceptual reason behind the width-dependence, consider the case of SGD for a single neuron, i.e., $K = 1$. We assume the input samples $\mathbf{x}_b$, the output gradients y'_b, and thus the gradient for each sample, $\mathbf{g}_b = \mathbf{x}_b \mathrm{y}'_b$, are all zero-mean IID random and mutually independent. The weight update then becomes $\Delta\mathbf{w} = -\eta\frac{1}{B}\sum_b \mathbf{g}_b$ and each element of the output change $\Delta\mathbf{y}$ has the form:

$$\Delta \mathrm{y}_i = \langle \Delta\mathbf{w}, \mathbf{x}_i \rangle = -\eta\tfrac{1}{B}\mathrm{y}_i\langle \mathbf{x}_i, \mathbf{x}_i\rangle + -\eta\tfrac{1}{B}\textstyle\sum_{b\neq i}\mathrm{y}_b\langle \mathbf{x}_b, \mathbf{x}_i\rangle \tag{9}$$

The output change for each input therefore involves one *self-contribution* term and $B-1$ *interference* terms from other samples. Each interference term has random alignment, making them $1/\sqrt{C}$ times smaller in expectation than for the full alignment that occurs in the self-contribution term. However, the $B-1$ interference terms add up randomly, giving an expected amplification of roughly $\sqrt{B}$ in the total interference. The interference dominates when $B \gg C$, giving an overall dependence on C. For $B \ll C$, the interference becomes negligible, making the update alignment invariant to C.

In practice the gradients are not uncorrelated. High initial correlation results in large update alignment that looks similar to the small batch-to-width setting that μP targets. This correlation decreases over time, resulting in a width dependence that violates μP's assumptions, see Appendix E.

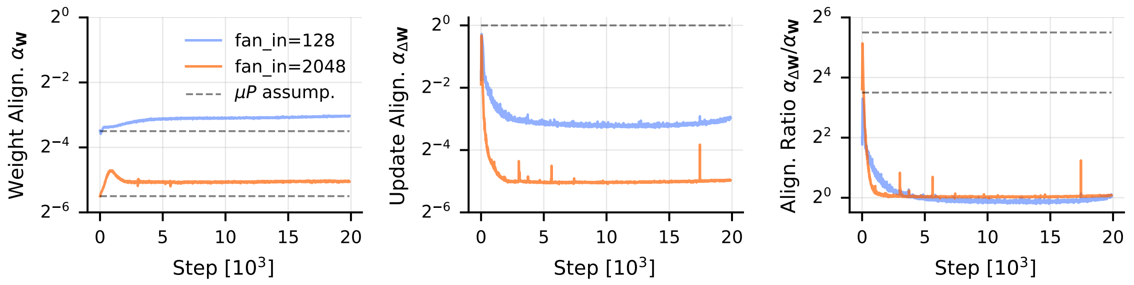

Figure 4: The alignment assumptions of μP do not hold in practice. Measurements of the weight alignment, the update alignment and the alignment ratio for llama training at different widths. The same hyperparameters are used for both scales, without any μP scaling. The update alignment varies significantly over time, becoming width-dependent. This violates μP's assumption which only holds very early in training. Plots shown for one representative MLP layer.

5.2. Weight Alignment

The initial weight alignment matches μP's assumptions, but analyzing it later in training is hard. Intuitively, we note that we can decompose the weights $\boldsymbol{W}_t$ at time t into weight updates $\Delta\boldsymbol{W}_\tau$ from prior time steps τ, where we assume $\Delta\boldsymbol{W}$ does not include the weight decay term (if used):

$$\boldsymbol{W}_t = (1-\eta\lambda)^t \cdot \boldsymbol{W}_0 + \textstyle\sum_{\tau=0}^{t-1}(1-\eta\lambda)^{t-\tau} \cdot \Delta\boldsymbol{W}_\tau \qquad \text{(update composition of weights)} \quad (10)$$

If the decay rate $\eta\lambda$ is large or $\|\boldsymbol{W}_0\|$ is small compared to the updates, the $\boldsymbol{W}_0$ term quickly becomes insignificant. Afterwards, the weight alignment at time t is determined primarily by the alignment of $\boldsymbol{X}_t$ and recent update terms $\Delta\boldsymbol{W}_\tau$. If this alignment is similar to the update alignment, one might reasonably expect the overall weight alignment to approximate the update alignment, resulting in an alignment ratio $\alpha_{\Delta\boldsymbol{W}}/\alpha_{\boldsymbol{W}} \approx 1$ as we often observe in practice (see Appendix H, Figure 16).

6. With Independent Weight Decay μP can act as Additional Learning Rate Warmup

We have shown that for the majority of training weight decay aids learning rate transfer by effectively counteracting μP's learning rate scaling. However in the very early phases of training μP still has an impact by decreasing the relative size of early updates. Overall this has an effect similar to a learning rate warmup. We measure and analyze this effect in Appendix B. We find that it can sometimes have benefits that are surprisingly hard to obtain via standard learning rate warmup strategies.

7. Discussion & Conclusion

We have shown that in practice μP's learning rate scaling is only relevant at the start of training due to alignment assumptions that quickly break down. Instead weight decay seems to be the main mechanism that stabilizes representation changes across widths, thus facilitating leaning rate transfer via μP's core design objective. However it curiously achieves this by effectively undoing the effects of μP's scaling for the majority of training. These findings challenge the conventional view on learning rate transfer and the training of large networks. For example it suggests wide networks can be trained with similarly sized updates as narrower ones after the initial phases of training.

In Appendix D we show that our findings generalize to ResNet training on ImageNet. Disabling WD interestingly results in alignment and relative update behavior similar to independent WD, see Appendix C. However, we note that significantly different update alignment dynamics can occur with matrix-level optimizers. For example, Muon [18] prevents the update alignment from changing over time, making it a promising alternative to μP for effectively controlling feature learning in practice.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rkxQ-nA9FX. arXiv:1812.03981.

[3] Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in llm pre-training. *arXiv preprint arXiv:2505.13738*, 2025.

[4] Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. *Advances in Neural Information Processing Systems*, 33:21370–21381, 2020. arXiv:2002.03432.

[5] 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. arXiv:2407.17465.

[6] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. Symbolic discovery of optimization algorithms. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=ne6zeqLFCZ. arXiv:2302.06675.

[7] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. arXiv:1905.05894.

[8] Team Cohere, Arash Ahmadian, Marwan Ahmed, Jay Alammar, Milad Alizadeh, Yazeed Alnumay, Sophia Althammer, Arkady Arkhangorodsky, Viraat Aryabumi, Dennis Aumiller, et al. Command a: An enterprise-ready large language model. *arXiv preprint arXiv:2504.00698*, 2025.

[9] Francesco D'Angelo, Maksym Andriushchenko, Aditya Varre, and Nicolas Flammarion. Why do we need weight decay in modern deep learning? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=YrAxxscKM2. arXiv:2310.04415.

[10] Nolan Dey, Gurpreet Gosal, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, Joel Hestness, et al. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023.

[11] Nolan Dey, Quentin Anthony, and Joel Hestness. The practitioner's guide to the maximal update parameterization, 2024. URL https://www.cerebras.ai/blog/the-practitioners-guide-to-the-maximal-update-parameterization.

[12] 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.

[13] Katie E 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. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=0ksNeD1SJT. arXiv:2407.05872.

[14] Falcon-LLM Team. Falcon-h1: A family of hybrid-head language models redefining efficiency and performance, May 2025. URL https://falcon-lm.github.io/blog/falcon-h1.

[15] Moritz Haas, Sebastian Bordt, Ulrike von Luxburg, and Leena Chennuru Vankadara. On the surprising effectiveness of large learning rates under standard width scaling. *arXiv preprint arXiv:2505.22491*, 2025.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. arXiv:1512.03385.

[17] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

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

[19] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. arXiv:1412.6980.

[20] Atli Kosson, Bettina Messmer, and Martin Jaggi. Analyzing & reducing the need for learning rate warmup in GPT training. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=ZgDNrpS46k. arXiv:2410.23922.

[21] Atli Kosson, Bettina Messmer, and Martin Jaggi. Rotational equilibrium: How weight decay balances learning across neural networks. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=MQirNNU2pC. arXiv:2305.17212.

[22] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas

Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Kamal Mohamed Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Joshua P Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander T Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-LM: In search of the next generation of training sets for language models. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL `https://openreview.net/forum?id=CNWdWn47IE`. arXiv:2406.11794.

[23] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rJg8TeSFDH`. arXiv:1910.07454.

[24] Zhiyuan Li, Kaifeng Lyu, and Sanjeev Arora. Reconciling modern deep learning with traditional optimization analyses: The intrinsic learning rate. *Advances in Neural Information Processing Systems*, 33:14544–14555, 2020. arXiv:2010.02916.

[25] Zhiyuan Li, Srinadh Bhojanapalli, Manzil Zaheer, Sashank Reddi, and Sanjiv Kumar. Robust training of neural networks using scale invariant architectures. In *International Conference on Machine Learning*, pages 12656–12684. PMLR, 2022. arXiv:2202.00980.

[26] Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, et al. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*, 2023.

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`. arXiv:1711.05101.

[28] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[29] Christian H.X. Ali Mehmeti-Göpel and Michael Wand. On the weight dynamics of deep normalized networks. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=AzUCfhJ9Bs`. arXiv:2306.00700.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. arXiv:1912.01703.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. arXiv:1409.0575.

[32] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/shazeer18a.html. arXiv:1804.04235.

[33] 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. URL https://arxiv.org/abs/2302.13971.

[34] Twan Van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

[35] Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

[36] Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. Spherical motion dynamics: Learning dynamics of normalized neural network using sgd and weight decay. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 6380–6391. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/326a8c055c0d04f5b06544665d8bb3ea-Paper.pdf. arXiv:2006.08419.

[37] Xi Wang and Laurence Aitchison. How to set adamw's weight decay as you scale model and dataset size. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=IszVnczhfz. arXiv:2405.13698.

[38] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[39] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie E Everett, Alexander A Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ. arXiv:2309.14322.

[40] Ge 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. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17084–17097. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/8df7c2e3c3c3be098ef7b382bd2c37ba-Paper.pdf. arXiv:2203.03466.

[41] Greg Yang and Edward J. Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International*

Conference on Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/yang21c.html. arXiv:2011.14522.

[42] Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning. *ArXiv*, abs/2310.17813, 2023. URL https://api.semanticscholar.org/CorpusID:264555180. arXiv:2310.17813.

[43] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: Feature learning in infinite depth neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=17pVDnpwwl. arXiv:2310.02244.

[44] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1lz-3Rct7. arXiv:1810.12281.

Appendix A. Related Work

Adam [19] is a widely used optimizer in deep learning. It was originally used with ℓ_2-regularization but Loshchilov and Hutter [27] empirically found that decoupled weight decay works better, resulting in AdamW. The version of AdamW we use in this paper is shown in Algorithm 1 and corresponds to the most popular variant and is the default in PyTorch [30].

Weight decay is unfortunately still widely believed to act primarily as an explicit regularizer despite a long line of work that explains this is not the case in deep learning [7, 9, 21, 23–25, 34, 36, 44]. Instead weight decay modules some notion of an "effective" learning rate, which can be seen as measures of the relative update size. Our work builds closely upon Kosson et al. [21], who described the equilibrium dynamics of weight decay in detail for AdamW. This included originally noting the scheduling effects, how weight decay can be replaced by controlling the relative weight update, and how the balanced equilibrium behavior of AdamW explains its effectiveness over Adam with ℓ_2-regularization.

Yang and Hu [41] proposed the Maximal Update Parameterization μP as a way to ensure stable feature learning in infinitely wide networks. Yang et al. [40] demonstrated how μP allows transferring the learning rates across different network widths and how this beneficial for hyperparameter tuning. Later works have expanded μP to network depth rather than only width [12, 43]. The original formulation of μP is quite theoretical but Yang et al. [42] show a simplified and more accessible derivation based on spectral norms which we base our discussion on. Dey et al. [11] provides a useful overview of μP.

Everett et al. [13] shows hyperparameter transfer is possible for other parameterizations than μP, including standard initialization schemes without the special multipliers used in μP, using analogous learning rate scaling. We adopt this approach in our experiments as it is simpler and results in better performance according to Everett et al. [13]. They also describe how the learning rate scaling depends on alignment metrics analogous to those we define in Equation 4, but use the total weight change from the start of training rather than a single optimization step. Our findings fit particularly well with one case they consider, the no-alignment setting. Their analysis of AdaFactor [32] with weight-proportional updates in this case predicts that learning rates should transfer without any scaling. This is similar to what we observe in practice, for both LionAR and standard training in rotational equilibrium, updates are proportional to the weights. We expand upon their work with measurements of the update-alignment, describing the role of weight decay, and how the batch size affects alignment.

Haas et al. [15] is very recent work that investigates why higher learning rates than μP prescribes can be used. They investigate alignment changes as a potential cause but dismiss them in favor of the role of the loss function, showing that typical cross entropy losses help stabilize the final layer compared to ℓ_2 based losses. Our alignment measure is based on the current update rather than the total update across training, which might explain the different conclusions.

Wang and Aitchison [37] empirically demonstrated that independent scaling is necessary for learning rate transfer and is the only work we are aware of that explores why this is the case. Their analysis revolves around approximating AdamW as an exponential moving average (EMA) over past updates, which has a form similar to the geometric sum in Equation 10. From this perspective the decay rate defines a time scale for the EMA (analogous to a half-life) which is an alternative characterization of the relative update size in equilibrium. They argue the time scale should intuitively remain constant across networks sizes in order for the final network parameters to form a similar average over different data points, but do not justify this formally. We note that although the timescale interpretation can be a useful mental model and approximation, it does not account for the dependency of later updates upon earlier ones. This dependency is exactly what prevents optimization from being performed in a single step by simply taking an appropriate weighted average over all the data. We take a different approach by relating weight decay directly to the goals of μP through the rate of representation changes.

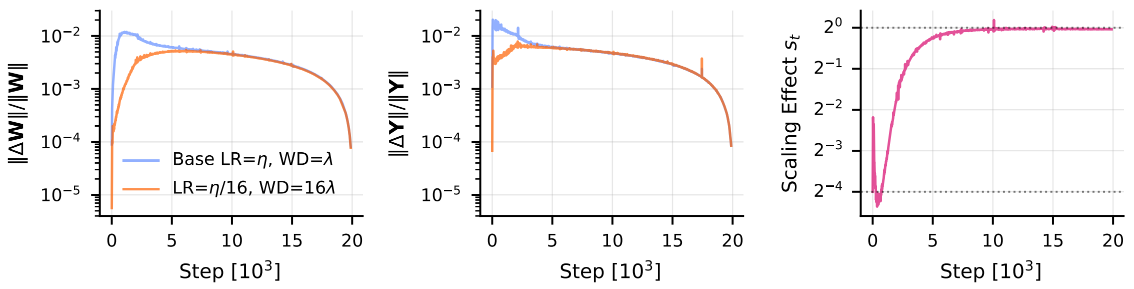

Figure 5: μP with independent WD scaling has a warmup-like effect on relative updates. Measurements of the relative weight update (left) and relative representation change (middle) for different learning rate (LR) and weight decay (WD) combinations in a llama layer with 2048 input features. Independent weight decay scaling achieves a given LR-WD product using a lower LR and higher WD (orange) compared to not performing μP scaling at all (blue) or equivalently via μP with standard WD scaling (at a higher shifted base LR). For a given LR-WD product, low-LR high-WD configurations result in smaller relative updates early in training but not later, similar to using a stronger learning rate warmup. The right panel shows the ratio of the relative weight updates for the runs, which goes from 1/16 to asymptotically approaching 1 in a roughly exponential manner, similar to an additional warmup factor. See G.3.4 for experimental details.

Appendix B. μP Can Act as Additional Learning Rate Warmup

Based on the order one alignment ratios that we find to dominate practical training, the relative weight updates should be kept roughly constant across widths. Kosson et al. [21] described how the magnitude of the relative updates in the steady-state only depends on the product of the learning rate η and weight decay λ rather than their individual values. However, (η, λ) pairs with the same product generally affect the initial phases of training differently. Specifically, high weight decay pairs like those obtained from independent μP scaling via $(\eta, \lambda) \mapsto (\eta/m, m\lambda)$ have a warmup-like effect, where the relative updates at the start of training are comparatively smaller. Therefore, **μP acts as a special form of additional learning rate warmup when used with independent WD scaling.** We measure this effect empirically for llama training in Figure 5, confirming the general effect is still present despite the lack of the perfect scale-invariance typically assumed by works on weight decay.

B.1. Simple Analytical Model

In Appendix F we analyze a simple idealized setting where the gradient updates are orthogonal to the weights and have a fixed RMS value of one. This lets us model the RMS norm ρ_t of the weight matrices as:

$$\rho_{t+1}^2 = a_t^2\rho_t^2 + \eta_t^2, \qquad a := 1 - \eta\lambda \qquad \text{(simple RMS model)} \quad (11)$$

For a constant learning rate $\eta_t = \eta$ the weight RMS exponentially approaches a stable equilibrium value ρ_∞ which also determines the average relative weight update η_t/ρ_t:

$$\rho_t^2 = \rho_\infty^2 + (\rho_0^2 - \rho_\infty^2)a^{2t}, \quad \rho_\infty^2 = \frac{\eta^2}{1-a^2} \approx \frac{\eta}{2\lambda}, \quad \frac{\eta}{\rho_\infty} \approx \sqrt{2\eta\lambda} \quad \text{(simple equilibrium)} \quad (12)$$

The warmup effect is the ratio of the relative weight updates for the high weight decay configuration and the standard configuration. In the simplified setting this becomes:

$$s_t = \sqrt{\frac{1 + (\rho_0^2/\rho_\infty^2 - 1)a^{2t}}{1 + (m^2\rho_0^2/\rho_\infty^2 - 1)a^{2t}}} \underset{\text{if } \rho_0=\rho_\infty}{=} \frac{1}{\sqrt{1 + (m^2 - 1)a^{2t}}} \qquad \text{(simple warmup effect)} \quad (13)$$

Note how this factor goes from $1/m$ to 1 over time as we experimentally observed in Figure 5. However the overall behavior is complex and depends on the specific values of the learning rate η, weight decay λ and initialization value ρ_0. In the special case where $\rho_0 = \rho_\infty$ the behavior simplifies to a roughly exponential approach towards 1 rather than the ratio of two such functions.

The length of the warmup effect depends on the learning rate and weight decay through a as well as the ratio ρ_0/ρ_∞. The shape of s_t can thus greatly vary between different hyperparameter configurations, even within a standard learning rate sweep. In practice deviations from the simple RMS model prevent us from accurately predicting s_t. We believe the two biggest sources of error are how temporal gradient correlations interact with momentum and potentially non-orthogonal gradients from a lack of normalization layers.

Finally we note that μP's learning rate scaling in AdamW is used to both counteract assumed changes in the alignment ratio and the size of the relative updates. Recall that the learning rate is scaled $\eta \propto 1/m$ whereas the relative updates should only scale with $1/\sqrt{m}$. Even without any assumptions about the alignment ratio, it would make sense to scale the learning rate and weight decay $(\eta, \lambda) \mapsto (\eta/\sqrt{m}, \sqrt{m} \cdot \lambda)$. This scales $\rho_\infty \propto 1/\sqrt{m}$, matching how variance preserving initialization scales $\rho_0 \propto 1/\sqrt{m}$. Such scaling preserves both a and ρ_0/ρ_∞ in Equation 13, and is thus more likely to preserve the relative weight update behavior across training. In comparison independent μP scaling has an additional warmup effect of $1/\sqrt{m}$ in the relative updates (which is desirable), but also decreases $\rho_\infty \propto 1/m$. These smaller weight norms could potentially cause signal propagation issues if there are not learnable gains or normalization layers to counteract them.

B.2. Does Standard Learning Rate Warmup Suffice?

It is natural to ask whether the additional warmup effect is needed in practice. After all typical training often includes a learning rate warmup already, especially transformer training with AdamW. In the first two panels of Figure 6 we explore replacing the hyperparameter scaling of μP with a longer warmup for llama training. Somewhat surprisingly, we find that even very long linear warmup does not give the same benefits for learning rate transfer or even in terms of stabilizing higher learning rates. In Appendix D we repeat this experiment for ResNet training, finding the additional warmup effect is not needed there. This could be a part of the reason μP like learning rate scaling did not appear until transformer training became popular, well-normalized convolutional networks may simply not need it at reasonable scales.

We experiment with modified warmup schedules to see if they can replicate the benefits of independent scaling for transformers. The alignment ratios do start out high in practice, warranting the $1/m \to 1$ scaling from s_t. This could be done through a directly scaling the existing learning rate schedule (including the linear warmup) by an exponentially increasing factor of the form:

$$\hat{\eta}_t = \eta_t \cdot m^{\min(0,\, t/T_W - 1)} \qquad \text{(exp-increasing warmup)} \quad (14)$$

where T_W is the length of the exponential warmup. Alternatively we could take inspiration from the simpler $\rho_0 = \rho_\infty$ case of Equation 13. We note that although this might be a useful starting point,

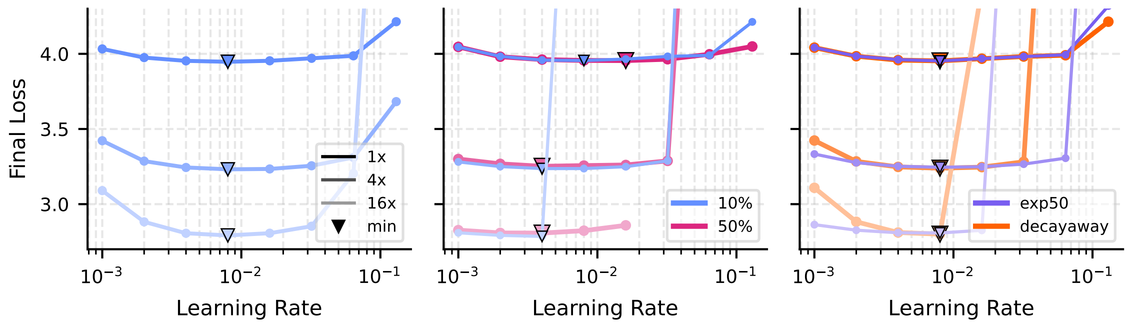

Figure 6: Stronger warmup-schedules may replicate the benefit of μP with independent scaling. Learning rate transfer plots for llama training using different warmup approaches (details in Appendix G.3.5). **Left:** μP with independent scaling transfers well. **Middle:** No μP scaling with either the base 10% linear warmup or a longer 50% warmup. These work relatively poorly in this case. **Right:** No μP scaling with additional warmup incorporating the width scaling factor m can give good transfer, but generally requires some tuning. Additional 50% exponential warmup (Equation 14) compared to a decay-away warmup (Equation 15).

scaling the learning rate schedule by s_t does not result in an exact scaling of the relative weight update by s_t since the behavior of the weight RMS ρ_t is also affected. This results in a form of warmup the scaling factor decays over time:

$$\hat{\eta}_t = \eta_t \cdot \left(1 + (m^2 - 1)\prod_{\tau=0}^{t-1}(1 - \eta_\tau \lambda)^2\right)^{-1/2} \qquad \text{(decay-away warmup)} \quad (15)$$

Both types of additional warmup factors are shown in Figure 7.

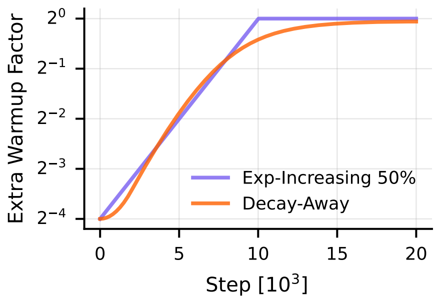

Figure 7: Additional warmup factors applied on top of a linear decay schedule with 10% linear warmup for $m = 16$, $\eta = 0.004$, and $\lambda = 0.1$, matching the minimum in the right panel of Figure 6.

The final panel of Figure 6 shows the effect of applying these additional warmup factors on top of the existing 10% linear warmup. This works well, offering better stability and learning rate transfer than with the linear schedules suggesting: **Additional warmup can at least sometimes replace μP's learning rate scaling in practice.** However the warmup effect from independent scaling still seems to give better stability. We also note that this is applied to all parameters of the network unlike the effect of the independent scaling which only affects certain layers.

In general the additional warmup factors likely require tuning which might partially defeat the purpose of hyperparameter transfer by requiring repeated experiments at scale. We believe the warmup effect from independent μP scaling may have similar limitations although it seems to be a good heuristic. The key to its effectiveness may be that the length of the warmup-effect depends on the rate at which the weights are updated. This rate likely affects both the time it takes to reach equilibrium and the time it takes for the alignment ratio to stabilize. However this also means that the effective length of the warmup effect changes between different points in a learning rate sweep. For example, note how quickly the performance of both the decay-away schedule and independent scaling drops as the learning rate is lowered compared to other schedules in Figure 6. Although these lower learning rates generally require less warmup for stability, they actually get a longer warmup-effect from independent weight decay scaling. The longer warmup could potentially give better transfer of the relative representation changes if the alignment ratio takes longer to stabilize, but is likely excessive causing the comparatively fast loss of performance. This could contribute to the perceived stability of the learning rate transfer by making lower rates perform worse.

The equilibrium dynamics of weight decay are a confounding factor in the design of both warmup schedules and learning rate schedules in general. One potential solution would be to use relative or rotational optimizers which control the relative update size directly rather than relying on the effects of weight decay. This would allow the exact application of a scaling factor in the relative updates like s_t by simply scaling the learning rate schedule.

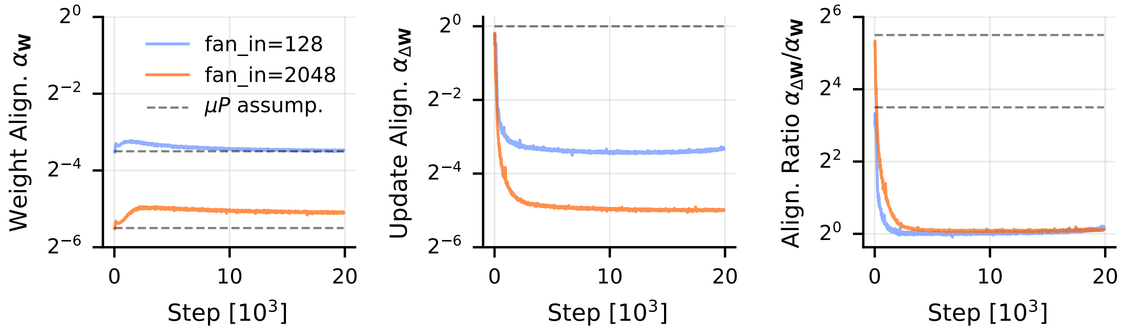

Figure 8: The alignment assumptions of μP break down even without weight decay. Measurements of the weight alignment, the update alignment and the alignment ratio for llama training at different widths. The update alignment varies significantly over time, becoming width-dependent. This violates μP's assumption which only holds very early in training. A base learning rate of $\eta = 1.6 \cdot 10^{-2}$ is used for the narrower network and scaled down $16\times$ for the wider network via μP's learning rate scaling. Plots shown for one representative MLP layer, `layers.13.feed_forward.w1`.

Appendix C. μP's Alignment Assumptions Break Down Even Without Weight Decay

In Figure 1 we observed that no weight decay seems to provide better learning rate transfer than the use of standard weight decay, although it is still not as good as with independent weight decay. The final loss is also slightly worse than the best configuration with either weight decay approach. In this section we explore whether the assumptions of μP hold without weight decay.

In Figure 8 we see that this is not the case. The alignment ratio still becomes roughly one independent of the width, meaning that the alignment assumptions of μP break down exactly like in the weight decay case. **Weight decay is therefore not the cause of μP's violated assumptions.** This matches our analysis of the update alignment where the width dependence at large batch sizes does not depend on weight decay directly. With an alignment ratio that does not vary across width, the relative updates should be preserved in order to keep the relative representation changes comparable.

In Figure 9 we measure how the relative updates behave across different learning rates. Notably, **without weight decay the peak learning rate has little effect on the size of relative updates later in training**. This could explain why the no weight decay setting seems to be less sensitive to the specific value of the learning rate in Figure 1. Without weight decay the size of the relative updates falls over time compared to when weight decay is used, similar to an extra decay factor in the learning rate schedule. The way that this decay occurs seems to cause the relative weight updates to lose their dependence on the peak learning rate. In Appendix F.2 we show this can happen in a simplified setting for constant learning rates. The conceptual reason is that if the total weight growth dominates the initial value, both the update and the total weight norm have the same proportional dependency on the learning rate later in training. These factors cancel out resulting in roughly identical relative weight updates across peak learning rates. This assumes that the network behaves similar to a scale-invariant network, i.e., that there is no strong gradient signal that affects the weight norms. A similar effect of the initial learning rate losing its relevance over time has been observed for SGD and normalization layers before [2, 29].

The optimal learning rate likely becomes a balance between two factors, meaningful learning and stability. Higher learning rates decrease the contribution of the initial weights to the final model, which may allow the model to better learn the distribution. However, high learning rates can also

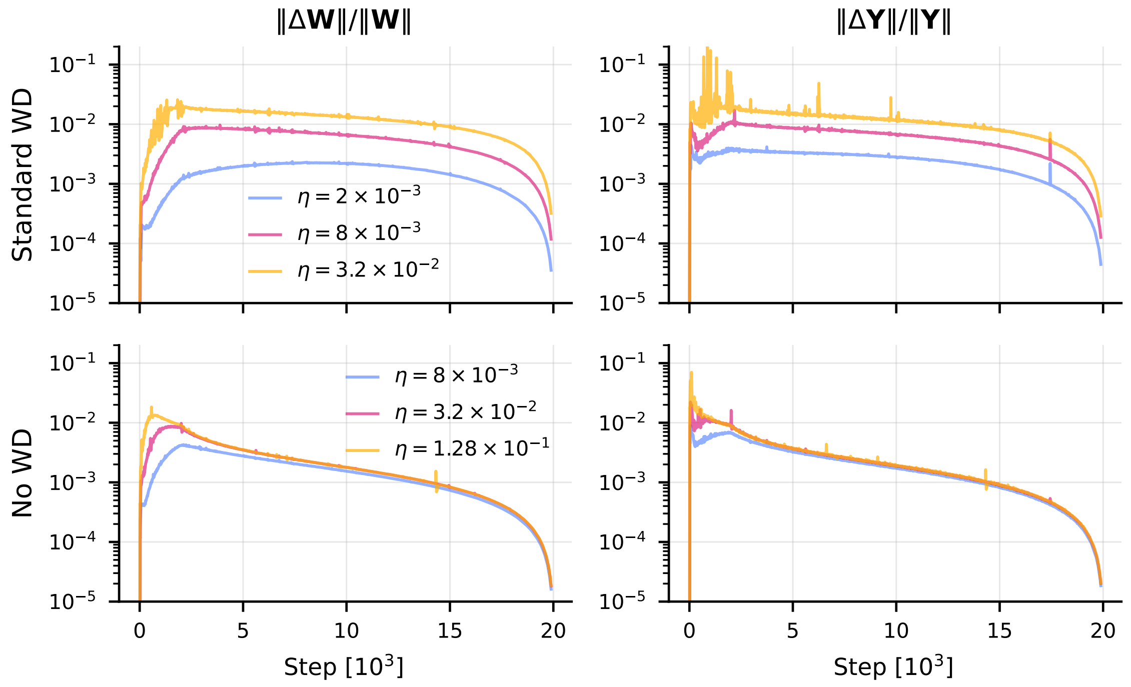

Figure 9: Without weight decay relative updates become similar across learning rates. Measurements of the relative weight update size and relative representation change for llama training. Plots shown for one representative MLP layer, `layers.13.feed_forward.w1`.

cause large relative updates early in training that can have detrimental effects as explored by Kosson et al. [20], perhaps by saturating non-linearities or causing instability. If these detrimental effects occur sufficiently early, the alignment assumptions of μP may still roughly hold. In this case the optimal learning rate may be decided primarily by early behavior which may aid optimal learning rate transfer under μP's scaling, despite its core assumptions only holding briefly at the start of training. We suspect the length of the learning rate warmup may also impact whether the key limiting stability period occurs when μP's alignment assumptions still hold or not.

Overall the reason for learning rate transfer when it occurs would then be similar as in the independent weight decay case. Early in training μP correctly scales down the relative updates which helps transfer relative representation changes under the high, width-dependent, initial alignment ratios. Later in training the relative updates become approximately the same across widths which helps stabilize the rate of feature learning when the alignment ratios are identical across widths. Only standard weight decay follows μP's prescribed relative weight update scaling from Equation 6 throughout training, which is exactly why learning rates fail to transfer with it. With either no weight decay or independent weight decay, the relative updates deviate from the theoretical scaling which helps counteract the violated alignment assumptions. However, independent weight decay controls the relative updates later in training more tightly, resulting in better transfer overall.

Appendix D. ResNet Experiments

In this section we repeat some of our experiments for ResNet training to see if they generalize to other architectures and settings. The training setup is described in Appendix G.4 Overall our main conclusions from the llama experiments hold well. The main differences are that the learning rate transfer is worse, likely due to regularization effects and that the warmup effects seem to matter less. The results are shown in the following figures.

- Figure 10 compares learning rate transfer between standard and independent weight decay scaling. Independent weight decay gives better transfer but there is a shift in the optimal learning rates for both scaling approaches.
- Figure 11 shows that independent weight decay scaling helps preserve relative representation changes. The behavior is essentially identical to the llama setting.
- Figure 12 measures the weight alignment, update alignment and alignment ratio. Just like in the llama setting μP's assumptions only hold for a very brief period at the start of training. The weight alignment changes more than we observed with llama.
- Figure 13 empirically measures the warmup effect from the use of high weight decay. The conclusions are identical to the llama setting.
- Figure 14 compares learning rate transfer between μP with independent learning rate scaling and not applying μP scaling. We find that both learning rate transfer quality and stability are similar. The μP configuration results in a slightly higher accuracy which may at least in part be due to the warmup effect.

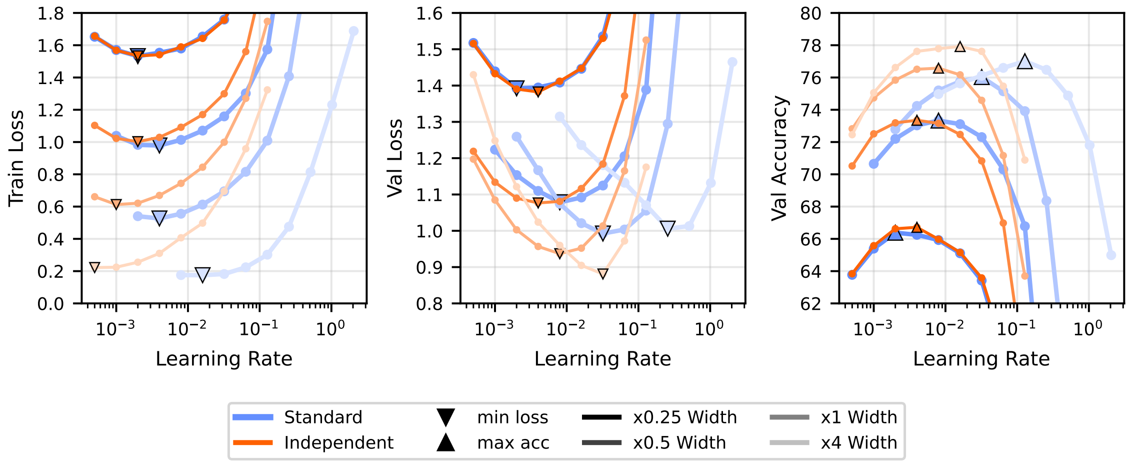

Figure 10: Comparison of standard vs independent μP scaling for ResNet training on ImageNet. Learning rates generally transfer much worse than for llama. Independent scaling still gives better results than standard scaling overall. Note how optimal learning rates in terms of training loss shift downwards with independent scaling while they shift upwards for the validation loss and accuracy, a strong indication of a confounding regularization effect. Widths are specified relative to a standard ResNet-50 but the base width for scaling is the $0.25\times$ variant. Learning rates for parameters that do not scale with width are frozen before the sweep at the optimal value for the $0.25\times$ network.

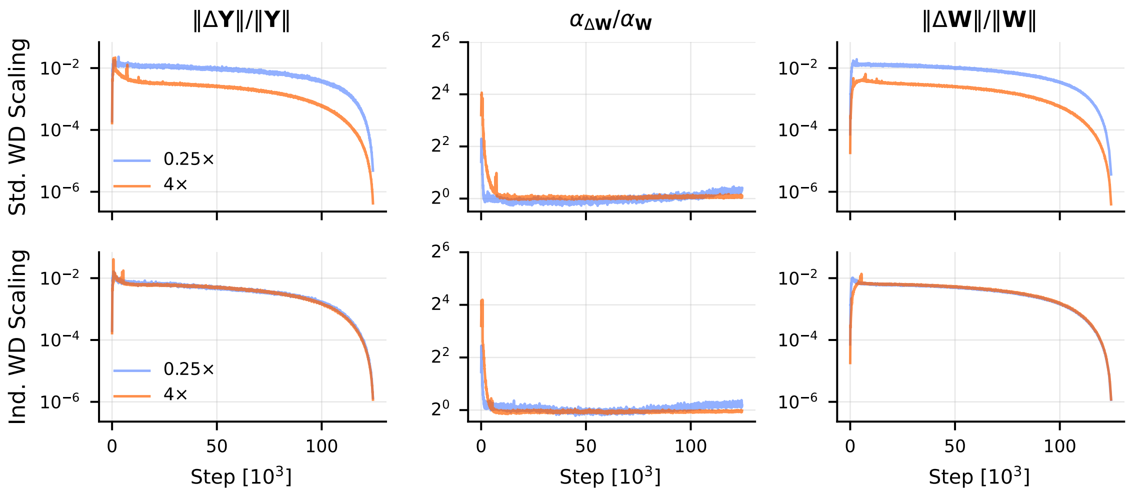

Figure 11: ResNet variant of Figure 3. Similar to the llama case, standard weight decay scaling fails to preserve the relative representation change across widths. The alignment ratio is similar across widths contrary to μP's assumptions, requiring maintaining relative weight updates as achieved by independent weight decay scaling. Plots show for `layer3.2.conv1` which is a 1×1 convolutional layer with `fan_in=4096` for the $4\times$ variant. The learning rate used here is $\eta = 4 \times 10^{-3}$ with a 1 epoch warmup (1%) followed by cosine decay to zero.

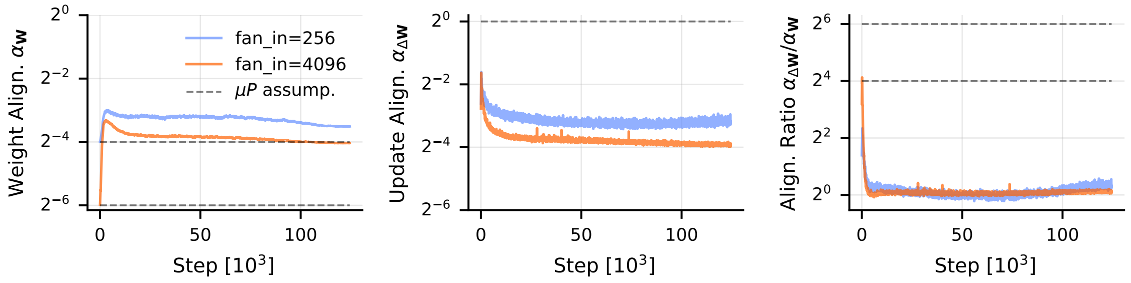

Figure 12: ResNet variant of Figure 4. Plots show for `layer3.2.conv1` which is a 1×1 convolutional layer. Both the weight alignment and update alignment deviate significantly from μP's assumptions. Note the larger changes in the weight alignment over time compared to the llama experiment. Learning rate $\eta = 4 \times 10^{-3}$, no μP scaling applied.

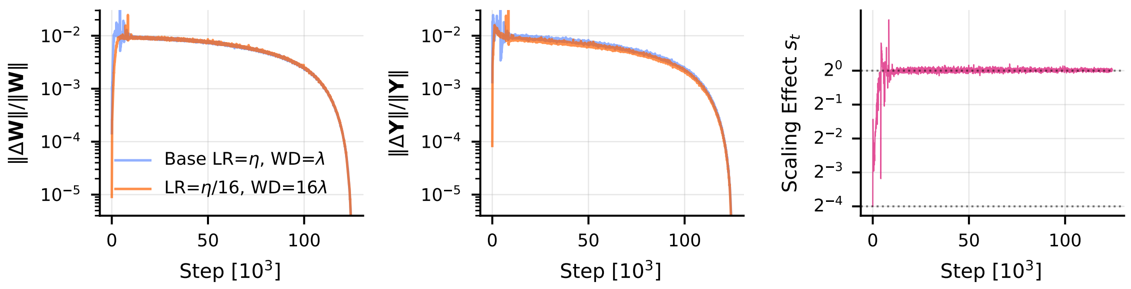

Figure 13: ResNet version of Figure 5. Plots show for `layer3.2.conv1` which is a 1×1 convolutional layer. Hyperparameters used are $\eta = 8 \times 10^{-3}$, $\lambda = 0.1$ corresponding to the best standard WD configuration in Figure 10 (after μP scaling). The additional warmup effect here is comparable in absolute length to the llama setting but proportionally shorter.

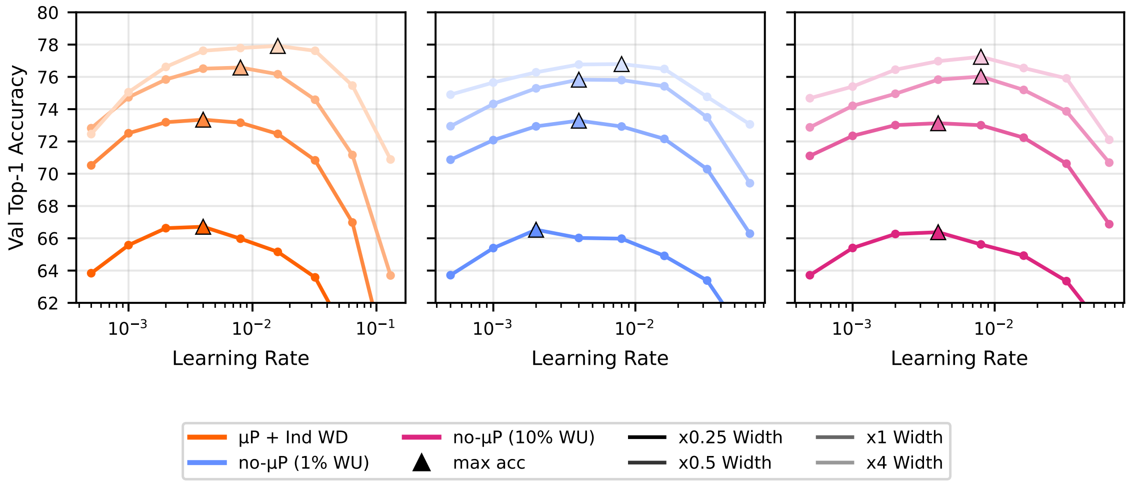

Figure 14: Learning rate transfer comparison between μP with independent weight decay scaling (left) and no-μP scaling for ResNet training (middle/right). The left and middle panels use a 1% linear warmup, the right panel uses a 10% linear warmup (WU). The additional warmup effect of μP with independent weight decay scaling is not needed for stability or transfer, but may contribute to the slightly higher accuracy achieved on the left. Other differences such as the resulting weight norms could also matter, particularly for the final FC layer. For normalized layers the weight norms also inversely scale the gradient norms which may affect the result through interactions with the epsilon value of Adam. A similar gap also appears in Figure 10 between standard and independent weight decay scaling.

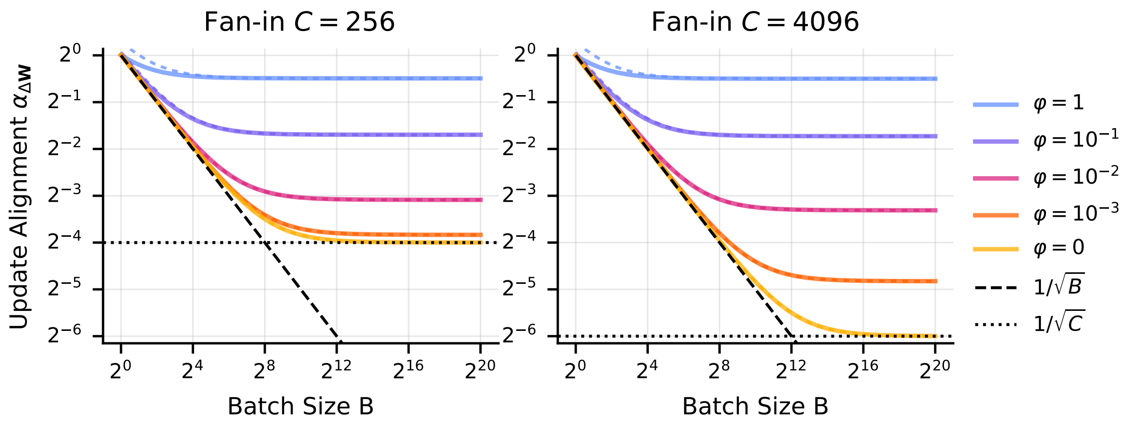

Figure 15: Predicted update alignment $\alpha_{\Delta W}$ in the simple noise model based on Equation 27. Note how the alignment for a low signal-to-noise ratio $\varphi \approx 0$ is roughly $1/\sqrt{B}$ until $B > C$ when it becomes $1/\sqrt{C}$. Higher values of φ always increase the alignment regardless of B and C. This is most prominent when $\varphi \gg 1/C$ in which case we get $\alpha_{\Delta W} \approx 1/\sqrt{\varphi^{-1}+1}$ once $B \gg \varphi^{-1}$. This can be viewed as the batch size being effectively capped at $\varphi^{-1}+1$, an estimate of the critical batch size [28]. The overall behavior is well approximated as $\alpha_{\Delta W} \approx \sqrt{C^{-1} + B^{-1} + \varphi/(1+\varphi)}$ shown by the dashed color lines.

Appendix E. Analysis of the Update Alignment for a Simple Noise Model

In this section we analyze the update alignment for a simple probabilistic model of a single neuron with a fan-in of C at batch size B. On the forward pass we have:

$$\mathbf{y}^\top = [\mathrm{y}_1, \ldots, \mathrm{y}_B] = \mathbf{w}^\top \mathbf{X} = \mathbf{w}^\top [\mathbf{x}_1, \ldots, \mathbf{x}_B] \tag{16}$$

where $\mathbf{w} \in \mathbb{R}^C$ are the weights of the neuron, $\mathbf{X} \in \mathbb{R}^{C \times B}$ is a batch of B input vectors with dimension C each, and $\mathbf{y} \in \mathbb{R}^B$ is the output for each input. The gradient of the final average sample loss $\mathscr{L} = \frac{1}{B}\sum_{b=1}^{B} \mathscr{L}_b$ with respect to the weights $\mathbf{w}$ is given by:

$$\mathbf{g} := \frac{\partial \mathscr{L}}{\partial \mathbf{w}} = \frac{1}{B}\sum_{b=1}^{B} \frac{\partial \mathscr{L}_b}{\partial \mathrm{y}_b}\mathbf{x}_b = \frac{1}{B}\sum_{b=1}^{B} \mathrm{y}'_b \mathbf{x}_b \tag{17}$$

where we have defined $\mathrm{y}'_b := \partial \mathscr{L}_b / \partial \mathrm{y}_b$. Note how the gradient for each sample $\mathbf{g}_b := \mathrm{y}'_b \mathbf{x}_b$ is simply a scaled version of the input vector $\mathbf{x}_b$ and is thus fully aligned with it.

We want to be able to introduce analytically tractable correlations between the random variables in our probabilistic model. Taking inspiration from Kosson et al. [20], we use the Signal-to-Noise ratio φ to capture the correlation. It is defined as the square ratio of the norms of the expected gradient (signal) to the expected deviation (noise):

$$\varphi := \frac{\|\bar{\mathbf{g}}\|^2}{\mathbb{E}_b[\|\mathbf{g}_b - \bar{\mathbf{g}}\|^2]}, \qquad \bar{\mathbf{g}} := \mathbb{E}_b[\mathbf{g}_b] \tag{18}$$

We can now define a distributions for $\mathbf{x}_b$ and y'_b that result in a specific φ while respecting the mathematical form of the gradient and being simple enough to analyze. We chose the form of the input samples as:

$$\mathbf{x}_b = (\tilde{\mathbf{x}}_b + \sqrt{\varphi} \cdot \mathbf{s} \cdot \frac{1}{\mathrm{y}'_b}) / \sqrt{1+\varphi} \tag{19}$$

where the noise component $\tilde{\mathbf{x}}_b$ and signal component $\mathbf{s}$ are independent random vectors drawn from a standard multivariate normal distribution, i.e., $\tilde{\mathbf{x}}_b, \mathbf{s} \sim \mathcal{N}(\mathbf{0}, I_C)$. We choose the distribution of the output gradient to be $\mathrm{y}'_b = \pm 1$ with equal probability to avoid complexities in modeling the inverse. We could scale the variance of the distributions but this would not affect the alignment ratio. In this model the gradient becomes:

$$\mathbf{g}_b = \mathrm{y}'_b \mathbf{x}_b = (\tilde{\mathbf{x}}_b \mathrm{y}'_b + \sqrt{\varphi} \cdot \mathbf{s}) / \sqrt{1+\varphi} \tag{20}$$

Assuming $\|\mathbf{s}\|^2 = C$ (or alternatively in expectation over $\mathbf{s}$), this results in the targeted signal-to-noise ratio φ:

$$\frac{\|\bar{\mathbf{g}}\|^2}{\mathbb{E}_b[\|\mathbf{g}_b - \bar{\mathbf{g}}\|^2]} = \frac{\varphi \|\mathbf{s}\|^2/(1+\varphi)}{\mathbb{E}_b[\|\mathrm{y}'_b \tilde{\mathbf{x}}_b / \sqrt{1+\varphi}\|^2]} = \varphi \frac{C/(1+\varphi)}{C/(1+\varphi)} = \varphi \tag{21}$$

Note that there are other forms for $\mathbf{x}_b$ that result in the desired signal-to-noise ratio including some that give $\bar{\mathbf{g}} = \mathbf{s}$, but our choice results in bounded input and gradient norms in expectation for both $\varphi \to 0$ and $\varphi \to \infty$. We will also assume that the update is obtained via simple gradient descent giving:

$$\Delta \mathbf{w} = -\eta \mathbf{g} \tag{22}$$

The update alignment involves a ratio that is hard to analyze under expectation. Instead we will approximate this as the ratio of the square expectations:

$$\mathbb{E}[\alpha_{\Delta \boldsymbol{W}}] = \mathbb{E}\left[\frac{\|\Delta \mathbf{w}^\top \mathbf{X}\|}{\|\Delta \mathbf{w}\| \|\mathbf{X}\|}\right] \approx \sqrt{\frac{\mathbb{E}[\|\Delta \mathbf{w}^\top \mathbf{X}\|^2]}{\mathbb{E}[\|\Delta \mathbf{w}\|^2] \mathbb{E}[\|\mathbf{X}\|^2]}} \tag{23}$$

we thus need to compute $\mathbb{E}[\|\Delta \mathbf{w}^\top \mathbf{X}\|^2]$, $\mathbb{E}[\|\Delta \mathbf{w}\|^2]$, and $E[\|\mathbf{X}\|^2]$.
Starting with the input size we get:

$$\mathbb{E}[\|\mathbf{X}\|^2] = B \cdot \mathbb{E}_b[\|\mathbf{x}_b\|^2] \tag{24a}$$

$$= B \cdot \mathbb{E}_b\left[\frac{1}{1+\varphi}\left\langle \tilde{\mathbf{x}}_b + \sqrt{\varphi} \cdot \mathbf{s} \cdot \frac{1}{\mathrm{y}'_b}, \tilde{\mathbf{x}}_b + \sqrt{\varphi} \cdot \mathbf{s} \cdot \frac{1}{\mathrm{y}'_b}\right\rangle\right] \tag{24b}$$

$$= \frac{B}{1+\varphi}\left(\mathbb{E}_b[\|\tilde{\mathbf{x}}_b\|^2] + 2\sqrt{\varphi}\mathbb{E}_b\left[\frac{1}{\mathrm{y}'_b}\langle \tilde{\mathbf{x}}_b, \mathbf{s}\rangle\right] + \varphi \mathbb{E}_b\left[\frac{1}{(\mathrm{y}'_b)^2}\|\mathbf{s}\|^2\right]\right) \tag{24c}$$

$$= \frac{B}{1+\varphi}(C + 0 + \varphi \cdot C) \tag{24d}$$

$$= BC \tag{24e}$$

For the update size we get:

$$\mathbb{E}[\|\Delta \mathbf{w}\|^2] = \eta^2 \mathbb{E}[\|\mathbf{g}\|^2] \tag{25a}$$

$$= \eta^2 \mathbb{E}\left[\left\|\frac{1}{B}\sum_{b=1}^{B} \frac{\mathrm{y}'_b \tilde{\mathbf{x}}_b + \sqrt{\varphi}\mathbf{s}}{\sqrt{1+\varphi}}\right\|^2\right] \tag{25b}$$

$$= \frac{\eta^2}{B^2(1+\varphi)}\mathbb{E}\left[\left\|\left(\sum_{b=1}^{B} \mathrm{y}'_b\tilde{\mathbf{x}}_b\right) + B\sqrt{\varphi}\mathbf{s}\right\|^2\right] \tag{25c}$$

$$= \frac{\eta^2}{B^2(1+\varphi)}\left(\mathbb{E}\left[\left\|\sum_{b=1}^{B} \mathrm{y}'_b\tilde{\mathbf{x}}_b\right\|^2\right] + 2\mathbb{E}\left[\left\langle\sum_{b=1}^{B} \mathrm{y}'_b\tilde{\mathbf{x}}_b, B\sqrt{\varphi}\mathbf{s}\right\rangle\right] + \mathbb{E}\left[\|B\sqrt{\varphi}\mathbf{s}\|^2\right]\right) \tag{25d}$$

$$= \frac{\eta^2}{B^2(1+\varphi)}\left(\sum_{b=1}^{B}\mathbb{E}[\|\tilde{\mathbf{x}}_b\|^2] + 0 + B^2\varphi\mathbb{E}[\|\mathbf{s}\|^2]\right) \tag{25e}$$

$$= \frac{\eta^2}{B^2(1+\varphi)}\left(BC + B^2\varphi C\right) \tag{25f}$$

$$= \frac{\eta^2 C}{B}\frac{1+B\varphi}{1+\varphi} \tag{25g}$$

Finally we can compute the expected output change as:

$$\mathbb{E}[\|\Delta\mathbf{w}\mathbf{X}\|^2] = B\mathbb{E}[\langle\mathbf{x}_b, -\eta\mathbf{g}\rangle^2] \tag{26a}$$

$$= \frac{B\eta^2}{(1+\varphi)^2}\mathbb{E}\Big[\Big\langle\tilde{\mathbf{x}}_b + \sqrt{\varphi}\mathbf{s}/\mathrm{y}'_b, \frac{1}{B}\sum_{i=1}^{B}\left(\tilde{\mathbf{x}}_i\mathrm{y}'_i + \sqrt{\varphi}\mathbf{s}\right)\Big\rangle^2\Big] \tag{26b}$$

$$= \frac{\eta^2}{(1+\varphi)^2B}\mathbb{E}\Big[\Big(\mathrm{y}'_b\|\tilde{\mathbf{x}}_b\|^2 + \sum_{i\neq b}\mathrm{y}'_i\langle\tilde{\mathbf{x}}_b,\tilde{\mathbf{x}}_i\rangle + (1+B)\sqrt{\varphi}\langle\tilde{\mathbf{x}}_b,\mathbf{s}\rangle + \frac{\sqrt{\varphi}}{\mathrm{y}'_b}\sum_{i\neq b}\langle\mathbf{s},\tilde{\mathbf{x}}_i\rangle\mathrm{y}'_i + \frac{B\varphi}{\mathrm{y}'_b}\|\mathbf{s}\|^2\Big)^2\Big] \tag{26c}$$

$$= \frac{\eta^2}{(1+\varphi)^2B}\Bigg[\mathbb{E}[\|\tilde{\mathbf{x}}_b\|^4] + \mathbb{E}\Big[\Big(\sum_{i\neq b}\mathrm{y}'_i\langle\tilde{\mathbf{x}}_b,\tilde{\mathbf{x}}_i\rangle\Big)^2\Big] + (1+B)^2\varphi\mathbb{E}[\langle\tilde{\mathbf{x}}_b,\mathbf{s}\rangle^2] + \varphi\mathbb{E}\Big[\Big(\sum_{i\neq b}\langle\mathbf{s},\tilde{\mathbf{x}}_i\rangle\mathrm{y}'_i\Big)^2\Big] + B^2\varphi^2\mathbb{E}\Big[\|\mathbf{s}\|^4\Big] + 2B\varphi\mathbb{E}\Big[\|\tilde{\mathbf{x}}_b\|^2\|\mathbf{s}\|^2\Big]\Bigg] \tag{26d}$$

$$= \frac{\eta^2}{(1+\varphi)^2B}\Bigg[\mathbb{E}[\|\tilde{\mathbf{x}}_b\|^4] + \sum_{i\neq b}\mathbb{E}[\langle\tilde{\mathbf{x}}_b,\tilde{\mathbf{x}}_i\rangle^2] + (1+B)^2\varphi\mathbb{E}[\langle\tilde{\mathbf{x}}_b,\mathbf{s}\rangle^2] + \varphi\sum_{i\neq b}\mathbb{E}[\langle\mathbf{s},\tilde{\mathbf{x}}_i\rangle^2] + B^2\varphi^2\mathbb{E}[\|\mathbf{s}\|^4] + 2B\varphi\mathbb{E}[\|\tilde{\mathbf{x}}_b\|^2\|\mathbf{s}\|^2]\Bigg] \tag{26e}$$

$$= \frac{\eta^2}{(1+\varphi)^2B}\Bigg[C(C+2) + (B-1)C + (1+B)^2\varphi C + \varphi(B-1)C + B^2\varphi^2C(C+2) + 2B\varphi C^2\Bigg] \tag{26f}$$

Now we have all the factors needed to approximate the update alignment via Equation 23 giving:

$$\mathbb{E}[\alpha_{\Delta \boldsymbol{W}}] \approx \sqrt{\frac{(C+2)+(B-1)+(1+B)^2\varphi+\varphi(B-1)+B^2\varphi^2(C+2)+2B\varphi C}{(1+B\varphi)(1+\varphi)CB}} \tag{27}$$

Figure 15 plots this prediction showing C-dependent behavior for large batch-to-width ratios as expected. We find the overall behavior in Equation 27 well approximated by:

$$\mathbb{E}[\alpha_{\Delta \boldsymbol{W}}] \approx \sqrt{C^{-1}+B^{-1}+\varphi/(1+\varphi)} \tag{28}$$

The last term is the inverse of $\varphi^{-1}+1$ which is an estimate of the critical batch size [28] where the overall gradient starts to become dominated by the signal component and the benefit of increasing the batch size further diminishes quickly. In the fully uncorrelated case $\varphi = 0$ we roughly get $\alpha_{\Delta \boldsymbol{W}} \approx 1/\sqrt{B}$ when $B \ll C$ which is the regime targeted by μP, infinitely wide networks trained with finite batch sizes. In the practical training settings we often have $B \gg C$ which results in $\alpha_{\Delta \boldsymbol{W}} \approx 1/\sqrt{C}$. This C dependence breaks the scaling behavior of μP. When the critical batch size is small compared to C this can instead become $\alpha_{\Delta \boldsymbol{W}} \approx \sqrt{\frac{\varphi}{1+\varphi}}$ which eliminates the C dependence again. The overall alignment behavior is thus highly dependent on the specific settings of B and C as well as the signal-to-noise ratio φ which dynamically changes throughout training.

Appendix F. Analyzing the Scheduling Effects of Weight Decay

F.1. The Evolution of the Weight Norm Over Time

Here we construct a simple model of how the weight norms evolve over time, similar to the one used in prior work [21]. In this section we use $\Delta \boldsymbol{W}$ to refer to this "gradient component" of the update, treating the weight decay component separately. We will assume that the weight update (excluding weight decay) at each timestep is orthogonal to the current weights. This results in a simple recurrence relation for the weight norms:

$$\|\mathbf{W}_{t+1}\|^2 = (1 - \eta_t \lambda)^2 \|\mathbf{W}_t\|^2 + \|\Delta \boldsymbol{W}_t\|^2 \tag{29}$$

We will assume that the elementwise RMS size of the weight update is roughly one, for example due to the second moment normalization in Adam. Defining $\rho_t := \|\boldsymbol{W}_t\|/\sqrt{KC}$ as the RMS value of the elements in $\|\boldsymbol{W}_t\|$ and $a_t := 1 - \eta_t \lambda$ as the decay multiplier we get a recurrence relation:

$$\rho_{t+1}^2 = a_t^2 \rho_t^2 + \eta_t^2 \tag{30}$$

This model involves several significant simplifications:

- The assumption that the update is orthogonal to the current weights can either come from randomness or due to scale-invariance from normalization layers. The gradient of a scale-invariant weight is always orthogonal to the weight and zero-mean uncorrelated random gradient is orthogonal to the weights on average. We note that with momentum the update is not necessarily orthogonal on average, even in these special cases.

- This model only holds for momentum when the gradients are uncorrelated over time. In this case we can replace the update term by the total contribution of the current gradient over time (the sum of the impulse response) as done by Kosson et al. [21]. This also assumes β_1 is not too large, so that the impact of a single gradient happens over a short timeframe compared to the t values we are interested in. If there is correlation the behavior of the system changes in a way we can not predict without additional information. See Wan et al. [36] for an analysis that partially accounts for this in SGDM.

- The size of the update assumes that β_2 can accurately track the expected magnitude of the gradient over time. This either requires small β_2 values in comparison to how fast the gradient magnitude changes. We note that at the start of training the gradient magnitude often changes very rapidly.

This means that in practice Equation 30 does not accurately predict the behavior of the weight norm in neural network training. However, we believe it is the best we can do based on the hyperparameters alone without measurements of the temporal gradient correlation and gradient magnitude over time (which we expect strongly depend on the other hyperparameters too). We still find the predictions of this model to be informative for real training, particularly those based on the equilibrium value that satisfies $\rho_{t+1} = \rho_t$. The RMS values seen in real training often stabilize near this value [21], but how fast this happens depends strongly on the complex temporal effects described above.

With this disclaimer, we proceed with the analysis of Equation 30. Rolling out the recurrence relation we get:

$$\rho_{t+1}^2 = \rho_0^2 \prod_{\tau=0}^{t} a_\tau^2 + \sum_{\tau=0}^{t} \eta_\tau^2 \prod_{i=\tau+1}^{t} a_i^2 \tag{31}$$

In the case where the learning rate is constant $\eta_t = \eta$, making $a_t = a$ constant as well, this simply gives:

$$\rho_t^2 = \rho_0^2 \cdot a^{2t} + \eta^2 \sum_{\tau=0}^{t-1} a^{2(t-\tau)} = \rho_0^2 \cdot a^{2t} + \eta^2 \frac{1-a^{2t}}{1-a^2} \tag{32}$$

assuming $0 < a < 1$. If η_t is not constant, we can still easily compute the behavior of the weight norm over time using the recurrence relation or the rolled-out form directly.

In the constant learning rate case the RMS value approaches the equilibrium value ρ_∞ exponentially:

$$\rho_t^2 = \rho_\infty^2 + (\rho_0^2 - \rho_\infty^2)a^{2t} \qquad\qquad \rho_\infty^2 = \frac{\eta^2}{1-a^2} \approx \frac{\eta}{2\lambda} \tag{33}$$

We can use this to estimate how long it takes for the weights to approximately reach equilibrium. If we define this by the timestep T_{EQ} when the RMS value falls within a factor $\sqrt{2}$ of the equilibrium value we get:

$$T_{\text{EQ}} \approx \begin{cases} \frac{1}{2\eta\lambda} \ln\left(\frac{1-\rho_0^2/\rho_\infty^2}{1-1/\sqrt{2}}\right) & \text{if } \rho_0 < \frac{1}{\sqrt{2}}\rho_\infty \\ \frac{1}{2\eta\lambda} \ln\left(\frac{\rho_0^2/\rho_\infty^2-1}{\sqrt{2}-1}\right) & \text{if } \rho_0 > \sqrt{2}\rho_\infty \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

F.2. The Evolution of the Relative Weight Update Over Time

In this simplified model the relative weight update becomes:

$$\frac{\|\Delta \boldsymbol{W}_t\|}{\|\boldsymbol{W}\|} = \frac{b\eta_t}{\rho_t} \tag{35}$$

which we can compute exactly given the expressions for ρ_t from the previous subsection.

In the constant learning rate case we get:

$$\frac{\|\Delta \boldsymbol{W}_t\|}{\|\boldsymbol{W}\|} = \sqrt{\frac{b^2\eta^2}{\rho_t^2}} = \sqrt{\frac{b^2\eta^2}{\rho_0^2 \cdot a^{2t} + b^2\eta^2\frac{1-a^{2t}}{1-a^2}}} \xrightarrow[t\to\infty]{} \sqrt{2\eta\lambda - \eta^2\lambda^2} \approx \sqrt{2\eta\lambda} \tag{36}$$

showing that the product $\eta\lambda$ determines the size of relative weight updates in the steady-state. However, at the start of training, e.g. the first step, the relative weight update (excluding the weight decay component) is determined by just the learning rate and the initialization norm.

It is also interesting to see what happens without weight decay, $\lambda = 0$. For a constant learning rate, this gives:

$$\rho_t^2 = \rho_0^2 + t \cdot \eta^2 \qquad\qquad \frac{\eta_t}{\rho_t} = \frac{\eta}{\sqrt{\rho_0^2 + t\cdot\eta^2}} = \frac{1}{\sqrt{(\rho_0^2/\eta^2) + t}} \tag{37}$$

which means that the relative updates decay towards zero over time roughly as $t^{-0.5}$, rather than stabilizing at a fixed value. For this reason it is important to use learning rate schedules with weight decay, otherwise the effective learning rate measured in terms of relative updates never decays which typically leads to suboptimal outcomes. This decay in the relative update size can also be detrimental, if the relative update size decreases too quickly the network stops learning (sometimes referred to as a loss of plasticity). An example of this was seen in the AdamW paper [27] where the optimal weight decay value in an experiment was zero for a constant learning rate but non-zero when a cosine schedule was used. It is better to vary the effective learning rate over time explicitly according to our desired schedule, ideally exactly through optimizers like LionAR [20] or approximately via weight decay.

F.3. The Warmup Effect of High Weight Decay Configurations

Let us now consider two configurations for the learning rate and weight decay, (η, λ) vs $(\eta/m, m\lambda)$. These correspond to the configurations from independent weight decay scaling compared to no scaling (or in standard scaling at a higher base learning rate). We first note that their product is identical so the relative weight update will converge to the same equilibrium value. However, the very first update will also be m times smaller for the high weight decay configuration. A learning rate warmup starting at η/m and reaching η over time has an identical effect on the initial update and the steady-state updates. However, the exact shape of this warmup effect is not trivial, i.e. how it affects the relative update size over time.

We use the previous symbols to denote quantities for the base hyperparameters (η, λ) and ˆ over the symbol to denote the corresponding quantities for the scaled configuration. We are interested in the ratio of the relative weight updates between the two configurations:

$$s_t := \frac{\hat{b\eta_t}/\hat{\rho}_t}{b\eta_t/\rho_t} \tag{38}$$

which captures the size of the warmup effect. For the simple weight norm model we can always compute this using Equation 30, even when the learning rate varies over time. For constant learning rates we would get:

$$s_t = \frac{1}{m}\frac{\rho_t}{\hat{\rho}_t} = \frac{1}{m}\sqrt{\frac{\rho_0^2 \cdot a^{2t} + b^2\eta^2\frac{1-a^{2t}}{1-a^2}}{\rho_0^2 \cdot a^{2t} + b^2\hat{\eta}^2\frac{1-a^{2t}}{1-a^2}}} = \sqrt{\frac{\rho_\infty^2 + (\rho_0^2 - \rho_\infty^2)a^{2t}}{\rho_\infty^2 + (m^2\rho_0^2 - \rho_\infty^2)a^{2t}}} = \sqrt{\frac{1 + (\rho_0^2/\rho_\infty^2 - 1)a^{2t}}{1 + (m^2\rho_0^2/\rho_\infty^2 - 1)a^{2t}}} \tag{39}$$

This is already a relatively complicated expression with behavior that depends on the initialization magnitude ρ_0 as well as the exact values of η and λ used. The effective length of the additional warmup can vary from a single step, e.g. when $\rho_0 \to 0$, to the whole training, for example when ρ_0 is large or decays slowly. We note it will also depend significantly on $\eta\lambda$, so when sweeping learning rates the length of the warmup effect will differ between points.

Finally we note that simply scaling the learning schedule in for the base configuration by s_t will generally not give relative updates matching the high weight decay configuration. This is because modifying the learning rate schedule also changes how the norms evolve over time. A scaling factor that would make the profiles match exactly could still be computed using the recurrence relations by keeping track of how the modifications affect the norms over time. However, exactly matching the warmup effect from higher weight decay values may not be that important since it is largely arbitrary as far as we know in the sense that it does not depend on the alignment ratio. Applying the scaling factor from Equation 38 to a relative or rotational optimizer like LionAR [20] would result in the desired warmup effect.

Algorithm 1 AdamW (PyTorch variant, differs from the original by Loshchilov and Hutter [27])

```
Require: Learning rate $\eta_t$, weight decay $\lambda$, momentum $\beta_1$, magnitude smoothing $\beta_2$, $\epsilon$ for numerical stability
1: Initialize: Time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_0$, momentum vector $\boldsymbol{m}_0 \leftarrow 0$, magnitude vector $\boldsymbol{v}_0 \leftarrow 0$
2: while stopping criteria not met :
3:     $t \leftarrow t + 1$
4:     $\boldsymbol{g}_t \leftarrow$ Mini-batch gradient w.r.t. $\boldsymbol{\theta}_{t-1}$
5:     $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
6:     $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
7:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$
8:     $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$
9:     $\boldsymbol{\theta}_t \leftarrow (1 - \eta_t\lambda)\boldsymbol{\theta}_{t-1} - \eta_t\hat{\boldsymbol{m}}_t/(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon)$
```

Appendix G. Experimental Details

G.1. Algorithms

The AdamW variant we use is shown in Algorithm 1. Note how the weight decay hyperparameter in line 9 is scaled by the learning rate. This differs from the original variant described in Loshchilov and Hutter [27], which is often referred to as "independent" or "fully-decoupled". This variant only has λ instead of $\eta\lambda$ in line 9, but the weight decay hyperparameter is still varied according to the same schedule as the learning rate.

G.2. Learning Rate Transfer Experiments

Our learning rate transfer experiments consist of two phases. We start from an existing hyperparameter configuration and first sweep the global learning rate used across all parameter types i.e. the input layer, hidden layers, output layer, biases and gains. This is done for the $1\times$ scale model. We then freeze the learning rate of the input layer, biases and gains, changing only the learning rate for the layers that scale with the width, i.e., the hidden layer and output layer. This second base learning rate is scaled via μP's rules before applying it in the optimizer, but the plots show the base rate before scaling (which is what transfers across scale). This second sweep gives the final performance shown in the learning rate transfer plots.

We adopt this approach because it better captures how well the learning rate scaling works by leaving the parameters it is not applied to unaffected. Compared to sweeping a single global learning rate across scales, this typically gives better performance with standard scaling but highlights the shift more. The reason is that without freezing the learning rate for the other layers, they can become unstable at the shifted (higher) base learning rates needed for the hidden layers to learn well. This gives worse overall results since no single global learning rate works well for all parameters anymore, but the optimal learning rates may shift less as a result.

Instead of using the full Maximal Update Parameterization μP for training we combine standard parameterization with the learning rate scaling of μP as proposed by Everett et al. [13]. They find this simpler approach empirically works just as well or better. We note there are many variants of μP that achieve the same effects slightly differently, see for example Tables 3, 8 and 9 in Tensor Programs V [40]. However, these are generally not equivalent with standard weight decay, which would complicate the use of full μP for our experiments.

For transformer training the main differences between full μP and our experiments are the handling of the final output layer and the normalization factor applied in attention. The output layer is initialized using a smaller standard deviation that scales with $1/C$ rather than $1/\sqrt{C}$ like in standard parameterization. This results in an initial relative update size that is not scaled with the width compared to the $1/\sqrt{m}$ scaling that occurs in the approach from Everett et al. [13]. The other difference is that the scaling factor in the attention heads is $1/d_{\text{head}}$ rather than $1/\sqrt{d_{\text{head}}}$. However we keep d_{head} constant in our scaling experiments, opting to scale the number of heads instead. This difference could thus be absorbed into a tunable factor that does not change across scales.

G.3. Llama experiment configuration

Our llama [33] experiments are performed using a modified version of Lingua [35]. The networks have 20 transformer blocks, each consisting of one attention and one MLP sub-block. We vary the embedding dimension d_{emb} between 128 and 2048, which we sometimes refer to as $1\times$ and $16\times$. The MLP block has an expansion factor of exactly $3\times$ giving exact scaling ratios for all layers. We used a fixed attention head size of $d_{\text{head}} = 128$, varying the number of heads across scale. All linear layers are initialized using a standard deviation of $1/\sqrt{C}$. We do not use embedding tying but the embeddings are still initialized with a standard deviation of $1/\sqrt{d_{\text{emb}}}$ like the final FC layer (default behavior in Lingua).

The training task is standard next token prediction using a cross-entropy loss. The dataset used is DLCM [22], specifically a 10% subset obtained by using the first global shard from the Hugging Face dataset. We use the llama3 tokenizer[5] with a vocabulary size of 128,256.

Unless otherwise stated we use the PyTorch variant of the AdamW optimizer shown in Algorithm 1. We vary the learning rate between runs but other hyperparameters are fixed at $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and $\lambda = 0.1$ (except when scaled through independent weight decay scaling). Weight decay is applied to all parameters except the gains. We train for 20,000 steps at a global batch size of 256 and sequence length of 4096 giving 1,048,576 (2^{20}) tokens in each batch and 20.97 billion tokens in total. This results in 19.23 tokens per non-embedding parameter in the largest configuration we use (embedding dim 2048). Unless otherwise noted, the learning rate schedule consists of a 10% linear warmup from 0 to the specified peak learning rate, followed by a linear decay down to 0. The losses reported are the average of the last 100 training steps (we had issues with the evaluation part of the code base).

For an embedding dim of 2048 it takes roughly 14 hours to complete a single run using 8 H200 GPUs. This was not optimized and is slowed down by heavy logging of a wide range of alignment related metrics. Training was performed in bfloat16 with mixed precision.

G.3.1. DETAILS FOR FIGURE 1

This experiment follows our base llama training G.3 and learning rate transfer setup G.2. However these are more recent runs with working validation losses. For the weight decay runs, the learning rate for the input layer and gains is frozen at $\eta = 1.6 \times 10^{-2}$ based on an initial sweep of a global learning rate at the $1\times$ scale. For the no weight decay runs, this secondary learning rate is set to $\eta = 6.4 \times 10^{-2}$. The best loss without weight decay is 2.815 at $\eta_{\text{base}} = 6.4 \times 10^{-2}$, for independent weight decay it is 2.801 at $\eta_{\text{base}} = 8 \times 10^{-3}$, and for standard weight decay it is 2.798 at $\eta_{\text{base}} = 6.4 \times 10^{-2}$.

5. https://huggingface.co/meta-llama/Meta-Llama-3-8B

G.3.2. DETAILS FOR FIGURE 3

This experiment shows `layers.13.feed_forward.w1` for a base learning rate $\eta = 8 \times 10^{-3}$, the optimal for value for the $1\times$ width in Figure 1. The learning rate for gains and the input layer is $\eta = 6.4 \times 10^{-2}$. The metrics are logged and plotted every 10 steps, the final 0.5% of training is excluded for numerical reasons (the changes go to zero with the learning rate). The representation based metrics are computed for 1/64th of a batch (corresponding to one micro-batch on the master rank). Other details follow the base setup described in G.3.

G.3.3. DETAILS FOR FIGURE 4

This experiment shows `layers.13.feed_forward.w1` for a learning rate $\eta = 4 \times 10^{-3}$ and weight decay $\lambda = 0.1$. Note that this is the LR-WD product that gives the best results for standard scaling of the $16\times$ configuration in Figure 1, after applying the μP scaling to the LR. No μP scaling is performed between the two configurations, the hyperparameters are exactly the same and the specified learning rate is applied to all parameters. The metrics are logged and plotted every 10 steps, the final 0.5% of training is excluded for numerical reasons. The representation based metrics are computed for 1/64th of a batch (corresponding to one micro-batch on the master rank). See Figure 16 for the alignment ratio of other layers. Other details follow the base setup described in G.3.

G.3.4. DETAILS FOR FIGURE 5

This experiment shows `layers.13.feed_forward.w1` for a learning rate $\eta = 4 \times 10^{-3}$ (after μP scaling) and weight decay $\lambda = 0.1$. This corresponds to the best configuration for standard weight decay scaling in Figure 1 and the independent scaling run with the same LR-WD product. Metrics are logged and displayed every 10 steps, the final 0.5% are excluded for numerical reasons, representation changes are computed on 1/64th of a batch. Other details follow the base setup described in G.3.

G.3.5. DETAILS FOR FIGURE 6

This experiment follows our base setup for llama training G.3 and learning rate transfer experiments G.2. The learning rate for the gains and input layer is fixed at $\eta = 1.6 \times 10^{-2}$. The modified learning rate schedules are applied to all parameters, not only the width dependent ones. The scaling factor for `exp50` and `decayaway` varies between widths as in μP, they have no effect at the $1\times$ scale. The additional warmup factors are applied on top of the existing 10% warmup and linear decay. For the longer 50% linear warmup, the run corresponding to $\eta = 4 \times 10^{-3}$ was repeated due to sudden divergence half way through, which did not occur in the repeated run with exactly the same seed and configuration.

G.4. ResNet experiment configuration

Our ResNet [16] experiments are performed using a modified version of PyTorch Image Models [38]. All networks are 50 layers deep with a bottleneck block configuration, specifically the original post-norm variant but with the downsampling on the residual stream performed on the 3x3 convolutions ("ResNet v1.5"). When scaling the width we scale all hidden dimensions of all layers evenly relative to the original network, going down to $0.25\times$ width up to $4\times$. The $4\times$ variant is fairly large for a convolutional network with roughly 384M parameters.

The training task is image classification using cross-entropy loss. We train on ImageNet-1k [31] for 100 epochs. Data augmentation consists of random cropping and random horizontal flips.

Optimization is performed using the PyTorch variant of AdamW. The learning rate varies between experiments. Other hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\lambda = 0.1$ (before independent weight decay scaling). Unless otherwise stated we use a 1 epoch linear warmup from 0 followed by a cosine decay to 0. The batch size is 1024 spread out over 8 GPUs without the use of synchronized batch normalization during training.

The training of the $4\times$ variant takes roughly 10 hours to complete using 8 H200 GPUs. This was not optimized and is slowed down by heavy logging of a wide range of alignment related metrics. Training was performed in bfloat16 with mixed precision.

Appendix H. Supplementary Figures

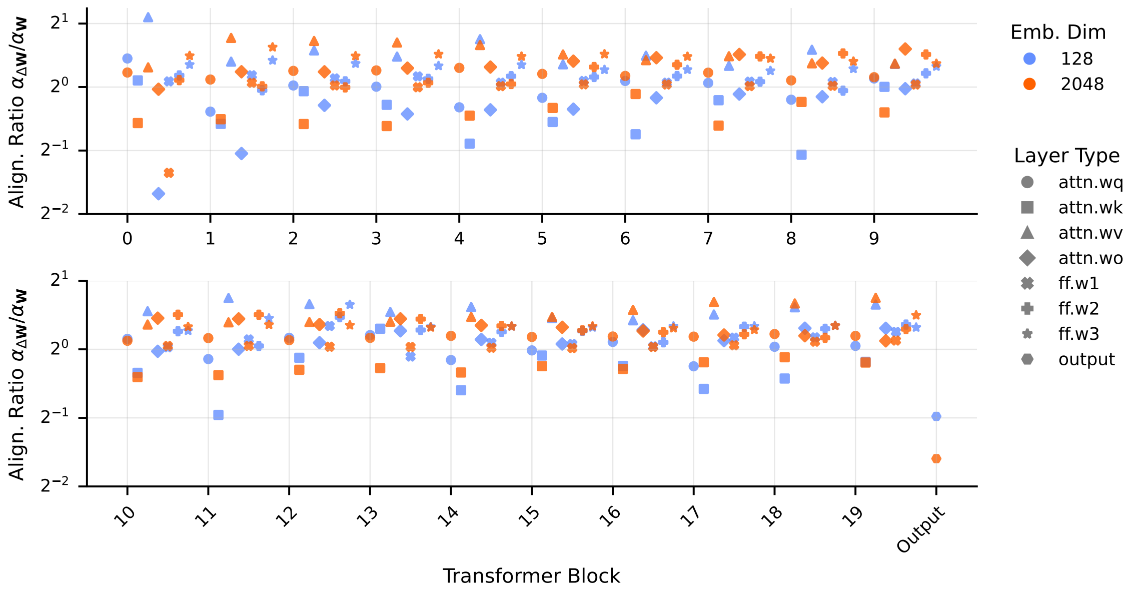

Figure 16: Halfway through training the alignment ratio of all layers is approximately one. The alignment ratio for each layer in the llama experiments shown in Figure 4 measured at step 10,000. The ratio is similar across widths, contrary to μP's assumptions.