Rotational Equilibrium: How Weight Decay Balances Learning Across Neural Networks

Atli Kosson [1 *] **Bettina Messmer** [1 *] **Martin Jaggi** [1]

Abstract

This study investigates how weight decay affects the update behavior of individual neurons in deep neural networks through a combination of applied analysis and experimentation. Weight decay can cause the expected magnitude and angular updates of a neuron's weight vector to converge to a steady state we call *rotational equilibrium*. These states can be highly homogeneous, effectively balancing the average rotation—a proxy for the effective learning rate—across different layers and neurons. Our work analyzes these dynamics across optimizers like Adam, Lion, and SGD with momentum, offering a new simple perspective on training that elucidates the efficacy of widely used but poorly understood methods in deep learning. We demonstrate how balanced rotation plays a key role in the effectiveness of normalization like Weight Standardization, as well as that of AdamW over Adam with ℓ_2-regularization. Finally, we show that explicitly controlling the rotation provides the benefits of weight decay while substantially reducing the need for learning rate warmup.

1. Introduction

The use of weight decay or ℓ_2-regularization has become ubiquitous in deep learning optimization. Although originally proposed as an explicit regularization method, Van Laarhoven (2017) showed that this interpretation does not hold for modern networks with normalization layers. This is because normalization can make a weight vector scale-invariant, meaning that the network output is unaffected by the magnitude of the vector (see §2). For scale-invariant vectors, weight decay instead acts as a scaling factor for some notation of an "effective" learning rate with varying definitions (Van Laarhoven, 2017; Zhang et al.,

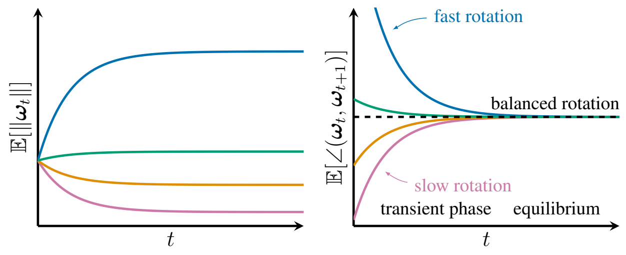

Figure 1: Conceptual figure of the norm (left) and angular updates (right) of the weight vector $\boldsymbol{\omega}_t$ for different neurons (each line color) over time t with a constant learning rate. Weight decay modulates and stabilizes both metrics.

2019; Li & Arora, 2020; Wan et al., 2021). We explore this idea further, aiming to describe the effects of weight decay on the optimization dynamics of modern neural network (NN) training through applied analysis and experimentation.

We specifically examine the ***update dynamics*** of individual neurons. A ***neuron*** computes a scalar feature by comparing a learnable weight vector $\boldsymbol{\omega}_t \in \mathbb{R}^C$ with an incoming activation vector $\boldsymbol{x}_t \in \mathbb{R}^C$, followed by a non-linear activation function φ and an optional learnable bias $b_t \in \mathbb{R}$:

$$\varphi(\langle\boldsymbol{\omega}_t, \boldsymbol{x}_t\rangle + b_t) = \varphi(\|\boldsymbol{\omega}_t\|\|\boldsymbol{x}_t\| \cos(\angle(\boldsymbol{\omega}_t, \boldsymbol{x}_t)) + b_t) \quad (1)$$

Here $\langle\cdot\rangle$ denotes a dot product which can be rewritten with a cosine similarity, showing that the direction of $\boldsymbol{\omega}_t$ determines which "patterns" in $\boldsymbol{x}_t$ the neuron responds to. We describe the neuronal update dynamics through the ***expected weight norm*** $\mathbb{E}[\|\boldsymbol{\omega}_t\|]$, ***root-mean-square (RMS) update size*** η_g for the bias, and ***expected angular update size*** η_r:

$$\eta_g := \sqrt{\mathbb{E}[(b_{t+1} - b_t)^2]} \quad (2)$$

$$\eta_r := \mathbb{E}[\angle(\boldsymbol{\omega}_t, \boldsymbol{\omega}_{t+1})] = \mathbb{E}\left[\arccos\left(\frac{\langle\boldsymbol{\omega}_t, \boldsymbol{\omega}_{t+1}\rangle}{\|\boldsymbol{\omega}_t\|\|\boldsymbol{\omega}_{t+1}\|}\right)\right] \quad (3)$$

where the expectation accounts for noise from randomly sampled minibatches (e.g., data shuffling). We assume weight decay is only applied to $\boldsymbol{\omega}$, not b or other parameters, which is common and good practice (Jia et al., 2018).

Figure 1 shows how the weight norm and angular updates (η_r) of different neurons could behave over time in a typical case. This behavior is caused by Spherical Motion Dynamics (Wan et al., 2021) which arise from the interaction of

*Primary contributor [1]EPFL, Switzerland.
Correspondence to: <atli.kosson@epfl.ch>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

weight decay and stochastic gradient updates (described further in §3). Over time, the weight norm reaches a stable ***equilibrium magnitude*** in expectation, at which point the opposing effects of gradient updates (which increase the norm) and weight decay (which reduces it) cancel each other out. Interestingly, this simply stems from the geometry of the stochastic optimizer updates (especially with normalization layers), not the convergence of the underlying loss function, and can therefore be analyzed for a random walk.

The angular updates shown in fig. 1R have an inverse dependency on the weight magnitude. During an initial ***transient phase*** the rotation is somewhat arbitrary, depending on the initial weight magnitude and gradient norm (for some optimizers). The convergence of the weight norm results in a ***steady-state*** we call ***rotational equilibrium***, characterized by the average angular update having a specific, stable magnitude. For some setups the rotational equilibrium is identical for different layers and neurons (even if the weight norms differ), resulting in ***balanced rotation*** which we empirically observe aids optimization. We discuss our intuition for why this helps in appx. M.

This study significantly expands upon prior investigations into the interactions between weight decay and normalization, demystifying the effects of weight decay and other common tricks in deep learning. While we touch upon certain previous works throughout, please refer to appx. A for an extended discussion. Earlier research has primarily focused on the general mechanisms and properties of weight decay and normalization, especially for plain SGD. We focus on two main new directions, rotational equilibrium in other optimizers like AdamW (Loshchilov & Hutter, 2019), and the importance of balanced rotation in the optimization of neural networks. Compared to prior work, our analysis also targets more fine-grained update dynamics at the neuron level, applies to networks without scale-invariance from perfectly placed normalization layers, investigates additional aspects of the dynamics (e.g. transient phase, bias behavior), and relates standard optimizers to the LARS-style optimizers (You et al., 2017). Our key contributions are:

- Deriving the steady-state neuronal update dynamics of AdamW, Adam with ℓ_2-regularization, Lion and SGD with momentum for a random walk. We experimentally validate that the results hold for NN training in practice.
- Showing how the interaction of weight decay and learning rate shapes the rotation in the initial transient phase and results in two distinct "effective step sizes" in the steady state, η_g for biases and η_r for weights.
- Demonstrating how explicitly controlling the angular updates via Rotational Optimizer Variants provides an alternative way of achieving the benefits of weight decay and normalization, while also simplifying the update dynamics and reducing the need for learning rate warmup.
- Revealing how balanced rotation contributes to the performance benefit of AdamW vs Adam+ℓ_2, and certain normalization layers like Weight Standardization.

2. Preliminaries

Scale-Invariance: Li & Arora (2020); Wan et al. (2021) describe how properly placed normalization can make the weight vector $\boldsymbol{\omega}$ of a neuron/layer scale invariant w.r.t. a loss $\mathscr{L}$ and the resulting properties of the gradient $\nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega},\cdot)$:

$$\text{Scale Invariance: } \mathscr{L}(r\boldsymbol{\omega},\cdot) = \mathscr{L}(\boldsymbol{\omega},\cdot), \forall r>0 \quad (4)$$

$$\text{Gradient orthogonality: } \nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega},\cdot) \perp \boldsymbol{\omega} \quad (5)$$

$$\text{Inverse proportionality: } \|\nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega},\cdot)\| \propto \|\boldsymbol{\omega}\|^{-1} \quad (6)$$

See appx. B for an overview. Note that different normalization operations can result in scale-invariance at a different granularity, for example individual neurons for Batch Normalization (Ioffe & Szegedy, 2015) and Weight Standardization (Qiao et al., 2019; Huang et al., 2017b) but only whole layers for Layer Normalization (Ba et al., 2016).

The Effective Learning Rate: In related literature there are multiple definitions of "effective" learning rates (Van Laarhoven, 2017; Chiley et al., 2019; Wan et al., 2021) which aim to describe how fast the neural network is being updated based on some metric. We use the average angular change η_r for this purpose, but will refer to it as an ***effective update size***. Note that the direction of $\boldsymbol{\omega}$ in eq. (1) controls which patterns in the inputs $\boldsymbol{x}$ the neuron detects, and η_r thus directly captures how fast this important aspect in the underlying functional representation of the neuron changes. This applies to all neurons, but particularly when the weight vector is scale-invariant and the direction thus fully determines its effect. Simple alternatives like $\|\boldsymbol{\omega}_{t+1} - \boldsymbol{\omega}_t\|$ are scale dependent and do not directly measure changes in the encoded function. For other parameters we use the RMS change η_g as a measure of the update size. This is generally not a "functional" update measure (which would vary based on the architecture), but still informative and easy to analyze. Finally we note that an update size only measures the size of individual updates. Momentum affects the correlation of the updates over time, which also influences long-term "learning speed", see appx. L.

3. Analysis

In this section we analyze the rotational equilibrium of a weight vector $\boldsymbol{\omega}$ to obtain simple expressions for the equilibrium magnitude $\widehat{\|\boldsymbol{\omega}\|}$ and the expected angular update in equilibrium, $\widehat{\eta_r}$. We focus on a simplified setting where updates are dominated by noise, resulting in a type of random walk. The equilibrium dynamics derived in this analytically tractable setting are predictive of the behavior we empirically observe in neural networks (see experiments), but this

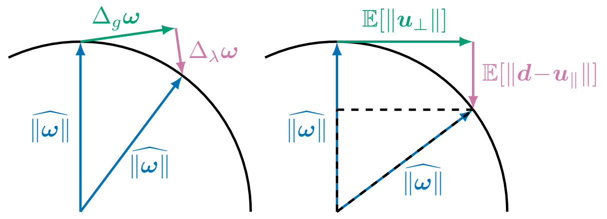
$\Delta_g \boldsymbol{\omega}$
$\Delta_\lambda \boldsymbol{\omega}$
$\widehat{\|\boldsymbol{\omega}\|}$
$\widehat{\|\boldsymbol{\omega}\|}$
$\mathbb{E}[\|\boldsymbol{u}_\perp\|]$
$\mathbb{E}[\|\boldsymbol{d}-\boldsymbol{u}_\parallel\|]$
$\widehat{\|\boldsymbol{\omega}\|}$
$\widehat{\|\boldsymbol{\omega}\|}$

Figure 2: Two views of equilibrium where the weight norm $\widehat{\|\omega\|}$ is preserved because the gradient and weight decay components balance out on average. **Left:** Standard optimizer update eq. (7). **Right:** The total update contributions over the course of training, $\boldsymbol{u}$ and $\boldsymbol{d}$, derived from the gradient and weight decay of a given timestep, respectfully.

is not meant to be a formal theoretical analysis that fully captures all the intricacies of neural network optimization.

Specifically, we assume the loss is in the form of empirical risk, i.e. $\mathscr{L}(\boldsymbol{\omega}, \mathbb{X}) = \frac{1}{|\mathbb{X}|}\sum_{\boldsymbol{x}\in\mathbb{X}} \mathscr{L}(\boldsymbol{\omega}, \boldsymbol{x})$ where $\mathbb{X}$ is our training dataset, $\boldsymbol{\omega}$ are our weights and $\boldsymbol{x}$ is a data point. The true noiseless gradient is then $\boldsymbol{g}_\mathbb{X} = \nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}, \mathbb{X})$ and the gradient for a minibatch $\mathbb{B}$ is $\boldsymbol{g}_\mathbb{B} = \nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}, \mathbb{B})$. We can define the noise in the gradient as $\boldsymbol{g}_N = \boldsymbol{g}_\mathbb{B} - \boldsymbol{g}_\mathbb{X}$ with $\mathbb{E}_\mathbb{B}[\boldsymbol{g}_N] = 0$, because $\mathbb{E}_\mathbb{B}[\boldsymbol{g}_\mathbb{B}] = \boldsymbol{g}_\mathbb{X}$ for a randomly sampled $\mathbb{B}$. Our simplifying assumption of the noise dominating can be stated $\boldsymbol{g}_\mathbb{B} = \boldsymbol{g}_\mathbb{X} + \boldsymbol{g}_N \approx \boldsymbol{g}_N$, resulting in a *random walk* for the neural network parameters. Analogous assumptions have been successfully used elsewhere, e.g. with stochastic differential equations to derive the batch size scaling behavior of optimizers (Li et al., 2021; Malladi et al., 2022). In our experiments we find that the final predictions hold very well for a variety of networks despite being derived for this simplified setting. Appx. G further describes the analytical setting and explores how the differences between a random walk and real neural network optimization affect the predictions.

3.1. Geometric Model for Equilibrium

In this section we present a simple geometric derivation of the equilibrium norm $\widehat{\|\omega\|}$ for different optimizers inspired in part by the analysis in Online Normalization (Chiley et al., 2019). We divide a parameter update $\boldsymbol{\omega}_t \to \boldsymbol{\omega}_{t+1}$ into:

$$\boldsymbol{\omega}_{t+1} - \boldsymbol{\omega}_t = \Delta_g \boldsymbol{\omega} + \Delta_\lambda \boldsymbol{\omega} \tag{7}$$

where $\Delta_g\boldsymbol{\omega}$ comes from the gradients and $\Delta_\lambda\boldsymbol{\omega}$ from the weight decay. Equilibrium is an abstract state where the effects of $\Delta_g\boldsymbol{\omega}$ and $\Delta_\lambda\boldsymbol{\omega}$ on the expected weight magnitude balance out on average. These components typically have different monotonic dependencies on the weight magnitude, with weight decay being proportional while the gradient component is either constant or inversely proportional, depending on the setting. As a result, the effects of these components can balance out in expectation at the equilibrium magnitude $\widehat{\|\omega\|}$. As shown in fig. 2L, the geometry of this is not necessarily simple. Due to the averaging effects of momentum over time, $\Delta_g\boldsymbol{\omega}$ is not necessarily orthogonal to the weights even in cases where individual gradients are (e.g. for scale-invariant weights). Similarly, the weight decay (or ℓ_2-regularization) component $\Delta_\lambda\boldsymbol{\omega}$ may not be perfectly anti-parallel to the weights with momentum.

To simplify the effects of momentum, we instead consider a different view of equilibrium shown in fig. 2R. Here we consider the total weight change throughout training arising from the weight decay term and gradient from a given time step, instead of the update that is applied in that iteration. The ***Total Update Contribution (TUC)*** of the gradient at time step t, denoted $\boldsymbol{u}$, is the sum of the contributions of $\nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}_t, \cdot)$ to subsequent updates $\boldsymbol{\omega}_t \to \boldsymbol{\omega}_{t+1}$, $\boldsymbol{\omega}_{t+1} \to \boldsymbol{\omega}_{t+2}$, and so on. Analogously, the TUC of the weight decay, denoted $\boldsymbol{d}$, is the total change due to the weight decay or ℓ_2-regularization of the weights $\boldsymbol{\omega}_t$ at iteration t. Note that without momentum $\boldsymbol{u} = \Delta_g\boldsymbol{\omega}$, $\boldsymbol{d} = \Delta_\lambda\boldsymbol{\omega}$ and that if $\Delta_g\boldsymbol{\omega}$ and $\Delta_\lambda\boldsymbol{\omega}$ balance out on average, then so must $\boldsymbol{u}$ and $\boldsymbol{d}$.

In many cases $\boldsymbol{u}$ is orthogonal to the weights on average due to scale-invariance or randomness. Otherwise, we can split it into an orthogonal $\boldsymbol{u}_\perp$ and a radial (outwards) $\boldsymbol{u}_\parallel$ component. The $\boldsymbol{u}_\parallel$ term then has a similar effect as the weight decay term $\boldsymbol{d}$ which is anti-parallel to the weights in the cases we consider. If we can obtain an expression for the orthogonal $\|\boldsymbol{u}_\perp\|$ and radial $\|\boldsymbol{d} - \boldsymbol{u}_\parallel\|$ total update contributions, we can apply the Pythagorean theorem to the dashed triangle in fig. 2R:

$$(\widehat{\|\boldsymbol{\omega}\|} - \mathbb{E}[\|\boldsymbol{d} - \boldsymbol{u}_\parallel\|])^2 + \mathbb{E}[\|\boldsymbol{u}_\perp\|]^2 = \widehat{\|\boldsymbol{\omega}\|}^2 \tag{8}$$

We can then solve for $\widehat{\|\omega\|}$, making sure to account for the dependency of $\boldsymbol{u}$ and $\boldsymbol{d}$ on the weight norm.

Once we have an expression for $\widehat{\|\omega\|}$, we can compute a prediction for the corresponding RMS update size $\widehat{\eta_g} = \sqrt{E[\|\Delta_g\boldsymbol{\omega}\|^2]}$. This gives a prediction for the expected relative update size $\widehat{\eta_g}/\widehat{\|\omega\|} := \widehat{\eta_r}$, which closely approximates the angular update η_r in equilibrium. We do this for AdamW in the next subsection and for SGDM, Lion (Chen et al., 2023) and Adam with ℓ_2-regularization in appx. C, D and E. The resulting predictions for the update dynamics of each optimizer are summarized in table 1.

3.2. AdamW Equilibrium

We write AdamW (Loshchilov & Hutter, 2019) updates as:

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \tag{9}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \tag{10}$$

$$\boldsymbol{p}_t = \boldsymbol{p}_{t-1} - \eta \cdot \left(\frac{\boldsymbol{m}_t/(1-\beta_1^t)}{\sqrt{\boldsymbol{v}_t/(1-\beta_2^t)}+\varepsilon} + \lambda \boldsymbol{p}_{t-1}\right) \tag{11}$$

Table 1: Analytical predictions for the steady state neuronal update dynamics of different optimizers. The $\widehat{\eta_g}$ values apply to any parameter $\boldsymbol{p} \in \mathbb{R}^C$ with gradient $\boldsymbol{g}$. The $\widehat{\eta_r}$ and $\widehat{\|\boldsymbol{\omega}\|}$ values apply to scale-invariant weights $\boldsymbol{\omega} \in \mathbb{R}^C$ in equilibrium. Expressions with $\tilde{\boldsymbol{g}} := \|\boldsymbol{p}\|\boldsymbol{g}$ use the inverse proportionality from eq. (6) and would therefore differ without scale-invariance.

	SGDM (45)	AdamW (11)	Adam+ℓ_2 (78)	Lion (57)
RMS update size $\widehat{\eta_g}$	$\eta\sqrt{\frac{\mathbb{E}[\|\boldsymbol{g}\|^2]}{1-\alpha^2}}$	$\eta\sqrt{C\frac{1-\beta_1}{1+\beta_1}}$	$\eta\sqrt{C\frac{1-\beta_1}{1+\beta_1}}$	$\eta\sqrt{C}$
Expected rotation $\widehat{\eta_r}$	$\sqrt{\frac{2\eta\lambda}{1+\alpha}}$	$\sqrt{2\eta\lambda\frac{1-\beta_1}{1+\beta_1}}$	$\sqrt[3]{\frac{2\eta^2\lambda}{\langle\mathbf{1},\sqrt{\mathbb{E}[\tilde{\boldsymbol{g}}^2]}\rangle}\sqrt{\frac{1-\beta_1}{1+\beta_1}C}}$	$\sqrt{\pi\eta\lambda}\left((1-\beta_1)^2+\beta_1^2\frac{1-\beta_2}{1+\beta_2}\right)^{\frac{1}{2}}$
Equilibrium norm $\widehat{\|\boldsymbol{\omega}\|}$	$\sqrt[4]{\frac{\eta\mathbb{E}[\|\tilde{\boldsymbol{g}}\|^2]}{2\lambda\cdot(1-\alpha)}}$	$\sqrt{\frac{\eta C}{2\lambda}}$	$\sqrt[3]{\frac{\eta}{2\lambda}\cdot\langle\mathbf{1},\sqrt{\mathbb{E}[\tilde{\boldsymbol{g}}^2]}\rangle}$	$\sqrt{\frac{\eta C}{\pi\lambda}}\left((1-\beta_1)^2+\beta_1^2\frac{1-\beta_2}{1+\beta_2}\right)^{-\frac{1}{2}}$

where $\boldsymbol{p}_t \in \mathbb{R}^C$ is a parameter vector at iteration t, $\boldsymbol{g}_t = \nabla_{\boldsymbol{p}}\mathscr{L}(\boldsymbol{p}_t, \ldots)$ is the gradient, $\boldsymbol{m}$ is the first moment and $\boldsymbol{v}$ is the second moment. The learning rate $\eta \geq 0$, weight decay $\lambda \geq 0$ (zero expect for weight vectors), moment coefficients $\beta_1, \beta_2 \in (0, 1)$ and $\varepsilon \geq 0$ are hyperparameters. For simplicity we assume that ε and the bias correction can be ignored, i.e. that ε, β_1^t and β_2^t are all ≈ 0.

Equilibrium Magnitude: When applying AdamW to a weight vector $\boldsymbol{\omega} \in \mathbb{R}^C$, the total update contributions are:

$$\boldsymbol{u} = -\eta \sum_{k=t}^{\infty} \beta_1^{k-t}(1-\beta_1)\frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}_k}}, \quad \boldsymbol{d} = -\eta\lambda\boldsymbol{\omega} \tag{12}$$

We note that due to symmetry, each coordinate of $\boldsymbol{u}$ has a zero-mean distribution in the random walk setup. Since $\boldsymbol{u}$ is independent from $\boldsymbol{\omega}$, this makes them orthogonal in expectation i.e. $\mathbb{E}[\langle \boldsymbol{u}, \boldsymbol{\omega}\rangle] = 0$. It is also reasonable to assume that the variance of each coordinate remains constant when the gradient distribution is not changing over time, resulting in $\forall t, k : \mathbb{E}[\|\boldsymbol{g}_t/\sqrt{\boldsymbol{v}_k}\|^2] = C$ (the vector dimension) and therefore $\mathbb{E}[\|\boldsymbol{u}\|^2] = \eta^2 C$. Defining $\omega = \|\boldsymbol{\omega}\|$, $u = \|\boldsymbol{u}\|$, $u_\| = \langle\boldsymbol{\omega}, \boldsymbol{u}\rangle/\|\boldsymbol{\omega}\|$, $u_\perp^2 = u^2 - u_\|^2$ and $d = \|\boldsymbol{d}\|$ we can write a recurrence relation based on eq. (8):

$$\begin{aligned}\mathbb{E}[\omega_{i+1}^2] &= \mathbb{E}[(\omega_i - d + u_\|)^2 + u_\perp^2] && (13)\\ &= \mathbb{E}[\omega_i^2 - 2d\omega_i + 2u_\|\omega_i - 2du_\| \\ &\quad + d^2 + u_\|^2 + (u^2 - u_\|^2)] && (14)\\ &= \mathbb{E}[\omega_i^2](1 - 2\eta\lambda + \eta^2\lambda^2) + \eta^2 C && (15)\end{aligned}$$

where we used independence, $\mathbb{E}[u_\|] = 0$, and $\mathbb{E}[u] = \eta^2 C$. The solution is:

$$\begin{aligned}\mathbb{E}[\omega_i^2] &= \mathbb{E}[\omega_0^2]a^i + \tfrac{\eta^2 C}{2\eta\lambda - \eta^2\lambda^2}(1 - a^i) && (16)\\ a &= 1 - 2\eta\lambda + \eta^2\lambda^2 && (17)\end{aligned}$$

The recurrence relation is written in terms of the $\boldsymbol{u}$ and $\boldsymbol{d}$ instead of $\Delta_g\boldsymbol{\omega}$ and $\Delta_\lambda\boldsymbol{\omega}$. This is thus only an approximation of how the real system converges to equilibrium over time, but still informative. It may be a good approximation if $\|\boldsymbol{\omega}\|$ changes slowly compared to how fast $\boldsymbol{u}$ is applied and $\boldsymbol{v}$ is updated, i.e. when β_1, β_2 are low compared to a. The limit $i \to \infty$ gives us the equilibrium norm listed in table 1:

$$\widehat{\|\boldsymbol{\omega}\|} = \sqrt{\tfrac{\eta C}{2\lambda - \eta\lambda^2}} \approx \sqrt{\tfrac{\eta C}{2\lambda}} \qquad \text{(for } \lambda\eta \ll 2\text{)} \tag{18}$$

Expected Update Size: We can estimate the RMS update size η_g of a parameter $\boldsymbol{p} \in \mathbb{R}^C$ with $\Delta_g\boldsymbol{p} = \frac{\eta\boldsymbol{m}_t}{\sqrt{\boldsymbol{v}_t}}$ as follows:

$$\begin{aligned}\mathbb{E}[\|\Delta_g\boldsymbol{p}\|^2] &= \mathbb{E}[\|\tfrac{\eta}{\sqrt{\boldsymbol{v}_t}}(1-\beta_1)\textstyle\sum_{k=0}^{t-1}\beta_1^{t-k}\boldsymbol{g}_{t-k}\|^2] && (19)\\ &= \eta^2(1-\beta_1)^2\textstyle\sum_{k=0}^{t-1}\beta_1^{2t-2k}\mathbb{E}[\|\tfrac{\boldsymbol{g}_{t-k}}{\sqrt{\boldsymbol{v}_t}}\|^2] && (20)\\ &\approx \eta^2\tfrac{1-\beta_1}{1+\beta_1}C && (21)\end{aligned}$$

where we have approximated the geometric sum with its limit $t \to \infty$, used the fact that for the random walk $\forall j \neq k : \mathbb{E}[\langle\boldsymbol{g}_j, \boldsymbol{g}_k\rangle] = 0$ as well as our previous assumption $\forall t, k : \mathbb{E}[\|\boldsymbol{g}_t/\sqrt{\boldsymbol{v}_k}\|^2] = C$. This gives us the prediction $\widehat{\eta_g} \approx \mathbb{E}[\sqrt{\|\Delta_g\boldsymbol{p}\|^2}]$ listed in table 1. Approximating the equilibrium angular update size with the expected relative update size $\sqrt{\mathbb{E}[\|\Delta_g\boldsymbol{\omega}\|^2]}/\widehat{\|\boldsymbol{\omega}\|}$ gives the $\widehat{\eta_r}$ value. This approximation is good for small relative updates and a relatively small radial component in $\Delta_g\boldsymbol{\omega}$.

3.3. AdamW vs Adam with ℓ_2-Regularization

Loshchilov & Hutter (2019) proposed the use of decoupled weight decay instead of ℓ_2-regularization in Adam (Kingma & Ba, 2015). In their experiments they find that Adam with decoupled weight decay (i.e. AdamW, see eq. (11)) outperforms the ℓ_2-regularized form (i.e. Adam+ℓ_2, eq. (78)) across a wide range of settings. Since then AdamW has been widely adopted, but as far as we know the reason for its effectiveness over Adam+ℓ_2 is not well understood.

Our analysis of Adam+ℓ_2 in appx. E reveals that both the equilibrium norm and angular update size depend on the gradient magnitude (through $\tilde{\boldsymbol{g}}$) unlike AdamW, see table 1. When the gradient norm varies between neurons or layers, this results in imbalanced rotation. We believe the balanced vs imbalanced equilibrium rotation is a key difference between AdamW and Adam+ℓ_2, which may explain why decoupled weight decay is more effective for Adam-like methods. In our experiments (§5.3) we explore this further along with the general impact of imbalanced rotation.

Algorithm 1 Rotational Wrapper for constrained dynamics.

```
Require: Inner optimizer F, decay factor $0 \leq \beta < 1$, $\varepsilon \geq 0$ for
         numerical stability, iteration count $T$, rotational set $\Omega$
 1: for $\boldsymbol{p}$ in $\Omega$ :                    # For each neuronal weight vector
 2:     $\nu_{\boldsymbol{p}} \leftarrow 0$                # Initialize the update RMS tracker
 3:     $n_{\boldsymbol{p}} \leftarrow \|\boldsymbol{p}\|$       # Save the initial magnitude
 4:     $\boldsymbol{p} \leftarrow n_{\boldsymbol{p}} \cdot \frac{\boldsymbol{p}-\bar{\boldsymbol{p}}}{\|\boldsymbol{p}-\bar{\boldsymbol{p}}\|}$   # Remove the mean component of $\mathbf{p}$
 5: for $t \in \{1, ..., T\}$ :
 6:     Perform backpropagation, obtain gradients for all params
 7:     for all $\boldsymbol{p}$ :                          # For each parameter
 8:         # Get update components (§ 3.1):
 9:         $\Delta_g\boldsymbol{p}, \Delta_\lambda\boldsymbol{p} \leftarrow$ F.get_update$(\boldsymbol{p}, \nabla_{\boldsymbol{p}}\mathscr{L}(\boldsymbol{p}, \ldots))$
10:         if $\boldsymbol{p} \in \Omega$ :                # If $\mathbf{p}$ is a neuronal weight vector
11:             $\Delta_g\boldsymbol{p} \leftarrow \Delta_g\boldsymbol{p}/\eta$          # Undo $\eta$ used in F
12:             # Remove projections, s.t. $\Delta_g\mathbf{p} \perp \mathbf{p}$, $\Delta_g\mathbf{p} \perp \mathbf{1}$:
13:             $\Delta_g\boldsymbol{p} \leftarrow \Delta_g\boldsymbol{p} - \frac{\langle \Delta_g\boldsymbol{p}, \boldsymbol{p}\rangle}{\|\boldsymbol{p}\|^2}\boldsymbol{p} - \frac{\langle \Delta_g\boldsymbol{p}, \mathbf{1}\rangle}{\|\mathbf{1}\|^2}\mathbf{1}$
14:             # Update RMS tracker:
15:             $\nu_{\boldsymbol{p}} \leftarrow \beta \cdot \nu_{\boldsymbol{p}} + (1-\beta) \cdot \|\Delta_g\boldsymbol{p}\|^2$
16:             # Rotate $\mathbf{p}$ by $\widehat{\eta_r}$ from table 1 on average:
17:             $\boldsymbol{p} \leftarrow \boldsymbol{p} + \widehat{\eta_r} \cdot n_{\boldsymbol{p}} \cdot \frac{\Delta_g\boldsymbol{p}}{\sqrt{\nu_{\boldsymbol{p}}/(1-\beta^t)}+\varepsilon}$
18:             # Normalize $\mathbf{p}$ to the initial magnitude:
19:             $\boldsymbol{p} \leftarrow n_{\boldsymbol{p}} \cdot \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}$
20:         else:                                # $\mathbf{p}$ is not a neuronal weight vector
21:             $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta_g\boldsymbol{p} + \Delta_\lambda\boldsymbol{p}$     # Standard update
```

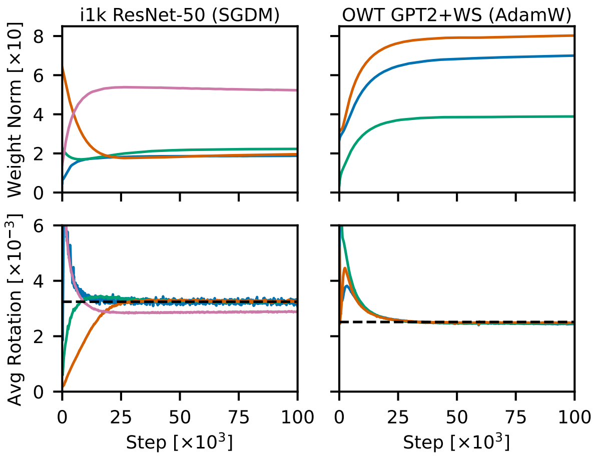

Figure 3: Measured weight norms and average rotation for different layers (solid colors) in two real neural network training tasks, ResNet-50 on ImageNet-1k (SGDM) and Weight Standardized GPT2-124M on OpenWebText (AdamW). The predicted equilibrium rotation (dashed black) from table 1 holds very well for all scale-invariant layers. The final fully-connected layer in RN-50 (pink) is not scale-invariant with a radial gradient component that decreases the effective weight decay, slowing the rotation (see §3.4). The learning rate is constant for easier comparison.

3.4. Rotational Dynamics of Scale-Sensitive Parameters

Prior work has primarily focused on the dynamics of scale-invariant weights. Note that any weight vector can be made scale-invariant by simply applying normalization to it, e.g. Weight Standardization (Qiao et al., 2019). For a random walk the gradient component is always orthogonal in expectation, but for real tasks scale-sensitive weights can have an average radial gradient component $\mathbb{E}[\boldsymbol{u}_{\|}] \neq 0$ (fig. 2R). In appx. H we explore how this affects the rotational dynamics of these weights (for SGDM). We find that a radial gradient component acts like an additional weight decay $\lambda_u = -\mathbb{E}[\boldsymbol{u}_{\|}]/(\eta\|\boldsymbol{\omega}\|)$ giving a new "effective" weight decay of $\lambda_e = \lambda + \lambda_u$, resulting in update dynamics similar to scale-invariant weights with the adjusted value.

4. Rotational Variants of Optimizers (RVs)

Our analysis shows how weight decay causes standard optimizers to transition towards equilibrium over time. In the steady state, weight decay balances the rotation across weight vectors and scales η_r relative to η_g. The same effect can be achieved without weight decay by directly controlling the average angular update as shown in algo. 1. We keep the weight magnitude constant and optionally introduce a learnable gain to compensate, which can matter for scale-sensitive weights and avoids numerical issues. These ***Rotational Variants (RVs)*** of existing optimizers serve as valuable research tools that allow us to verify our understanding of weight decay and perform ablation studies. Since RVs mimic the steady-state behavior of standard optimizers we expect similar performance. However, the update dynamics also differ in certain ways:

- The RVs can perfectly balance the average rotation without relying on scale-invariance from e.g. normalization.
- The RVs are always in equilibrium so there is no transient-phase. This means that the specified learning rate schedule directly controls η_r unlike in standard optimizers.

This simplifies the optimization dynamics and makes them more robust to architectural choices such as normalization. Note that the RVs closely resemble relative optimizers like LARS (You et al., 2017) and LAMB (You et al., 2020), revealing how they relate to and differ from standard optimizers. The RVs are further described in appx. I.

5. Experiments & Discussion

In this section we experimentally validate our analysis of the neuronal update dynamics and explore their impact on training. See appx. K for experimental setup and details.

5.1. Measuring & Constraining the Update Dynamics

In §3 we derived how weight decay affects the neuronal update dynamics of a network undergoing a carefully constructed analytically tractable random walk. Here we show that these dynamics occur in practical problems and that they suffice for obtaining the benefits of weight decay.

Table 2: Test set performance (mean±std) over three seeds for the baseline optimizer, AdamW, and its rotational variant (RV). We use the baseline hyperparameters directly for the no weight decay ($\lambda = 0$) and zero-shot results, but do minor tuning for the few-shot RV results where needed. The performance parity of the RV suggests that the benefits of weight decay, which are clear from the baseline degradation without it, can be achieved by directly controlling angular updates.

				AdamW		**RV-AdamW**	
Dataset	**Model**	**Batch Size**	**Metric** (⇅)	**Baseline**	$\boldsymbol{\lambda = 0}$	**(zero-shot)**	**(few-shot)**
CIFAR-10	ResNet-20	128	Top-1 Acc. [%] (↑)	92.2±0.2	91.1±0.3	92.4±0.3	N/A
CIFAR-10	ResNet-20	2048	Top-1 Acc. [%] (↑)	91.5±0.3	90.4±0.3	91.4±0.3	92.2±0.3
Imagenet-1k	DeiT tiny	1024	Top-1 Acc. [%] (↑)	72.1	69.3	71.3	72.5
IWSLT2014 de-en	Transformer-S	4096	Bleu (↑)	34.6±0.1	34.6±0.1	20.0±0.1	34.7±0.1
Wikitext	GPT2-124M	165×512	Perplexity (↓)	19.6±0.1	*unstable*	19.0±0.2	N/A
OpenWebText	GPT2-124M	480×1024	Loss (↓)	3.18±0.02	3.21±0.01	3.18±0.01	N/A
OpenWebText (25k)	GPT2-124M	480×1024	Loss (↓)	2.94±0.01	*unstable*	2.93±0.01	N/A

Measurements: Figure 3 shows the weight norm and average angular updates over time for several layers in RN-50 and a GPT2 variant. For scale-invariant layers, the average rotation converges to the predicted equilibrium rotation $\widehat{\eta_r}$ over time as anticipated. This value depends solely on the hyperparameters with no dependency on other unknown or time-varying quantities such as the gradient magnitude.

For layers that are not scale-invariant, the equilibrium rotation is affected by the average radial gradient component as expected (§3.4), but this deviation is often not very significant as seen for the final FC layer in fig. 3. Note that transformers are not scale-invariant without additional tricks like Weight Standardization (WS), but we find the equilibrium rotation is still often close to the scale-invariant value (see appx. J, fig. 15 for GPT2 without WS).

For fig. 3 the length of the transient phase is small compared to the length of typical training. ResNet-50 was originally trained for 90 epochs (∼450k steps) and GPT2 for ∼600k steps. However, this need not be the case, and will also depend on the hyperparameters as predicted by eq. (16).

Constraining the Rotational Dynamics: Table 2 shows the impact of replacing weight decay in AdamW via algo. 1 (RV-AdamW) for various network architectures and tasks. To quantify the effect of weight decay on the baselines, we repeat them with the same hyperparameters aside from disabling weight decay, i.e. with $\lambda = 0$. Training without weight decay is similar to using a different learning rate schedule (Li & Arora, 2020). In particular, for scale-invariant weights, disabling weight decay is similar to multiplying the learning rate schedule for η_r with an exponentially decaying function. An example of this can be seen in fig. 14 in appx. J, which also shows the accompanying increase in the weight norms. The effects of this will vary with training length, the weight decay value used, the initial weight norms etc. Without weight decay GPT training eventually diverges (at least in bfloat16), which was also observed before by Andriushchenko et al. (2023).

Table 2 shows that in most cases the RV perform similarly to the original baseline using exactly the same hyperparameters (zero-shot). This is expected if the baseline training is dominated by the steady-state equilibrium phase that the RV is designed to model. This is not always the case, for example in the IWSLT2014 the weight decay of the baseline is too low to have a significant impact over the course of training. The resulting training never reaches the equilibrium the RV models, where the average rotation would be very low (remember that the weight decay value affects the convergence rate towards equilibrium, see eq. (16)). In such cases we also perform a "few-shot" experiment where we do light tuning of the weight decay parameter. We can roughly predict the modified value based on measurements of the observed rotation during the baseline training using table 1.

Appx. J shows experiments for other base optimizers. In all cases we are able to recover the baseline performance with the RV. This suggests that controlling the average angular update size is sufficient to obtain the benefits of weight decay. Note that our intention here is not to outperform the baselines, although we believe the insights from the RVs could help with this in the future.

5.2. Transient Effects & The Interaction of η and λ

In our analysis we describe two different update sizes, the angular update size η_r (for neuronal weight vectors) and the RMS update size η_g (for biases, gains etc). The steady state value of η_r depends on both the learning rate η and weight decay λ hyperparameters, while η_g is not directly affected by weight decay (which is not applied to biases etc). The initial η_r also only depends on η, but λ modulates it over time creating an induced "effective" learning rate schedule $\eta_r(t)$ that can deviate from the specified learning rate schedule $\eta(t)$. Here we explore these effects experimentally.

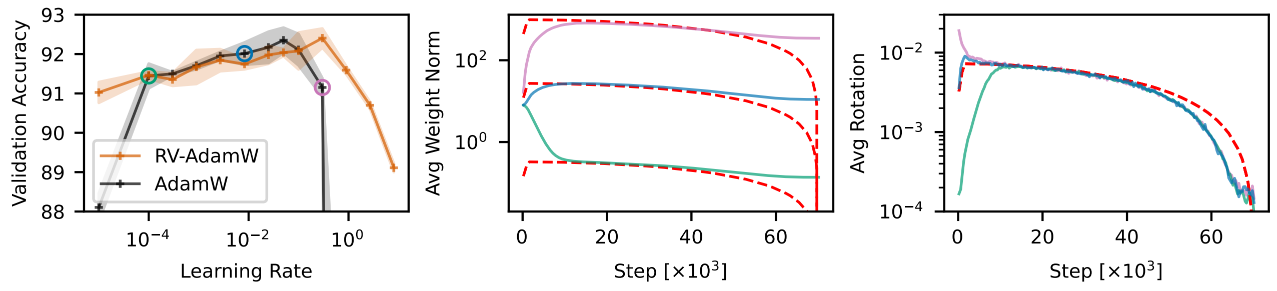

Figure 4: Weight decay influences transient behavior and how fast weights are updated relative to biases. **Left:** Validation accuracy for ResNet-20 on CIFAR-10 for learning rate, weight decay pairs with a constant product ($\eta\lambda = 5 \cdot 10^{-4}$) resulting in a specific $\widehat{\eta_r}$ but different $\widehat{\eta_g}$ and $\widehat{\|\omega\|}$ (table 1). **Middle/Right:** The weight norm $\|\boldsymbol{\omega}\|$ and angular update size η_r over time for three (η, λ) pairs corresponding to the colored circles on the left with equilibrium predictions in dashed red.

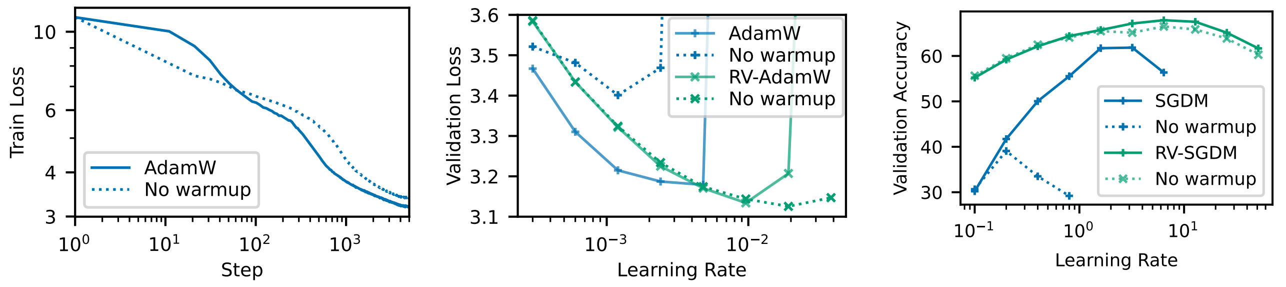

Figure 5: The RVs display a reduced need for learning rate warmup compared to standard optimizers, offering insights into the utility of warmups. **Left:** GPT2-124M OWT loss curve with/without learning rate warmup. **Middle:** GPT2-124M OWT final validation loss for different learning rates with AdamW/RV-AdamW and with/without warmup. **Right:** ResNet-50 i1k validation accuracy (short, large batch training) for different learning rates with SGDM/RV-SGDM, with/without warmup.

Learning Rate vs Weight Decay: Figure 4L shows the final network performance obtained for different (η, λ) pairs with a constant $\eta\lambda$ product and therefore the same equilibrium rotation value. With the RV, this is equivalent to changing the size of the bias updates via η_g, while keeping the rotation η_r constant. The performance varies quite a bit, the leftmost values at $\eta_g \approx 0$ roughly match the results with frozen gains/biases (91.1%), while the rightmost values have rapidly changing biases degrading performance. Although we don't show it here, the η_r value also matters significantly so η and λ jointly determine two distinct effective step sizes, η_r and η_g, both of which affect performance. Weight decay can therefore be seen as a scaling factor for the effective (equilibrium) update size of the weights (η_r) relative to other parameters such as biases (η_g). As a result we need to keep the λ hyperparameter in the RVs even if we don't actually decay the weights. The baseline optimizer (black in fig. 4L) shows a similar trend but is also affected by changes in the transient phase which we will look at next.

Transient Effects: Although the weight decay and learning rate have an identical effect on the equilibrium rotation $\widehat{\eta_r}$ for AdamW, they affect the norm differently (see table 1). In fig. 4M we plot the weight norm and equilibrium prediction for three different (η, λ) pairs corresponding to the colored circles in fig. 4L. These experiments use a cosine decay learning rate schedule and a five epoch warmup. Note how configurations with a higher learning rate (and lower weight decay) converge towards a higher equilibrium norm after the initial transient phase. Towards the later stages of training the weights fall back out of equilibrium as the weights can not decay fast enough to keep up with the shifting equilibrium norm (which decreases with the learning rate schedule). We expect longer training would keep the weights in equilibrium for proportionally longer.

Figure 4R shows the measured angular updates corresponding to the previous (η, λ) pairs. We note how the rotational behavior out of equilibrium is inverted compared to that of the norm. Weight norms below the equilibrium magnitude result in rotation that is faster than in equilibrium, and vice versa. This means that depending on the initialization magnitude of the weights compared to the equilibrium magnitude, we either observe overly fast or slow rotation during the initial transient phase. The purple example shows that even though we use a learning rate warmup, the induced "effective" update schedule of η_r can still include overly fast rotation. The RVs eliminate these transient phases causing differences at both the start and end of training which can be significant, but could be replicated by scheduling η or λ appropriately if desired.

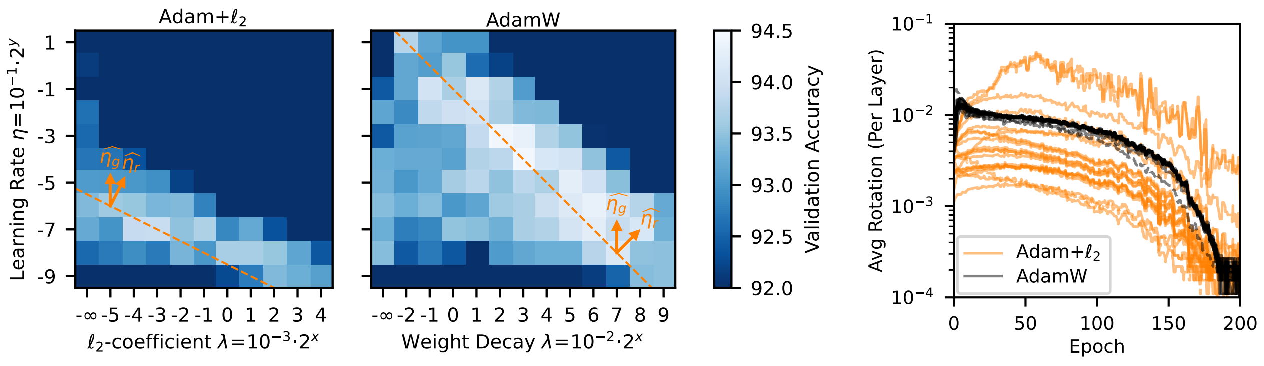

Figure 6: Balanced vs imbalanced rotation appears to be a key difference between AdamW and Adam+ℓ_2. **Left:** A sweep of the learning rate and ℓ_2-regularization / weight decay of Adam+ℓ_2 and AdamW on CIFAR-10 ResNet-18. Adam+ℓ_2 is unable to match the performance of AdamW. The orange dashed lines represent contour lines for $\widehat{\eta_r}$ based on table 1 along which $\widehat{\eta_g}$ varies, demonstrating how both update sizes matter. This dashed line is analogous to the sweep depicted in fig. 4. **Right:** The observed average rotation of each layer during training with the best setting for each optimizer. Here the rotation is affected by a cosine learning rate schedule (no warmup).

Need for Learning Rate Warmup: We conjecture that the irregular update dynamics observed in the initial transient phase can hinder optimization. In fig. 5L we show training curves for GPT2-124M OWT with and without a 5% learning rate warmup. The run without warmup is stable and makes faster initial progress but eventually falls behind, creating a loss gap that is not closed throughout training. Figure 5M compares the final validation loss with and without warmup for different learning rates for both AdamW and its rotational variant. The AdamW runs show a significant gap between the two, but the difference is minimal with the RV. We even observe the no-warmup runs being stable at slightly higher learning rates with the RV, but believe this is noise due to non-deterministic divergence.

In fig. 5R we do a similar experiment with 10 epoch ResNet-50 i1k training with a large batch size of 8k. We again observe that the baseline benefits significantly from warmups, but the RV only marginally (which could be due to the dynamics of other parameters like biases which the RV does not stabilize). We observed the same trend when extending the best runs to full 90 epochs (see appx. J).

This suggests learning rate warmup may aid training in part by stabilizing the transient phase, and that explicitly controlling the angular updates could offer similar benefits.

5.3. The Importance of Balanced Rotation

The previous sections have mostly glossed over the fact that equilibrium forms independently for various neurons and layers. In certain settings this leads to different (imbalanced) equilibrium rotation between network components. Here we explore the performance impact of this experimentally.

Adam vs AdamW: Our analysis revealed the the equilibrium rotation of Adam+ℓ_2 has a dependency on the gradient magnitude unlike AdamW. When the gradient norm differs between neurons/layers this should lead to imbalanced rotation (even for scale-invariant layers). Figure 6L reproduces the performance gap between AdamW and Adam+ℓ_2 observed by Loshchilov & Hutter (2019), around 0.5% on the validation set. Note how the structure of these heatmaps corresponds to changes in the two equilibrium update sizes $\widehat{\eta_r}$ and $\widehat{\eta_g}$, demonstrating how both matter in practice as suggested in §5.2. The original decoupled version of AdamW described by Loshchilov & Hutter (2019) differs from the one used in PyTorch and described here, replacing $\eta\lambda$ in the weight decay term of eq. (11) with just λ, but scaling both η and λ over time according to the learning rate schedule. In this case $\widehat{\eta_r}$ depends only on λ and $\widehat{\eta_g}$ on η, explaining the separability of the hyperparameter space they observe.

Figure 6R shows the rotation of different layers over training for the best configuration of each optimizer. In AdamW all layers behave similarly, even the scale-sensitive final FC layer (dashed) does not deviate significantly, unlike Adam+ℓ_2 where the rotation differs drastically. The $\sim$30× range in observed rotation corresponds to a factor of $\sim$1000× in the learning rate (table 1). In appx. J we show that enforcing balanced rotation in Adam+ℓ_2 roughly closes the gap. We also observe imbalanced rotation impeding training in other settings (see later), so we hypothesize that balanced equilibrium rotation is the main benefit of AdamW over Adam+ℓ_2.

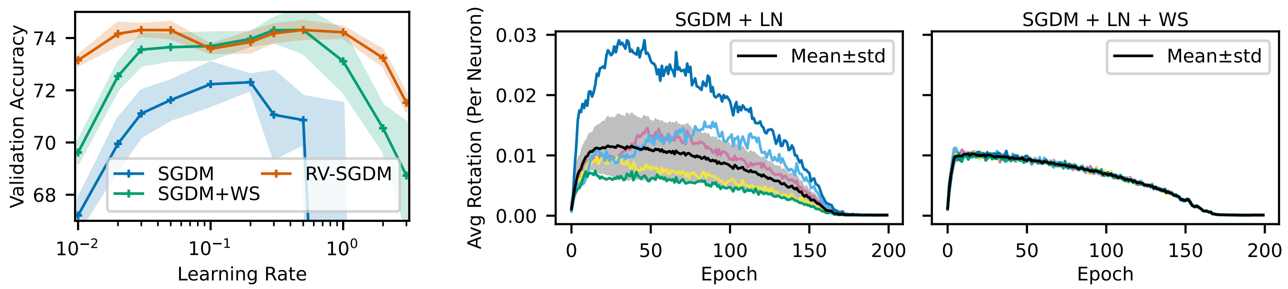

Figure 7: Balanced neuronal rotation obtained through e.g. RVs or Weight Standardization seems to aid training. **Left:** Final accuracy for a Layer-Normalized (LN) ResNet-18 on CIFAR-100 for different peak learning rates (cos+warmup schedule). Baseline compared with additional Weight Standardization (WS) and RV training. **Right:** The baseline has imbalanced rotation across neurons, WS balances them. Example neurons (each color) and mean±std (black/gray) shown for one layer.

The Benefit of Weight Standardization: Layer Normalization can make a whole layer scale-invariant but not individual neurons (unlike e.g. Batch Normalization). This means that the gradient is orthogonal to the weights of the whole layer (see eq. (5)) but individual neurons may have a non-zero radial component. In fact, the normalized output magnitude of one neuron can be increased by decreasing the output magnitude of the other neurons in the layer and vice versa, potentially encouraging radial gradient components on the neuron level. This changes the "effective" weight decay as discussed in §3.4, which can lead to imbalanced rotation between neurons. Weight Standardization (WS) makes each neuron scale invariant eliminating this effect. Note that WS is fully independent of the network inputs and can thus be seen as a reparameterization or an optimization trick. Other types of normalization like Batch Normalization can have additional effects like affecting signal propagation (Brock et al., 2021a) or causing a dropout-like regularization effect (Kosson et al., 2024).

Figure 7L shows the final accuracy achieved when training a Layer-Normalized ResNet-18 on CIFAR-100 for different learning rates. Training using either Weight Standardization or an RV significantly outperforms standard training. fig. 7R shows imbalanced neuronal rotation in the baseline which the WS eliminates as expected (the RV does as well). Weight Standardization thus appears to facilitate optimization by balancing the equilibrium dynamics across neurons, especially when applied on top of Layer Normalization or Group Normalization (Wu & He, 2018).

Imbalanced Rotation: We can modify the RVs to directly test the impact of imbalanced rotation without potential confounding effects. In appx. J (fig. 18) we show that an RV with artificially imbalanced neuronal rotation, where some neurons are intentionally rotated slower or faster, suffers from a performance degradation compared to a standard balanced version. This gap persists despite tuning the hyperparameters for each configuration. We observe that even small imbalances in the rotational speed (η_r) can significantly degrade performance.

6. Conclusion

In this work we have described how weight decay modulates the update dynamics of individual neurons and explored how their variance across components (balanced vs imbalanced) and time (transient vs equilibrium phases) impacts training. Despite their simplicity, neural update dynamics offer a surprisingly insightful perspective of neural network optimization and demystify the effectiveness of widely adopted techniques. We believe this area presents numerous opportunities for further theoretical and practical research.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

We thank Maksym Andriushchenko for insightful conversations related to this study. We also appreciate Alex Hagele, Dongyang Fan and Nikita Doikov for their feedback on the manuscript which enhanced its quality and clarity.

References

Andriushchenko, M., D'Angelo, F., Varre, A., and Flammarion, N. Why do we need weight decay in modern deep learning? *arXiv preprint arXiv:2310.04415*, 2023. URL `https://arxiv.org/abs/2310.04415`. 6, 13

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL `https://arxiv.org/abs/1607.06450`. 2

Brock, A., De, S., and Smith, S. L. Characterizing signal propagation to close the performance gap in unnormalized resnets. In *9th International Conference on Learning Representations, ICLR*, 2021a. arXiv:2101.08692. 9, 29

Brock, A., De, S., Smith, S. L., and Simonyan, K. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pp. 1059–1071. PMLR, 2021b. arXiv:2102.06171. 14, 29

Cettolo, M., Niehues, J., Stüker, S., Bentivogli, L., and Federico, M. Report on the 11th IWSLT evaluation campaign. In *Proceedings of the 11th International Workshop on Spoken Language Translation: Evaluation Campaign*, pp. 2–17, Lake Tahoe, California, December 4-5 2014. URL `https://aclanthology.org/2014.iwslt-evaluation.1`. 33

Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., Lu, Y., and Le, Q. V. 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. 3, 17

Chiley, V., Sharapov, I., Kosson, A., Koster, U., Reece, R., Samaniego de la Fuente, S., Subbiah, V., and James, M. Online normalization for training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. arXiv:1905.05894. 2, 3, 13, 17, 36

Fu, J., Wang, B., Zhang, H., Zhang, Z., Chen, W., and Zheng, N. When and why momentum accelerates sgd: An empirical study. *arXiv preprint arXiv:2306.09000*, 2023. URL `https://arxiv.org/abs/2306.09000`. 36

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 13

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. arXiv:1512.03385. 33

Heo, B., Chun, S., Oh, S. J., Han, D., Yun, S., Kim, G., Uh, Y., and Ha, J.-W. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Iz3zU3M316D`. arXiv:2006.08217. 13

Hoffer, E., Banner, R., Golan, I., and Soudry, D. Norm matters: efficient and accurate normalization schemes in deep networks. *Advances in Neural Information Processing Systems*, 31, 2018. arXiv:1803.01814. 13

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. URL `https://arxiv.org/abs/2203.15556`. 34

Huang, L., Liu, X., Lang, B., and Li, B. Projection based weight normalization for deep neural networks. *arXiv preprint arXiv:1710.02338*, 2017a. URL `https://arxiv.org/abs/1710.02338`. 14

Huang, L., Liu, X., Liu, Y., Lang, B., and Tao, D. Centered weight normalization in accelerating training of deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2803–2811, 2017b. URL `https://openaccess.thecvf.com/content_iccv_2017/html/Huang_Centered_Weight_Normalization_ICCV_2017_paper.html`. 2, 15, 29

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015. arXiv:1502.03167. 2, 14

Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018. arXiv:1807.11205. 1

Karpathy, A. nanogpt. `https://github.com/karpathy/nanoGPT/`, 2023. 33, 34

Karras, T., Aittala, M., Lehtinen, J., Hellsten, J., Aila, T., and Laine, S. Analyzing and improving the training dynamics of diffusion models. *arXiv preprint arXiv:2312.02696*, 2023. 14

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. arXiv:1412.6980. 4, 14, 19

Kodryan, M., Lobacheva, E., Nakhodnov, M., and Vetrov, D. P. Training scale-invariant neural networks on the sphere can happen in three regimes. *Advances in Neural Information Processing Systems*, 35:14058–14070, 2022. arXiv:2209.03695. 13

Kosson, A., Fan, D., and Jaggi, M. Ghost noise for regularizing deep neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(12):13274–13282, Mar. 2024. doi: 10.1609/aaai.v38i12.29228. URL `https://ojs.aaai.org/index.php/AAAI/article/view/29228`. arXiv:2305.17205. 9

Krizhevsky, A. Learning multiple layers of features from tiny images. *self-published*, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`. 33

Li, Z. and Arora, S. 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. 1, 2, 6, 13

Li, Z., Lyu, K., and Arora, S. 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. 13

Li, Z., Malladi, S., and Arora, S. On the validity of modeling SGD with stochastic differential equations (SDEs). In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=goEdyJ_nVQI`. arXiv:2102.12470. 3

Li, Z., Bhojanapalli, S., Zaheer, M., Reddi, S., and Kumar, S. Robust training of neural networks using scale invariant architectures. In *International Conference on Machine Learning*, pp. 12656–12684. PMLR, 2022a. arXiv:2202.00980. 14

Li, Z., Wang, T., and Yu, D. Fast mixing of stochastic gradient descent with normalization and weight decay. *Advances in Neural Information Processing Systems*, 35: 9233–9248, 2022b. URL `https://proceedings.neurips.cc/paper_files/paper/2022/hash/3c215225324f9988858602dc92219615-Abstract-Conference.html`. 13

Liu, Y., Bernstein, J., Meister, M., and Yue, Y. Learning by turning: Neural architecture aware optimisation. In *International Conference on Machine Learning*, pp. 6748–6758. PMLR, 2021. arXiv:2102.07227. 14

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`. arXiv:1711.05101. 2, 3, 4, 8, 13, 29, 34

Malladi, S., Lyu, K., Panigrahi, A., and Arora, S. On the SDEs and scaling rules for adaptive gradient algorithms. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=F2mhzjHkQP`. arXiv:2205.10287. 3

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018. 23

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Byj72udxe`. arXiv:1609.07843. 33

Neyshabur, B., Salakhutdinov, R. R., and Srebro, N. Path-sgd: Path-normalized optimization in deep neural networks. *Advances in neural information processing systems*, 28, 2015. arXiv:1506.02617. 27, 37

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019. arXiv:1904.01038. 33, 34

Pagliardini, M. llm-baseline. `https://github.com/epfml/llm-baselines`, 2023. 33, 34

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. arXiv:1912.01703. 22

Qiao, S., Wang, H., Liu, C., Shen, W., and Yuille, A. L. Weight standardization. *CoRR*, abs/1903.10520, 2019. URL `http://arxiv.org/abs/1903.10520`. 2, 5, 15, 29

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *self-published*, 2019. URL `https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`. 33

Roburin, S., de Mont-Marin, Y., Bursuc, A., Marlet, R., Perez, P., and Aubry, M. A spherical analysis of adam with batch normalization. *arXiv preprint arXiv:2006.13382*, 2020. URL `https://arxiv.org/abs/2006.13382`. 14

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. 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. 33

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016. arXiv:1602.07868. 13, 14

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. arXiv:1811.03600. 23

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.1556`. 27

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. Training data-efficient image transformers & distillation through attention. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/touvron21a.html`. arXiv:2012.12877. 33

Van Laarhoven, T. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017. URL `https://arxiv.org/abs/1706.05350`. 1, 2, 13

Wan, R., Zhu, Z., Zhang, X., and Sun, J. Spherical motion dynamics: Learning dynamics of normalized neural network using sgd and weight decay. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 6380–6391. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/326a8c055c0d04f5b06544665d8bb3ea-Paper.pdf`. arXiv:2006.08419. 1, 2, 13, 16, 17

Wightman, R. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 33, 34

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018. arXiv:1803.08494. 9, 34

Xie, Z., zhiqiang xu, Zhang, J., Sato, I., and Sugiyama, M. On the overlooked pitfalls of weight decay and how to mitigate them: A gradient-norm perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=vnGcubtzR1`. arXiv:2011.11152. 13

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017. URL `https://arxiv.org/abs/1708.03888`. 2, 5, 14

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Syx4wnEtvH`. arXiv:1904.00962. 5, 14

Zhang, G., Wang, C., Xu, B., and Grosse, R. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1lz-3Rct7`. arXiv:1810.12281. 1, 13

Zhou, Y., Sun, Y., and Zhong, Z. Fixnorm: Dissecting weight decay for training deep neural networks. *arXiv preprint arXiv:2103.15345*, 2021. URL `https://arxiv.org/abs/2103.15345`. 13

A. Expanded Related Work

In this section we discuss more related works divided into five main categories.

A.1. Scale-Invariance and Effective Learning Rates

Several works have investigated how the scale-invariance results in a certain "effective learning rate" based on the relative change that varies based on the norm of the weights, often in the form of $\eta/\|\boldsymbol{\omega}\|^2$. The works in this section do not describe how $\|\boldsymbol{\omega}\|$ can converge to an equilibrium value that results in a fixed relative or rotational learning rate. In Weight Normalization, Salimans & Kingma (2016) point out how normalization can make parameters scale-invariant and that the gradient magnitude varies based on the weight magnitude. They describe how the gradient can "self-stabilize its norm", with larger gradients becoming smaller over time due to growth in the weight magnitude, but do not consider the effects of weight decay on this process. Zhang et al. (2019) and Hoffer et al. (2018) empirically find that the regularization effects of weight decay are primarily caused by increases in the effective learning rate due to decreased weight norms. Li & Arora (2020) show that weight decay can be replaced by an exponentially increasing learning rate when optimizing scale-invariant weights with SGDM.

A.2. Equilibrium

The works in this section also consider the fact that the weight norm converges to a specific value and they explore the resulting effects on the relative update size. Van Laarhoven (2017) points out the scale-invariance property of normalization and how it interacts with ℓ_2-regularization. They derive the $\eta/\|\boldsymbol{\omega}\|^2$ as the effective learning rate and show there exists a fixed point where the weight norms are stable. Their work does not consider convergence of the weight magnitude as a separate process from the overall convergence of the loss and weights. In Online Normalization, Chiley et al. (2019) show a simple derivation of the equilibrium condition in SGD and how it results in a relative update size that is identical across layers. The Spherical Motion Dynamics (SMD) (Wan et al., 2021) expands on prior work by deriving the convergence of the weight norm and extending the analysis to include momentum. They also show plots of the weight norm over the course of training, providing empirical evidence for early convergence of the weight norm and note how it can fall out of equilibrium with sudden learning rate changes or when the learning rate becomes too small. They also consider the angular updates, empirically and analytically showing that they converge to an equilibrium value. Li et al. (2020) analyze the convergence of SGD to equilibrium by modelling it as a stochastic differential equation, arriving at similar conclusion as the SMD paper (without momentum). This is expanded upon by Li et al. (2022b).

A.3. Understanding and Improving Weight Decay

In traditional literature (e.g. Goodfellow et al. (2016)), ℓ_2-regularization is introduced as a simple and commonly used regularization method achieved by adding $\frac{1}{2}\lambda\|\boldsymbol{\omega}\|^2$ to the objective function. This strategy is commonly referred to as weight decay, but this is confusing since it is technically different from directly decaying the weights as pointed out by Loshchilov & Hutter (2019). As previously mentioned, Van Laarhoven (2017) showed that this interpretation does not hold for modern networks with normalization layers. Normalization can make a weight vector scale-invariant, meaning that the network output is unaffected by the magnitude of the vector (see §2). In this work, we focus on building upon the insight that weight decay serves as a scaling factor for an "effective" learning rate, as suggested by previous research (Van Laarhoven, 2017; Zhang et al., 2019; Li & Arora, 2020; Wan et al., 2021). However, other lines of work have explored different facets of understanding and improving weight decay. Xie et al. (2023) have aimed at mitigating pitfalls of weight decay and improving optimization performance. They propose a weight decay schedule that ensures better convergence towards the end of training. Lastly, Andriushchenko et al. (2023) investigated weight decay's role in regularization. They specifically, examined how weight decay influences the implicit regularization of SGD noise and the *bias-variance trade-off*.

A.4. Projected Optimization

Some existing works remove weight decay and rely on projections of either the updates or weights instead. AdamP (Heo et al., 2021) orthogonalizes the update of the Adam and SGDM optimizers by removing the radial component of $\Delta_g\boldsymbol{\omega}$. The main reason for this is to avoid a rapid increase in the weight norm during the initial phases of training. Zhou et al. (2021) propose keeping the weight magnitude constant, projecting it onto a sphere after every step and removing the weight decay. Kodryan et al. (2022) analyze training using projections onto the unit sphere after every optimizer update. Both of these works consider the union of all scale-invariant weights. Fixing the norm of this total parameter vector has a similar effect as weight decay and will balance the effective learning rates for each scale-invariant group over time (but does not eliminate the transient phase). This differs from Huang et al. (2017a) which projects the weights of each neuron individually, which does generally not result in balanced rotation when used with SGD as in their case. Performing a similar operation with Adam or Lion could result in balanced rotation when the weights of different neurons have identical RMS values. Roburin et al. (2020) analyzes the spherical projection of the Adam optimization trajectory during standard training.

A.5. Relative Optimization

LARS (You et al., 2017) and LAMB (You et al., 2020) are variants of SGDM and AdamW that scale the update of a weight to be proportional to its norm (sometimes a clamped version of the weight norm). They apply this to linear and convolutional layer weights, keeping the original update for weights and biases. LARS and LAMB were proposed for large batch size training and found to work well there. Although they are not inspired by the Spherical Motion Dynamics, their form is quite similar to the Rotational Optimizer Variants (algo. 1) with a few important differences. The default form of the RVs is applied filter-wise, centers the weights and allows the update magnitude to vary between steps while keeping the average relative update constant. The RV also doesn't apply weight decay while LARS and LAMB consider it a part of the update and take into account when scaling the relative update. Finally, the RVs adjust the learning rate based on the rotational equilibrium value. This makes it more compatible with the underlying optimizer variants in terms of hyperparameters. One notable difference is the square root dependency on the relative updates in equilibrium, while LARS and LAMB are directly proportional. This means that any learning rate schedule for these optimizers is more similar to applying a squared version of this schedule to standard optimizers in equilibrium or the RVs. This does not fully eliminate the differences however, because changing the schedule also affects gains and biases where the update magnitude is directly proportional to the learning rate for all the optimizers and variants discussed here.

Nero (Liu et al., 2021) is another optimizer that applies relative updates that are directly proportional to the learning rate and weight magnitude. Like LARS and LAMB, Nero is not inspired by the neuronal update dynamics of standard optimizers with weight decay, and to the best of our knowledge their relationship has not been pointed out before. Like the RVs, Nero is applied filter-wise and centers the weights. Overall, Nero is similar to the SGDM RV without momentum and the hyperparameter mapping, but also applies Adam like updates to the gains and biases, using a separate learning rate. By making the relative updates directly proportional to the learning rate, it has the same learning rate scheduling differences as LARS and LAMB mentioned above. Nero lacks momentum which is something that we observed can hurt large batch size training (exploratory experiments not shown).

Instead of controlling the average relative update size, Brock et al. (2021b) and Li et al. (2022a) clip the maximum relative update size instead. The Adaptive Gradient Clipping from Brock et al. (2021b) is applied on a per filter basis and is constant throughout training, i.e. does not vary with the learning rate or weight decay. The clipping introduced in Li et al. (2022a) scales with the learning rate and weight decay in a manner inspired by the equilibrium norm for SGD. They seem to apply this globally (i.e., not per neuron or layer).

In parallel with this work, Karras et al. (2023) studied diffusion model training and proposed combining neuron-wise Weight Normalization (Salimans & Kingma, 2016), projections and the Adam algorithm (Kingma & Ba, 2015) for optimization. They also forgo weight decay, ending up with a training approach that is very similar to our RV-AdamW, showing this significantly improves the optimization of their networks. The main differences are that they do not use a separate parameter such as the weight decay coefficient to set the relative update size of non-weight parameters like gains and biases (which they remove entirely), do not center the weights, and in their case the rotation is proportional to the learning rate.

B. Normalization and Scale-Invariance

This section provides an overview of normalization and scale-invariance.

Setup: We use Batch Normalization (Ioffe & Szegedy, 2015) as an example of a normalization operation. Let $\boldsymbol{x} = \boldsymbol{Z}\boldsymbol{\omega}$ for $\boldsymbol{x} \in \mathbb{R}^{B\times 1}$, $\boldsymbol{\omega} \in \mathbb{R}^{C\times 1}$ and $\boldsymbol{Z} \in \mathbb{R}^{B\times C}$ correspond to a single output feature of a linear layer (i.e. a neuron). We can write the batch normalization of this feature as:

$$\hat{\boldsymbol{x}} = N(\boldsymbol{x}) = \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \qquad \mu = \frac{1}{B}\sum_{i=1}^{B} x_i, \qquad \sigma^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu)^2 \tag{22}$$

where $\boldsymbol{x} = [x_1, \ldots, x_B]^\top \in \mathbb{R}^B$ is a vector and $\varepsilon \geq 0$ is a small hyperparameter added for numerical stability. Backpropagation accurately treats μ and σ as functions of $\boldsymbol{x}$. When ε is sufficiently small to be ignored, the output of the normalization is not affected by a positive scaling of the input:

$$N(r\boldsymbol{x}) = (r\boldsymbol{x} - r\mu)/\sqrt{r^2\sigma^2 + \varepsilon} \approx (\boldsymbol{x} - \mu)/\sqrt{\sigma^2 + \varepsilon} = N(\boldsymbol{x}), \qquad r > 0 \tag{23}$$

If the training loss $\mathscr{L}$ does not depend on $\boldsymbol{x}$ in other ways than through $N(\boldsymbol{x})$, this makes $\boldsymbol{x}$ scale-invariant with respect to the loss, i.e. $\mathscr{L}(r\boldsymbol{\omega}) = \mathscr{L}(\boldsymbol{\omega})$ for $r > 0$. Note that although we sometimes write $\mathscr{L}(\boldsymbol{\omega})$ for brevity the loss generally depends on other weights and inputs as well, $\boldsymbol{\omega}$ is generally only a portion of the parameters used in the network, and could for example be a particular row in the weight matrix of a fully connected layer. Some normalization operations like Centered Weight Normalization (Huang et al., 2017b) a.k.a. Weight Standardization (Qiao et al., 2019) are performed directly on the weights instead of activations. This also makes the weight scale-invariant and in case of the aforementioned methods also makes $\nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}) \perp \mathbf{1}$.

Properties: Scale-invariance results in the properties stated in eq. (5) and eq. (6), repeated below:

$$\text{Gradient orthogonality:} \qquad \nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}) \perp \boldsymbol{\omega} \tag{24}$$
$$\text{Inverse proportionality:} \qquad \|\nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega})\| \propto \|\boldsymbol{\omega}\|^{-1} \tag{25}$$

Intuition: The first property is a result of the loss surface being invariant along the direction of $\boldsymbol{\omega}$. Hence the directional derivative of $\mathscr{L}(\boldsymbol{\omega})$ in the direction of $\boldsymbol{\omega}$ is zero:

$$\begin{aligned}\langle \nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}), \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\rangle &= \lim_{h\to 0} \frac{\mathscr{L}(\boldsymbol{\omega} + h\boldsymbol{\omega}/\|\boldsymbol{\omega}\|) - \mathscr{L}(\boldsymbol{\omega})}{h} && (26)\\ &= \lim_{h\to 0} \frac{\mathscr{L}(\boldsymbol{\omega}) - \mathscr{L}(\boldsymbol{\omega})}{h} && (27)\\ &= \lim_{h\to 0} \frac{0}{h} = 0 && (28)\end{aligned}$$

The second property is a result of the backpropagation through N, which scales the gradient by the factor used on the forward pass $1/\sqrt{\sigma^2 + \varepsilon} \approx \sigma^{-1}$ as if it were a constant, and the fact that $\sigma \propto \|\boldsymbol{\omega}\|$.

Backpropagation: The properties can also be shown using expressions for the backpropagation through the normalization layers. For completeness we include the learnable affine transformation that typically follows normalization operations:

$$\boldsymbol{y} = \gamma\hat{\boldsymbol{x}} + \beta \tag{29}$$

For the backpropagation we have:

$$\begin{aligned}&\nabla_{\gamma}\mathscr{L}(\boldsymbol{p}) = \langle \hat{\boldsymbol{x}}, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle && (30)\\ &\nabla_{\beta}\mathscr{L}(\boldsymbol{p}) = \langle \mathbf{1}_B, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle && (31)\\ &\nabla_{\boldsymbol{x}}\mathscr{L}(\boldsymbol{p}) = \tfrac{\gamma}{\sqrt{\sigma^2+\varepsilon}} \cdot \left[\nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p}) - \tfrac{1}{B}\langle \mathbf{1}_B, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle \mathbf{1}_B - \tfrac{1}{B}\langle \hat{\boldsymbol{x}}, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle \hat{\boldsymbol{x}}\right] && (32)\end{aligned}$$

Assuming that ε is small gives:

$$\nabla_{\boldsymbol{x}}\mathscr{L}(\boldsymbol{p}) = \tfrac{\gamma}{\sigma}\left[\nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p}) - \tfrac{1}{B}\langle \mathbf{1}_B, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle \mathbf{1}_B - \tfrac{1}{B}\langle \hat{\boldsymbol{x}}, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle \hat{\boldsymbol{x}}\right] \tag{33}$$

In this case we have:

$$\begin{aligned}\langle \nabla_{\boldsymbol{x}}\mathscr{L}(\boldsymbol{p}), \mathbf{1}_B\rangle &= \tfrac{\gamma}{\sigma}\left[\langle \mathbf{1}_B, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle - \tfrac{1}{B}\langle \mathbf{1}_B, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle \underbrace{\langle \mathbf{1}_B, \mathbf{1}_B\rangle}_{=B} - \tfrac{1}{B}\langle \hat{\boldsymbol{x}}, \nabla_{\boldsymbol{y}}\mathscr{L}(\boldsymbol{p})\rangle \underbrace{\langle \hat{\boldsymbol{x}}, \mathbf{1}_B\rangle}_{=0}\right]\\ &= 0 && (34)\end{aligned}$$

and similarly:

$$\begin{aligned}\langle \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}), \hat{\boldsymbol{x}} \rangle &= \tfrac{\gamma}{\sigma} \Bigg[\langle \hat{\boldsymbol{x}}, \nabla_{\boldsymbol{y}} \mathscr{L}(\boldsymbol{p}) \rangle - \tfrac{1}{B} \langle \mathbf{1}_B, \nabla_{\boldsymbol{y}} \mathscr{L}(\boldsymbol{p}) \rangle \underbrace{\langle \mathbf{1}_B, \hat{\boldsymbol{x}} \rangle}_{=0} - \tfrac{1}{B} \langle \hat{\boldsymbol{x}}, \nabla_{\boldsymbol{y}} \mathscr{L}(\boldsymbol{p}) \rangle \underbrace{\langle \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}} \rangle}_{=B} \Bigg] \\ &= 0 \end{aligned} \tag{35}$$

which gives:

$$\langle \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}), \boldsymbol{x} \rangle = \langle \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}), \sigma \hat{\boldsymbol{x}} + \mu \mathbf{1}_B \rangle = 0 \tag{36}$$

This allows us to obtain the properties of the weight gradient:

$$\nabla_{\boldsymbol{p}} \mathscr{L}(\boldsymbol{p}) = \boldsymbol{Z}^\top \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}) \tag{37}$$

First we note that:

$$\|\nabla_{\boldsymbol{p}} \mathscr{L}(\boldsymbol{p})\| \propto \|\nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p})\| \propto \sigma^{-1} \propto \|\boldsymbol{p}\|^{-1} \tag{38}$$

where the second proportionality follows from eq. (33) and the final one from eq. (22). This gives the inverse proportionality listed in eq. (25).

We can also derive the gradient orthogonality in eq. (24) as follows:

$$\begin{aligned} \langle \nabla_{\boldsymbol{p}} \mathscr{L}(\boldsymbol{p}), \boldsymbol{p} \rangle &= \langle \boldsymbol{Z}^\top \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}), \boldsymbol{p} \rangle && (39) \\ &= \boldsymbol{p}^\top \boldsymbol{Z}^\top \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}) && (40) \\ &= \boldsymbol{x}^\top \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}) && (41) \\ &= \langle \nabla_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{p}), \boldsymbol{x} \rangle && (42) \\ &= 0 && (43) \end{aligned}$$

These properties can also be shown directly from the scale-invariance using calculus theorems as done in Wan et al. (2021).

C. SGDM Equilibrium

The standard version of SGD with momentum (SGDM) can be written as:

$$\begin{aligned} \boldsymbol{m}_t &= \alpha \boldsymbol{m}_{t-1} + \boldsymbol{g}_t + \lambda \boldsymbol{p}_{t-1} && (44) \\ \boldsymbol{p}_t &= \boldsymbol{p}_{t-1} - \eta \cdot \boldsymbol{m}_t && (45) \end{aligned}$$

where $\boldsymbol{p}_t \in \mathbb{R}^C$ is a parameter vector at time t, $\boldsymbol{g}_t = \nabla_{\boldsymbol{p}} \mathscr{L}(\boldsymbol{p}_t)$ is the gradient and $\boldsymbol{m}$ is the first moment (i.e. momentum or velocity). The learning rate ($\eta \geq 0$), weight decay ($\lambda \geq 0$), and momentum coefficient ($0 < \alpha < 1$) are hyperparameters.

We compute the total update contribution due to $\boldsymbol{g}_t$, i.e. $\boldsymbol{u}$ in eq. (8) as:

$$\boldsymbol{u} = -\eta \textstyle\sum_{k=t}^{\infty} \alpha^{k-t} \boldsymbol{g}_t = \frac{-\eta}{1-\alpha} \boldsymbol{g}_t \tag{46}$$

Analogously, the total update contribution of the ℓ_2-regularization of $\boldsymbol{w}_{t-1}$, i.e. $\boldsymbol{d}$ in eq. (8), is:

$$\boldsymbol{d} = -\eta \textstyle\sum_{k=t}^{\infty} \alpha^{k-t} \lambda \boldsymbol{p}_{t-1} = \frac{-\eta\lambda}{1-\alpha} \boldsymbol{p}_{t-1} \tag{47}$$

Combining eq. (46) and eq. (47), this allows us to solve eq. (8) for a scale-invariant weight vector $\boldsymbol{\omega}$. Here we assume scale-invariance since it slightly changes the resulting expression due to the dependency of $\|\boldsymbol{u}\|$ on $\|\boldsymbol{\omega}\|$. It also simplifies the math a bit, with $\boldsymbol{u} \perp \boldsymbol{\omega}$, not just in expectation. We get:

$$\begin{aligned} \widehat{\|\boldsymbol{\omega}\|}^2 &= \mathbb{E}\left[(\widehat{\|\boldsymbol{\omega}\|} - \|\boldsymbol{d}\|)^2 + \|\boldsymbol{u}\|^2 \right] && (48) \\ &= \mathbb{E}\left[\left(\widehat{\|\boldsymbol{\omega}\|} - \tfrac{\eta\lambda}{1-\alpha} \widehat{\|\boldsymbol{\omega}\|} \right)^2 + \left(\tfrac{\eta}{1-\alpha} \tfrac{\|\tilde{\boldsymbol{g}}\|}{\widehat{\|\boldsymbol{\omega}\|}} \right)^2 \right] && (49) \end{aligned}$$

Where we define $\tilde{\boldsymbol{g}}_t = \boldsymbol{g}_t\|\boldsymbol{\omega}_t\|$ using $\|\boldsymbol{g}_t\| \propto \|\boldsymbol{\omega}_t\|^{-1}$ due to the inverse proportionality of the gradient magnitude, see eq. (6) or eq. (25). We can interpret $\tilde{\boldsymbol{g}}_t$ as the gradient for weights of unit norm $\|\boldsymbol{\omega}_t\| = 1$.

Solving for $\widehat{\|\boldsymbol{\omega}\|}$ and assuming that $\eta\lambda \ll 2 \cdot (1-\alpha)$ gives:

$$\widehat{\|\boldsymbol{\omega}\|} = \sqrt[4]{\frac{\eta\mathbb{E}[\|\tilde{\boldsymbol{g}}\|^2]}{2\lambda \cdot (1-\alpha) - \eta\lambda^2}} \approx \sqrt[4]{\frac{\eta\mathbb{E}[\|\tilde{\boldsymbol{g}}\|^2]}{2\lambda \cdot (1-\alpha)}} \tag{50}$$

To obtain the absolute size of an update, we further assume that $\mathbb{E}[\|\boldsymbol{g}_t\|^2]$ can be approximated as a constant $\mathbb{E}[\|\boldsymbol{g}\|^2]$ when computing the size of $\boldsymbol{m}_t$, and that successive gradients are roughly orthogonal giving $\boldsymbol{m}_{t-1} \perp \boldsymbol{g}_t$ in expectation. For the random walk setting, the first is reasonable when the norm is stable e.g. around equilibrium and the second always holds. The average square size of an update is then:

$$\mathbb{E}[\|\Delta_g \boldsymbol{p}\|^2] = \eta^2\mathbb{E}\left[\|\alpha\boldsymbol{m}_{t-1} + \boldsymbol{g}_t\|^2\right] \tag{51}$$

$$= \eta^2\mathbb{E}\left[\|\alpha\boldsymbol{m}_{t-1}\|^2\right] + \mathbb{E}\left[\|\boldsymbol{g}_t\|^2\right] \tag{52}$$

$$= \eta^2 \textstyle\sum_{k=0}^{t} \mathbb{E}[\left(\alpha^{t-k}\|\boldsymbol{g}_k\|\right)^2] \tag{53}$$

$$\approx \eta^2 \tfrac{\mathbb{E}[\|\boldsymbol{g}\|^2]}{1-\alpha^2} \tag{54}$$

where (52) comes from the orthogonality, (53) by recursively writing out $\boldsymbol{m}$ in terms of $\boldsymbol{g}$, and (54) from assuming that t is high enough to approximate the sum of the geometric series as an infinite sum.

Simplifying, we get the $\eta_g = \sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{p}\|^2]}$ and $\eta_r = \sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{\omega}\|^2]}/\widehat{\|\boldsymbol{\omega}\|}$ in table 1. We note that the derived rates are consistent with the ones derived in the Spherical Motion Dynamics (Wan et al., 2021) and Online Normalization (Chiley et al., 2019) (when $\alpha = 0$).

D. Lion Equilibrium

The standard version of Lion (Chen et al., 2023) can be written as:

$$\boldsymbol{v}_t = \operatorname{sign}(\beta_1\boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t) \tag{55}$$

$$\boldsymbol{m}_t = \beta_2\boldsymbol{m}_{t-1} + (1-\beta_2)\boldsymbol{g}_t \tag{56}$$

$$\boldsymbol{p}_t = \boldsymbol{p}_{t-1} - \eta \cdot (\boldsymbol{v}_t + \lambda\boldsymbol{p}_{t-1}) \tag{57}$$

where $\boldsymbol{p}_t \in \mathbb{R}^C$ is a parameter vector at time t, $\boldsymbol{g}_t = \nabla_{\boldsymbol{p}}\mathscr{L}(\boldsymbol{p}_t)$ is the gradient, $\boldsymbol{m}$ is the first moment and $\boldsymbol{v}$ is the update velocity. The learning rate ($\eta \geq 0$), weight decay ($\lambda \geq 0$), and moment coefficients ($0 < \beta_1 < 1, 0 < \beta_2 < 1$) are hyperparameters.

In our analysis we look at the arguments of the sign function which we define as:

$$\boldsymbol{n}_t := \beta_1\boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \qquad \boldsymbol{v}_t = \operatorname{sign}(\boldsymbol{n}_t) \tag{58}$$

To obtain an estimate of the magnitude $\|\boldsymbol{n}_t\|$, we assume that the gradient magnitude $\mathbb{E}[\|\boldsymbol{g}_t\|^2]$ can be approximated as a constant $\mathbb{E}[\|\boldsymbol{g}\|^2]$, and that successive gradients are roughly orthogonal giving $\boldsymbol{m}_{t-1} \perp \boldsymbol{g}_t$ in expectation. For the random walk setting, the first is reasonable when the norm is stable e.g. around equilibrium and the second always holds. This gives:

$$\mathbb{E}[\|\boldsymbol{n}_t\|^2] = \mathbb{E}\left[\|\beta_1\boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t\|^2\right] \tag{59}$$

$$= \beta_1^2\mathbb{E}\left[\|\beta_2\boldsymbol{m}_{t-1} + (1-\beta_2)\boldsymbol{g}_{t-1}\|^2\right] + (1-\beta_1)^2\mathbb{E}\left[\|\boldsymbol{g}_t\|^2\right] \tag{60}$$

$$= \beta_1^2(1-\beta_2)^2 \sum_{k=0}^{k=t-1} \beta_2^{t-1-k}\mathbb{E}[\|\boldsymbol{g}_k\|^2] + (1-\beta_1)^2\mathbb{E}\left[\|\boldsymbol{g}_t\|^2\right] \tag{61}$$

$$\approx \beta_1^2(1-\beta_2)^2 \sum_{k=0}^{\infty} \beta_2^k\mathbb{E}[\|\boldsymbol{g}\|^2] + (1-\beta_1)^2\mathbb{E}\left[\|\boldsymbol{g}\|^2\right] \tag{62}$$

$$= \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right) \mathbb{E}[\|\boldsymbol{g}\|^2] \tag{63}$$

where we have used the gradient orthogonality and constant magnitude, and approximated the geometric sum as extending to infinity.

To compute the total update contribution of the gradient, $\|\boldsymbol{u}\|$ in eq. (8), we first need to model how the sign non-linearity affects the magnitude and direction of the update. We note that for a parameter of dimension C, we have:

$$\|\boldsymbol{v}_t\| = \sqrt{C} \tag{64}$$

so the sign function has an average scaling effect:

$$\frac{\|\boldsymbol{v}_t\|}{\|\boldsymbol{n}_t\|} = \sqrt{\frac{C}{\left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right) \mathbb{E}[\|\boldsymbol{g}\|^2]}} \tag{65}$$

The sign function will also rotate $\boldsymbol{n}_t$ resulting in two components, one parallel to $\boldsymbol{n}_t$ and the other orthogonal. We will assume that the orthogonal one cancels out on average without significantly affecting equilibrium and focus on the parallel component. This component depends on the average angle between $\boldsymbol{n}_t$ and $\operatorname{sign}(\boldsymbol{n}_t)$ which is determined by the distribution and correlation between the elements. In the random walk setting, we can assume the components of $\boldsymbol{n}_t = [n_1, \dots, n_C]$ are normally distributed with mean zero. However, the expression for the average angle is still complicated unless the components are independent and identically distributed (i.i.d.) so we make this assumption for this step with $n_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for all k. Then we can use the known expected absolute value for a centered normal distribution to get:

$$\mathbb{E}[\langle \boldsymbol{n}_t, \operatorname{sign}(\boldsymbol{n}_t)\rangle] = C \cdot \mathbb{E}[|n_k|] = C \cdot \sqrt{\frac{2\sigma^2}{\pi}} = \sqrt{\frac{2}{\pi}} \cdot \|\boldsymbol{n}_t\| \cdot \|\operatorname{sign}(\boldsymbol{n}_t)\| \tag{66}$$

Note that the angle is still bounded regardless of the distribution but will result in a different factor in the range that $\|\boldsymbol{n}\|_1/(\sqrt{C}\|\boldsymbol{n}\|_2)$ can take, i.e. $[C^{-\frac{1}{2}}, 1]$ instead of $\sqrt{2/\pi}$.

Based on the preceding analysis we will model the sign function for the computation of $\|\boldsymbol{u}\|$ as:

$$\operatorname{sign}(\boldsymbol{n}_t) \approx \sqrt{\frac{2}{\pi}} \frac{\|\boldsymbol{v}_t\|}{\|\boldsymbol{n}_t\|} \boldsymbol{n}_t = \sqrt{\frac{2C}{\mathbb{E}[\|\boldsymbol{g}\|^2] \cdot \pi \cdot \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)}} \boldsymbol{n}_t \tag{67}$$

which gives:

$$\mathbb{E}[\|\boldsymbol{u}\|^2] = \frac{2\eta C}{\mathbb{E}[\|\boldsymbol{g}\|^2] \cdot \pi \cdot \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)} \mathbb{E}\left[\left\|(1-\beta_1)\boldsymbol{g} + \beta_1(1-\beta_2)\sum_{k=0}^{\infty} \beta_2^k \boldsymbol{g}\right\|^2\right] \tag{68}$$

$$= \frac{2\eta C}{\pi \cdot \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)} \tag{69}$$

Combined with the total update contribution for the weight decay, $\boldsymbol{d} = -\eta\lambda\boldsymbol{\omega}_{t-1}$, this allows us to write eq. (8) for $\boldsymbol{\omega} \in \mathbb{R}^C$:

$$\widehat{\|\boldsymbol{\omega}\|}^2 = \mathbb{E}\left[(\widehat{\|\boldsymbol{\omega}\|} - \|\boldsymbol{d}\|)^2 + \|\boldsymbol{u}\|^2\right] \tag{70}$$

$$= (\widehat{\|\boldsymbol{\omega}\|} - \eta\lambda\widehat{\|\boldsymbol{\omega}\|})^2 + \mathbb{E}[\|\boldsymbol{u}\|^2] \tag{71}$$

$$= (1 - 2\eta\lambda + \eta^2\lambda^2)\widehat{\|\boldsymbol{\omega}\|}^2 + \frac{2\eta C}{\pi}\left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)^{-1} \tag{72}$$

Solving for $\widehat{\|\boldsymbol{\omega}\|}$ and assuming $\eta\lambda \ll 2$ gives:

$$\widehat{\|\boldsymbol{\omega}\|} = \sqrt{\frac{2\eta C}{\pi \cdot (2\lambda - \eta\lambda^2)}} \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)^{-\frac{1}{2}} \approx \sqrt{\frac{\eta C}{\pi\lambda}} \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)^{-\frac{1}{2}} \tag{73}$$

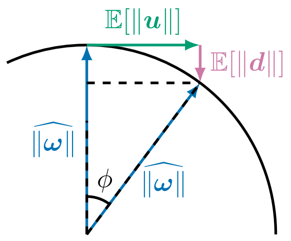
$\mathbb{E}[\|\boldsymbol{u}\|]$
$\mathbb{E}[\|\boldsymbol{d}\|]$
$\widehat{\|\boldsymbol{\omega}\|}$
ϕ
$\widehat{\|\boldsymbol{\omega}\|}$

Figure 8: Weight norm equilibrium. The total update contribution of the gradient is $\boldsymbol{u}$ and the TUC of the weight decay is $\boldsymbol{d}$.

Combined with $\|\boldsymbol{v}_t\| = \sqrt{C}$ for $\boldsymbol{\omega}, \boldsymbol{p} \in \mathbb{R}^C$ we get the expected equilibrium rotation and RMS update size:

$$\widehat{\eta_g} = \sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{p}\|^2]} = \eta\sqrt{C} \tag{74}$$

$$\widehat{\eta_r} = \sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{\omega}\|^2]}/\widehat{\|\boldsymbol{\omega}\|} = \sqrt{\pi\eta\lambda} \cdot \left((1-\beta_1)^2 + \beta_1^2 \frac{1-\beta_2}{1+\beta_2}\right)^{\frac{1}{2}} \tag{75}$$

E. Adam+ℓ_2 Equilibrium

In this section we apply a modified form of the geometric model from §3.1 to Adam (Kingma & Ba, 2015) with ℓ_2-regularization (Adam+ℓ_2 for short) to gain insight into how the rotational equilibrium differs from that of Adam with decoupled weight decay (AdamW, see §3.2).

E.1. Adam+ℓ_2 Formulation

We will write the Adam+ℓ_2 update as follows:

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)(\boldsymbol{g}_t + \lambda \boldsymbol{p}_{t-1}) \tag{76}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t + \lambda \boldsymbol{p}_{t-1})^2 \tag{77}$$

$$\boldsymbol{p}_t = \boldsymbol{p}_{t-1} - \eta \cdot \left(\frac{\boldsymbol{m}_t/(1-\beta_1^t)}{\sqrt{\boldsymbol{v}_t/(1-\beta_2^t)}+\varepsilon}\right) \tag{78}$$

Similar to AdamW, $\boldsymbol{p}_t \in \mathbb{R}^C$ is a parameter vector at time t, $\boldsymbol{g}_t = \nabla_{\boldsymbol{p}} \mathscr{L}(\boldsymbol{p}_t)$ is the gradient, and all operations (e.g. division, squaring) are performed elementwise. In Adam+ℓ_2, both the first and second moment of the gradient ($\boldsymbol{m}$ and $\boldsymbol{v}$) include contributions from the ℓ_2-regularization term. This differs from AdamW (see eq. (11)) where the ℓ_2-regularization (or technically weight decay) does not affect $\boldsymbol{m}$ and $\boldsymbol{v}$. The learning rate ($\eta \geq 0$), ℓ_2-regularization coefficient ($\lambda \geq 0$), moment coefficients ($0 < \beta_1 < 1, 0 < \beta_2 < 1$) and $\varepsilon \geq 0$ are hyperparameters similar to AdamW. Like before, we use $\boldsymbol{\omega}$ to specifically denote the weight vector of a neuron or a layer that can form rotational equilibrium (as opposed to $\boldsymbol{p}$ that we use as a general symbol for any parameter vector, and could denote e.g. a bias).

E.2. Simplifications

The rotational dynamics of Adam+ℓ_2 are more complicated than those of AdamW. The main difference is that the strength of the "weight decay" is affected by the gradient norm. As we will see, this makes the equilibrium norm and angular update depend on the gradient magnitude. Furthermore, the weight decay can be scaled differently for each coordinate of the weight vector as the gradient distribution may vary between them. This complicates the analysis, forcing us to treat each coordinate separately.

Our analysis is based on the random walk setup introduced in §3 and described in appx. G. We further make several assumptions and simplifying approximations that allow us to obtain simpler expressions for the cases of interest:

1. We assume the rotational equilibrium exists as a steady state where hyperparameters are fixed (not varying over time), the expected weight norm $\sqrt{\mathbb{E}[\|\boldsymbol{\omega}_t\|^2]} = \widehat{\|\boldsymbol{\omega}\|}$ is constant, and the second moment of the gradient $\mathbb{E}[\boldsymbol{g}_t^2]$ is constant over time. For simplicity we will drop the t subscript.

2. We focus on the case where the weights are scale-invariant, defining $\tilde{\boldsymbol{g}} = \|\boldsymbol{\omega}\|\boldsymbol{g}$ as the gradient corresponding to a unit weight norm based on the inverse proportionality from eq. (6).

3. We will assume that ε and the bias correction can be ignored, i.e. that ε, β_1^t and β_2^t are all effectively zero.

4. We will assume the second moment tracker $\boldsymbol{v}_t$ is dominated by the gradient component, i.e. that $\boldsymbol{g}^2 \gg \lambda^2\boldsymbol{\omega}^2$, and that it perfectly tracks the expected value, i.e. that $\mathbb{E}[\boldsymbol{v}] = \mathbb{E}[\boldsymbol{g}^2]$. This is a non-trivial approximation based on the geometry of equilibrium when the angular updates are small. For a small ϕ in fig. 8 we can can approximate:

$$\mathbb{E}[\|\boldsymbol{u}\|] = \widehat{\|\boldsymbol{\omega}\|} \cdot \tan(\phi) \approx \widehat{\|\boldsymbol{\omega}\|} \cdot \phi \tag{79}$$

$$\mathbb{E}[\|\boldsymbol{d}\|] = \widehat{\|\boldsymbol{\omega}\|} \cdot (1 - \cos(\phi)) \approx \widehat{\|\boldsymbol{\omega}\|} \cdot \frac{\phi^2}{2} \tag{80}$$

As a result $\mathbb{E}[\|\boldsymbol{u}\|] \gg \mathbb{E}[\|\boldsymbol{d}\|]$. For Adam+$\ell_2$ we have $\boldsymbol{u} = -\eta\boldsymbol{g}_t/\sqrt{\boldsymbol{v}}$ and $\boldsymbol{d} = -\eta\lambda\boldsymbol{\omega}_{t-1}/\sqrt{\boldsymbol{v}}$. As long as $\boldsymbol{v}$ is relatively homogeneous across coordinates, we therefore have $\mathbb{E}[\|\boldsymbol{g}\|] \gg \mathbb{E}[\lambda\|\boldsymbol{\omega}\|]$. We assume this holds roughly coordinate wise as well, giving $\boldsymbol{g}^2 \gg \lambda^2\boldsymbol{\omega}^2$. We note that this fourth assumption is not strictly necessary but significantly simplifies the resulting expressions, giving us an interpretable closed form solution instead a solution expressed as the root of a third-degree polynomial.

E.3. Equilibrium Norm

The total update contribution of the gradient $\boldsymbol{g}_t$, i.e. $\boldsymbol{u}$ in eq. (8), is given by:

$$\boldsymbol{u} = -\eta \textstyle\sum_{k=t}^{\infty} \beta_1^{k-t}(1-\beta_1)\frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}}} = -\eta\frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}}} \tag{81}$$

Similarly, the total update contribution due to the weight decay of $\boldsymbol{\omega}_{t-1}$ is:

$$\boldsymbol{d} = -\eta \sum_{k=t}^{\infty} \beta_1^{k-t}(1-\beta_1)\frac{\lambda\boldsymbol{\omega}_{t-1}}{\sqrt{\boldsymbol{v}}} = -\eta\frac{\lambda\boldsymbol{\omega}_{t-1}}{\sqrt{\boldsymbol{v}}} \tag{82}$$

Due to the coordinate-wise differences in the weight decay, we analyze a single element ω_k at coordinate k in $\boldsymbol{\omega}$ with corresponding elements d_k, u_k, g_k, v_k in $\boldsymbol{d}$, $\boldsymbol{u}$, $\boldsymbol{g}$, $\boldsymbol{v}$, respectively. Although the geometric model is not well defined coordinate-wise, we can still use the concept of orthogonality as defined for random variables. This gives us:

$$\begin{aligned}
\mathbb{E}[\omega_k^2] &= \mathbb{E}[(\omega_k - d_k)^2 + u_k^2] &(83)\\
&= (1 - \tfrac{\eta\lambda}{\sqrt{v_k}})^2\mathbb{E}[\omega_k^2] + \eta^2\tfrac{\mathbb{E}[g_k^2]}{v_k} &(84)\\
&= (1 - \tfrac{2\eta\lambda}{\sqrt{v_k}} + \tfrac{\eta^2\lambda^2}{v_k})\mathbb{E}[\omega_k^2] + \eta^2(1 - \tfrac{\lambda^2\mathbb{E}[\omega_k^2]}{v_k}) &(85)\\
&= (1 - \tfrac{2\eta\lambda}{\sqrt{v_k}})\mathbb{E}[\omega_k^2] + \eta^2 &(86)
\end{aligned}$$

where we have used the fact that $v_k = \mathbb{E}[g_k^2] + \lambda^2\mathbb{E}[\omega_k^2]$.

Since we are targeting the scale-invariant case (Assumption 2) we can write:

$$\mathbb{E}[g_k^2] = \mathbb{E}[\tilde{g}_k^2]\mathbb{E}[\|\boldsymbol{\omega}\|^2]^{-1} \tag{87}$$

where $\tilde{g}_k$ corresponds to an element of the unit norm weight gradient $\tilde{\boldsymbol{g}}$. Accordingly we can write:

$$v_k = \mathbb{E}[\tilde{g}_k^2]\mathbb{E}[\|\boldsymbol{\omega}\|^2]^{-1} + \lambda^2\mathbb{E}[\omega_k^2] \approx \mathbb{E}[\tilde{g}_k^2]\mathbb{E}[\|\boldsymbol{\omega}\|^2]^{-1} \tag{88}$$

where we used Assumption 4.

Plugging this form of v into eq. (86), squaring and simplifying gives:

$$\mathbb{E}[\omega_k^2] = \frac{\eta}{2\lambda}\sqrt{\frac{\mathbb{E}[\tilde{g}_k^2]}{\mathbb{E}[\|\boldsymbol{\omega}\|^2]}} \tag{89}$$

We can now write an expression for the equilibrium norm:

$$\widehat{\|\boldsymbol{\omega}\|}^2 = \mathbb{E}[\|\boldsymbol{\omega}\|^2] = \sum_{k=1}^{C} \mathbb{E}[\omega_k^2] = \frac{\eta}{2\lambda} \sum_{k=1}^{C} \sqrt{\frac{\mathbb{E}[\tilde{g}_k^2]}{\mathbb{E}[\|\boldsymbol{\omega}\|^2]}} \tag{90}$$

which gives:

$$\widehat{\|\boldsymbol{\omega}\|} = \sqrt[3]{\frac{\eta}{2\lambda} \cdot \langle \mathbf{1}, \sqrt{\mathbb{E}[\tilde{\boldsymbol{g}}^2]} \rangle} \tag{91}$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product. Note that when the elements of $\boldsymbol{g}$ have the same second moment, e.g. when they are identically distributed, we can write $\langle \mathbf{1}, \sqrt{\mathbb{E}[\tilde{\boldsymbol{g}}^2]} \rangle = \sqrt{C}\sqrt{\mathbb{E}[\|\tilde{\boldsymbol{g}}\|^2]}$. Also note how this behavior differs from that of AdamW, here the equilibrium norm depends on the gradient magnitude. Finally we note that without scale-invariance we would get a square root instead of a cube root.

E.4. Equilibrium Angular Update

To obtain the absolute size of an update for $\boldsymbol{\omega}$ in equilibrium, we use the fact that in the random walk successive gradients are orthogonal in expectation.

Similar to AdamW, we can then write the average square size of an update as:

$$\mathbb{E}[\|\Delta_g \boldsymbol{\omega}\|^2] = \eta^2 (1-\beta_1)^2 \textstyle\sum_{k=0}^{t} \beta_1^{2t-2k} \mathbb{E}\left[\left\| \frac{\boldsymbol{g}_k}{\boldsymbol{v}} \right\|^2 \right] \tag{92}$$

$$\approx \eta^2 \tfrac{1-\beta_1}{1+\beta_1} C \tag{93}$$

where we approximated the geometric sum with its limit and used $\boldsymbol{v} \approx \mathbb{E}[\boldsymbol{g}^2]$ based on Assumption 4. Note that the use of Assumption 4 gives the same result as for AdamW.

We can then approximate the expected angular update in equilibrium as:

$$\widehat{\eta_r} = \frac{\sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{\omega}\|^2]}}{\widehat{\|\boldsymbol{\omega}\|}} = \sqrt[3]{\frac{2\eta^2\lambda}{\langle \mathbf{1}, \sqrt{\mathbb{E}[\tilde{\boldsymbol{g}}^2]} \rangle}} \sqrt{\frac{1-\beta_1}{1+\beta_1} C} \tag{94}$$

Note that the average angular update depends on the gradient magnitude unlike for other optimizers. Also note the different dependency on η and λ, here the angular update depends on the product $\eta^2\lambda$, not $\eta\lambda$ like for other optimizers. This pattern is visible in the hyperparameter heatmap seen in fig. 6, the performance varies faster along the direction of increasing $\widehat{\eta_r}$ than where it is constant. Finally there is an odd dependency on C that is not present in the other optimizers. Without scale-invariance, the first cube root would be replaced by a square root and the gradient dependency on C would cancel the C in the second root.

F. Random Walk Experiments

F.1. Measurements of a Random Walk

We can directly perform a random walk to validate the expressions given in table 1. Figure 9 shows measurements for a random walk in a simple system described below for each of the metrics given in the table. We show four neurons (colored solid lines), the average over a layer (black) and the predicted equilibrium value (red dashed line) from table 1. As we can see the analytically derived expressions accurately describe the neuronal dynamics of the random walk for each optimizer. We use reasonable hyperparameters values in each case that we have found to work well for either ResNet-18 or ResNet-20 on CIFAR-10 (see details in the next section).

F.2. Simple System for Random Walks

Definition: We define the simple system as:

$$f(\boldsymbol{X}) = \boldsymbol{\gamma}_{\text{out}} \odot N\big(\boldsymbol{W}(\boldsymbol{\gamma}_{\text{in}} \odot \boldsymbol{X})\big) \tag{95}$$

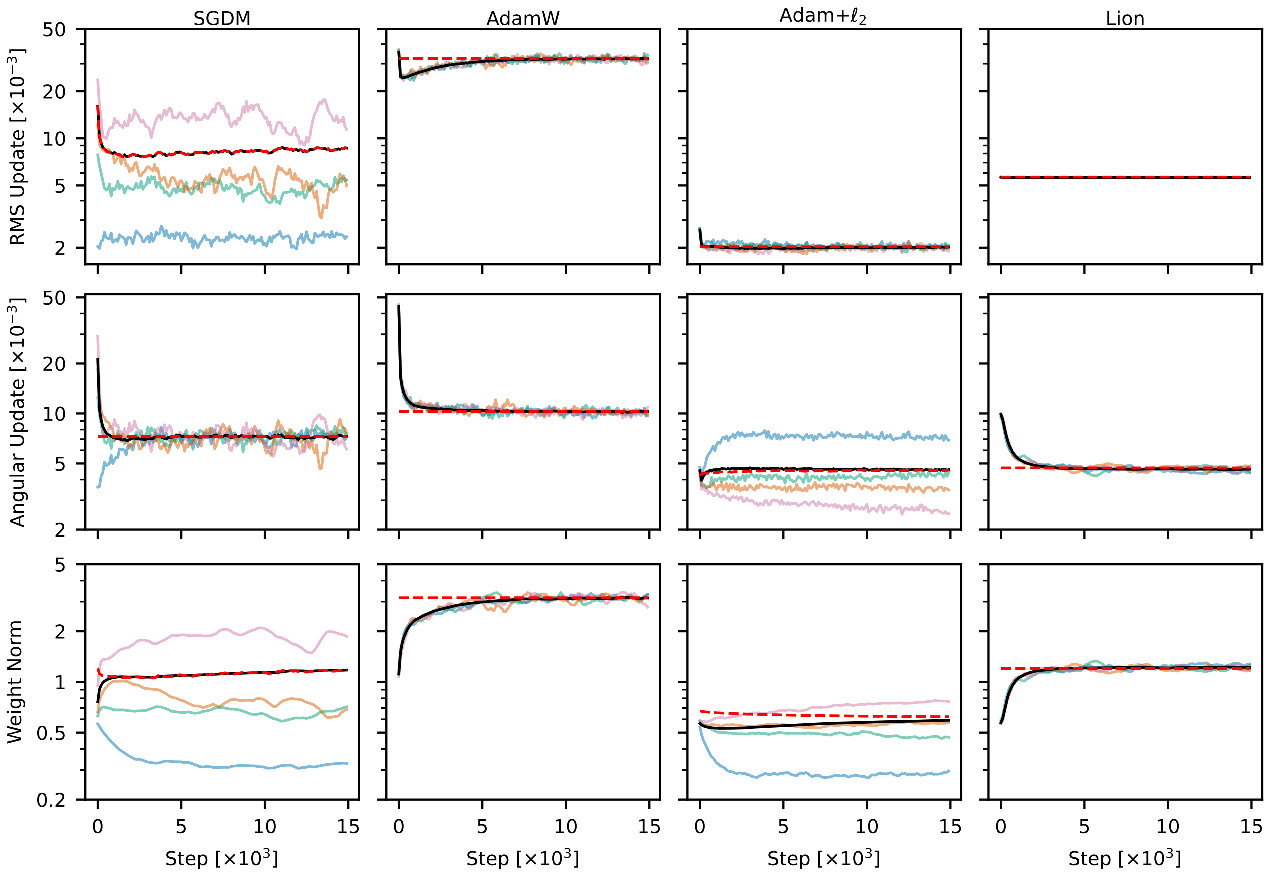

Figure 9: Measurements of the neuronal update dynamics in a random walk, comparing them with the our analytical equilibrium predictions. Colors (pink, orange, green, blue) correspond to individual neurons with varying gradient norms from a single layer. Black lines represent averages over the whole layer, and red dashed lines show equilibrium predictions for the layer average from table 1. The predictions accurately describe the steady state value in each case.

where $\boldsymbol{X} \in \mathbb{R}^{C\times B}$, $\boldsymbol{W} \in \mathbb{R}^{K\times C}$, $\gamma_{\text{in}} \in \mathbb{R}^{C\times 1}$, $\gamma_{\text{out}} \in \mathbb{R}^{K\times 1}$ and N is a batch normalization function (see eq. (22)) applied to each feature independently. The only learnable parameters are the weights $\boldsymbol{W}$, the gammas γ_{in} and γ_{out} are kept constant. We initialize the weights using the default initialization for a linear layer in PyTorch (Paszke et al., 2019) i.e. each element is sampled independently and uniformly from the interval $[-\frac{1}{\sqrt{C}}, \frac{1}{\sqrt{C}}]$. The gammas are initialized with elements independent and identically distributed (i.i.d.) following a standard normal distribution. The inputs are also sampled i.i.d. from a standard normal distribution at each iteration. The gradients of $f(\boldsymbol{X})$, which are used to compute other gradients via the chain-rule or backpropagation, are sampled i.i.d. from a normal distribution with standard deviation $\frac{1}{KB}$ where the B simulates the typical averaging of the loss over a batch and the K gives a scale more similar to the derivatives of softmax-cross-entropy (the difference of two vectors with an L_1-norm of 1 each). We can also scale the output gradients (that get backpropagated to compute the parameter gradients) further with a loss scale to obtain different gradient norms (especially important for Adam).

Rationale: We use this system to study a random walk in a neural network as described in §3, which serves as a simplified model of a real optimization problem. The gammas give different variances for each input and output channel, causing the second gradient moment in Adam/AdamW to vary between elements of $\boldsymbol{W}$ like they may in real neural network training due to the rest of the network. The normalization ensures that the system is scale-invariant for each row of $\boldsymbol{W}$. The randomly sampled inputs and output gradients ensure that everything is orthogonal in expectation. Compared to a real neural network training, the dynamics of this system are simplified with no loss converging over time and steady input / gradient distributions. Other complicated effects such as dead ReLUs do also not happen in this system. This makes this simple system a good setting to study the equilibrium dynamics in a controlled manner.

Details of Figure 9: Here we use $B = 32, C = K = 128$. We use the CIFAR-10 ResNet-20 hyperparameters from table 4 for SGDM and Lion (batch size 128), and the optimal configuration from the learning rate, weight decay sweep on CIFAR-10 ResNet-18 for AdamW and Adam+ℓ_2 (fig. 6L, see details for that experiment in appx. K). The learning rate is constant and the experiments run for 15k steps, with the plots downsampled by a factor of 100x using RMS averaging.

G. Differences between Real Networks and a Random Walk

The random walk model we use assumes the gradient is dominated by the noise. As discussed by McCandlish et al. (2018), the mini-batch gradient noise is inversely proportional to the batch size. At the so called critical batch size, the noise term has a magnitude roughly equal to the true gradient. For sufficiently small batch sizes relative to the critical batch size, which is often large in practice (McCandlish et al., 2018; Shallue et al., 2019), the noise term thus dominates the mini-batch gradient. In this case, the optimization trajectory will locally look like a random walk, but its properties can change over time. Our analysis focuses on the simplified setting of a fully random walk, disregarding the influence of the true gradient on its dynamic behavior. This simplification allows us to make various assumptions that would not strictly hold for real neural network optimizations. Consequently, for real network optimization, we make several approximations that can affect the accuracy of the predictions in table 1. Here we evaluate how well the main simplifications from the random walk analysis of the AdamW optimizer in §3 apply to real neural network training tasks. Specifically, we cover a standard and unnormalized CNN trained on CIFAR-10 (fig. 10 and fig. 11) and a Transformer model trained on Wikitext (fig. 12). Below we discuss the key quantities we approximate and how this affects the predicted equilibrium norm and rotation.

Orthogonality Between the Weights and the Gradient $\angle(\omega_{t-1}, \boldsymbol{g}_t)$: We use the orthogonality between the weights and the gradient as a measure of $\mathbb{E}[\langle \boldsymbol{u}, \boldsymbol{\omega} \rangle] = 0$. If we measure a small positive or negative bias, the gradient contributes to the effective weight decay λ_e. When the gradient $\boldsymbol{g}_t$ is aligned with the weight ω_{t-1}, it increases the weight decay. Conversely, when the gradient $\boldsymbol{g}_t$ is negatively aligned with the weight ω_{t-1} the weight decay decreases.

As a consequence, we tend to overestimate (or underestimate) the measured weight norm $\|\boldsymbol{\omega}\|$ and underestimate (or overestimate) the expected angular update η_r.

By defining the error as a scaling factor of λ (represented as $\lambda_{err} = \frac{\lambda_e}{\lambda}$), we observe the following impact on our prediction

$$\eta_r \approx \sqrt{2\eta(\lambda \cdot \lambda_{err})\frac{1-\beta_1}{1+\beta_1}} = \widehat{\eta_r} \cdot \sqrt{\lambda_{err}} \tag{96}$$

$$\|\boldsymbol{\omega}\| \approx \sqrt{\frac{\eta C}{2(\lambda \cdot \lambda_{err})}} = \widehat{\|\boldsymbol{\omega}\|} \cdot \frac{1}{\sqrt{\lambda_{err}}} \tag{97}$$

Orthogonality Between the Momentum and the Gradient $\angle(g_t, m_{t-1})$: We use the orthogonality between the momentum and the gradient as an approximation of $\forall j \neq k : \mathbb{E}[\langle \boldsymbol{g}_j, \boldsymbol{g}_k \rangle] = 0$. It gives us information about the orthogonality of $\boldsymbol{g}_t$ and the previous update directions. This means we have additional negative (or positive) terms in eq. (20) and thus tend to overestimate (or underestimate) the approximated RMS update size η_g in eq. (21).

Scaled Gradient Norm $\|\frac{g_t}{\sqrt{v_t + \varepsilon}}\|$: The scaled norm directly measures the assumption $\forall t, k : \mathbb{E}[\|\boldsymbol{g}_t / \sqrt{\boldsymbol{v}_k}\|] = \sqrt{C}$. If $\mathbb{E}[\|\boldsymbol{g}_t / \sqrt{\boldsymbol{v}_k}\|]$ is larger than $\sqrt{C}$, we underestimate the weight norm. If it is smaller, we overestimate the weight norm.

For an error C_{err} in the estimate, defined as $\mathbb{E}[\|\boldsymbol{g}_t / \sqrt{\boldsymbol{v}_k}\|] = C \cdot C_{err}$, we have:

$$\|\boldsymbol{\omega}\| \approx \sqrt{\frac{\eta(C \cdot C_{err})}{2\lambda}} = \widehat{\|\boldsymbol{\omega}\|} \cdot \sqrt{C_{err}} \tag{98}$$

Radial Gradient Component λ_u/λ: The radial gradient component approximates the impact of $\mathbb{E}[\boldsymbol{u}_{\|}]$ relative to λ. It measures how much the scaled gradient component influences the effective weight decay $\lambda_e = \lambda + \lambda_u$. A large λ_u affects our predictions similarly to what is described in eq. (96) and eq. (97).

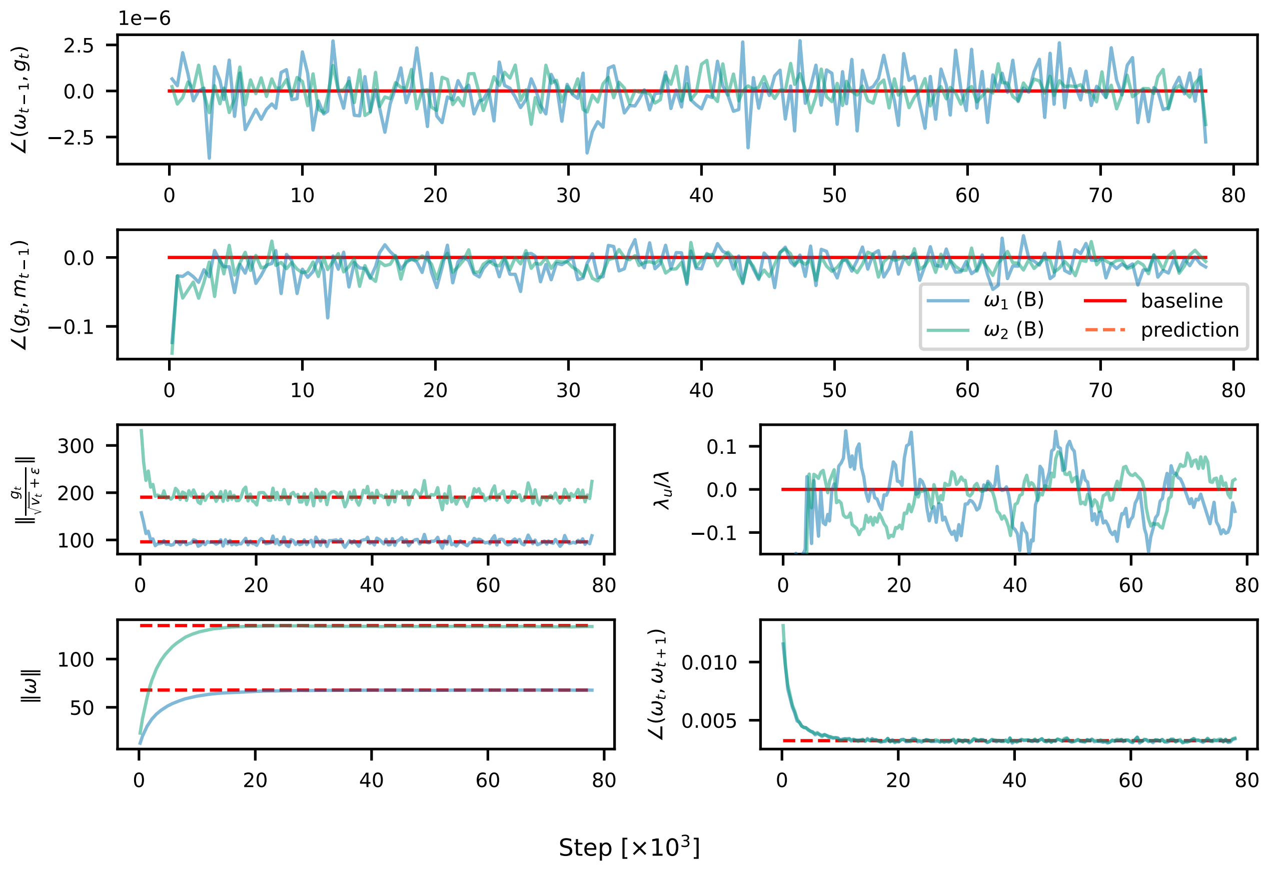

Figure 10: Measurement of how closely the random walk model approximates the training dynamics of two convolutional filters in a ResNet-20 trained on CIFAR-10 using AdamW, with a constant learning rate of $\eta = 0.01$ and a weight decay of $\lambda = 0.01$. This standard ResNet employs Batch Normalization after each convolutional layer, ensuring scale-invariant convolutional weights. As expected, $\mathbb{E}[\langle \boldsymbol{u}, \boldsymbol{\omega} \rangle]$ and $\lambda_u = 0$ are close to zero. Similarly, $\angle(g_t, m_{t-1})$ is nearly zero, indicating that the approximation $\forall j \neq k : \mathbb{E}[\langle \boldsymbol{g}_j, \boldsymbol{g}_k \rangle] = 0$ holds well in practice. Additionally, the approximation $\forall t, k : \mathbb{E}[\|\boldsymbol{g}_t / \sqrt{\boldsymbol{v}_k}\|] = \sqrt{C}$ appears to hold in this case. Finally, we observe that the predictions closely match the measurements after the initial transient phase in this setup.

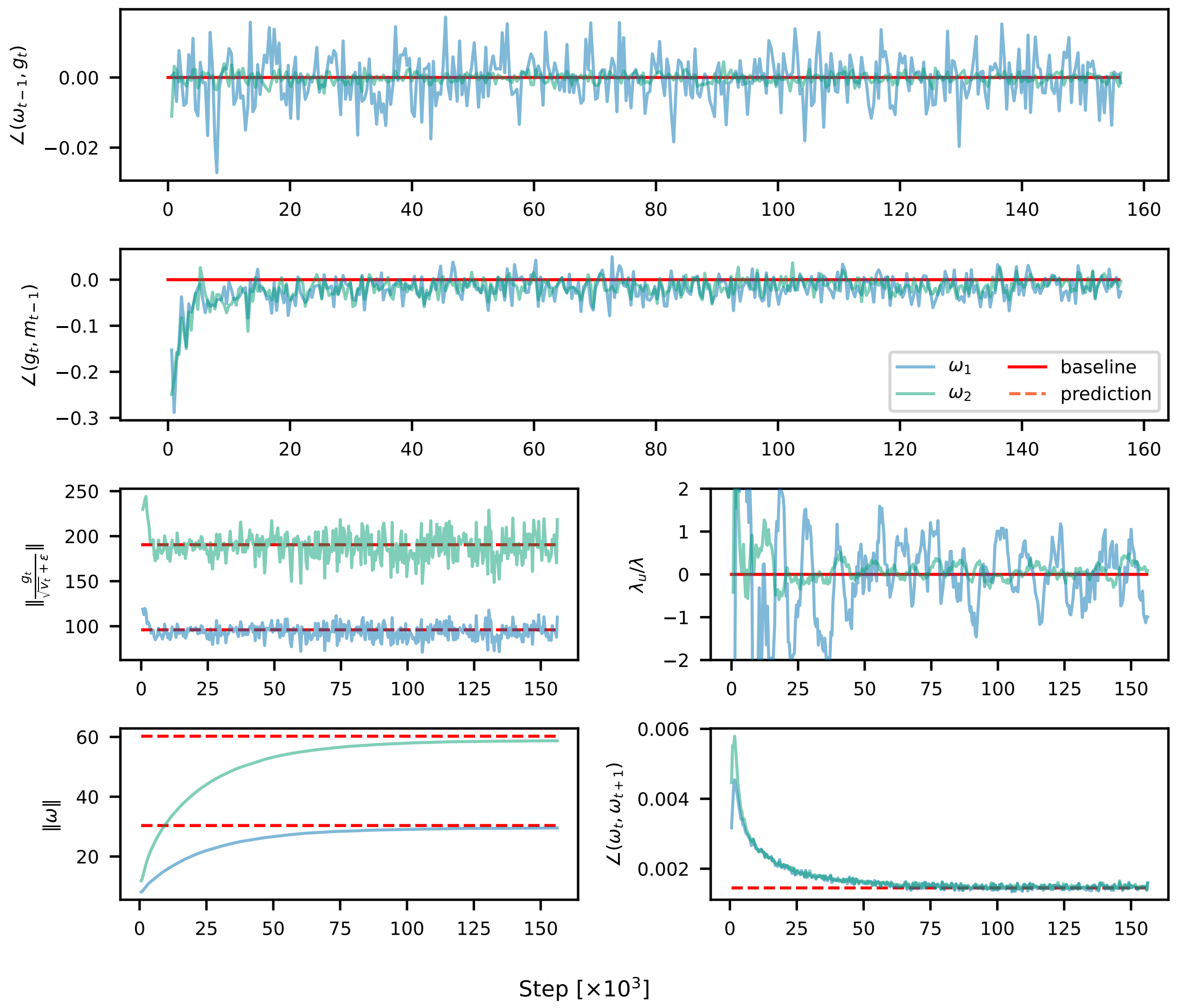

Figure 11: Measurement of how closely the random walk model approximates the training dynamics of two convolutional filters in an unnormalized ResNet-20 trained on CIFAR-10 using AdamW, with a constant learning rate of $\eta = 0.002$ and a weight decay of $\lambda = 0.01$. We observe similar overall behavior as in fig. 10, but the gradients are not necessarily fully orthogonal to the weights as we would expect. There is a slight alignment between the gradients and weights, causing the effective weight decay to change. Nevertheless, we observe that the predictions closely match the measurements after the initial transient phase in this setup.

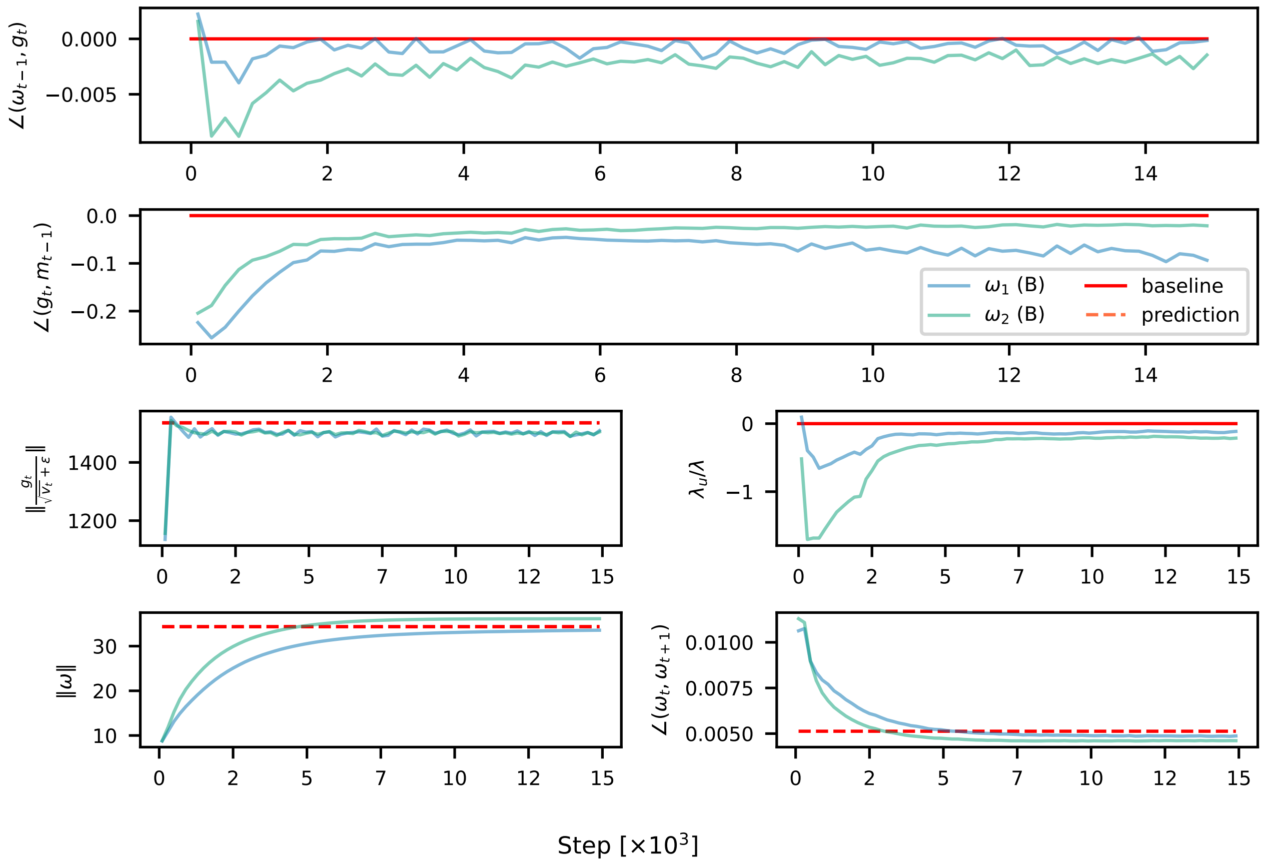

Figure 12: Measurement of how closely the random walk model approximates the training dynamics for a GPT2-124M model trained on Wikitext using AdamW, with a constant learning rate of $\eta = 0.0005$ and a weight decay of $\lambda = 0.5$. In this model, the weights are not fully scale-invariant. We observe a small but consistent negative alignment between the gradients and weights, resulting in a reduced weight decay effect. Additionally, there is a slight overestimation of the scaled gradient and a consistent negative alignment between the gradients and momentum. Despite these deviations, the random walk approximations yield reasonable predictions, as evidenced by the measurements in the last row.

H. Rotational Dynamics of Scale-Sensitive Parameters

Most neural network architectures have some scale-sensitive parameters. This commonly includes gains and biases as well as a final fully connected (FC) layer that is typically not followed by normalization. In networks without normalization, with infrequent normalization, or poorly placed normalization, most weight vectors can be scale-sensitive. The original, un-normalized, VGG (Simonyan & Zisserman, 2015) architecture is a good example of this. VGG consists of a series of convolutional layers, with ReLUs and occasional pooling layers between them, and series of fully connected layers towards the end. In this section we use it to investigate the rotational dynamics of scale-sensitive weights.

First we would like to note that the weight and gradient magnitude of scale-sensitive weights can also be largely arbitrary, similar to scale-invariant weights. Although they can't be scaled directly without affecting the loss, we can often scale two of them without affecting the network output. Consider two successive layers with a ReLU between them:

$$f(\boldsymbol{X}, \boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2) = \text{ReLU}(\boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{99}$$

where $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{C\times C}$ are weight matrices, $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{R}^{1\times C}$ are vectors, $\boldsymbol{X} \in \mathbb{R}^{B\times C}$ are inputs and we broadcast the operations. Note that the ReLU is positively homogeneous, so for a positive scalar $r > 0$ we have:

$$f(\boldsymbol{X}, r\boldsymbol{W}_1, r^{-1}\boldsymbol{W}_2, r\boldsymbol{b}_1, \boldsymbol{b}_2) = \text{ReLU}(\boldsymbol{X}\boldsymbol{W}_1 r + \boldsymbol{b}_1 r)\boldsymbol{W}_2 r^{-1} + \boldsymbol{b}_2 = f(\boldsymbol{X}, \boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2) \tag{100}$$

Assuming the weights are scaled in-place (i.e. we don't modify the computation graph, only the weight values), this type of rescaling operation scales the relative update of $\boldsymbol{W}_1$ by r^{-2} and $\boldsymbol{W}_2$ by r^2 when optimizing using SGD. This can significantly affect the learning dynamics as studied in e.g. Path-SGD (Neyshabur et al., 2015).

For a scale-sensitive weight $\boldsymbol{\omega}$, the gradient orthogonality eq. (5) and inverse scaling eq. (6) do not necessarily hold. The inverse scaling holds in terms of rescaling operations like the ones mentioned above if they are applicable. Generally, the gradient has some radial component in the direction of the weight. The expected magnitude of this component depends on the average angle between the gradient and the weight as well as the expected gradient magnitude itself. If we separate the gradient into radial and perpendicular components and view the radial component as a modification of the weights decay, we have a very similar setup to the one we analyzed for scale-invariant weights. If a stable equilibrium exists, this could give rise to rotational dynamics which may vary from weight to weight based on the "effective weight decay" for each one.

We explore this with VGG-13 training on CIFAR-10 using SGDM. We compare two versions, a standard unnormalized network and a variant where weight standardization is applied to every convolutional and fully connected layer. For each setup, we measure the angular updates, the weight norms, and the relative radial gradient magnitude:

$$\lambda_u = \mathbb{E}[\langle \boldsymbol{\omega}, \nabla_{\boldsymbol{\omega}}\mathscr{L}\rangle / \|\boldsymbol{\omega}\|^2] = E[\cos(\angle(\boldsymbol{\omega}, \nabla_{\boldsymbol{\omega}}\mathscr{L})) \cdot \|\nabla_{\boldsymbol{\omega}}\mathscr{L}\| / \|\boldsymbol{\omega}\|] \tag{101}$$

Note that we have written this in the case of no momentum by using $-\eta\nabla_{\boldsymbol{\omega}}\mathscr{L}$ instead of $\boldsymbol{u}$, but for the standard implementation of SGDM the momentum magnifies both this version of λ_u and the standard "weight decay" (ℓ_2-regularization) term the same way so they are comparable. The λ_u term can therefore be viewed as modifying the weight decay, the effective weight decay parameter is $\lambda_e = \lambda + \lambda_u$ and accounts for the entire radial portion of a weight update. We replace the standard λ with λ_e when showing predicted values for the unnormalized network.

The results can be seen in fig. 13 for two weights. For the first one on the left, λ_u is relatively small compared to $\lambda = 5 \cdot 10^{-4}$ and the weight behaves similarly in both setups, with "standard" update dynamics in the unnormalized setup. The equilibrium predictions match well early in training after the initial transient phase, but the weight falls out of equilibrium towards the end when it can't decay fast enough to keep up with the equilibrium weight magnitude. For the second weight shown on the right, λ_u is large causing a significant difference between the scale-invariant and scale-sensitive setups. The modified equilibrium predictions using λ_e capture the behavior well in the middle phase of training, after the initial transition before the weight falls out of equilibrium towards the end. We note that in the unnormalized setup λ_u changes over the course of training, starting out around 0 corresponding to an orthogonal gradient and growing larger in the later phases. This is likely due to the cross-entropy loss used, which is minimized with large (infinite) output magnitudes once the network has learned to accurately classify (overfit) the training data. This encourages the network to increase the output magnitude to fully overfit the data, resulting in a decreased effective weight decay and slower rotation.

Our results for VGG13 suggest that scale-sensitive weights can also have rotational dynamics in real neural networks. The dynamics are less regular than in the normalized setup, with weights rotating at different speeds depending on the size of the

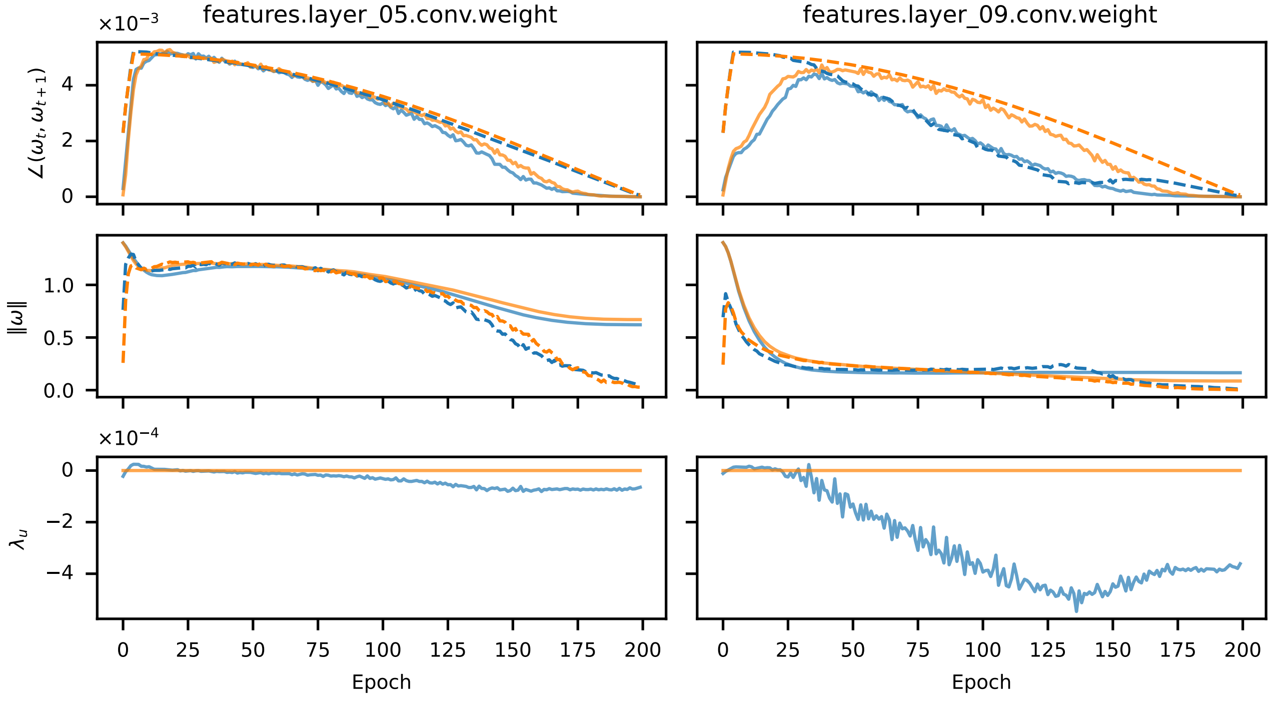

Figure 13: Measured (solid) and predicted equilibrium values (dashed) when training unnormalized (blue) and weight standardized (orange) variants of VGG-13 on CIFAR-10. The blue predictions account for the modified "effective" weight decay caused by the radial component of the gradient.

radial gradient component. The weight magnitude can also not vary freely like for scale-invariant weights, where we can trade off the weight decay and learning rate without affecting the dynamics much (once equilibrium is achieved). Using large amounts of weight decay in unnormalized networks can bring the weight norms out of balance, resulting in issues like vanishing gradients or activations. In unnormalized networks the magnitude of one weight matrix also affects the gradient magnitude of all others layers, further complicating the effect of weight decay. Our rotational optimizer variants constrain the dynamics to match the equilibrium dynamics of weight standardized networks throughout training, eliminating some of these effects.

I. Rotational Optimizer Wrapper

In this section, we provide further details on the algorithmic design choices used in our rotation optimizer wrapper, as shown in algo. 1. Note that the method can act as a wrapper around any given existing optimizer F with a known $\widehat{\eta}_r$. In cases where the true value is unknown or undesirable, we can also specify some different value of our choice.

Rotational and Non-Rotational Updates: We use Ω to specify weights we apply rotational updates to, so a parameter $\boldsymbol{p}$ is treated differently based on whether $\boldsymbol{p} \in \Omega$ or not. By default we consider each neuronal weight vector to be a separate vector in Ω, but we could also apply the RVs at a coarser scale like whole layers (which is done in e.g. LARS). The rotational wrapper leaves the update of non-rotational parameters unchanged. Rotational parameters are rotated by $\widehat{\eta}_r$ on average and their magnitude is kept constant. Some weights in Ω may be scale-sensitive meaning their magnitude can matter for efficient training. Since the RVs constrain the weights we optionally introduce a learnable gain to allow the network to learn the right magnitude for these weights. This gain can be absorbed into the weights for inference.

Keeping the Weight Magnitude Constant: Alternatively, we could vary the weight magnitude according to our derived value for the equilibrium norm. However, with a learning rate schedule this value can become arbitrarily small causing numerical issues. For scale-invariant weights the magnitude doesn't matter so we simply keep it constant. This has the added benefit of removing the inverse scaling effect of the weight norm on the gradient magnitude eq. (6), potentially making it a more informative metric over the course of training with a learning rate schedule.

Controlling the Rotation Instead of the Relative Update: The rotation of a scale-invariant weight $\boldsymbol{\omega}$ is generally caused by both $\Delta_g \boldsymbol{\omega}$ and $\Delta_\lambda \boldsymbol{\omega}$ as can be seen in fig. 2L and 2R. In equilibrium, the sum of these components is roughly orthogonal to the weight vector. We want to avoid having to apply the weight decay and our constrained magnitude is generally not equal to the equilibrium magnitude. We therefore project $\Delta_g \boldsymbol{p}$ to be orthogonal to $\boldsymbol{p}$ and control the average size of this projected version of $\Delta_g \boldsymbol{p}$ instead of the original $\boldsymbol{p}$. This lets us explicitly control the angular update, regardless of any radial component in $\Delta_g \boldsymbol{p}$ that the weight decay would eliminate on average. If we apply rotational updates to scale-sensitive weights, performing line 15 after line 13 prevents any radial component in the gradient from affecting the rotational speed.

Centering the Weights: Different normalization setups can result in slightly different SMD properties. Layer Normalization typically makes an entire weight matrix scale-invariant whereas Batch Normalization makes individual filters (i.e., rows or columns) independent. The default form of the rotational wrapper corresponds to the rotational equilibrium dynamics obtained with Weight Standardization (Qiao et al., 2019) also known as Centered Weight Normalization (Huang et al., 2017b), where each filter is scale and shift invariant. We remove the mean $\bar{\boldsymbol{p}} = \frac{1}{C}\sum_{i=1}^{C} p_i$ of $\boldsymbol{p} = [p_1, \ldots, p_C]$ since it is irrelevant in this setup. This removal was also found to be beneficial in NF-Nets (Brock et al., 2021a;b). Note how we remove the mean component of the update before updating the RMS tracker. This ensures that the average rotation is not decreased when there is a significant mean component in the update.

Hyperparameters: The algorithm requires an ε value for numerical stability but otherwise only adds one hyperparameter, a decay factor β similar to those in Adam. It determines the rate at which we update our estimate of the average update magnitude (Line 15). This in turn controls how much we let the rotation vary between steps. We could potentially derive an analytical value for β based on the convergence speed towards equilibrium. For example β should perhaps be roughly equal to $\sqrt{a}$ from eq. (16) for AdamW, when trying to match the dynamics exactly. However, this rate may not be optimal and generally depends on the learning rate (which may vary according to a learning rate schedule). We use a default of $\beta = 0.9$ which should keep the expected angular update close to the equilibrium value over time, while still allowing some variation from step to step. There is likely batch size dependence in the optimal value of β, with larger batches potentially benefiting from smaller values since balancing the average rotation within in each step could be sufficient in these cases. An Adam-like bias correction is applied to the average update magnitude when it is used (Line 17).

Resource Requirements: We need to keep track of two scalars $\nu_{\boldsymbol{p}}$ and $n_{\boldsymbol{p}}$ for each rotational parameter. Since $\boldsymbol{p}$ is generally a vector, such as a row in a weight matrix, the memory requirement is negligible compared to quantities like momentum that store a scalar for every element of $\boldsymbol{p}$. The computational requirements in terms of floating-point operations are also relatively small, linear in the number of network (scalar) parameters like standard optimizers. However, the rotational variants are not applied fully elementwise, making efficient (parallel) implementations slightly harder.

J. Additional Experiments

Dynamics for a Random Walk: In appx. F we measure the update dynamics in a simplified system undergoing a random walk matching our analytical setting. We observe that our analytical predictions from table 1 accurately describe the dynamics of this system for each of the optimizers we analyzed.

Dynamics Without Weight Decay: In §5.1, we discussed that training without weight decay is similar to multiplying the learning rate schedule for η_r with an exponentially decaying function. This can be beneficial in some scenarios, e.g. Loshchilov & Hutter (2019) found that for a constant learning rate schedule a weight decay of zero can be optimal, unlike when a cosine decay schedule is used. The exponential decay in the angular updates without weight decay evident in fig. 14, where we present measurements of a ResNet-20 trained on CIFAR-10 with a learning rate of $\eta = 0.05$ and no weight decay ($\lambda = 0$), as well as the same setup with a weight decay of $\lambda = 0.01$ for both AdamW and RV-Adam. All experiments use a cosine learning rate schedule. Note the equilibrium norm for RV-Adam is constant throughout training by design.

Dynamics of Scale-Sensitive GPT Model: In §5.1 and fig. 3 we show the dynamics of a modified GPT2 model where Weight Standardization (WS, Qiao et al. (2019); Huang et al. (2017b)) is applied to all linear layers. Although we don't show the results here, we found the Weight Standardized model to generally perform equally well or slightly better than the baseline. WS makes individual neurons scale-invariant which in turn helps balance and regulate the dynamics as discussed in §5.3. Without scale-invariance, radial components in the gradient can modify the effective weight decay (see §3.4) leading to different equilibrium values for the magnitude and rotation. We can predict modified values for SGDM (appx. H) based on measurements of the radial component, but have not attempted to do so for AdamW.

In fig. 15 we compare the angular updates of the Weight Standardized GPT2 model to a standard one without scale-invariance.

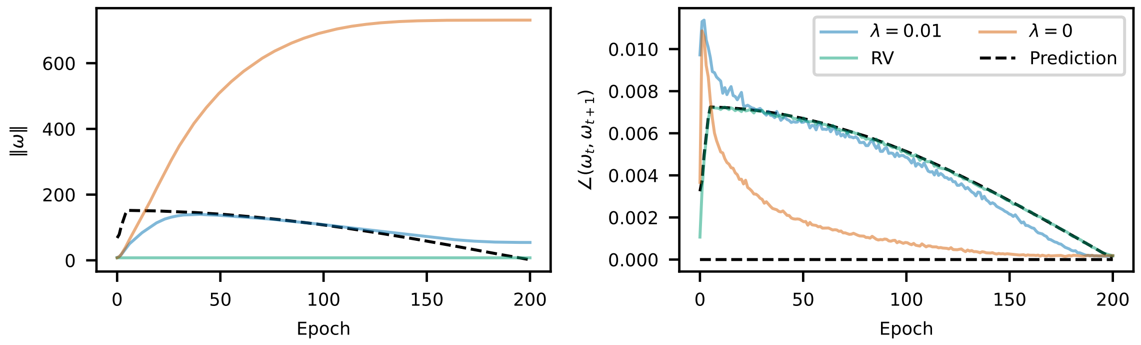

Figure 14: Measurements of the weight norm $\|\omega\|$ and angular updates η_r for CIFAR10 ResNet20 training for AdamW and RV-AdamW compared to Adam without weight decay ($\lambda = 0$). The black line represents our equilibrium and angular update size predictions from table 1. Note how the angular updates without weight decay quickly approach zero, which would even happen with a constant learning rate schedule.

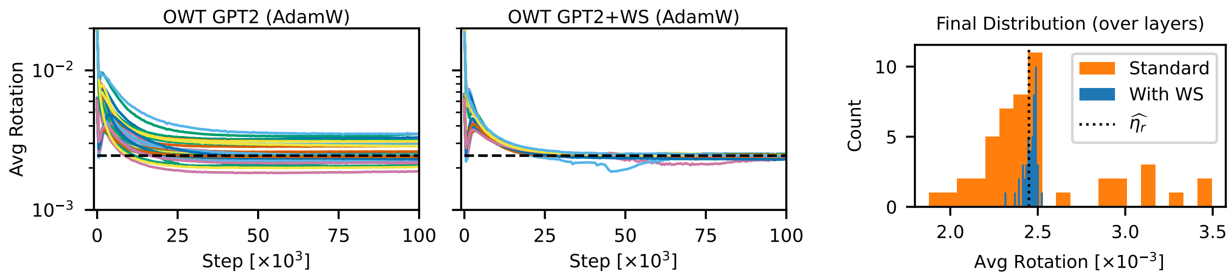

Figure 15: Left: The measured rotation of every linear layer of GPT2-124M, with and without Weight Standardization, for 100k steps on OpenWebText using a constant learning rate. The predicted equilibrium rotation is shown in dashed black (not adjusted for the scale-sensitivity). **Right:** The distribution of the average rotation of the layers at step 100k for each setting.

We show the predicted values based on the formula for scale-invariant layers. As can be seen the unmodified GPT2 model has layers that deviate from this value although it is a reasonable approximation in most cases. With Weight Standardization all layers end up very close to the predicted equilibrium value. In fact we include the extra $\eta\lambda^2$ term from eq. (18) in the prediction here since it is significant compared to the width of the final distribution. Two layers (in the first attention block) temporarily deviate slightly for reasons we are not fully sure of, but they coincide with significant changes in the gradient norms. Effects such as dead neurons or a negative average alignment of successive gradients could also cause deviations from the random walk behavior our analysis assumes (see appx. G).

Constraining the Rotational Dynamics of Other Optimizers: In this section, we examine the performance of the Rotational Variants of SGDM and Lion. Table 3 shows that the RVs for both SGDM and Lion can match the baseline similar to what we observed for AdamW in table 2. In most cases no further hyperparameter tuning of the baseline values is needed (zero-shot) but otherwise light tuning (few-shot) suffices. This shows that the RVs can replicate the benefits of weight decay while simplifying the training dynamics by removing the transient phase and fully balancing rotation.

Table 3: Test set performance for baseline optimizers SGDM and Lion and their RVs. Zero-shot results for RVs use the baseline hyparparameters, the few-shot is lightly tuned.

Dataset	**Model**	**Optimizer**	**Batch Size**	**Metric (⇅)**	**Baseline**	**RV Zero-Shot**	**RV Few-Shot**
CIFAR-10	ResNet-20	SGD	128	Top-1 Acc. (↑)	92.7±0.1	92.4±0.2	N/A
CIFAR-10	ResNet-20	SGD	2048	Top-1 Acc. (↑)	92.0±0.2	92.0±0.3	N/A
CIFAR-10	ResNet-20	Lion	128	Top-1 Acc. (↑)	92.1±0.2	91.9±0.2	N/A
CIFAR-10	ResNet-20	Lion	2048	Top-1 Acc. (↑)	91.8±0.3	91.5±0.3	91.8±0.2
Imagenet-1k	ResNet-50	SGD	256	Top-1 Acc. (↑)	77.4	77.3	N/A

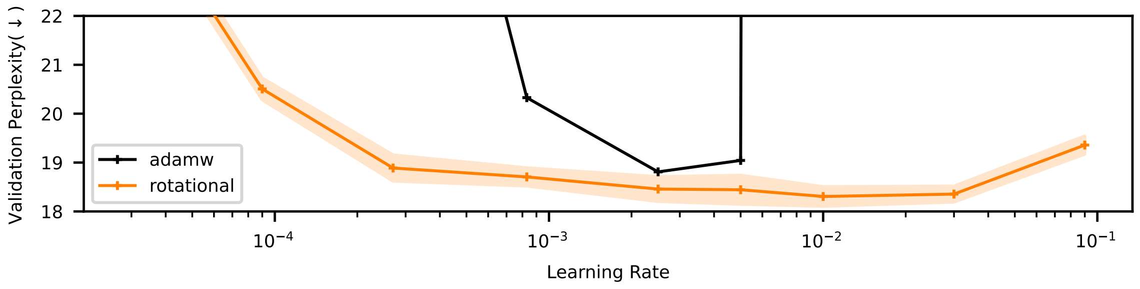

Figure 16: Validation perplexity for GPT2-124M model on Wikitext for different learning rate, weight decay pairs with a constant product ($\eta\lambda = 2.5 \cdot 10^{-3}$) resulting in a specific $\widehat{\eta_r}$ (table 1).

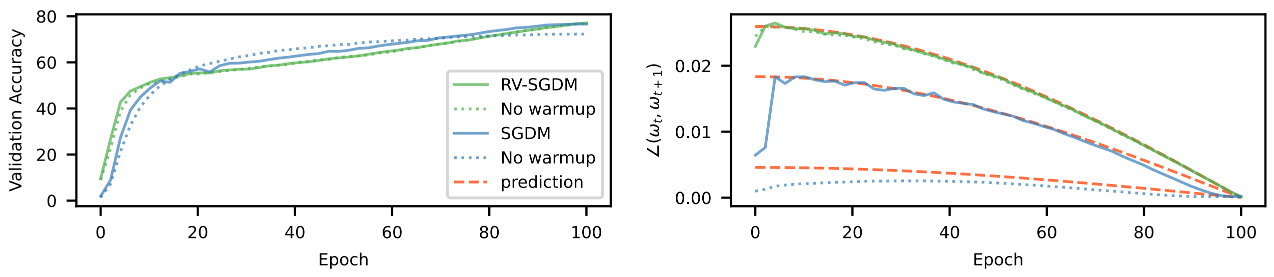

Figure 17: Left: Validation accuracy curves for SGDM and RV-SGDM with and without learning rate warmup for the best configuration of each based on fig. 5R. The final accuracy is 76.6% for SGDM with warmup significantly higher than 72.3% without warmup, compared to 77.1% for RV-SGDM with warmup which is only slightly higher than 76.9% without warmup. **Right:** Measured average angular updates averaged across all convolutional layers of the ResNet50. For both RV-SGDM runs, a learning rate of $\eta = 6.4$ was applied. For standard SGDM a learning rate of $\eta = 3.2$ compared to a significantly lower $\eta = 0.2$ was utilized for the SGDM without warmup. We believe the reduced learning rate for SGDM without warmup is necessary due to the instability encountered during the initial transient phase at higher learning rates.

Learning Rate vs Weight Decay for a Transformer model: Here we repeat the Learning Rate vs Weight Decay Experiment from §5.2, fig. 4, for a GPT2-124M model trained on Wikitext. We vary the learning rate η while keeping the product of the weight decay λ and learning rate, i.e. $\lambda\eta$ constant. We include a learnable gain for each neuron when using the RV, as they are not scale-invariant and their magnitude may matter for learning. The results are shown in fig. 16. Varying the learning rate while keeping the $\eta\lambda$ product constant only affects the updates of biases and gains in the RV as the rotational rate is constant. For the baseline the relative size of the bias/gain updates compared to the angular updates is also affected but additional transient effects are also introduced. We observe that the RV is less sensitive, displaying better results across a wide range of learning rates. We believe this is primarily due to variations in the effective (rotational) step size schedule for the baseline, which can suffer from overly fast initial (transient) rotation at higher learning rates and an slow rotation for lower learning rates.

Need for Learning Rate Warmup: In fig. 5R, we demonstrate that SGDM significantly benefits from a warmup period, while the RV-SGDM exhibits only marginal performance improvements. For the learning rate that performed best in this experiment, we extended the run to span the full 100-epoch duration. Figure 17 shows the validation accuracy curve over the course of training for each of these runs. Notably, while SGDM without warmup exhibits significant performance gains over the course of the 100 epochs, it fails to achieve the performance of SGDM with warmup. In contrast, training the ResNet50 with RV without a warmup period closely matches the performance with warmup. This finding reinforces our belief that a learning rate warmup may aid in stabilizing the transient phase of training, an effect that could potentially be achieved by directly controlling the expected angular update.

Balancing Adam+ℓ_2: Our analysis (§3.3) shows that in Adam+ℓ_2 the equilibrium rotation depends on the gradient norm unlike AdamW and the other optimizers listed in table 1. In our experiments (§5.3) we found that this indeed holds for ResNet-18 on CIFAR-10, where we also reproduced a sweep from the original AdamW paper that shows a performance gap between the two. Although we observe imbalanced rotation causing performance degradation in multiple settings, the experiment does not directly show that the imbalanced rotation causes the difference in this case.

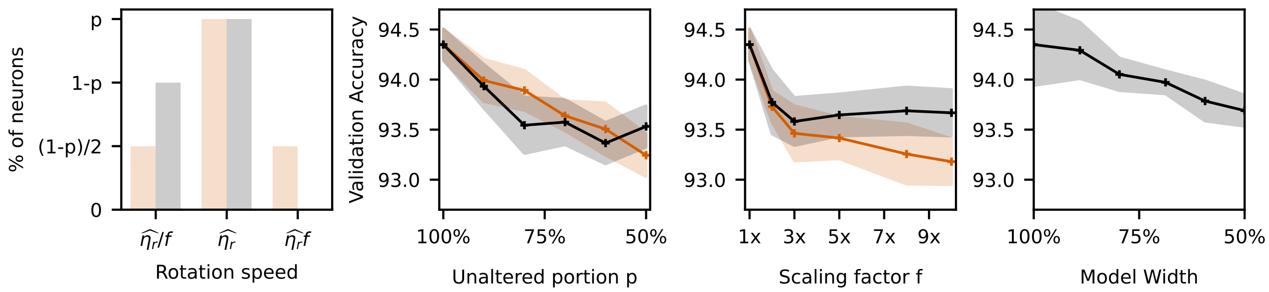

Figure 18: The more imbalanced the rotation, the lower the final performance for ResNet-18 training on CIFAR-10. This happens even with extensive hyperparameter tuning of each setting, which does not compensate for the imbalance. **Left:** Two artificially imbalanced angular update size distributions (black/orange). A portion $1-p$ of the neurons is rotated f times slower and/or faster than rest using a modified RV. **Right-1:** Varying the portion p for a fixed factor $f = 10$. **Right-2:** Varying the factor $f \in [1, 10]$ for a fixed portion $p = 50\%$. **Right-3:** Reducing the network width unsurprisingly also degrades performance. Imbalanced rotation with $p = 50\%$ and $f = 3$ gives similar results as a network of half the width (93.7%), suggesting we could be better off without neurons with even seemingly small deviations in the angular update size.

The updates of two optimizer can only differ in two ways, in the magnitude of the update and/or the direction of the update. We can construct a special RV that uses Adam+ℓ_2 as the inner optimizer (F in algo. 1) but uses the rotational update size $\widehat{\eta}_r$ computed for AdamW. If this RV performs similar to the standard RV-AdamW then the difference in the update direction should not be a significant factor. Note however that the impact on the update direction depends on the weight norm (which is kept fixed at initialization in the RV but varies in standard training) as well as the strength of the λ hyperparameter (which is much higher for the optimal AdamW configuration). We expect this to result in a larger change in the direction of $\Delta_g \boldsymbol{p}$ than would be observed along an Adam+ℓ_2 optimization trajectory. However, we do not validate this directly and expect the effect to vary over time and between network components (layers / neurons).

Running the two RVs at the optimal hyperparameter configuration from AdamW over 5 seeds gives 94.61±0.10% test accuracy for RV-AdamW and 94.53±0.11% for the special Adam+ℓ_2 RV. This is slightly lower than the 94.71±0.16% for the baseline AdamW but around 0.5% higher than the 94.08±0.16% for Adam+ℓ_2. The RV-AdamW performance is likely slightly lower than AdamW due to rotational scheduling effects as is often the case for a zero-shot hyperparameter transfer. The fact that the two RVs perform very similarly and noticeably better than the extensively tuned Adam+ℓ_2 supports our conjecture that irregular rotation contributes to the lower performance of Adam+ℓ_2.

Imbalanced Rotation: In §5.3, we explore the importance of balanced rotation in neural network training. We find that training configurations that result in more balanced rotation perform better, both in the case of AdamW compared to Adam+ℓ_2 and when comparing additional Weight Standardization to a baseline with only Layer Normalization. However, it is possible that other differences between these setups also effect the performance. To directly test the impact of imbalanced rotation, while eliminating as many confounding factors as possible, we construct a modified variant of RV-AdamW. In this modified RV, we additionally scale the angular updates of some fraction $1-p$ of the neurons in each layer by a factor of f. We consider two distributions shown in fig. 18L, either rotating all affected neurons slower or rotating half of them faster and the other half of them slower. For each configuration (p and f) we tune the weight decay, which shifts the rotational distribution through η_r while leaving the update size η_g for biases and gains unaffected.

The results can be seen in fig. 18R for a ResNet-18 trained on CIFAR-10. We see that the performance degrades when increasing either the portion of affected neurons or the scaling factor for either artificially imbalanced rotational distribution. The final panel shows how the previous imbalanced rotation compares with simply reducing the width of the network (completely removing neurons). Interestingly, the second and third panels show that a scaling factor of around $3\times$ applied to 50% of neurons results in performance comparable to a network with half the width. This indicates that imbalanced rotation directly and significantly harms performance. Why this happens is not entirely clear to us as we also discuss in appx. M. Neurons that rotate at a different speed may simply not learn effectively, which would work similarly to decreasing the number of neurons in the network. Intuitively, slow moving neurons could also contribute to overfitting (which is more prominent with lower global learning rates). We also speculate that fast rotating neurons may hinder the learning of other neurons, perhaps by limiting the maximum stable (global) learning rate or by changing the internal representations of the network too quickly.

Dataset	Model	Zero Mean	Invariance
CIFAR-10	ResNet-18	94.5±0.4 (**yn**)	94.5±0.4 (**yn**)
		94.3±0.2 (**nc**)	94.4±0.2 (**yt**)
Wikitext	GPT2-124M	18.5 ±0.42 (**yc**)	18.5±0.42 (**yc**)
		18.4 ± 0.4 (**nc**)	18.4±0.15 (**yt**)

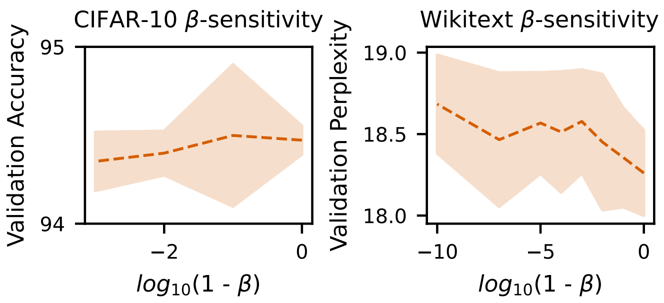

Figure 19: Experimental results for hyperparameter sensitivity. We report perplexity (↓) on Wikitext validation dataset and top-1 Acc. (↑) on a random validation split on CIFAR-10. In the default set up weight centering (y) and per neuron (n) scale-invariance is enabled.

Hyperparameter Sensitivity: The RVs introduce one significant hyperparameter, the decay rate β. Further, we can decide whether to enable (y) or disable (n) centering of the weights (zero-mean) and whether to control the angular updates on the layer (l) or neuron (channel, c) level. In this section we study the sensitivity of these choices in two different setups. We train a ResNet-18 on a random train split of CIFAR-10 with the RV of SGDM and a GPT2-124M model on Wikitext with the RV of AdamW. The results are shown in fig. 19. They indicate that the performance of the RVs remains relatively stable when the hyperparameters are varied. Note that we tuned the effective update size for each configuration by varying the weight decay for the ResNet-18 experiments, but run the sensitivity study only for one learning rate, weight decay setting for the GPT2-124M model due to resource constraints. Here we find little difference between the neuron and layer levels, but believe this difference may be larger in other settings that are more prone to imbalanced rotation on the neuron level.

K. Experimental Details

We perform our experiments on several popular datasets, i.e., CIFAR-10/100 (Krizhevsky, 2009) and Imagenet-1k (Russakovsky et al., 2015) for image classification, IWSLT2014 (Cettolo et al., 2014) for German-English translation, and Wikitext (Merity et al., 2017) and OpenWebText (Radford et al., 2019) for language modelling. Our code utilizes the TIMM library (Wightman, 2019) for vision tasks, FairSeq (Ott et al., 2019) for translation, and NanoGPT (Karpathy, 2023) and LLM-Baselines (Pagliardini, 2023) for language modelling.

We train commonly used architectures, namely ResNet-20, ResNet-18, ResNet-50 (He et al., 2016), DeiT tiny (Touvron et al., 2021), a small transformer, and a GPT2-124M network (Radford et al., 2019) from scratch.

General Setup: For all experiments trained with SGDM we use a momentum of 0.9, for experiments trained with AdamW we used $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and for experiments trained with our RVs we used $\beta = 0.99$, unless otherwise stated. For Lion we used $\beta_1 = 0.9$ and $\beta_2 = 0.999$ exclusively. Most of the experiments are run on a single NVIDIA A100-SXM4-40GB GPU.

Note that we used default architectures for the baseline experiments, but used a learnable gain for DeiT trained on Imagenet-1k, Transformer-S trained on IWSLT2014 de-en and the GPT2-124M architecture on Wikitext and OpenWebText. As mentioned in §4 we do this since transformers are not scale-invariant so constraining the weight norms may be harmful.

Here we list additional details referenced in the **details**-column in table 4 and 5:

D-1 We pre-process the data by normalizing it with mean $(0.49140.48220.4465)$ and std $(0.20230.19940.2010)$. For training we used simple data augmentation from He et al. (2016).

D-2 We use the standard data augmentation from He et al. (2016) for Imagenet.

D-3 Analogously to Touvron et al. (2021) we apply strong data augmentation. We use color jitter brightness up to 0.3, auto-augmentation, random erase with probability 0.25, drop path with probability 0.1, mixup with probability 0.8 and cutmix with probability 1.0. Additionally, we use label smoothing of 0.1.

Note that in this experiment, we implemented a $\cos^2$ schedule for the RV few-shot training. We observed that the baseline training exhibited a smaller rotation towards the end. Consequently, we adjusted the effective update size for the RV training to more closely align with the baseline run's effective learning rate schedule.

D-4 We use standard FairSeq library (Ott et al., 2019) with dropout probability 0.3.

Note that additionally to using weight standardization with learnable gain, we set the weight decay to 0 for scale-sensitive weights. This is done by default for the vision tasks in TIMM library (Wightman, 2019), but not by FairSeq library (Ott et al., 2019) we used for this task. We observed no difference in performance for the baseline model, yet this adjustment allowed to tune the effective update size for the scale-invariant weights, without affecting the learning rate of the scale-sensitive weights significantly.

D-5 For this experiment we use the llm-baseline library (Pagliardini, 2023). For the GPT2-124M architecture, we use vocabulary size of 50304, sequence length of 512, embedding size of 768. The model features 12 repeated blocks, each comprising a self-attention block followed by a two-layer MLP block with hidden dimension 3072. This results in a total of 124 million parameters. We use a drop probability of 0.2 for dropout.

D-6 For this experiment we use the nanoGPT library (Karpathy, 2023). The details are the same as for D-1, except the sequence length is 1024 and no dropout is used. The model is trained for 5000 iterations which corresponds to roughly 20 tokens per parameter, inspired by Chinchilla (Hoffmann et al., 2022). The learning rate schedule goes down to 0, which we found to be easier to combine with learning rate and weight decay sweeps without affecting the performance significantly in exploratory experiments.

Constraining the Rotational Dynamics: The experimental details for the experiments reported in tables 2 and 3 can be found in table 4.

Measuring the Update Dynamics & Dynamics of Scale-Sensitive GPT Model: For the experiments in fig. 3, we generally used the same settings for ResNet-50 on ImageNet-1k trained with SGDM and the GPT2-124M model trained with AdamW on OpenWebText. However, we use a constant learning rate without warmup, train for a 100k steps and use the default learning rate from NanoGPT, $6\mathrm{e}{-4}$ for the GPT2 model. We use Weight Standardization (with learnable gains) for GPT2 in fig. 3 but not fig. 15.

Learning Rate vs Weight Decay: For the ResNet-20 experiment on CIFAR-10 and language model task on Wikitext with a GPT2-124M model we use the few shot (zero shot) setting reported in table 4 as default. We then sweep over the learning rate keeping $\eta\lambda = 5{\cdot}10^{-4}$, $\eta\lambda = 2.5{\cdot}10^{-3}$ respectively, constant.

Transient Effects: In the experiment described in the preceding paragraph, we monitored $\|\boldsymbol{\omega}\|$, $\angle(\boldsymbol{\omega}_t, \boldsymbol{\omega}_{t+1})$ during training for one of the runs with the chosen learning rates: $1{\cdot}10^{-4}$, $8.3{\cdot}10^{-3}$, and $3{\cdot}10^{-1}$.

Need for Learning Rate Warmup: For the ResNet-50 experiment on ImageNet-1k we follow the same base setup as we report in table 4. We train for a total of 10 epochs using a cosine decay schedule (applied stepwise) and no warmup. We use 32 local accumulations on top of batches of size 256 to emulate a batch size of 8'192 and sweep the learning λ in the range $2^i \cdot 0.1$, with $i \in [0, \dots, 9]$. The GPT2 experiments follow the setup in table 4 with the learning rate swept over $\{6 \cdot 10^{-4} \cdot 2^i\}$ for $i \in [-1, \dots, 5]$.

Adam vs AdamW: For the sweep we train a ResNet-18 on a 90/10 train/val split from the original train set. We use a step-wise cosine schedule and train for 200 epochs without warmup. In this sweep we try to reproduce the results in figure 2 from (Loshchilov & Hutter, 2019), albeit with a slightly different network and training for 200 epochs instead of 100. The best configuration for Adam was $\eta = 7.813 \cdot 10^{-4}$, $\lambda = 1.250 \cdot 10^{-4}$ resulting in a validation set accuracy of 93.919%. The best configuration for AdamW was $\eta = 1.25 \cdot 10^{-2}$, $\lambda = 8.0 \cdot 10^{-2}$ with a validation accuracy of 94.319%. On the test set we run each configuration over 5 different seeds, obtaining test accuracies of $94.08 \pm 0.16\%$ for Adam+ℓ_2 and $94.74 \pm 0.14\%$ for AdamW.

Benefit of Weight Standardization: We train a ResNet-18 with layer normalization and weight standardization on top of layer normalization on CIFAR-100 using the same augmentation, learning rate schedule and base hyperparameters as for the ResNet-20 on CIFAR-10 experiments, unless otherwise noted below. We train on a random subset containing 90% of the train set and use the remaining 10% for validation which we report. The inputs are normalized for mean `(0.5071, 0.4867, 0.4408)` and std `(0.2675, 0.2565, 0.2761)`. We use a weight decay of $5 \cdot 10^{-4}$ and a batch size of 256. The layer normalization is implemented with a Group Normalization (Wu & He, 2018) using a single group.

Table 4: Experimental set up (include training set and test set definition).

Dataset	Model	Optimizer	Batch Size	zero shot	few shot	details	lr schedule	warmup	epochs (e)/ iteration (it)	train duration	precision
CIFAR-10	ResNet-20	SGD	128	wd=1e−4 lr=0.5	N/A	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
CIFAR-10	ResNet-20	SGD	2048	wd=1e−4 lr=4.8	wd=2e−4 lr=4.8	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
CIFAR-10	ResNet-20	AdamW	128	wd=1e−2 lr=5e−2	$\beta = 0.9$	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
CIFAR-10	ResNet-20	AdamW	2048	wd=1e−2 lr=1.6e−1	wd=8e−2 lr=1.6e−1	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
CIFAR-10	ResNet-20	Lion	128	wd=1.0 lr=5e−4	N/A	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
CIFAR-10	ResNet-20	Lion	2048	wd=1.0 lr=1.6e−2	wd=2.0 lr=1.6e−2	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
Imagenet-1k	ResNet-50	SGD	256	wd=1e−4 lr=1e−1	N/A	(D-2)	cosine	lr=1e−6 5 epochs	90 (e)	30h	float16
Imagenet-1k	DeiT tiny	AdamW	1024	wd=5e−2 lr=5e−4	wd=2e−1 lr=5e−4	(D-3)	cosine	lr=1e−6 5 epochs	300 (e)	70h	float16
IWSLT2014 de-en	Transformer-S	AdamW	4096	wd=1e−4 lr=5e−4 $\beta_2 = 0.98$	wd=4e−1 lr=5e−4 $\beta_2 = 0.98$	(D-4)	cosine	4000 (it)	22021 (it)	50min	float16
Wikitext	GPT2-124M	AdamW	55×3	wd=0.5 lr=5e−3 $\beta_2 = 0.95$	N/A	(D-5)	cosine div_f=1e2 final_div_f=1e4	2e−2 (%)	15000 (it)	3h	bfloat16
OpenWebText	GPT2-124M	AdamW	12×40	wd=0.1 lr=4.8e−3 $\beta_2 = 0.95$	N/A	(D-6)	cosine min_lr=0	250 (it)	5000 (it)	4h	bfloat16

Table 5: Experimental details for hyperparameter senstivity study.

Dataset	Model	Optimizer	training dataset	validation dataset	hyper-parameters	details	lr schedule	warmup	epochs (e)/ iterations (it)	train duration	precision
CIFAR-10	ResNet-18	SGD	90% Train	10% Train	wd=N/A lr=1.0	(D-1)	cosine	lr=1e−6 5 epochs	200 (e)	35min	float32
Wikitext	GPT2-124M	AdamW	Train	Validation	wd=0.5 lr=4e−3 $\beta_2 = 0.95$	(D-5)	cosine div_f=1e2 final_div_f=1e4	2e−2 (%)	15000 (it)	3h	bfloat16

Imbalanced Rotation: We trained a ResNet-18 on a 90/10 train/val split from the original train set with learning rate $\eta = 0.1$, batch size 256 and varying weight decay to tune the effective updated size. For each rotation speed f, portion p and network width scaling we report the average performance over the best configuration of this sweep. All other settings are equivalent to the settings reported in table 4.

Hyperparameter Sensitivity: The experimental details for the experiments reported in fig. 19 can be found in table 5. Note, that we don't specify the weight decay used for the ResNet-18 experiment, since we tuned the weight decay for each configuration with $2^i \cdot \lambda, i \in [-64, 32]$. We use a batch size of 256 and a batch size of 55 with 3 accumulation steps for the GPT2-124M model.

Table 6: An extension of table 1 showing values for diffusion rates based on the expected Total Update Contribution (TUC) of the gradients from a single timestep, corresponding to the update sizes. The TUC attempts to capture the rate of longer-term changes rather than the instantaneous update sizes, accounting for the increased alignment from momentum.

Measure	Definition	SGDM eq. (45)	AdamW eq. (11)
Equilibrium norm $\widehat{\|\boldsymbol{\omega}\|}$	$\sqrt{\mathbb{E}[\|\boldsymbol{\omega}\|^2]}$	$\sqrt[4]{\frac{\eta\mathbb{E}[\|\tilde{\boldsymbol{g}}\|^2]}{2\lambda\cdot(1-\alpha)}}$	$\sqrt{\frac{\eta C}{2\lambda}}$
RMS update size $\widehat{\eta_g}$	$\sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{p}\|^2]}$	$\eta\sqrt{\frac{\mathbb{E}[\|\boldsymbol{g}\|^2]}{1-\alpha^2}}$	$\eta\sqrt{C\frac{1-\beta_1}{1+\beta_1}}$
RMS diffusion rate $\widehat{\tau_g}$	$\sqrt{\mathbb{E}[\|\boldsymbol{u}\|^2]}$	$\frac{\eta}{1-\alpha}\sqrt{\mathbb{E}[\|\boldsymbol{g}\|^2]}$	$\eta\sqrt{C}$
Expected rotation $\widehat{\eta_r}$	$\sqrt{\mathbb{E}[\|\Delta_g \boldsymbol{p}\|^2]}/\widehat{\|\boldsymbol{\omega}\|}$	$\sqrt{\frac{2\eta\lambda}{1+\alpha}}$	$\sqrt{2\eta\lambda\frac{1-\beta_1}{1+\beta_1}}$
Rotational diffusion rate $\widehat{\tau_r}$	$\sqrt{\mathbb{E}[\|\boldsymbol{u}\|^2]}/\widehat{\|\boldsymbol{\omega}\|}$	$\sqrt{\frac{2\eta\lambda}{1-\alpha}}$	$\sqrt{2\eta\lambda}$

L. Update Size vs Learning Rate

An update size, such as the average angular update η_r or RMS update η_g, specifically refers to a measure of the size of individual updates. This does not necessarily correlate directly to the net change over a longer period of multiple steps, e.g. an epoch of training. The longer term change is determined by the shape of the optimization trajectory which depends on the size and direction of the individual steps. For a fixed update size, a more consistent direction in the updates will cause a larger net change over a time period. For this reason we prefer to use the term "effective update size" rather than "effective learning rate" which is sometimes used to refer to measures of long term changes. We note that the update sizes are easier to measure and control, although the long term changes may be more informative for hyperparameter tuning and adjustment.

The difference is particularly important when using momentum which has a somewhat unintuitive effect in the random walk setting, e.g. for SGDM and AdamW. As seen in table 1 higher momentum coefficients decrease the average rotation in each step but we know they will also cause additional correlation between successive steps. This can be viewed as smoothing the optimization trajectory, it will have smaller random fluctuations in each step but the averaged "velocity" is not necessarily smaller. Comparing the average rotation per step between optimizers with different amounts of momentum will therefore not necessarily correlate with measures of how fast parameters are being updated over longer time intervals.

In the random walk setting the long term change is likely to be proportional to the total update contribution $\|\boldsymbol{u}\|$ rather than the size of a single update step $\|\Delta_g \boldsymbol{p}\|$ (see definitions in §3.1). Analogously, the total rotational contribution from a single gradient sample would be roughly $\|\boldsymbol{u}\|/\widehat{\|\boldsymbol{\omega}\|}$ rather then $\|\Delta_g \boldsymbol{p}\|/\widehat{\|\boldsymbol{\omega}\|}$ used in §3.2. These total update contributions may then add up orthogonally assuming successive gradients behave like in the random walk (i.e. that they are independent and zero mean in expectation). We refer to this longer term speed as **diffusion rates** based on a loose analogy with random walks like Brownian Motion. Table 6 shows the modified rates for SGDM and AdamW. The diffusion rates are more consistent with the way learning rate is typically scaled with the momentum for SGDM, i.e. keeping $\eta/(1-\alpha)$ constant rather than $\eta/(1+\alpha)$, see e.g. Chiley et al. (2019) and Fu et al. (2023).

M. Why Balanced Rotation Might Work

We empirically observe that balanced rotation seems to perform better than the imbalanced rotation resulting from various methods. The reasons for this are not fully clear to us, but we believe that balanced rotation is a heuristic that ensures that all neurons and layers are updated at a reasonable speed. For example, we intuitively expect that a layer which is updated very slowly, perhaps barely changing through the training process, will unlikely contribute optimally to the final model resulting in worse performance and wasted compute. Conversely, a rapidly changing layer may cause instability, limiting the maximum stable learning rate and preventing other layers from learning effectively. This suggests that sufficiently (and adversarially) imbalanced rates are not optimal. At the same time it seems unlikely that exactly balanced rates are optimal in all settings, exhaustively tuning the rotation speed of each component would likely result in some specific optimal, imbalanced, rotation for a given problem.

Modern neural networks have a complex structure which can give rise to various effects that scale the gradient of one component relative to another in arbitrary ways. Neyshabur et al. (2015) gives an example of this for ReLU networks (without branches), where one layer can be scaled up by a positive factor and another down. This preserves the encoded function but will change the optimization trajectory and can significantly degrade performance. Scale-invariance is another example, where scaling the weights of a layer will not affect the network outputs but changes the angular updates and therefore the optimization trajectory for standard optimizers. Finally we can change the network architecture by e.g. inserting a fixed scaling factor into the computational graph that scales a given parameter (whose value is adjusted to compensate) which in turns scales the gradients changing the optimization trajectory.

These examples show that the relative gradient magnitude of one component compared to another may simply not be very meaningful quantity in neural networks. Perhaps higher order information like curvature could allow us to determine what the relative update size should be. Without this information, normalizing the update magnitude through methods like Adam, Sign-SGD or Rotational Optimizer Variants may act as heuristics that are better than using the arbitrary gradient magnitude. Proving this might be infeasible, these heuristics may only help in specific cases dependent on the network architecture, initialization, dataset and so on, or it may at least be possible to construct adversarial scenarios where they hinder training. We do not pursue this further here and simply treat balanced rotation as an empirically useful heuristic that seems to play a role in the effectiveness of many influential methods in deep learning.