Understanding Optimization in Deep Learning with Central Flows

Jeremy Cohen*
Carnegie Mellon and Flatiron Institute
jmcohen.github.io

Alex Damian*
Princeton University
alex-damian.github.io

Ameet Talwalkar
Carnegie Mellon University

J. Zico Kolter
Carnegie Mellon University

Jason D. Lee
Princeton University

Abstract

Traditional theories of optimization cannot describe the dynamics of optimization in deep learning, even in the simple setting of deterministic training. The challenge is that optimizers typically operate in a complex, oscillatory regime called the *edge of stability*. In this paper, we develop theory that can describe the dynamics of optimization in this regime. Our key insight is that while the *exact* trajectory of an oscillatory optimizer may be challenging to analyze, the time-averaged (i.e. smoothed) trajectory is often much more tractable. To analyze an optimizer, we derive a differential equation called a *central flow* that characterizes this time-averaged trajectory. We empirically show that these central flows can predict long-term optimization trajectories for generic neural networks with a high degree of numerical accuracy. By interpreting these central flows, we are able to understand how gradient descent makes progress even as the loss sometimes goes up; how adaptive optimizers "adapt" to the local loss landscape; and how adaptive optimizers implicitly navigate towards regions where they can take larger steps. Our results suggest that central flows can be a valuable theoretical tool for reasoning about optimization in deep learning.

1 Introduction

While there is a rich body of work on the theory of optimization, few works attempt to analyze optimization in "real" deep learning settings. Instead, even works motivated by deep learning often rely on unrealistic assumptions such as convexity, or restrict their analyses to simplified models. Practitioners cannot use such theories to reason directly about their optimization problems. Our goal in this paper is to develop optimization theory that applies *directly* to deep learning problems. This is a difficult task: prior research has shown that, even in the seemingly simple setting of deterministic (i.e. full-batch) training, optimization typically operates in a complex, oscillatory regime called the *edge of stability* (EOS) (Xing et al., 2018; Wu et al., 2018; Jastrzębski et al., 2019, 2020; Cohen et al., 2021, 2022). The dynamics of optimization in this regime cannot be captured by traditional optimization theory.

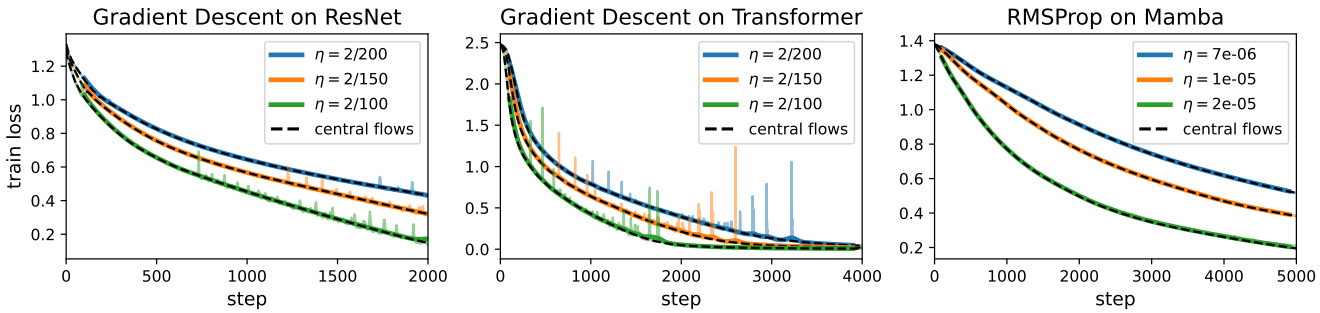


Figure 1: **Our theory accurately predicts long-term optimization trajectories of practical neural networks**. We hold our theory to the high standard of rendering accurate *numerical* predictions about the optimization of practical (i.e. non-toy) neural networks. For example, this figure shows that our central flows can accurately predict the time-averaged (smoothed) loss curves of gradient descent and RMSProp on various practical architectures.

*Equal contribution; author ordering determined by coin flip over a Zoom call (Kingma and Ba, 2015). Please direct correspondence to both the first authors (see websites for latest contact information). Alex Damian is now at Harvard and Jason D. Lee is now at U.C. Berkeley. This is the full version of a paper that was published at ICLR 2025.

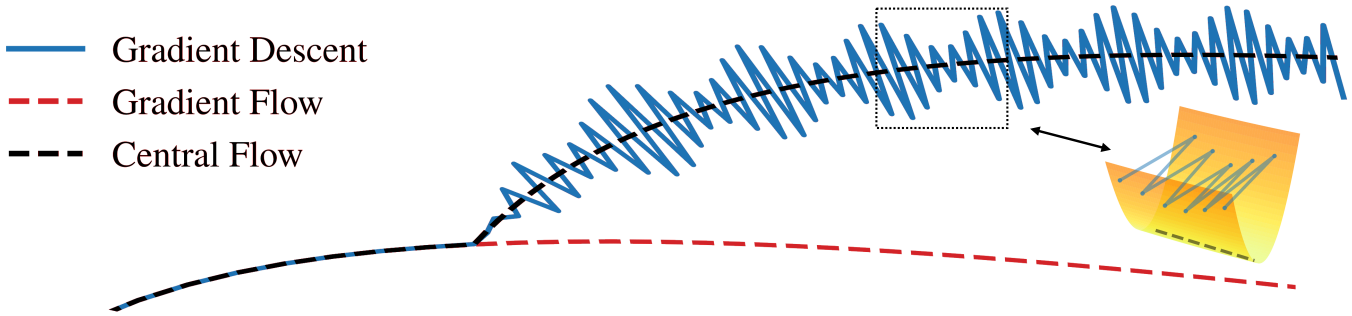


Figure 2: **The central flow models the time-averaged (i.e. smoothed) trajectory of the oscillatory optimizer**. In this illustrative cartoon of the weight-space dynamics, gradient descent (blue) takes an oscillatory path through weight space. The central flow (black) is a smooth curve that characterizes this trajectory, whereas gradient flow (red) takes a different path. As illustrated in the inset, an oscillatory optimizer can be visualized as moving through a "valley" while bouncing between the "valley walls" (Xing et al., 2018; Cohen et al., 2021; Wen et al., 2025).

In this paper, we devise a methodology for analyzing these oscillatory deep learning dynamics. Our key insight is that while the *fine-grained* trajectory of an oscillatory optimizer may be challenging to analyze, the *time-averaged* (i.e. locally smoothed) trajectory is often much more tractable. To analyze an optimization algorithm, we derive a differential equation called a *central flow* which explicitly captures this time-averaged trajectory (Figure 2). Being a smooth curve, the central flow is a simpler object than the original oscillatory trajectory. Hence, by interpreting the central flow, we can reason more easily about the original optimizer.

We start in Section 3 by analyzing gradient descent, the simplest optimizer. We first explain why traditional analyses cannot capture the typical dynamics of gradient descent in deep learning, and we then present a new analysis that does capture these dynamics. The product of this analysis is a central flow. We use this central flow to understand various aspects of gradient descent's behavior, such as how the train loss can behave non-monotonically over the short-term while nevertheless decreasing over the long term. We then examine a simple adaptive optimizer in Section 4, before turning to RMSProp (i.e. Adam without momentum) in Section 5. We show that much of the behavior of these optimizers is actually *implicit* in their oscillatory dynamics, and we render such behaviors *explicit* via our central flow analysis. In particular, our central flows reveal how these adaptive optimizers: (1) implicitly adapt their step size(s) to the local curvature, and (2) implicitly steer towards lower-curvature regions where they can take larger steps.

We focus in this paper on the simple, idealized setting of deterministic (i.e. full-batch) training. However, we emphasize that similar optimization dynamics have been observed in the practical stochastic setting (Jastrzębski et al., 2019, 2020; Andreyev and Beneventano, 2024). We view our analysis of deterministic optimization as a necessary stepping stone to a subsequent analysis of stochastic optimization.

While we derive each central flow using informal mathematical reasoning, we show that these flows can accurately predict long-term optimization trajectories in a variety of deep learning settings — a high standard of empirical proof. Thus, we believe that central flows hold promise as a framework for analyzing, reasoning about, and perhaps even inventing, deep learning optimizers.

We strongly encourage readers to look at the companion **blog version** of this paper for an interactive exposition with animations, as well as the accompanying **code** we used to simulate the central flows.[1]

[1]The blog version is `centralflows.github.io` and the code is at `github.com/centralflows/centralflows`.

Table of Contents

2 Related Work

Edge of stability The dynamics of optimization in deep learning remain poorly understood, even in the seemingly simple setting of deterministic (i.e. full-batch) training. Indeed, recent research showed that gradient descent on neural networks typically operates in a regime termed the "edge of stability" (EOS) in which (1) the largest Hessian eigenvalue equillibrates around the *critical threshold* $2/\eta$, and (2) the algorithm oscillates along high-curvature directions without diverging (Xing et al., 2018; Wu et al., 2018; Jastrzębski et al., 2019, 2020; Cohen et al., 2021). These dynamics could not be explained by existing optimization theory, which led Cohen et al. (2021) to observe that there was no explanation for how or why gradient descent can function properly in deep learning.

Subsequently, several studies sought to theoretically explain EOS dynamics. Some works rigorously analyzed EOS dynamics on specific objective functions (Agarwala et al., 2023; Ahn et al., 2024; Chen and Bruna, 2023; Even et al., 2024; Kreisler et al., 2023; Song and Yun, 2023; Li et al., 2022b; Wu et al., 2024; Zhu et al., 2023), while other works (Arora et al., 2022; Lyu et al., 2022; Damian et al., 2023) gave generic analyses based on a local *third-order* Taylor expansion of the loss, which is one order higher than is normally used in the theoretical analysis of gradient descent. Similar arguments were first used by Blanc et al. (2019) to study implicit regularization in SGD. Our work is most directly inspired by Damian et al. (2023), as their analysis applies to generic objective functions, and holds throughout training, not just near convergence. However, whereas they analyze the *fine-grained* oscillatory dynamics, we argue that analyzing the *time-averaged* dynamics is simpler, and is sufficient for most purposes.

Continuous-time models for optimization The standard continuous-time model for gradient descent is the gradient flow. Barrett and Dherin (2021); Smith et al. (2021) argued that gradient descent is, instead, better approximated by a *modified* gradient flow that is augmented with a penalty on the squared gradient norm. We find that on deep learning objectives, this modified flow improves slightly over gradient flow in the stable regime, but fails in the edge of stability regime, where most of the discrepancy between gradient descent and gradient flow originates (see Section C.1). Rosca et al. (2023) proposed a flow that can model oscillations by using complex numbers. However, this flow still cannot track the long-term trajectory of gradient descent in EOS regime.

Many works propose to model the dynamics of stochastic optimizers using stochastic differential equations (SDEs) (Li et al., 2017; Li et al., 2021, Malladi et al., 2022; Compagnoni et al., 2023, 2025). In the full batch limit, where SGD reduces to gradient descent, these SDEs reduce to gradient flow, which is a poor approximation to gradient descent at the edge of stability. Thus, these SDEs cannot be accurate in all hyperparameter regimes. Further, even when these SDEs do well-approximate the real optimizer trajectory, the SDE trajectories are themselves oscillatory, and accordingly can possess behaviors that are *implicit* in the oscillatory dynamics. Our central flows, by contrast, average out the oscillations and render all such behaviors *explicit*. Developing an analogue of the central flow for stochastic optimization is an interesting open question (see Section 7). While some works do aim to explicitly characterize the time-averaged trajectory of SGD (Blanc et al., 2019; Damian et al., 2021; Li et al., 2022a), existing analyses only apply in limiting regimes (e.g. $\eta \to 0$), and only when the loss is already near zero.

Understanding adaptive optimizers Ma et al. (2022) observed that RMSProp and Adam oscillate, and Cohen et al. (2022) showed that such dynamics can be viewed as an adaptive version of the edge of stability.Khaled et al. (2023) and Mishkin et al. (2024) observed that on quadratic functions, certain adaptive optimizers implicitly adapt their effective step size to the maximum stable step size; we show this holds more generally, beyond quadratics. The phenomenon we call "acceleration via regularization." explains experiments in Roulet et al. (2024) and Wang et al. (2024d). Many works have also conducted rigorous convergence analyses of adaptive optimizers, generally focused on deriving rates of convergence to a global minimizer or stationary point (Duchi et al., 2011; Reddi et al., 2018; Chen et al., 2019a,b; Zaheer et al., 2018; Zou et al., 2019; Défossez et al., 2022; Li and Lin, 2024; Chen et al., 2022; Wang et al., 2024a; Yang et al., 2024; Guo et al., 2021; Shi et al., 2021; Zhang et al., 2022; Crawshaw et al., 2022; Li et al., 2024; Wang et al., 2024b; Hong and Lin, 2024; Zhang et al., 2025; Wang et al., 2024c; Hübler et al., 2024).

Dynamical systems Our work likely has rich connections to various topics from the theory of dynamical systems such as the method of averaging and slow-fast systems (e.g., Guckenheimer and Holmes, 1983). We hope that these connections can be explored by future work.

3 Gradient Descent

The simplest first-order optimizer is deterministic gradient descent with a fixed learning rate η:

$$w_{t+1} = w_t - \eta \nabla L(w_t). \tag{1}$$

Perhaps surprisingly, Cohen et al. (2021) showed that traditional optimization analyses cannot capture the typical dynamics of gradient descent in deep learning. We now present a new analysis that *does* capture these dynamics.

- In Section 3.1, we describe the typical dynamics of gradient descent in deep learning, and we explain why these oscillatory *edge of stability* dynamics cannot be captured by traditional optimization theory.
- In Section 3.2, we show that while the *exact* oscillatory trajectory may be hard to analyze, the *time-averaged* trajectory is more tractable. We derive a central flow that characterizes this time-averaged trajectory.
- In Section 3.3 we use this central flow to understand the behavior of gradient descent. For example, we show that while gradient descent's loss curve is non-monotonic, it can be viewed as the superposition of the loss along the central flow, plus a contribution from the oscillations. The central flow loss is a smoothly varying quantity that monotonically decreases, and therefore constitutes a hidden progress metric for gradient descent.

Our analysis of gradient descent will set the stage for subsequent analyses of more complex optimizers.

3.1 The Dynamics of Gradient Descent

To understand the oscillatory dynamics of gradient descent in deep learning, it is instructive to first consider the case of quadratic objective functions. On quadratic functions, gradient descent oscillates if the *curvature* (i.e. Hessian) is too large relative to the learning rate. For example, consider a one-dimensional quadratic objective $L(x) = \frac{1}{2}Sx^2$, which has global curvature S. Under gradient descent with learning rate η, the iterates $\{x_t\}$ evolve via $x_{t+1} = (1-\eta S)x_t$. If S exceeds the *critical threshold* $2/\eta$, then $(1-\eta S) < -1$, so the iterate x_t flips signs and grows in magnitude at each step, i.e. gradient descent oscillates with exponentially growing magnitude, as shown in Figure 3. More generally, on a quadratic objective in multiple dimensions, the curvature is quantified by the Hessian matrix, and gradient descent oscillates with exponentially growing magnitude along Hessian eigenvectors with eigenvalues exceeding $2/\eta$.[2]

Of course, deep learning objectives $L(w)$ are not globally quadratic. Still, at any point w in weight space, the objective can be locally approximated by a quadratic Taylor expansion around w. The dynamics of gradient descent on this quadratic are controlled by the largest eigenvalue of the Hessian $H(w)$, which we call the *sharpness* $S(w)$:

$$S(w) := \lambda_1(H(w)). \tag{2}$$

If the sharpness $S(w)$ exceeds $2/\eta$, then gradient descent on the quadratic Taylor approximation would oscillate with exponentially growing magnitude along the top Hessian eigenvector(s). This argument suggests that gradient descent cannot function properly in regions of weight space where the sharpness $S(w)$ exceeds $2/\eta$.

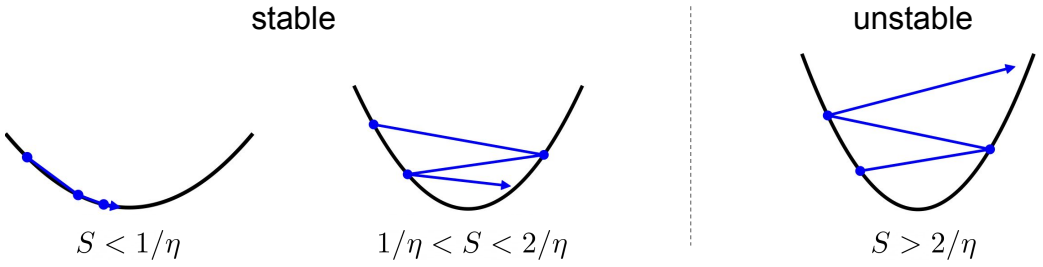
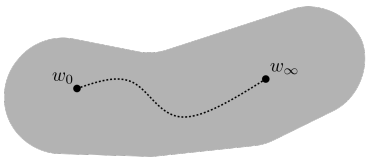
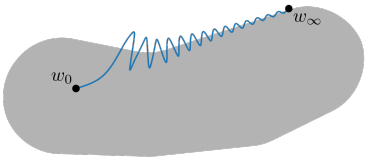


Figure 3: **Gradient descent on a quadratic function.** Consider gradient descent with learning rate η on a quadratic function $\frac{1}{2}Sx^2$, with sharpness S. If $S > 2/\eta$, gradient descent oscillates with exponentially growing magnitude.

[2]For an explicit derivation of this well-known fact, see Cohen et al. (2021, Proposition 1). Note that an exception is if the initial iterate has exactly zero alignment with these Hessian eigenvectors. However, this event has probability zero under any typical random initialization.



(a) **Expectation:** gradient descent stays throughout training inside the *stable region* (gray), the subset of weight space where the sharpness is bounded by $2/\eta$.



(b) **Reality:** gradient descent frequently exits the stable region, but dynamically steers itself back inside.

Figure 4: **Why does gradient descent converge in deep learning?** The reality (**right**) is dramatically different from the picture suggested by traditional theory (**left**).

In light of this discussion, why does gradient descent converge in deep learning? The natural explanation is that the sharpness remains below $2/\eta$ throughout training. In other words, if we define the "stable region" $\{w : S(w) \leq 2/\eta\}$ as the subset of weight space where the sharpness is bounded by $2/\eta$, then one might suppose that gradient descent remains inside the stable region throughout training, as depicted in the cartoon Figure 4(a). This is the picture suggested by traditional analyses of gradient descent.[3]

Yet, Cohen et al. (2021) observed a very different reality when training neural networks using gradient descent. Figure 5 depicts a typical gradient descent trajectory, with important events annotated (a) - (g). Initially, the sharpness rises (a).[4] Indeed, it is a robust empirical phenomenon, dubbed *progressive sharpening*, that the sharpness tends to rise when training neural networks.[5][6] Soon enough, the sharpness rises past the critical threshold $2/\eta$. Once this happens, gradient descent begins to oscillate with growing magnitude along the top Hessian eigenvector,[7] just as one would predict from a quadratic Taylor approximation (b). These oscillations grow large enough that the train loss starts to go up rather than down (c). Yet, gradient descent does not diverge. Instead, something odd happens: as if "by magic," the sharpness rapidly *drops* (d). Indeed, it drops all the way below the critical threshold $2/\eta$, after which point the oscillations start to shrink in magnitude (e), as one would expect from a new quadratic Taylor approximation. The unexplained rapid drop in the sharpness has conveniently prevented gradient descent from diverging.[8] Similar dynamics recur throughout the rest of training: gradient descent oscillates without diverging along the highest-curvature direction(s),[9] as the sharpness stays dynamically regulated around the critical threshold $2/\eta$ (f). Meanwhile, the train loss decreases over the long run, but behaves non-monotonically over the short run (g).

Intuitively, whereas the traditional theory implies that gradient descent remains inside the stable region throughout training, as in Figure 4(a), in reality gradient descent is frequently exiting the stable region, but is somehow steering itself back inside, as in Figure 4(b). Cohen et al. (2021) dubbed these dynamics *edge of stability* (EOS), and noted that they could not be explained by traditional optimization theory.

[3]We are referring to analyses which assume L-smoothness, i.e. Lipschitzness of the gradient / boundedness of the Hessian spectral norm. This assumption is usually stated as a global condition, but analyses generally only require it to hold locally, in the vicinity of the trajectory.

[4]Throughout this paper, we report the Hessian eigenvalues measured not at the iterates themselves, but rather at the second-order midpoints between the iterates (see Section B.1). This results in plots that are slightly crisper, while retaining all essential features.

[5]Progressive sharpening remains theoretically unexplained. Our goal in this paper is not to understand the origin of progressive sharpening in deep learning, but rather to understand the dynamics of gradient descent on objective functions which may or may not possess this property.

[6]In the literature, progressive sharpening has also been called a "narrowing valley" (Liu et al., 2025a,b) and "lower loss as sharper" loss landscape structure (Li et al., 2023; Bai et al., 2025).

[7]To compute "displacement along top Hessian eigenvector" for Figure 5, we let t_0 be the first step of the figure (i.e. step 2990), we let u be the top Hessian eigenvector computed at step t_0, and we report $u^\top(w_t - w_{t_0})$.

[8]This process is similar to the "catapult" phenomenon observed in Lewkowycz et al. (2020) at initialization. Indeed, the EOS dynamics with one unstable eigenvalue resemble a sequential series of catapults. However, the dynamics with >1 unstable eigenvalues are more complex.

[9]As gradient descent oscillates along the high-curvature directions, it can be visualized as moving through a "valley" while bouncing between the "walls" of the valley (Xing et al., 2018; Cohen et al., 2021; Wen et al., 2025).

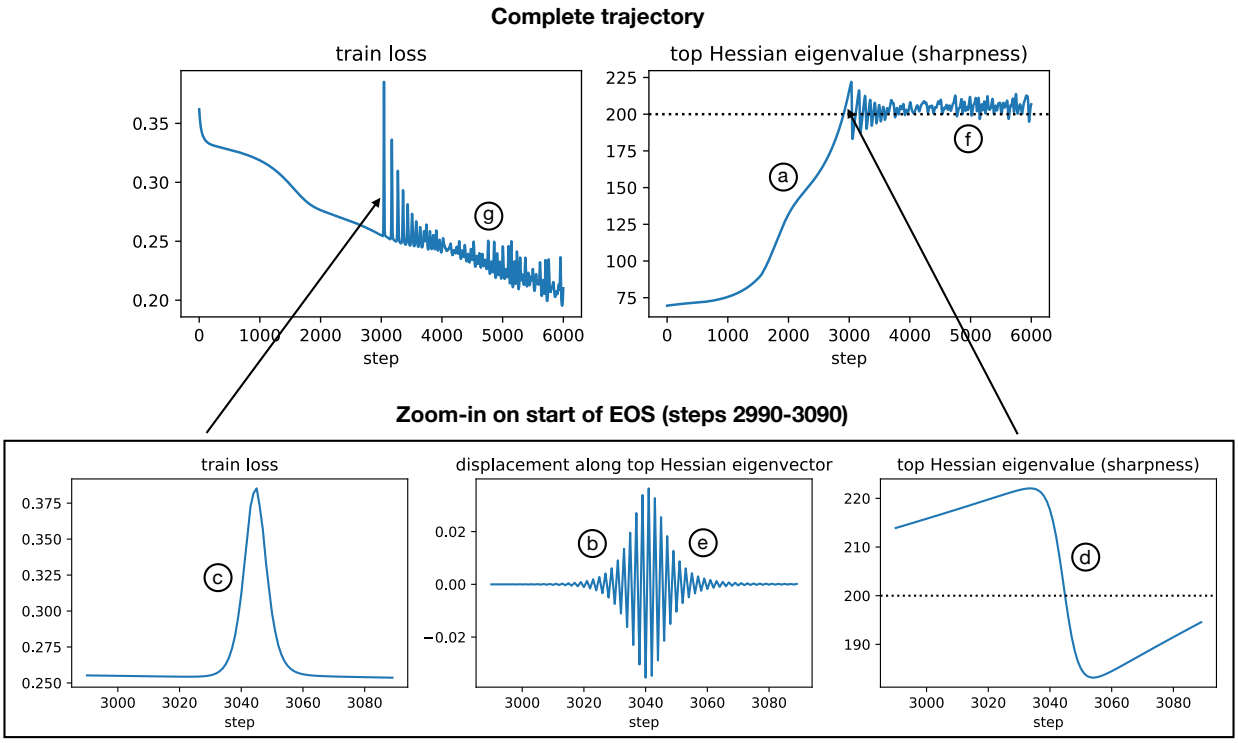
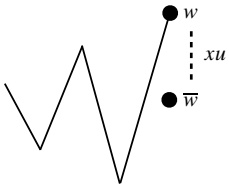


Figure 5: **A typical gradient descent trajectory in deep learning**. We train a neural network using gradient descent with step size $\eta = 0.01$. The top row shows the long-term trajectory, while the bottom row zooms in on a particular time segment. **(a)** The sharpness rises, reaching the critical threshold $2/\eta$ around step 2900. **(b)** Once the sharpness crosses the critical threshold $2/\eta$, gradient descent oscillates with growing magnitude along the top Hessian eigenvector. **(c)** These oscillations cause the train loss to go up rather than down. **(d)** However, gradient descent does not diverge; instead, as if "by magic", the sharpness decreases until falling below $2/\eta$. **(e)** Once the sharpness is below $2/\eta$, the oscillations shrink. Throughout the rest of training: **(f)** the sharpness stays regulated around the critical threshold $2/\eta$ and **(g)** the train loss behaves non-monotonically over short timescales, while decreasing over long timescales. *Details*: the network is a Vision Transformer trained on a subset of CIFAR-10 using MSE loss.

Damian et al. (2023) showed that the key for understanding these surprising dynamics is to Taylor-expand the objective to *third* order, which is one order higher than traditionally used in analyses of gradient descent. A third-order Taylor expansion reveals the crucial ingredient missing from traditional optimization theory:

Oscillations along the top Hessian eigenvector automatically trigger reduction of the top Hessian eigenvalue.

Let us informally sketch this argument. Suppose that gradient descent is oscillating around a reference point $\overline{w}$, along the top Hessian eigenvector u, with current magnitude x, so that (illustration on right):



$$w = \overline{w} + xu. \tag{3}$$

Due to the oscillation, the optimizer follows the gradient at w rather than the gradient at $\overline{w}$. How do the two relate? A Taylor expansion of ∇L around $\overline{w}$ yields:

$$\nabla L(\overline{w} + xu) = \underset{\text{first term}}{\nabla L(\overline{w})} + \underbrace{\underset{\text{second term}}{xH(\overline{w})u}}_{= xS(\overline{w})u} + \mathcal{O}(x^2). \tag{4}$$

Since u is an eigenvector of $H(\overline{w})$ with eigenvalue $S(\overline{w})$, we recognize the second term as $xS(\overline{w})u$. This term causes a negative gradient step computed at $\overline{w}+xu$ to move in the $-u$ direction. In other words, this term is causing gradient descent to oscillate back and forth along the top Hessian eigenvector u, as predicted by the traditional theory. The "magic" comes from the *next* term, which arises from third-order terms in the Taylor expansion of the loss:

$$\nabla L(\overline{w}+xu) = \overbrace{\nabla L(\overline{w})}^{\text{first term}} + \overbrace{xS(\overline{w})u}^{\text{second term}} + \overbrace{\underbrace{\tfrac{1}{2}x^2\nabla_{\overline{w}}[u^T H(\overline{w})u]}_{=\frac{1}{2}x^2\nabla S(\overline{w})}}^{\text{third term}} + \mathcal{O}(x^3). \tag{5}$$

Since $u^T H(\overline{w})u = S(\overline{w})$, we recognize this term as $\frac{1}{2}x^2\nabla S(\overline{w})$, where ∇S is none other than the *gradient of the sharpness*.[10] Thus, a negative gradient step computed at $\overline{w}+xu$ implicitly takes a negative gradient step *on the sharpness* with step size $\frac{1}{2}\eta x^2$. This is the key ingredient missing from the traditional theory. When gradient descent exits the stable region, it oscillates along the top Hessian eigenvector, just as the traditional theory predicts; but what the traditional theory fails to anticipate is that these oscillations in turn perform gradient descent *on the sharpness*, thereby steering the trajectory back into the stable region automatically.

Note that traditional optimization theory fails to capture the basic *causal structure* of the optimization process: gradient descent converges not because the sharpness is "already" small, but rather due to an automatic negative feedback mechanism that *keeps* the sharpness small.

Damian et al. (2023) analyzed the EOS dynamics in the special case where only one Hessian eigenvalue has crossed the critical threshold $2/\eta$, as in steps ~2900-3600 in Figure 6. In this setting, the dynamics consist of consecutive cycles in which: (1) the sharpness rises above $2/\eta$; (2) this triggers growing oscillations along the top Hessian eigenvector; (3) such oscillations reduce sharpness via eq. (5), pushing it below $2/\eta$; (4) the oscillations consequently shrink in magnitude.[11] However, a more common situation is when *multiple* Hessian eigenvalues have reached $2/\eta$, as in steps ~3600-4000 in Figure 6. Here, gradient descent oscillates simultaneously along all the corresponding eigenvectors,[12] and these oscillations cause all such eigenvalues to remain dynamically regulated around $2/\eta$.

Unfortunately, analyzing EOS dynamics in fine-grained detail is challenging (Damian et al., 2023). The difficulty arises from the need to account for the mutual interactions between the oscillations and the curvature. Even in the

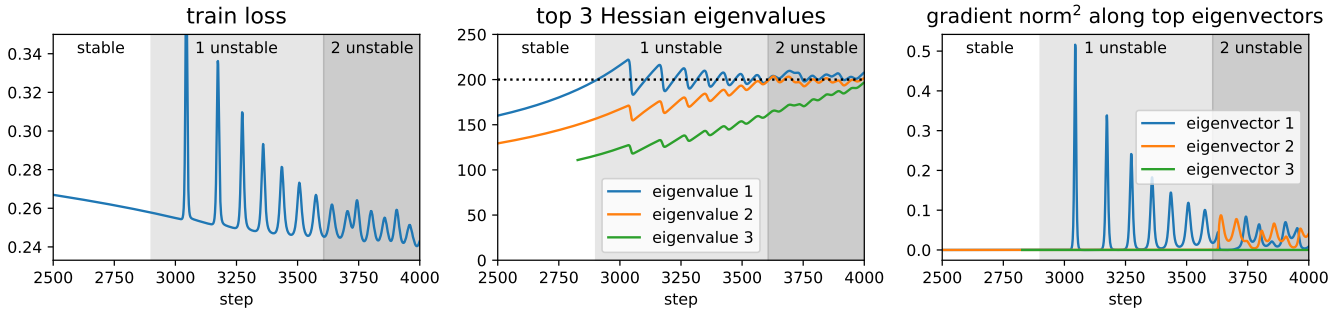


Figure 6: **Multiple Hessian eigenvalues can be at the edge of stability.** From steps ~2900-3600, one Hessian eigenvalue is at the edge of stability, and gradient descent oscillates along the top Hessian eigenvector. From steps ~3600-4000, two Hessian eigenvalues are at the edge of stability, and gradient descent oscillates simultaneously along both the corresponding eigenvectors. The number of oscillating directions can be easily read off from the right plot, which shows the squared norm of the gradient when projected onto each of the top 3 Hessian eigenvectors.

[10] Technically, equating $\nabla_{\overline{w}}[u^T H(\overline{w})u] = \nabla_{\overline{w}} S(\overline{w})$ requires invoking Danskin's theorem. This is made precise in Section A.8, Fact 1.

[11] Notice that the drop in the sharpness is rapid, yielding a sawtooth-like plot for the evolution of the sharpness. This is because the strength of the sharpness reduction effect is proportional to x^2. When x is small, the sharpness-reduction effect is negligible, but when x grows larger, the effect quickly becomes strong. See Damian et al. (2023) for a simplified ODE model of the joint dynamics between x and sharpness.

[12] With $k > 1$ unstable eigenvalues, the corresponding eigenvectors are not individually identifiable; instead, one should think of gradient descent as oscillating within the k-dimensional eigenspace spanned by the k eigenvectors at the edge of stability.

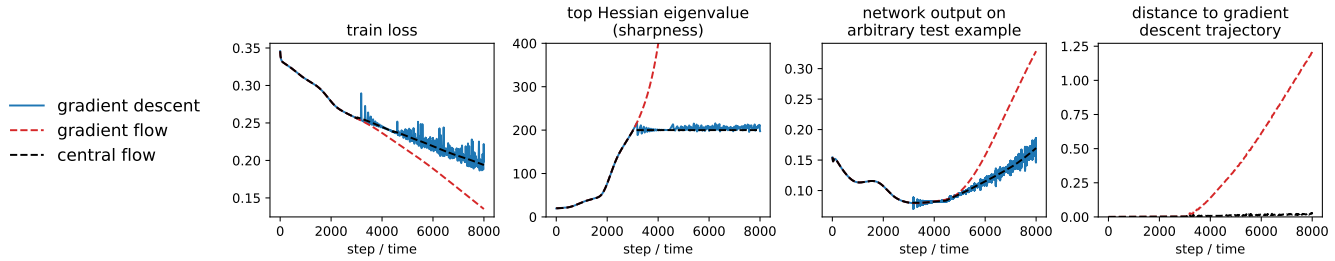
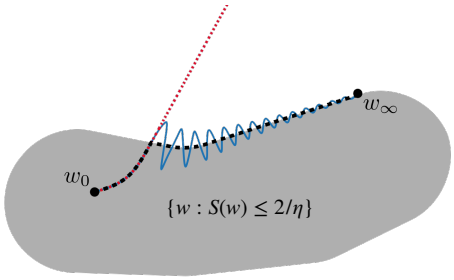


Figure 7: **What macroscopic path does gradient descent take?** Gradient descent (blue) is well-approximated by gradient flow (red) so long as the sharpness is below $2/\eta$. However, once gradient descent reaches the edge of stability, it takes a different path. Our *central flow* (black) approximates gradient descent even at the edge of stability. The plots on top present data from an experiment (same as Figure 5); the drawing on the right is a cartoon of the underlying weight-space dynamics.



special case of one unstable eigenvalue, these dynamics are nonlinear and highly sensitive to initial conditions. The more typical case of multiple unstable eigenvalues is even harder to analyze: the dynamics with k unstable eigenvalues do not decouple into k independent systems, and instead involve $O(k^2)$ mutually interacting quantities, yielding complex and often chaotic behavior.

Our key insight in this paper is that a fine-grained analysis of the EOS dynamics may not be necessary. Rather, we argue that the more important question is: what *macroscopic* (i.e. long-term) trajectory does gradient descent take through weight space? In the next section, we will use a heuristic time-averaging argument to characterize this macroscopic trajectory. Our analysis will not only recover the main finding of Damian et al. (2023) for a single unstable eigenvalue, but will also readily generalize to the challenging setting of multiple unstable eigenvalues.

3.2 Deriving the Gradient Descent Central Flow

The standard continuous-time approximation to gradient descent is the gradient flow:[13]

$$\frac{dw}{dt} = -\eta \nabla L(w). \tag{6}$$

Cohen et al. (2021) observed that trajectory of gradient descent is well-approximated[14][15] by that of gradient flow so long as training is *stable*, i.e. so long as the sharpness $S(w)$ remains below $2/\eta$. However, once the sharpness reaches $2/\eta$ and the dynamics enter the EOS regime, gradient descent departs from the gradient flow trajectory and takes a different path, as illustrated in Figure 7.[16]

[13] We fold η into the definition of gradient flow so that there is a correspondence between step t of gradient descent and time t of gradient flow. This will especially be useful when analyzing adaptive optimizers where the effective step size is a dynamic quantity.

[14] Barrett and Dherin (2021) argued that the accuracy of the gradient flow approximation can be improved by adding a penalty on the squared gradient norm. However, their modified flow does not hold in the EOS regime, and in the stable regime, we found that the accuracy improvement it brings is relatively small (Section C.1). Therefore, for simplicity, we leave out any such term from our flows.

[15] It remains theoretically unexplained why gradient flow is such a good fit to gradient descent. Existing bounds for the distance between gradient descent and gradient flow increase exponentially with time, with an exponent determined by the most negative Hessian eigenvalue (Elkabetz and Cohen, 2021). Empirically, such bounds are overly conservative.

[16] Even in the simplest setting of one unstable eigenvalue, capturing the EOS dynamics necessarily requires three variables: one for the oscillations along the top Hessian eigenvector, one for the top Hessian eigenvalue (sharpness), and one for the remaining directions. Since visualizing three-dimensional dynamics is difficult, we will frequently resort to two-dimensional cartoons (e.g. Figure 7). Such a "projection" will necessarily drop information. Accordingly, Figure 7 captures sharpness and remaining directions, but leaves out the back-and-forth oscillations along the top Hessian eigenvector. Figure 2, by contrast, captures these back-and-forth oscillations but leaves out the sharpness.

We now derive a more general differential equation, which we call a central flow, that approximates the trajectory of gradient descent even at the edge of stability. The central flow directly models the *time-averaged* (i.e. smoothed) trajectory of the oscillatory optimizer. In other words, the central flow averages out the oscillations while retaining their lasting effect on the macroscopic trajectory. We will derive the central flow using a heuristic time-averaging argument, and we will empirically demonstrate that it can accurately predict long-term optimization trajectories on a variety of neural networks with a high degree of numerical accuracy, as illustrated in Figure 7.

We will abuse notation and use $\mathbb{E}$ to denote "local time-averages" of deterministic quantities — see Section A.1.5 for discussion. The gradient descent central flow is intended to model the time-averaged trajectory $\mathbb{E}[w_t]$. To simplify notation, we will also use $\overline{w}_t := \mathbb{E}[w_t]$ to denote the time-averaged trajectory.

3.2.1 Warm-up: the Special Case of One Unstable Eigenvalue

We will introduce our time-averaging methodology by analyzing the special case when only the largest Hessian eigenvalue has crossed the critical threshold $2/\eta$, and gradient descent oscillates along a single direction — the corresponding eigenvector. Our fully general analysis, given later in Section 3.2.2, will allow for an arbitrary number of eigenvalues to be at the edge of stability, and for eigenvalues to enter and leave the edge of stability.

Thus, in this section, we start our analysis at the instant when the sharpness $S(w)$ first reaches $2/\eta$. From this point onward, we will model the gradient descent trajectory by:

$$w_t = \overline{w}_t + x_t u_t, \tag{7}$$

where w_t is the gradient descent iterate, $\overline{w}_t$ is the time-averaged iterate, u_t is the top Hessian eigenvector at $\overline{w}_t$, and x_t denotes the displacement between w_t and $\overline{w}_t$ along the u_t direction.[17] Note that by definition, $\mathbb{E}[x_t] = 0$, i.e. the time-averaged displacement is zero. To track the evolution of the time-averaged iterate $\overline{w}_t$, we time-average both sides of the gradient descent update eq. (1):

$$\overline{w}_{t+1} = \overline{w}_t - \eta \underbrace{\mathbb{E}[\nabla L(w_t)]}_{\text{time-averaged gradient}}. \tag{8}$$

That is, the time-averaged iterates follow the (negative) time-averaged gradient. To approximate the time-averaged gradient, we first Taylor-expand the gradient ∇L around the time-averaged iterate $\overline{w}_t$:

$$\nabla L(w_t) = \underbrace{\nabla L(\overline{w}_t)}_{\text{gradient at } \overline{w}_t} + \underbrace{x_t\, S(\overline{w}_t) u_t}_{\text{oscillation}} + \underbrace{\tfrac{1}{2}\, x_t^2\, \nabla S(\overline{w}_t)}_{\text{sharpness reduction}} + \mathcal{O}(x^3). \tag{9}$$

We then take the time average of both sides, averaging over the x oscillations. This reflects an implicit assumption that the x oscillations are happening fast relative to the remaining training dynamics:[18]

$$\mathbb{E}[\nabla L(w_t)] \;\approx\; \nabla L(\overline{w}_t) + \underbrace{\cancel{\mathbb{E}[x_t] S(\overline{w}_t) u_t}}_{0 \text{ because } \mathbb{E}[x_t]=0} + \underbrace{\tfrac{1}{2}\, \mathbb{E}[x_t^2] \nabla S(\overline{w}_t)}_{\text{implicit sharpness penalty}}. \tag{10}$$

This calculation shows that the time-averaged gradient $\mathbb{E}[\nabla L(w_t)]$ is equal to the gradient at the time-averaged iterate $\nabla L(\overline{w}_t)$, plus an implicit sharpness penalty whose strength is proportional to $\mathbb{E}[x_t^2]$, the variance of the oscillations at step t. Substituting eq. (10) into eq. (8) and switching to continuous time, we therefore model the time-averaged iterates $\overline{w}_t$ by the sharpness-penalized gradient flow $w(t)$ defined by:

$$\frac{dw}{dt} = -\eta \Big[\nabla L(w) + \underbrace{\tfrac{1}{2} \sigma^2(t) \nabla S(w)}_{\text{implicit sharpness penalty}} \Big]. \tag{11}$$

[17] This is a simplification. In reality, we know that gradient descent is displaced from $\overline{w}$ in at least *two* directions: the u direction and the $\nabla S(\overline{w})$ direction, with the latter responsible for the fluctuations in the sharpness. However, when modeling the time-averaged gradient, we will only account for the displacement in the u direction. This is analogous to Damian et al. (2023, Assumption 5). The success of our experiments validates this simplification.

[18] When time-averaging eq. (9), we assume that the eigenvector u changes slowly relative to the displacement x so that $\mathbb{E}[x_t u_t] \approx \mathbb{E}[x_t] u_t$.

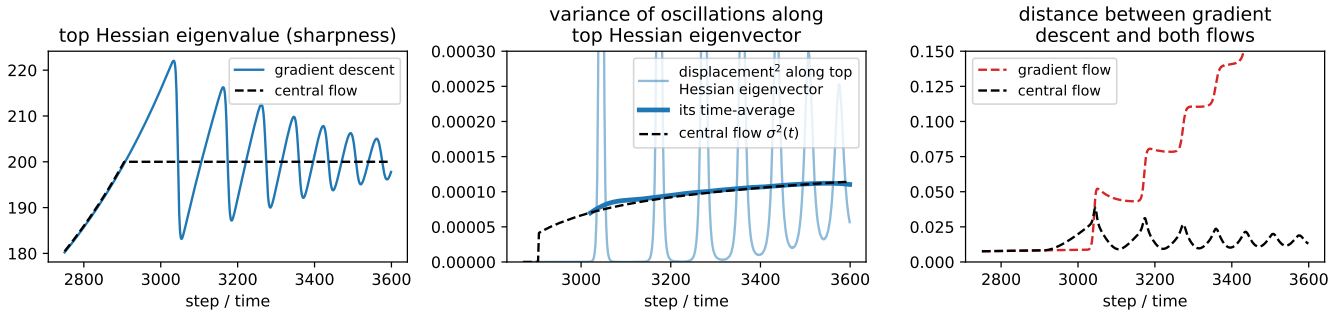


Figure 8: **Illustrating the central flow with one unstable eigenvalue**. **Left**: the sharpness (top Hessian eigenvalue) cycles around $2/\eta$ under gradient descent, and stays capped at $2/\eta$ under the central flow. **Center**: the central flow's $\sigma^2(t)$ accurately predicts the variance of the oscillations. In light blue, we plot $(u(t)^\top(w_t - w(t)))^2$, the squared displacement between gradient descent w_t and the central flow $w(t)$ along $u(t)$, the top Hessian eigenvector at $w(t)$. In dark blue, using Gaussian smoothing, we plot its time average, i.e. the empirical variance of the oscillations. Observe that this is well-predicted by the central flow's $\sigma^2(t)$, eq. (14), plotted in black. **Right**: the Euclidean distance $\|w_t - w(t)\|$ between gradient descent and the central flow (black) stays small over time, indicating that the central flow accurately predicts the long-term trajectory of gradient descent. In contrast, the distance (red) between gradient descent and the gradient flow eq. (6) grows large over time. This figure depicts the same ViT trajectory as Figure 5.

Here, $\sigma^2(t)$ is a still-unknown quantity intended to model $\mathbb{E}[x_t^2]$, the instantaneous variance of the oscillations at time t. This quantity also controls the strength of the implicit sharpness penalty. To determine $\sigma^2(t)$, we argue that only one value is consistent with the observed behavior of gradient descent. Empirically, once the sharpness reaches the critical threshold $2/\eta$, it does not continue to rise indefinitely; rather, it remains dynamically regulated around $2/\eta$. Thus, we will enforce that the central flow never increases the sharpness $S(w(t))$ past $2/\eta$. The time derivative of the sharpness under a flow of the form eq. (11) can be easily computed using the chain rule:

$$\frac{dS(w)}{dt} = \left\langle \nabla S(w), \frac{dw}{dt} \right\rangle = \underbrace{\eta \left\langle \nabla S(w), -\nabla L(w) \right\rangle}_{\substack{\text{change in sharpness} \\ \text{under gradient flow}}} - \underbrace{\tfrac{1}{2}\eta\sigma^2(t)\|\nabla S(w)\|^2}_{\substack{\text{sharpness reduction} \\ \text{from oscillations}}}. \tag{12}$$

When the first term, the change in sharpness under the gradient flow, is *negative*, gradient descent will leave the edge of stability and will once again follow gradient flow — this is made precise in Section 3.2.2. Therefore, we focus on the case where this first term is positive, i.e. where progressive sharpening holds. As the sharpness is currently at $2/\eta$ and must remain at $2/\eta$, we must have that $\frac{dS(w)}{dt} = 0$. Since $\frac{dS(w)}{dt}$ is affine in $\sigma^2(t)$, we can easily solve for the unique value of $\sigma^2(t)$ that ensures $\frac{dS(w)}{dt} = 0$:

$$\sigma^2(t) = \frac{2\left\langle \nabla S(w), -\nabla L(w) \right\rangle}{\|\nabla S(w)\|^2}. \tag{13}$$

Intuitively, this is the unique $\sigma^2(t)$ for which the downward force of oscillation-induced sharpness reduction "cancels out" the upwards force of progressive sharpening so the sharpness remains locked at $2/\eta$. The central flow for a single unstable eigenvalue is given by substituting this $\sigma^2(t)$ into eq. (11):

$$\frac{dw}{dt} = -\eta\Big[\nabla L(w) + \tfrac{1}{2}\sigma^2(t)\nabla S(w)\Big] \quad \text{where} \quad \sigma^2(t) = \frac{2\left\langle \nabla S(w), -\nabla L(w) \right\rangle}{\|\nabla S(w)\|^2}. \tag{14}$$

Figure 8 demonstrates this flow in action. We run gradient flow until the sharpness hits $2/\eta$, and then switch to eq. (14) at that time. (The complete central flow, defined in the next section, will handle such switches automatically.) Observe that the distance in weight space between gradient descent and the central flow remains small over time,[19]

[19] Observant readers might notice that the distance between gradient descent and the central flow starts to grow once the sharpness hits $2/\eta$ and is actually initially larger than the distance between gradient descent and the gradient flow. This is because the central flow has already started to apply sharpness regularization, but the discrete gradient descent trajectory has not yet done so.

verifying that the central flow accurately predicts the long-term trajectory of gradient descent. Moreover, observe that $\sigma^2(t)$ from eq. (13) accurately predicts the empirical variance of the oscillations along the top Hessian eigenvector, further demonstrating that our time-averaging argument is accurately capturing gradient descent's behavior.

Intuitively, whereas gradient descent reduces sharpness in impulse-like spurts which are triggered whenever the oscillations grow large, the central flow applies a sharpness-reduction force continuously, with the same average strength over time. That these two processes stay close over long timescales implies that the oscillations are only affecting the long-term gradient descent trajectory via their *variance* rather than via their fine-grained details (e.g the precise shape of the light blue line in Figure 8, center). This is good news: while the fine-grained oscillations may be challenging to analyze, we have shown that their variance is easy to analyze, as there is only one possible value that is consistent with the observed edge of stability equilibrium. In this way, we have successfully used a heuristic argument to solve for the time-averaged trajectory of gradient descent.

Interpretation as Projection While we have derived the central flow as a sharpness-penalized gradient flow, it can be equivalently interpreted as a *projected* gradient flow. In particular, simplifying eq. (14) gives:

$$\frac{dw}{dt} = -\eta\left[I - \frac{\nabla S(w)\nabla S(w)^\top}{\|\nabla S(w)\|^2}\right]\nabla L(w) = -\eta\Pi^\perp_{\nabla S(w)}\nabla L(w), \quad (15)$$

where $\Pi^\perp_v := I - \frac{vv^T}{\|v\|^2}$ denotes the projection matrix onto the orthogonal complement of v. This flow projects out the ∇S direction from the gradient ∇L to keep the sharpness fixed at $2/\eta$, as shown in the cartoon on the right.

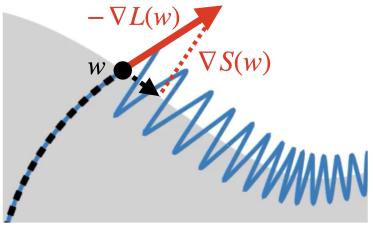


Figure 9: The central flow projects out the $\nabla S(w)$ direction from the loss gradient $\nabla L(w)$.

Previously, Damian et al. (2023) proved that under certain conditions, gradient descent at the edge of stability implicitly follows the trajectory of projected gradient descent constrained to the stable region. We have thus nearly[20] rederived their result in a simpler, albeit non-rigorous, manner.

The projection interpretation will be useful below for reasoning about gradient descent's behavior.

3.2.2 The Fully General Case

We will now derive the complete central flow, which applies in the fully general setting where any number of eigenvalues can be at the edge of stability, including zero. When no eigenvalues are at the edge of stability, the central flow will automatically reduce to the gradient flow. As above, we decompose the gradient descent trajectory as:

$$w_t = \overline{w}_t + \delta_t, \quad (16)$$

where w_t is the gradient descent iterate, $\overline{w}_t := \mathbb{E}[w_t]$ is the time-averaged iterate, and δ_t denotes the displacement between w_t and $\overline{w}_t$, i.e. the oscillation. Because gradient descent oscillates along the Hessian eigenvectors that are at the edge of stability, we model δ_t as lying within the span of these eigenvectors.[21] For example, in the case where only one direction is at the edge of stability, taking $\delta_t = x_t u_t$ recovers the analysis in Section 3.2.1. Note that by definition of $\overline{w}_t$, we have that $\mathbb{E}[\delta_t] = 0$. As before, the time-averaged iterates follow the time-averaged gradient $\mathbb{E}[\nabla L(w_t)]$. To compute the time-averaged gradient, we first Taylor-expand the gradient around $\overline{w}_t$:

$$\nabla L(w_t) = \underbrace{\nabla L(\overline{w}_t)}_{\text{gradient at } \overline{w}} + \underbrace{H(\overline{w}_t)\delta_t}_{\text{oscillation}} + \frac{1}{2}\underbrace{\nabla_{\overline{w}_t}\, \delta_t^T H(\overline{w}_t)\delta_t}_{\text{implicit curvature penalty}} + \mathcal{O}(\|\delta_t\|^3). \quad (17)$$

[20] The flow eq. (15) actually differs slightly from the constrained trajectory in Damian et al. (2023), beyond being continuous rather than discrete. In Damian et al. (2023), the stable region was defined as the set where $S(w) \leq 2/\eta$ *and* the the loss is directionally minimized along the top Hessian eigenvector. The latter condition prevents their theory from applying to certain models (e.g. Kreisler et al., 2023, Appendix A). Our central flow does not use the latter condition and hence does not suffer from such restrictions.

[21] Similar to footnote 17, this neglects the motion in the top Hessian eigenvalues. The success of our experiments justifies this simplification.

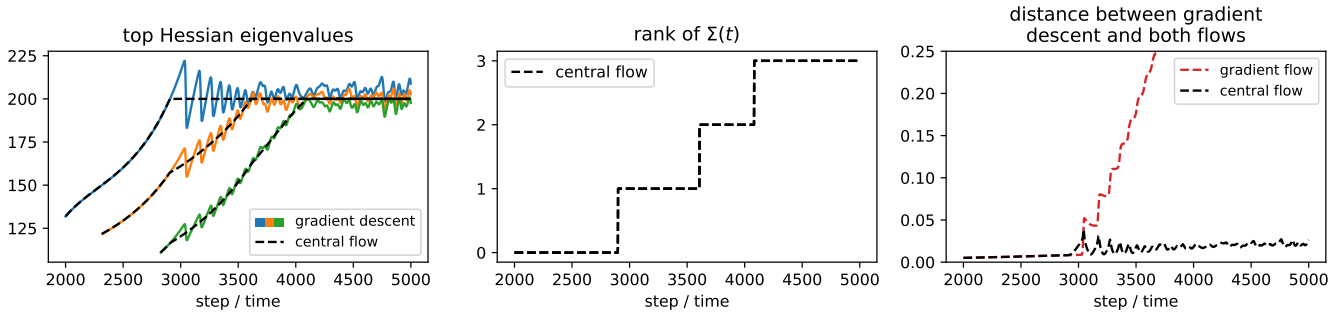


Figure 10: **Illustrating the central flow (general case)**. **Left**: whenever a Hessian eigenvalue rises to $2/\eta$, the central flow prevents it from increasing further. **Center**: since $\Sigma(t)$ models the covariance of the oscillations, its rank is always equal to the number of Hessian eigenvalues at the edge of stability. We show in Figure 11 that $\Sigma(t)$ accurately predicts the covariance of oscillations. **Right**: the Euclidean distance between gradient descent w_t and the central flow $w(t)$ stays small over time, indicating that the central flow accurately predicts the long-term trajectory of gradient descent. In contrast, the distance between gradient descent and the gradient flow eq. (6) grows large over time. This figure depicts the same ViT trajectory as Figure 5.

The third term in this Taylor expansion reveals that the negative gradient at the iterate w_t implicitly acts to decrease the directional curvature in the direction δ_t. Time-averaging both sides and rearranging the third term yields:

$$\mathbb{E}[\nabla L(w_t)] \approx \nabla L(\overline{w}_t) + \underbrace{\cancel{H(\overline{w}_t)\mathbb{E}[\delta_t]}}_{\text{0 because } \mathbb{E}[\delta_t]=0} + \frac{1}{2}\underbrace{\nabla_{\overline{w}_t}\left\langle H(\overline{w}_t), \mathbb{E}[\delta_t \delta_t^T]\right\rangle}_{\text{implicit curvature penalty}}, \tag{18}$$

where we use $\langle \cdot, \cdot \rangle$ to denote the Frobenius inner product between two matrices, equivalent to flattening the matrices into vectors and taking the dot product. Thus, we see that the time-averaged gradient is the gradient at the time-averaged iterate, plus an implicit curvature penalty whose strength and direction are determined by the *covariance* of the oscillations $\mathbb{E}[\delta_t \delta_t^T]$. Substituting eq. (18) into the time-averaged gradient descent update (eq. 8) and switching to continuous time, we model the time-averaged iterates $\overline{w}_t$ by a differential equation of the form:

$$\frac{dw}{dt} = -\eta\Big[\nabla L(w) + \frac{1}{2}\underbrace{\nabla_w \left\langle H(w), \Sigma(t)\right\rangle}_{\text{implicit curvature penalty}}\Big]. \tag{19}$$

Here, $\Sigma(t)$ is a still-unknown quantity intended to model $\mathbb{E}[\delta_t \delta_t^T]$, the instantaneous covariance of the oscillations at time t. This matrix also controls an implicit curvature penalty which penalizes the Σ-weighted Hessian $\langle \Sigma(t), H(w)\rangle$.[22] Similar as before, to determine $\Sigma(t)$, we impose three conditions:

1. Since Hessian eigenvalues which reach the critical threshold $2/\eta$ do not continue to rise further, we impose the condition that $\Sigma(t)$ should not allow any Hessian eigenvalues to rise beyond $2/\eta$.
2. Since gradient descent oscillates within the span of the unstable eigenvectors, we impose the condition that $\Sigma(t)$, which models the covariance of these oscillations, should be supported[23] within the span of the Hessian eigenvectors whose eigenvalue is equal to $2/\eta$.[24]
3. Since $\Sigma(t)$ models a covariance matrix, we impose the condition that $\Sigma(t)$ should be positive semidefinite.

These three conditions turn out to imply a unique value of $\Sigma(t)$. In particular, we detail in Section A.2 that $\Sigma(t)$ must be the unique solution to a type of convex program known as a *semidefinite complementarity problem* (SDCP), which

[22] This is a weighted sum of all entries in the Hessian matrix, where each entry is weighted by the corresponding entry of $\Sigma(t)$.

[23] We mean that $\text{span}[\Sigma] \subseteq \mathcal{U}$, where $\mathcal{U}$ is the span of the Hessian eigenvectors with eigenvalue $2/\eta$. Equivalently, we mean that Σ can be written as $\Sigma = UXU^T$ where the k columns of U form a basis for the k-dimensional subspace $\mathcal{U}$, and X is a $k \times k$ symmetric matrix.

[24] For gradient descent, the unstable eigenvectors have eigenvalues which fluctuate around $2/\eta$. However, for the central flow, the unstable eigenvectors will have eigenvalues which are exactly equal to $2/\eta$.

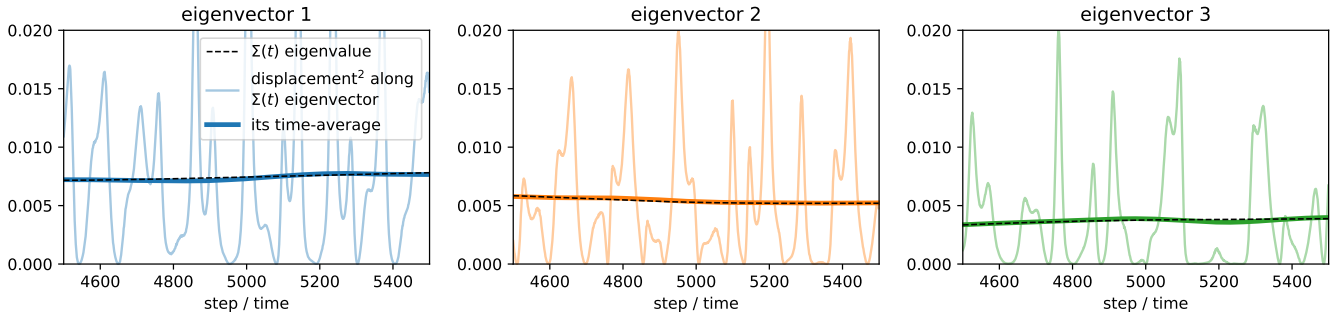


Figure 11: **Central flow's $\Sigma(t)$ accurately predicts covariance of oscillations.** We show that the central flow's $\Sigma(t)$ accurately predicts the covariance of gradient descent's oscillations. For this stretch of training, there are 3 Hessian eigenvalues at the edge of stability, so $\Sigma(t)$ has 3 nonzero eigenvalues (subpanels). In black, we plot each eigenvalue of $\Sigma(t)$; in colors, we plot the squared magnitude of gradient descent's displacement from the central flow along the corresponding eigenvectors (light = raw values, dark = time average using Gaussian smoothing). Observe that each eigenvalue of $\Sigma(t)$ accurately predicts the instantaneous variance of oscillations along the corresponding eigenvector, even though these oscillations might initially appear erratic. This figure depicts the same ViT trajectory as Figure 5.

are described in Section A.1.4.[25] The central flow is defined as eq. (19) with this $\Sigma(t)$:

$$\frac{dw}{dt} = -\eta\Big[\, \nabla L(w) + \tfrac{1}{2}\nabla_w \left\langle H(w), \Sigma(t)\right\rangle \,\Big] \quad \text{where} \quad \Sigma(t) \text{ solves the SDCP in eq. (70).} \tag{20}$$

A formal definition for the central flow is given in Section A.2, Definition 4. We note that $\Sigma(t)$ can be efficiently represented numerically as it is a low rank matrix, with rank at most the number of unstable eigenvalues.

We now elaborate on the behavior of the central flow:

1. **Stable regime:** If all Hessian eigenvalues are below $2/\eta$, then the SDCP returns $\Sigma(t) = 0$, and the central flow reduces to the gradient flow.
2. **One unstable eigenvalue**: If one Hessian eigenvalue is at $2/\eta$, and if the gradient flow would *increase* this eigenvalue above $2/\eta$, then our analysis reduces to that of Section 3.2.1. In particular, the SDCP returns a rank-one matrix of the form $\Sigma(t) = \sigma^2\, u\, u^\top$ where u is the top Hessian eigenvector at w, and σ^2 is defined in eq. (14). On the other hand, if the gradient flow would *decrease* this eigenvalue below $2/\eta$, then $\Sigma(t) = 0$, and the central flow will follow the gradient flow out of the edge of stability.
3. **Multiple unstable eigenvalues:** In general, the SDCP returns a $\Sigma(t)$ which constrains all Hessian eigenvalues currently at $2/\eta$ from rising above that value. Often, this $\Sigma(t)$ causes all Hessian eigenvalues currently at $2/\eta$ to remain fixed at $2/\eta$.[26]. However, it also allows for eigenvalues to leave EOS when appropriate.

Figure 10 demonstrates the central flow in action. Initially, all Hessian eigenvalues are below $2/\eta$, so $\Sigma(t) = 0$ and the central flow reduces to the gradient flow. Once the top Hessian eigenvalue reaches $2/\eta$ around step 2900, $\Sigma(t)$ becomes a rank-one matrix, and the central flow keeps the top Hessian eigenvalue locked at $2/\eta$, as it mimics the effects of oscillating along the the top eigenvector direction. Once the second Hessian eigenvalue also reaches $2/\eta$ around step 3600, $\Sigma(t)$ becomes a rank-two matrix, and the central flow keeps the top two eigenvalues both locked at $2/\eta$, as it mimics the effects of oscillating simultaneously along the top two eigenvector directions. Throughout, the Euclidean distance between gradient descent's w_t and the central flow's $w(t)$ stays small over time (right plot),

[25] Interestingly, complementarity problems arise frequently in the the study of contact mechanics. EOS can be interpreted as the gradient descent trajectory making "contact" with the boundary of the stable region, and then sliding along the boundary.

[26] There is a unique Σ that causes all Hessian eigenvalues currently at $2/\eta$ to remain fixed at $2/\eta$, and it can be found by solving a linear inverse, generalizing eq. (13). The solution to the SDCP coincides with this Σ at almost all times. However, this Σ cannot be used to define the central flow, as it would never allow an eigenvalue to leave the edge of stability, and it is not necessarily PSD.

indicating that the central flow accurately tracks the long-term trajectory of gradient descent. In contrast, the distance between gradient descent and the *gradient flow* eq. (6) grows large over time.

In Figure 11, we show that the central flow's $\Sigma(t)$ accurately predicts the covariance with which gradient descent is oscillating around the central flow. In particular, we show that each eigenvalue of $\Sigma(t)$ accurately predicts the instantaneous variance of oscillations along the corresponding eigenvector of $\Sigma(t)$. We find it striking that our theory is able to accurately predict the covariance of these oscillations. While the oscillations are erratic and might appear unpredictable, our findings reveal that a certain statistic — their covariance — is predictable after all. Moreover, predicting this covariance seems to be sufficient to predict the long-term trajectory of gradient descent.

Interpretation as projection The projection interpretation in Section 3.2.1 generalizes to the case of an arbitrary number of unstable eigenvalues. In particular, the central flow eq. (20) can be written as a flow which orthogonally projects the negative gradient onto the so-called *tangent cone* of the stable region $\mathbb{S} = \{w : S(w) \leq 2/\eta\}$, which is the set of directions in which one can move while still staying, to first order, within the stable region:[27]

$$\frac{dw}{dt} = \eta \underbrace{\text{proj}_{T_w\mathbb{S}}[-\nabla L(w)]}_{\substack{\text{project negative gradient onto} \\ \text{tangent cone } T_w\mathbb{S} \text{ of set } \mathbb{S}}} \quad \text{where} \quad \mathbb{S} = \underbrace{\{w : S(w) \leq 2/\eta\}}_{\text{stable region } \mathbb{S}}. \tag{21}$$

A formal definition is given in Definition 5. This projection interpretation will be used in Section 3.3 to show that the loss along the central flow decreases monotonically.

Where does deep learning come in? Our principal claim is that, if initialized stably, gradient descent will approximately follow the central flow over the long term. In the case where the sharpness does not rise to $2/\eta$ (e.g. on a quadratic objective, where the sharpness is constant), then the central flow reduces to the gradient flow, and so our claim reduces to the somewhat "uninteresting" claim that gradient descent will approximately follow the gradient flow. The central flow only becomes nontrivial, and our claim only becomes "interesting," in the event that the sharpness rises to $2/\eta$. This empirically tends to happen on deep learning objectives. However, we suspect that the central flow might also hold on other kinds of objectives where the sharpness rises to $2/\eta$ during gradient descent.

3.2.3 Understanding the train loss curve and more

Figure 12 shows that the loss along the actual gradient descent trajectory is consistently *higher* than the loss along the central flow. The intuitive explanation is that when gradient descent oscillates along the top Hessian eigenvector(s), it can be visualized as "bouncing between valley walls" (Xing et al., 2018; Cohen et al., 2021; Wen et al., 2025),

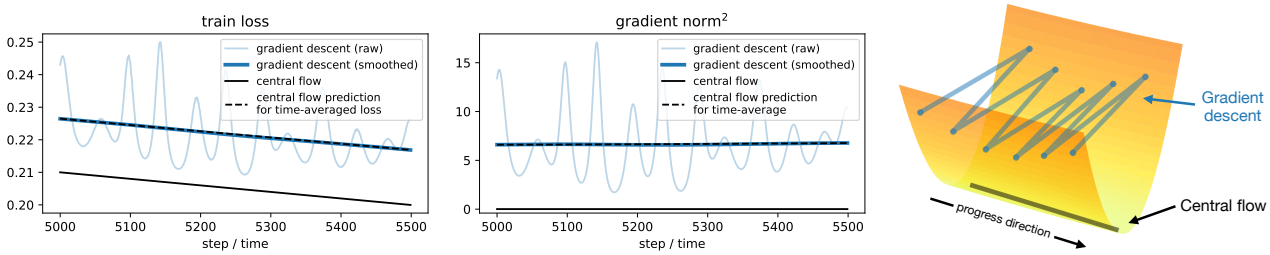


Figure 12: The train loss and gradient norm^2 are larger along the raw gradient descent trajectory (light blue) than along the central flow (solid black). The intuitive reason is that an oscillatory optimizer can be visualized as oscillating between the walls of a "valley" (see cartoon on right). Because the central flow models the covariance $\Sigma(t)$ of the oscillations, it can render predictions for the *time-averaged* values of the loss and gradient norm^2 along the gradient descent trajectory, via eq. (23) and eq. (74) respectively (dashed black). These predictions are close to the empirical time-averaged values, computed via Gaussian smoothing (dark blue). This figure depicts the same ViT as Figure 5.

[27] In the interior of the stable region (i.e. $S(w) < 2/\eta$), the tangent cone is the entire space, so this projection is a no-op and the central flow reduces to the gradient flow. When exactly one eigenvalue is at the edge of stability, the tangent cone is the half-space $\{v : \nabla S(w)^T v \leq 0\}$.

whereas the time-averaged iterates run nearly along the "valley floor" (called a "river" in Wen et al. (2025)), as illustrated on the right of Figure 12. The loss is higher on the valley walls, where the actual iterates are located, than on the valley floor, where the time-averaged iterates are located.[28]

Fortunately, the central flow framework will still let us reason about the loss along the actual gradient descent trajectory. Recall that the central flow predicts not just the time-averaged iterates $w(t)$, but also the covariance of the oscillations $\Sigma(t)$. In particular, the central flow models the gradient descent trajectory $\{w_t\}$ as:[29]

$$w_t = w(t) + \delta_t, \quad \text{where} \quad \mathbb{E}[\delta_t] = 0 \quad \text{and} \quad \mathbb{E}[\delta_t \delta_t^T] = \Sigma(t).$$

Thus, for any quantity $f(w)$ derived from the weights (e.g. loss or gradient norm), we can predict its time-averaged value $\mathbb{E}\, f(w_t)$ along the gradient descent trajectory w_t by taking a quadratic Taylor expansion along the central flow $w(t)$, and time-averaging over δ_t:

$$\underbrace{\mathbb{E}[f(w_t)]}_{\substack{\text{time-averaged} \\ \text{value along trajectory}}} \approx \underbrace{f(w(t))}_{\text{value along central flow}} + \underbrace{\tfrac{1}{2}\left\langle \nabla^2 f(w(t)), \Sigma(t) \right\rangle}_{\text{contribution from oscillations}}. \tag{22}$$

For example, if f is the loss L, then because $\Sigma(t)$ is supported on the Hessian eigenvectors with eigenvalue $2/\eta$:

$$\underbrace{\mathbb{E}[L(w_t)]}_{\substack{\text{time-averaged} \\ \text{loss along trajectory}}} \approx \underbrace{L(w(t))}_{\text{loss along central flow}} + \underbrace{\tfrac{1}{\eta}\operatorname{tr}\Sigma(t)}_{\text{contribution from oscillations}} := \bar{L}(t). \tag{23}$$

See Section A.2.2 for an explicit derivation. Figure 12 shows that this prediction $\bar{L}(t)$ for the time-averaged loss closely matches the actual time-averaged loss (computed with Gaussian smoothing). Both the central flow loss $L(w(t))$ and the predicted time-averaged loss $\bar{L}(t)$ model important quantities that are meaningful to DL practice:

- The central flow's prediction for the time-averaged loss $\bar{L}(t)$ models the smoothed training loss curve, often monitored in practice by Tensorboard (Abadi et al., 2015) or Weights and Biases (Biewald, 2020).
- The central flow loss $L(w(t))$ models the loss at the time-averaged iterate. This is similar to the loss at an exponential moving average of the weights, or the loss after annealing the learning rate (Sandler et al., 2023).

The central flow perspective allows us to quantify both of these loss values and reason about them separately.

A similar point holds for other quantities,[30] such as the gradient norm. Figure 12 shows that the squared gradient norm along the central flow, $\|\nabla L(w(t))\|^2$ is much smaller than at the actual iterates, $\|\nabla L(w_t)\|^2$. Intuitively, most of the gradient at the iterates is spent "oscillating across the valley" and cancels out over the long run. The central flow's gradient is smaller because it leaves out these oscillations. Yet, because the central flow models the covariance $\Sigma(t)$ of the oscillations, it can still predict the *time-average* of the squared gradient norm at the iterates, using eq. (74) from Section A.2.2. Figure 12 demonstrates the accuracy of this prediction.

3.2.4 Empirical verification

We empirically find that the central flow can accurately predict the long-term trajectory of gradient descent in a variety of deep learning settings. For example, Figure 13 shows the central flow on several different deep learning settings (details in Section 6). Observe that the central flow accurately predicts the weight-space trajectory, the covariance of the oscillations, and the time-averaged training loss curve. Figures 32(a) and 32(b) show that the central flow can accurately predict the time-averaged training loss curve at different learning rates across a variety of deep learning settings. Our full set of gradient descent experiments can be found in Section E.1.

[28] This does not mean that the loss landscape is "locally convex" in any deep sense (indeed, the Hessian generally has negative eigenvalues). It merely reflects that the optimizer is oscillating along directions of positive curvature.

[29] We emphasize that the central flow does not model the "distribution" (so to speak) of the oscillations δ_t. Rather, it only models the second moment $\mathbb{E}[\delta_t \delta_t^T]$, under the theory that the macroscopic trajectory of gradient descent is completely characterized by this second moment.

[30] Interestingly, for some quantities (such as the network outputs), we find that the value along the central flow is already an excellent approximation to the time-averaged value along the discrete optimizer trajectory, and eq. (22) is not necessary.

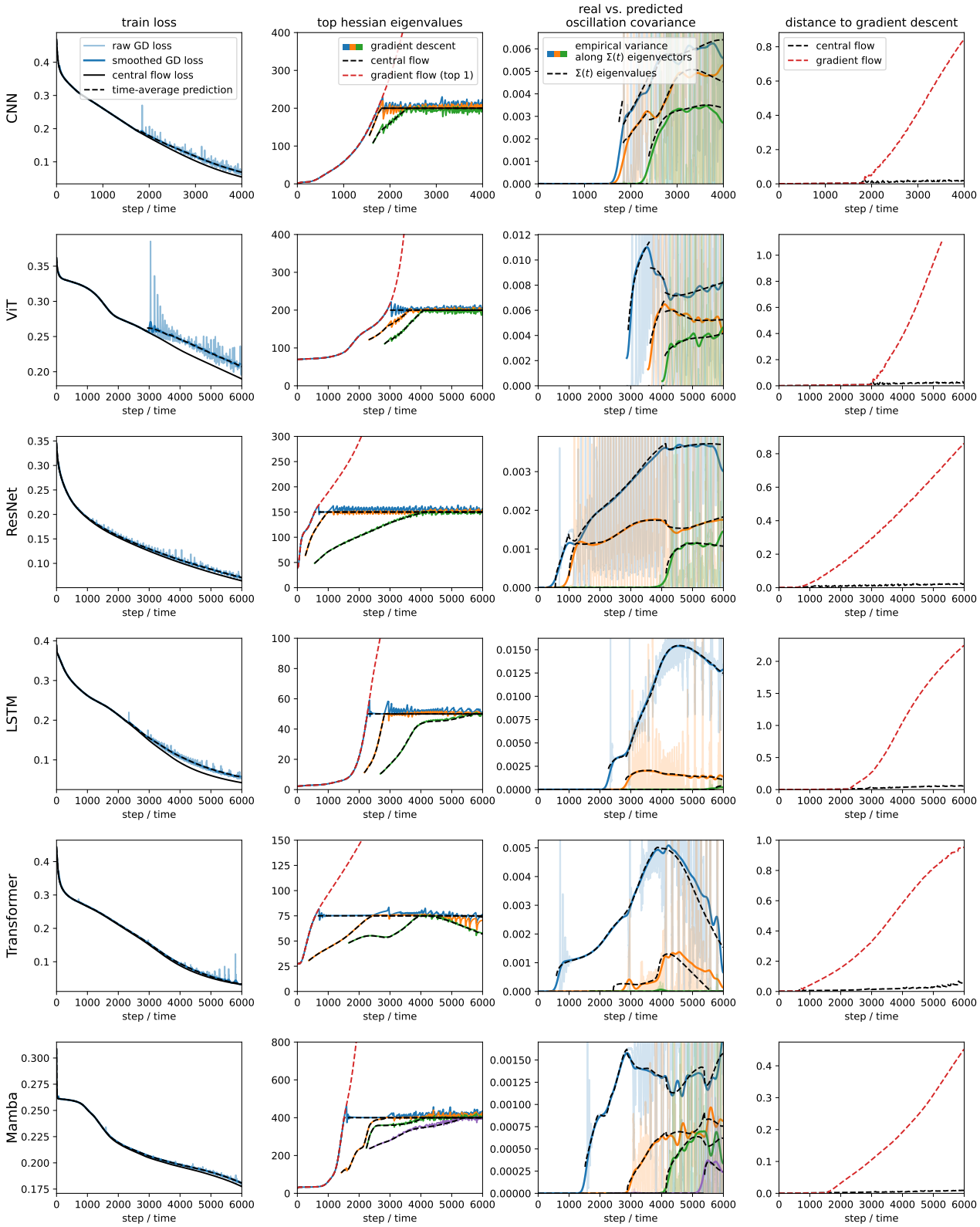


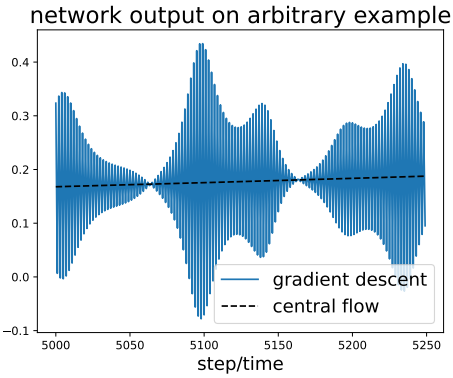
Figure 13: **Verifying the gradient descent central flow across various architectures.** Across various architectures, the central flow accurately predicts the weight-space trajectory, the covariance of the oscillations, and the time-averaged loss curve. These experiments all use MSE loss. See Section 6 for more experimental details and Section E for our full set of raw experiments.

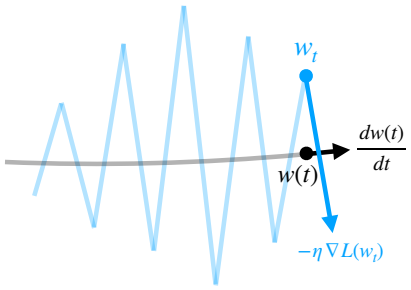
Nevertheless, our derivation relied on informal mathematical reasoning, and certain factors do empirically affect the quality of the central flow approximation. First, the central flow tends to become less accurate as the learning rate η is made increasingly large. Second, on some deep learning problems, higher-order terms cause the central flow to slightly mispredict $\Sigma(t)$, causing error to accumulate over the long run. Third, large spikes also can throw off the central flow. The latter two issues empirically seem to be more common when the loss criterion is cross-entropy rather than MSE. We discuss these points at greater length in Section 6. We hope that future work can rigorously understand the conditions under which the central flow does or does not approximate the gradient descent trajectory.

3.3 Understanding Gradient Descent via its Central Flow

We have shown that the central flow is a smooth curve that characterizes the macroscopic trajectory of gradient descent. We now explain why this makes it a useful theoretical tool for reasoning about optimization.

Averaging out oscillations reveals the underlying order At the edge of stability, gradient descent's oscillations lead to wild fluctuations in many training-related quantities, such as the training loss and the network's predictions (Rosenfeld and Risteski, 2024). For example, the figure on the right shows the evolution under gradient descent of the network's prediction on an example. One can see that along the actual gradient descent trajectory (blue), training proceeds erratically. In contrast, the central flow (black) is a more coherent training process which makes steady, continuous progress over time. By averaging out the oscillations, the central flow reveals the underlying order hidden beneath the chaotic oscillatory dynamics.[31]

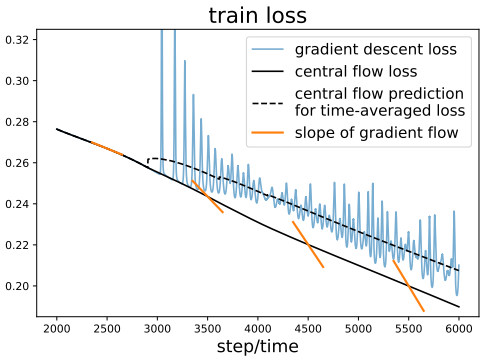




A smooth curve can be analyzed using calculus Because the central flow is a smooth curve, we can leverage calculus to reason quantitatively about the dynamics of training. Crucially, along the central flow, the time derivative $\frac{dw(t)}{dt}$ meaningfully reflects the optimizer's direction of motion over the near term (see cartoon on left). In comparison, along the gradient descent trajectory, the analogous update $-\eta \nabla L(w_t)$ is dominated by oscillations and hence does *not* meaningfully reflect the direction of motion over the near term — only over the current step.

For any quantity $f(w)$ derived from the weights w, we can use the chain rule to compute its rate of change under the central flow: $\frac{df}{dt} = \langle \nabla f(w), \frac{dw}{dt} \rangle$. We will now use this to reason about the rate of loss decrease.

Reasoning about training loss curve Consider the most basic question one can ask about an optimization algorithm: how fast is the loss going down? For the "raw" gradient descent trajectory, the loss *doesn't* always go down — instead, the loss behaves non-monotonically over short timescales, while only decreasing over long timescales. Thus, reasoning about the rate of loss decrease is challenging. In contrast, under the central flow, the loss evolves smoothly, and its rate of decrease can be quantified using the chain rule: $\frac{dL(w)}{dt} = \langle \nabla L(w), \frac{dw}{dt} \rangle$. Combining this with the projection interpretation (Definition 5), one can easily prove that the loss along the central flow $L(w(t))$ is monotonically decreasing. In other words, the central flow loss is a valid **potential function** for the optimization process:



Proposition 1. Under the central flow $w(t)$, we have $\frac{d}{dt} L(w(t)) \leq 0$.

[31] Arguably, the central flow can even be viewed as the "true" training process, with the actual gradient descent trajectory being merely a noisy realization of this idealized trajectory which is computationally cheap to obtain.

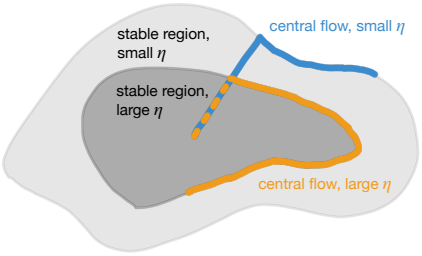
See Section A.2.3 for the proof. The intuition is that, even after the negative gradient is projected onto the tangent cone of the stable region, it will still be negatively aligned with the gradient.

While averaging out the oscillations yields a central flow with a smoothly decreasing loss curve, the oscillations still have an effect on this loss curve through their implicit curvature reduction effect, which can be shown to slow down training. In particular, whereas the unregularized gradient flow eq. (6) decreases the loss at the speed $\frac{dL}{dt} = -\eta\|\nabla L(w)\|^2$, it is straightforward to show that the central flow optimizes at a slower speed:

Proposition 2. Under the central flow $w(t)$, we have $\frac{d}{dt}L(w(t)) \geq -\eta\|\nabla L(w(t))\|^2$.

See Section A.2.3 for the proof. The intuition is that because the central flow projects out the components of the loss gradient that would cause the sharpness to rise above $2/\eta$, it has less gradient available with which to decrease the loss. This effect is illustrated in the figure on the previous page, which shows that at various points during training, the slope of the central flow loss curve is less steep than the rate of loss decrease under the gradient flow.

Understanding the effect of hyperparameters A notorious peculiarity of deep learning is that optimizer hyperparameters affect not just the speed of training, but also the particular path that the optimizer takes through weight space (e.g. Keskar et al., 2017; Jastrzębski et al., 2019). As a result, these hyperparameters can affect many properties of the final learned model, including its robustness and generalization.[32] Such effects are *implicit* in the gradient descent update eq. (1). In contrast, the central flow renders *explicit* all effects of the learning rate hyperparameter η on the optimization process, allowing us to disentangle these effects from one another. Recall from eq. (21) that the central flow is a projected gradient flow with learning rate η that is constrained to the stable region $\mathbb{S} = \{w : S(w) \leq 2/\eta\}$. From this characterization, we see that the learning rate hyperparameter η has two distinct effects on the central flow: (1) it acts as a time rescaling, which controls the speed of optimization without affecting the overall trajectory; and (2) it determines the stable region, which affects the overall trajectory. Thus, increasing the learning rate constrains gradient descent to a smaller subset of weight space, but also allows it to traverse this set at a faster speed.[33]



Having introduced the central flows framework with an analysis of gradient descent, we will now use this methodology to understand the behavior of two adaptive optimizers.

[32] Large learning rates are necessary for obtaining good generalization in some deep learning settings (e.g. Li et al., 2019). However, obtaining the best generalization performance usually also requires stochastic optimization with a sufficiently small batch size. Since our paper exclusively studies the deterministic setting, we decided to not focus on generalization in this paper.

[33] The learning rate η that is optimal from an optimization perspective (i.e. that will decrease the loss the fastest) will depend on the trade-off between these two effects. Empirically, we observe that for deterministic gradient descent, larger learning rates usually optimize faster (provided that training does not diverge), implying that the former effect is stronger.

4 Scalar RMSProp

As a stepping stone to the analysis of RMSProp in Section 5, we first study "Scalar RMSProp," a simplification of RMSProp which uses one global step size, rather than separate step sizes for each coordinate:[34][35]

$$\nu_t = \beta_2 \nu_{t-1} + (1-\beta_2)\|\nabla L(w_t)\|^2, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{\nu_t}} \nabla L(w_t). \tag{24}$$

The algorithm maintains an exponential moving average (EMA), ν, of the squared gradient norm, and takes gradient steps of size $\eta/\sqrt{\nu}$, which we call the *effective step size*.[36] The EMA hyperparameter β_2 is a knob that interpolates the algorithm between gradient descent when $\beta_2 = 1$ and normalized gradient descent (NGD) when $\beta_2 = 0$.[37]

While optimizers such as Scalar RMSProp are often said to utilize an "adaptive step size," it has remained unclear what precise property of the local landscape the step size is being adapted *to* (Orabona, 2020). In this section, we will use the central flows framework to answer this basic question. After describing the dynamics of Scalar RMSProp in Section 4.1 and deriving a central flow in Section 4.2, we will interpret this flow to understand the optimizer's behavior in Section 4.3. In particular:

- In Section 4.3.1, we make precise how Scalar RMSProp adapts its step size to the local loss landscape. Specifically, we show that the optimizer's dynamics implicitly set the effective step size to the value $2/S(w)$, where $S(w)$ is the current sharpness; this value is the *largest stable step size* at the current weights w.
- In Section 4.3.2, we show that step size adaptation is not the full story: Scalar RMSProp also implicitly regularizes curvature throughout training, and in fact, at EOS, the hyperparameters η, β_2 only affect the time-averaged trajectory by modulating the strength of this curvature regularization.
- Bringing it all together, in Section 4.3.3 we describe how the interplay between step size adaptation and curvature regularization gives rise to a mechanism we call *acceleration via regularization*, whereby the optimizer implicitly steers itself towards low-curvature regions where it can take larger steps. We show that this mechanism is key to the efficacy of Scalar RMSProp and to the function of its hyperparameters.

These points will generalize to RMSProp in Section 5, but are simpler to understand for Scalar RMSProp.

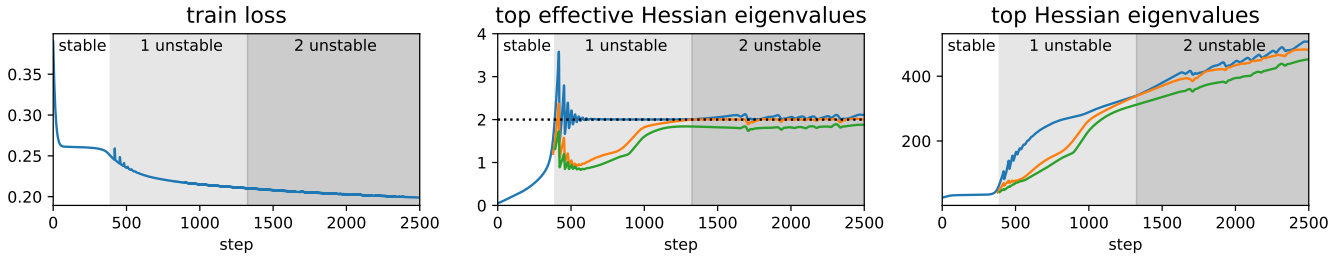


Figure 14: **A typical Scalar RMSProp trajectory**. We train a Mamba network on a sequence task using Scalar RMSProp with $\eta = 2/400$ and $\beta_2 = 0.99$. While the top eigenvalues of the "raw" Hessian $H(w)$ evolve freely (right), the top eigenvalue of the *effective* Hessian $\eta H(w)/\sqrt{\nu}$ equilibrates at the critical threshold 2 (center).

[34] Note that we have re-indexed ν compared to the standard definition of RMSProp (i.e. $\nu_{t+1} \to \nu_t$). This does not affect the trajectory and just ensures the effective learning rate at step t is determined by ν_t, rather than ν_{t+1}, which simplifies the notation.

[35] This algorithm was also studied by Lyu et al. (2022). However, their analysis only applies along a manifold of global minima, as $\eta \to 0$.

[36] The terms "learning rate" and "step size" are usually interchangeable. In this paper, to avoid ambiguity, we will use the phrase "learning rate" to denote the hyperparameter, and "step size" or "effective step size" to denote the actual step sizes that are taken.

[37] When $\beta_2 = 1$, Scalar RMSProp reduces to gradient descent with learning rate $\eta/\sqrt{\nu_0}$. Conversely, when $\beta_2 = 0$, it reduces to normalized gradient descent with learning rate η: $w_{t+1} = w_t - \eta \cdot \frac{\nabla L(w_t)}{\|\nabla L(w_t)\|}$.

4.1 The Dynamics of Scalar RMSProp

The dynamics of Scalar RMSProp revolve around the *effective sharpness*, defined as $S^{\text{eff}} := \eta S(w)/\sqrt{\nu}$.[38] First, the effective sharpness controls the oscillations: when $S^{\text{eff}} > 2$, Scalar RMSProp oscillates with growing magnitude along high curvature direction(s). Second, such oscillations in turn trigger a reduction of effective sharpness. This occurs via a combination of two distinct mechanisms. One mechanism, shared with gradient descent, is that oscillations implicitly reduce sharpness due to eq. (9), thereby decreasing the effective sharpness via its *numerator*. The other mechanism, new to Scalar RMSProp, is that oscillations increase the gradient norm and hence ν, thereby decreasing effective sharpness via its *denominator*. These dynamics give rise to a negative feedback loop that keeps the effective sharpness automatically regulated around the value 2, as depicted in Figure 14. The fine-grained dynamics are complex and challenging to analyze, even in the case of a single unstable eigenvalue. Fortunately, we will see in the next section that analyzing the *time-averaged* dynamics is much simpler.

4.2 Deriving the Scalar RMSProp Central Flow

Recall that while gradient descent trains stably, it is well-approximated by gradient flow. One can derive an analogous "stable flow" for Scalar RMSProp (Ma et al., 2022, cf.):[39]

$$\frac{dw}{dt} = -\frac{\eta}{\sqrt{\nu}}\nabla L(w), \qquad \frac{d\nu}{dt} = \frac{1-\beta_2}{\beta_2}\left[\|\nabla L(w)\|^2 - \nu\right]. \tag{25}$$

However, at the edge of stability, the trajectory of Scalar RMSProp deviates from eq. (25). We will now derive a more general *central flow* that characterizes the time-averaged trajectory even at EOS. In the main text, we will focus on the special case where one eigenvalue is, and remains at, the edge of stability. See Section A.3 for our full derivation which accounts for multiple eigenvalues at EOS and for eigenvalues entering and leaving EOS.

In Section 3.2.1, we derived an approximation for the time-averaged gradient, $\mathbb{E}[\nabla L(w_t)]$. Using the first two terms of eq. (9), we can also derive a time-averaged approximation for the squared gradient norm $\mathbb{E}[\|\nabla L(w_t)\|^2]$:

$$\mathbb{E}[\|\nabla L(w_t)\|^2] \approx \|\nabla L(\overline{w}_t)\|^2 + \cancel{2\left\langle \nabla L(\overline{w}_t), u \right\rangle S(\overline{w}_t)\,\mathbb{E}[x_t]} + S(\overline{w}_t)^2\,\mathbb{E}[x_t^2]$$

where we again used $\mathbb{E}[x_t] = 0$ to ignore the middle term. This calculation makes clear that larger oscillations (i.e. higher $\mathbb{E}[x_t^2]$) increase the squared gradient norm on average over time. Based on these time averages, we make the ansatz that the joint dynamics of (w_t, ν_t) follow a central flow $(w(t), \nu(t))$ of the form:

$$\frac{dw}{dt} = -\frac{\eta}{\sqrt{\nu}}\Big[\underbrace{\nabla L(w) + \tfrac{1}{2}\sigma^2(t)\nabla S(w)}_{\mathbb{E}[\nabla L(w_t)]}\Big], \qquad \frac{d\nu}{dt} = \frac{1-\beta_2}{\beta_2}\Big[\underbrace{\|\nabla L(w)\|^2 + S(w)^2\sigma^2(t)}_{\mathbb{E}[\|\nabla L(w_t)\|^2]} - \nu\Big], \tag{26}$$

where $\sigma^2(t)$ is a still-unknown quantity intended to model $\mathbb{E}[x_t^2]$, the instantaneous variance of the oscillations. As in our analysis of gradient descent, there is a unique value of $\sigma^2(t)$ that maintains $S^{\text{eff}}(w, \nu) = 2$. To compute it, we expand $\frac{dS^{\text{eff}}}{dt}$ using the chain rule: $\frac{dS^{\text{eff}}}{dt} = \langle \frac{\partial S^{\text{eff}}}{\partial w}, \frac{dw}{dt} \rangle + \frac{\partial S^{\text{eff}}}{\partial \nu} \cdot \frac{d\nu}{dt}$. Plugging in $\frac{dw}{dt}, \frac{d\nu}{dt}$ from eq. (26) shows that $\frac{dS^{\text{eff}}}{dt}$ is linear in σ^2. Thus, there is a unique value of σ^2 that will ensure $\frac{dS^{\text{eff}}}{dt} = 0$, which is given by:

$$\sigma^2(w; \eta, \beta_2) = \frac{\beta_2 \overbrace{\langle -\nabla L(w), \nabla S(w)\rangle}^{\text{progressive sharpening}} + (1-\beta_2)\overbrace{\left[S(w)^2/4 - \|\nabla L(w)\|^2/\eta^2\right]}^{\text{effect of mean reversion on } \nu}}{\beta_2 \underbrace{\frac{1}{2}\|\nabla S(w)\|^2}_{\text{sharpness reduction}} + (1-\beta_2)\underbrace{S(w)^2/\eta^2}_{\text{effect of oscillation on } \nu}}. \tag{27}$$

[38] While we could have defined effective sharpness as $S(w)/\sqrt{\nu}$ so that it would equilibrate at $2/\eta$, this version makes the analysis easier.

[39] The $1-\beta_2 \to \frac{1-\beta_2}{\beta_2}$ correction is necessary for small values of β_2. For example, when $\beta_2 = 0$ (i.e. normalized gradient descent), $\nu_t = \|\nabla L(w_t)\|^2$ so in the continuous time ODE, $\nu(t)$ needs to adapt "instantly" to $\|\nabla L(w(t))\|^2$. See Section A.7 for additional justification for this correction term.

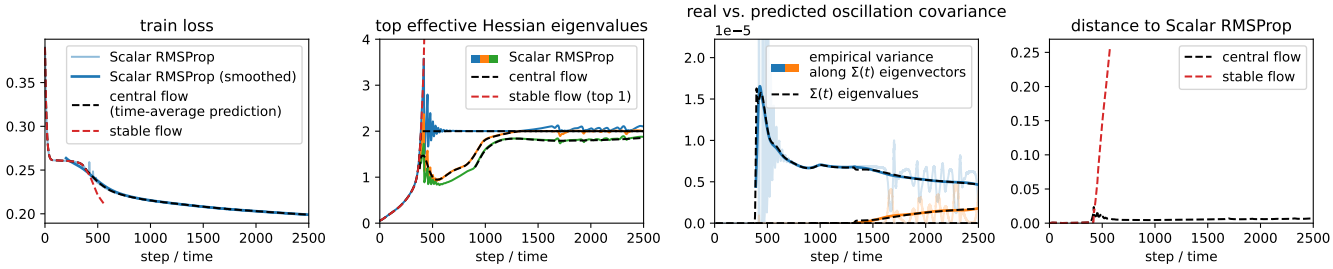


Figure 15: **Central flow for Scalar RMSProp.** The central flow (black) accurately models the time-averaged trajectory of Scalar RMSProp even at the edge of stability, whereas the naive stable flow (red) follows a different path. As with gradient descent, our analysis can accurately predict the covariance $\Sigma(t)$ with which Scalar RMSProp oscillates around the central flow (third panel). The setting is the same as Figure 14.

The central flow for Scalar RMSProp in the special case of one unstable eigenvalue is given by eq. (26) with this value of σ^2.[40] The fully general central flow is given in Section A.3, Definition 7. Figure 15 illustrates how this central flow can accurately predict the long-term trajectory of Scalar RMSProp as well as the covariance with which Scalar RMSProp oscillates around that trajectory. Figures 33(a) and 33(b) show, in a variety of deep learning settings, that this central flow can accurately predict the loss curve of Scalar RMSProp at different learning rates. Figure 36 shows that the central flow holds across different values of β_2. See Section E.2 for the full set of raw experiments.

The analysis in this section highlights the potential of our time-averaging methodology. With just a single invocation of the chain rule, we have characterized the long-term trajectory of a complex dynamical system involving mutual interactions between the oscillations, the sharpness, and the adaptive step size.

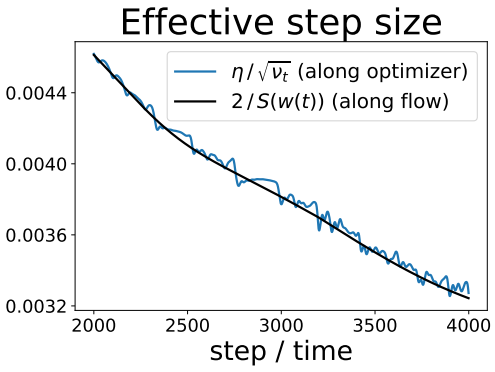
4.3 Understanding Scalar RMSProp via its Central Flow

We now interpret the Scalar RMSProp central flow to shed light on the behavior of the algorithm and the function of its hyperparameters η and β_2. Because the dynamics usually transition from stable to EOS quite early in training, we focus on interpreting the central flow in the EOS regime.[41]

4.3.1 Implicit step size selection

The central flow renders *explicit* the step size strategy that is *implicit* in the oscillatory dynamics of Scalar RMSProp. Recall that while the central flow is at EOS, the effective sharpness $S^{\text{eff}} := \eta S(w)/\sqrt{\nu}$ is fixed at 2. Indeed, this is the equilibrium condition that is automatically maintained by the dynamics of optimization. This EOS condition can be rearranged into a statement about the effective step size:

$$\eta/\sqrt{\nu} = 2/S(w). \tag{28}$$



That is, at EOS, the effective step size along the central flow is always equal to the value $2/S(w)$. Notably, the value $2/S(w)$ is the *largest stable step size* for gradient descent at location w. Thus, while Scalar RMSProp is at EOS, **the oscillatory dynamics continually adapt the effective step size to the current largest stable step size**, even as this value evolves throughout training. This is the precise sense in which Scalar RMSProp "adapts" its step size to the local loss landscape.

[40] The Scalar RMSProp central flow can be interpreted as a projected flow in the augmented space (w, ν) under a certain non-Euclidean norm. However, because this flow is not a gradient flow, it does not immediately suggest a decreasing potential function for Scalar RMSProp.

[41] In the stable regime ($S^{\text{eff}} < 2$), the central flow is given by the stable flow eq. (25). For this flow, $\frac{dw}{dt}$ is directly proportional to that of gradient flow, implying these flows traverse the same trajectory, just at a different speed (i.e. with a nonlinear time-rescaling). In this regime, the effective step size generally increases monotonically, so Scalar RMSProp follows gradient flow with a learning rate warmup.

In principle, it would be possible for an optimizer to *manually* compute the sharpness $S(w)$ at each iteration (e.g. by using the power method), and to manually set the step size to $2/S(w)$. However, computing the sharpness would incur some computational overhead, whereas we have shown that Scalar RMSProp finds the maximum stable step size of $2/S(w)$ *efficiently*, using no more computation than is already used by gradient descent (namely, one gradient computation per iteration). This rich behavior is implicit in the algorithm's oscillatory dynamics.

Furthermore, note that even comprehending this behavior requires an appeal to some notion of time-averaging. The effective step size is usually not *exactly* at $2/S(w)$, but rather is fluctuating around $2/S(w)$. The important point is that it is $2/S(w)$ on average over time. The central flow perspective gives a way to reason about this behavior.

4.3.2 Implicit curvature reduction

Understanding the implicit step size strategy employed by Scalar RMSProp is not sufficient to fully characterize the behavior of the algorithm. To do so, we need to return to the central flow, which additionally accounts for the curvature regularization induced by oscillations. In general, the Scalar RMSProp central flow is a joint flow over (w, ν). However, at EOS, because $\eta/\sqrt{\nu} = 2/S(w) \iff \nu = \frac{\eta^2 S(w)^2}{4}$, we can eliminate ν from the expression for $\frac{dw}{dt}$, and write the central flow in terms of w alone:

$$\frac{dw}{dt} = -\underbrace{\frac{2}{S(w)}}_{\text{effective step size}}\Big[\nabla L(w) + \underbrace{\frac{1}{2}\sigma^2(w;\eta,\beta_2)\nabla S(w)}_{\text{implicit sharpness penalty}}\Big] \tag{29}$$

where $\sigma^2(w;\eta,\beta_2)$ is given by eq. (27). In other words, the time-averaged trajectory of Scalar RMSProp at EOS is essentially equivalent to that of the following simpler-to-understand algorithm:

> At each iteration, compute the sharpness $S(w)$, and take a gradient step of size $2/S(w)$ on a sharpness-regularized objective, where the strength of the sharpness regularizer is given by eq. (27).

Interestingly, the hyperparameters η, β_2 are not used to determine the effective step size $2/S(w)$. Instead, their only role is to modulate σ^2, which controls the strength of the implicit sharpness penalty. The effect of the learning rate hyperparameter η is to *monotonically increase* σ^2 — indeed, the numerator of eq. (27) is increasing in η while the denominator is decreasing in η, which implies the overall expression for σ^2 is increasing in η. The simplest case is that of NGD, i.e. when $\beta_2 = 0$, for which eq. (27) reduces to $\sigma^2 \approx \frac{\eta^2}{4}$ (see Section A.3). Meanwhile, the effect of the hyperparameter β_2 is to monotonically interpolate σ^2 between that of normalized gradient descent when $\beta_2 = 0$ and that of gradient descent when $\beta_2 = 1$.[42] The interpretations of η, β_2 generalize to the setting of multiple unstable eigenvalues, as detailed in Section A.3, Proposition 5.

4.3.3 Acceleration via regularization

To fully grasp the *modus operandi* of Scalar RMSProp, it is necessary to consider the link between step size adaptation and curvature regularization. By regularizing sharpness $S(w)$, Scalar RMSProp is able to steer itself towards regions where the maximal locally stable step size of $2/S(w)$ is larger. In such regions, Scalar RMSProp can and does take larger steps. Thus, **by regularizing sharpness, Scalar RMSProp enables itself to take larger steps later in training**. We call this mechanism *acceleration via regularization*. Our experiments suggest that this mechanism is a critical component of the algorithm's effectiveness. In Figure 16, we compare the Scalar RMSProp central flow to an ablated version which adapts the step size to $2/S(w)$ but does not regularize sharpness. Over the long term, this ablated flow optimizes slower than the Scalar RMSProp central flow, because it traverses sharper regions of weight space in which it is forced to take smaller steps. (See Section D, Figure 40 for more settings.)

[42] We note that which of these is larger is situation dependent, so σ^2 can be either monotonically increasing or monotonically decreasing in β_2. That said, because when $\beta_2 = 0$, $\sigma^2(w;\eta,0) \approx \eta^2/4$ and when $\beta_2 = 1$, $\sigma^2(w;\eta,1)$ is independent of η, a general rule is that for small learning rates, σ^2 is monotonically increasing in β_2, while for large learning rates, σ^2 is monotonically decreasing in β_2.

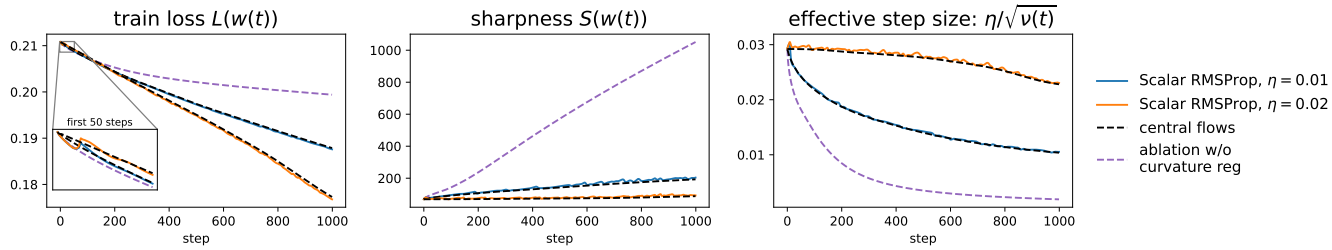


Figure 16: **Implicit curvature regularization accelerates optimization for Scalar RMSProp**. Starting from the same initial point, we run Scalar RMSProp at two different learning rates (blue and orange), alongside the corresponding central flows (black). We also run an ablated flow $\frac{dw}{dt} = -\frac{2}{S(w)}\nabla L(w)$ which has curvature regularization removed (purple). All three flows use the same step size strategy, and differ only in the strength of implicit curvature regularization. Initially (see inset), the flows with higher curvature regularization optimize slower; however, over the longer run, they take larger steps and optimize faster. This figure is in the same setting as Figure 15.[43]

The mechanism of "acceleration via regularization" is also key for understanding the function of the learning rate hyperparameter η. We have seen that at EOS, the only direct effect of η on the central flow is to modulate the strength of sharpness regularization, with higher η inducing stronger sharpness regularization. Thus, counterintuitively, the *instantaneous* effect of a higher η is often to *slow down* optimization. However, as we illustrate in Figures 16 and 40, over longer timescales, higher η steers the trajectory into lower-sharpness regions, in which Scalar RMSProp's effective step size will be larger, thereby tending to *speed up* optimization. Thus, as one would expect of a learning rate hyperparameter, larger η can accelerate optimization; however they do so through this *indirect* mechanism.

[43] In this figure, for Scalar RMSProp, we show the train loss at the second-order midpoints between iterates (see Section B.1).

5 RMSProp

We now study RMSProp (Tieleman and Hinton, 2012), which is equivalent to Adam (Kingma and Ba, 2015) without momentum. RMSProp maintains an EMA ν of the elementwise squared gradients $\nabla L(w)^{\odot 2}$, and uses *per-coordinate* effective step sizes of $\eta/\sqrt{\nu}$:[44]

$$\nu_t = \beta_2 \nu_{t-1} + (1-\beta_2)\nabla L(w_t)^{\odot 2}, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{\nu_t}} \odot \nabla L(w_t), \tag{30}$$

where $\odot$ represents the entrywise product. RMSProp can also be viewed as preconditioned gradient descent $w_{t+1} = w_t - P_t^{-1}\nabla L(w_t)$ with the dynamic preconditioner $P_t := \mathrm{diag}(\sqrt{\nu_t}/\eta)$.[45] While Adam employs the same dynamic preconditioner and has achieved widespread success in deep learning, it has remained unclear why this specific preconditioning strategy is so effective (Kunstner et al., 2019; Orabona, 2020; Martens, 2020). A common folklore belief is that Adam/RMSProp adapts to the local "curvature" (i.e. Hessian). However, it is a priori unclear how this can be so, since the algorithm uses the (squared) *gradient*, not the Hessian, to update its preconditioner.

In this section, we use the central flows framework to understand the behavior of RMSProp. We will show that RMSProp *does* adapt to the local Hessian after all, but the reason is inextricably tied to its oscillatory dynamics, which have not been previously studied.

We start by describing the dynamics of RMSProp in Section 5.1. We then derive a central flow in Section 5.2. Finally, in Section 5.3, we interpret this flow to understand the optimizer's behavior. In particular:

- In Section 5.3.1, we show that RMSProp's preconditioner is *implicitly* determined by the algorithm's oscillatory dynamics, and we make this preconditioner *explicit* for the first time. Specifically, we show that RMSProp computes its preconditioner by solving a convex program (eq. 35) involving the Hessian. This clarifies that RMSProp is **implicitly a second-order optimizer**, despite only accessing the loss through first-order gradients.
- In Section 5.3.2, we show that, like Scalar RMSProp, the success of RMSProp relies not only on this preconditioning strategy, but also on an *acceleration via regularization* mechanism whereby implicitly regularizing curvature allows the optimizer to take larger steps later in training.

These basic insights into the functioning of RMSProp are a prerequisite for a similar understanding of Adam, and may guide us in using these optimizers more effectively and in improving upon them.

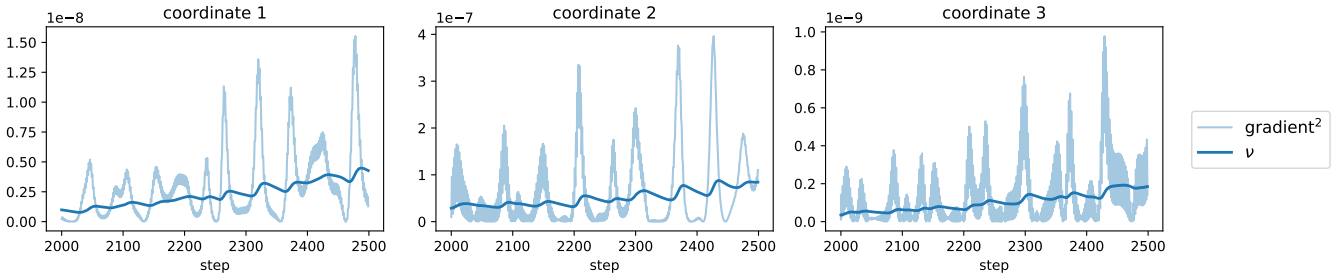


Figure 17: **RMSProp ν is determined by oscillations.** While training a network using RMSProp, we plot the squared gradient $\nabla L(w_t)^{\odot 2}$ (light blue) and its EMA ν_t (dark blue) at three coordinates (subpanels). Due to the EOS oscillations, the squared gradient fluctuates, causing the EMA ν_t to also fluctuate. Since this EMA is used to determine the effective step sizes $\eta/\sqrt{\nu_t}$, analyzing these dynamics is necessary for understanding RMSProp's adaptivity. This network is a ResNet trained on a subset of CIFAR-10 using $\eta =$ 2e-5, $\beta_2 = 0.99$ and MSE loss.

[44] Our analysis can accommodate both bias correction and an ϵ-dampening (dividing by $\sqrt{\nu+\epsilon}$ rather than $\sqrt{\nu}$) which are used by Adam (see Section A.5). However, to simplify exposition, the main text focuses on this simpler version of RMSProp.

[45] Folding η into the preconditioner is unconventional, but will make the analysis clearer.

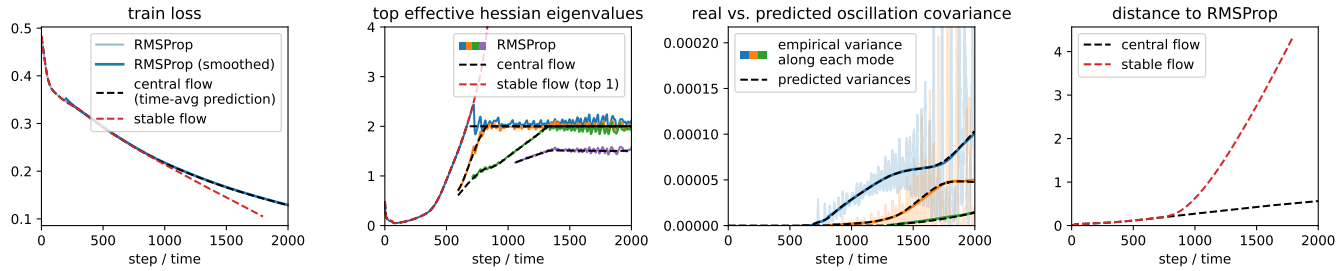


Figure 18: **Central flow for RMSProp.** The RMSProp central flow (black) accurately models the macroscopic trajectory of RMSProp even at EOS, whereas the naive stable flow (red) follows a different path. As for gradient descent and Scalar RMSProp, we are able to predict the covariance $\Sigma(t)$ with which RMSProp oscillates around the central flow (third panel). This figure is in the same setting as Figure 17.

5.1 The Dynamics of RMSProp

To give some intuition into RMSProp's behavior, Figure 17 plots the dynamics of the squared gradient $\nabla L(w_t)^{\odot 2}$ and its EMA ν_t at several coordinates over a stretch of training. Observe that the entries of the squared gradient fluctuate rapidly, causing their EMA to also fluctuate. Since this EMA ν directly determines the effective step sizes $\eta/\sqrt{\nu}$, understanding the origin of this behavior is necessary to understand how RMSProp sets its effective step sizes.

These fluctuations in the gradient arise because RMSProp is operating in an oscillatory *edge of stability* regime. To understand why RMSProp oscillates, first consider running preconditioned gradient descent $w_{t+1} = w_t - P^{-1}\nabla L(w_t)$ on a quadratic function with Hessian H. The resulting dynamics are controlled by the *effective Hessian* $P^{-1}H$. Namely, if any eigenvalues of this matrix exceed the critical threshold 2, then preconditioned GD will oscillate with exponentially growing magnitude along the corresponding (right) eigenvectors.[46] For RMSProp in deep learning, both the Hessian $H(w_t)$ and the preconditioner $P_t = \mathrm{diag}(\sqrt{\nu_t}/\eta)$ can vary. However, a local quadratic Taylor approximation suggests that RMSProp will oscillate if the largest eigenvalue of the *current* effective Hessian $P_t^{-1}H(w_t)$ exceeds the critical threshold 2.[47] We refer to this quantity as the effective sharpness $S^{\text{eff}}(w_t, \nu_t)$:

$$S^{\text{eff}}(w_t, \nu_t) := \lambda_1(P_t^{-1}H(w_t)). \tag{31}$$

Paralleling the dynamics of gradient descent, Cohen et al. (2022) observed that RMSProp typically operates in an oscillatory EOS regime that revolves around the effective sharpness eq. (31). On the one hand, oscillations ensue whenever the effective sharpness rises above the critical threshold 2.[48] On the other hand, such oscillations reduce the effective sharpness, both by inducing implicit regularization of curvature (i.e. shrinking $H(w_t)$), and by growing the gradient and hence the preconditioner P_t. The net result is that the effective sharpness equilibrates around the value 2 (see Figure 18), as the optimizer oscillates along the top eigenvectors of the effective Hessian.

5.2 Deriving the RMSProp Central Flow

Similar as before, we now derive a central flow $(w(t), \nu(t))$ that jointly models the time-averaged dynamics of w_t, ν_t. We defer the full details to Section A.4 and sketch the argument here.

If RMSProp is oscillating around its time-averaged trajectory $\{\overline{w}_t\}$, so that $w_t = \overline{w}_t + \delta_t$, then the time average of

[46] On a quadratic $\frac{1}{2}w^T H w$, this algorithm evolves via: $w_{t+1} = (I - P^{-1}H)w_t \implies w_t = (I - P^{-1}H)w_0$. If $P^{-1}H$ has any eigenvalues greater than 2, $(I - P^{-1}H)$ has eigenvalues less than -1, and the iterates diverge along the corresponding right eigenvectors.

[47] With this argument, we are also implicitly assuming that the preconditioner evolves sufficiently slowly that its movement can be neglected.

[48] For RMSProp (and Scalar RMSProp), the effective sharpness eq. (31) tends to rise *both* because the curvature tends to rise (progressive sharpening) *and* because the gradient (and hence ν) tends to shrink. Due to the second effect, RMSProp often enters EOS sooner, and for a large range of learning rates, than gradient descent. Further, RMSProp enters EOS even on quadratics, whereas gradient descent does not.

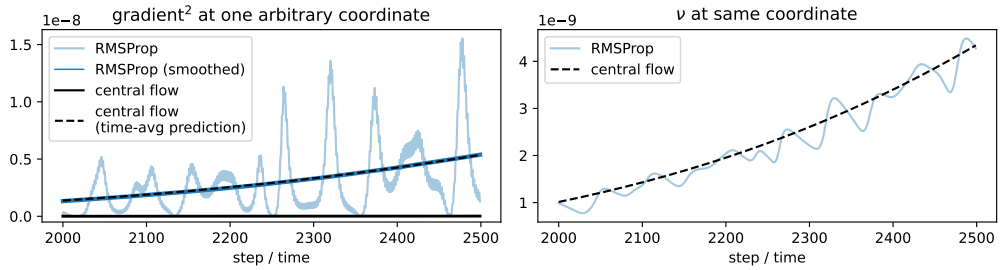


Figure 19: **Central flow can successfully predict both the time-averaged gradient^2 (left) and the EMA ν (right).** On the left, we show that although the squared gradient is fluctuating erratically (recall Figure 17), its time-average can be predicted by the central flow using eq. (33). On the right, we show that the central flow's $\nu(t)$ accurately tracks the macroscopic trajectory of the real EMA ν_t. This figure is in the same setting as Figure 17.

the elementwise squared gradient is approximately:

$$\underbrace{\mathbb{E}[\nabla L(w_t)^{\odot 2}]}_{\substack{\text{time-average of}\\ \text{squared gradient}}} \approx \underbrace{\nabla L(\overline{w}_t)^{\odot 2}}_{\substack{\text{squared gradient at}\\ \text{time-averaged iterate}}} + \underbrace{\operatorname{diag}[H(\overline{w}_t)\,\mathbb{E}[\delta_t \delta_t^T] H(\overline{w}_t)]}_{\text{contribution from oscillations}}. \tag{32}$$

The first term is the squared gradient at the time-averaged iterate; the second term is the contribution to the squared gradient that originates from oscillating with covariance $\mathbb{E}[\delta_t \delta_t^T]$.

If we further assume then these oscillations are contained within the right eigenspace of the effective Hessian $\operatorname{diag}(\eta/\sqrt{\nu_t})H(w_t)$ that corresponds to the eigenvalue 2, then the rightmost term simplifies as follows:

$$\underbrace{\mathbb{E}[\nabla L(w_t)^{\odot 2}]}_{\substack{\text{time-average of}\\ \text{squared gradient}}} \approx \underbrace{\nabla L(\overline{w}_t)^{\odot 2}}_{\substack{\text{squared gradient at}\\ \text{time-averaged iterate}}} + \underbrace{\frac{4\nu}{\eta^2} \odot \operatorname{diag}[\mathbb{E}[\delta_t \delta_t^T]]}_{\text{contribution from oscillations}} \tag{33}$$

Based on this calculation, and on the time-averaged gradient computed in eq. (18), we make the ansatz that the time-averaged dynamics of w_t, ν_t follow a central flow $(w(t), \nu(t))$ with the functional form:

$$\frac{dw}{dt} = -\frac{\eta}{\sqrt{\nu}} \odot \left[\underbrace{\nabla L(w) + \tfrac{1}{2}\nabla\langle \Sigma(t), H(w)\rangle}_{\mathbb{E}[\nabla L(w_t)]}\right], \quad \frac{d\nu}{dt} = \frac{1-\beta_2}{\beta_2}\left[\underbrace{\nabla L(w)^{\odot 2} + \frac{4\nu}{\eta^2} \odot \operatorname{diag}[\Sigma(t)]}_{\mathbb{E}[\nabla L(w_t)^{\odot 2}]} - \nu\right]. \tag{34}$$

To determine $\Sigma(t)$, we impose three conditions on this flow, analogous to those from Section 3.2.2. As before, it can be shown that there is a unique matrix $\Sigma(t)$ satisfying these three conditions, and this matrix can be characterized as the solution to a semidefinite complementarity problem. The RMSProp central flow is defined as eq. (34) with this value of $\Sigma(t)$. See Section A.4, Definition 9 for a formal statement.

Figure 18 illustrates how this central flow can accurately predict the macroscopic trajectory $w(t)$ of RMSProp, as well as the covariance $\Sigma(t)$ with which RMSProp is oscillating around that trajectory.[49] Figure 19 shows how the central flow can accurately predict the time-average of the elementwise squared gradient via eq. (33), as well as the macroscopic trajectory $\nu(t)$ of the EMA. Figure 34(a) and Figure 34(b) in Section D show in a variety of deep learning settings that the central flow can accurately predict the RMSProp loss curve across different learning rates. Figure 35 and Figure 37 show that the central flow can accurately predict the RMSProp trajectory at different values of β_2 and ϵ, respectively. The full set of raw RMSProp experiments can be found in Section E.3.

[49]To match the preconditioned geometry of the optimizer, we assess whether each eigenvalue of $P(\nu(t))^{1/2}\Sigma(t)\,P(\nu(t))^{1/2}$ accurately predicts the P-whitened variance of oscillations along the corresponding eigenvector; see eq. (126) in Section A.4.

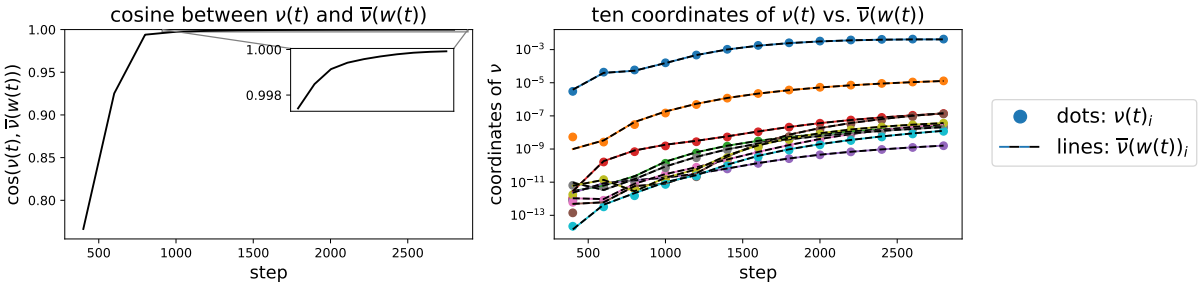


Figure 20: **The EMA ν converges to its stationary value.** While running the RMSProp central flow, we compare the actual EMA $\nu(t)$ to its stationary value $\overline{\nu}(w(t))$ w.r.t the current weights $w(t)$. On the left, we plot the cosine similarity between $\nu(t)$ and $\overline{\nu}(w(t))$; on the right, we compare ten individual coordinates (colors), spaced uniformly throughout the network. The plots begin when training enters EOS, just before step 500. Observe that after a bit of time, the cosine similarity between $\nu(t)$ and $\overline{\nu}(w(t))$ reaches high values (near 1), and the individual coordinates coincide as well. This figure depicts the same setting as Figure 17; see Figures 42(a) to 43(b) for more settings.

As with gradient descent, we find that the central flow approximation tends to become less accurate as the learning rate η grows; see Section 6 for our general discussion about the accuracy of the central flow. In addition, we observe that the central flow for RMSProp tends to be a bit less accurate overall than that for gradient descent, at least as measured by weight-space distance between the flow and the discrete optimizer. Finally, we expect the central flow for RMSProp to break down when β_2 becomes too close to zero (i.e. the sign GD limit), as then RMSProp would no longer resemble preconditioned gradient descent with a slowly-changing preconditioner.

5.3 Understanding RMSProp via its Central Flow

We now interpret the RMSProp central flow to understand the behavior of RMSProp, including how the algorithm sets its effective step sizes $\eta/\sqrt{\nu}$. Because the dynamics usually transition from stable to EOS early in training, we focus on the EOS regime.

5.3.1 The stationary preconditioner

Stationarity of ν Unfortunately, even at the edge of stability, $\nu(t)$ cannot be expressed as a closed-form function of $w(t)$ (as it could for Scalar RMSProp in Section 4), and instead remains an independent variable that must be tracked. This reflects the fact that for any w, there are potentially many values for ν that could stabilize optimization, and the actual value used by RMSProp depends on the history. Nevertheless, we will now see that under the RMSProp central flow, ν often implicitly converges to a value that depends on the current w alone.

Intuitively, the RMSProp central flow eq. (34) involves two simultaneous processes of optimization (the w dynamics) and preconditioner adaptation (the ν dynamics). Suppose that the ν dynamics of preconditioner adaptation occur *fast* relative to the w dynamics of optimization, so that ν reaches a stationary point w.r.t the current weights w. In Proposition 8 we show that for any w, there is in fact a *unique* ν that satisfies the stationarity condition $\frac{d\nu}{dt} = 0$. We call this unique ν the *stationary* ν for the weights w, denoted as $\overline{\nu}(w)$. Empirically, we observe that $\nu(t)$ usually starts to attain its stationary value $\overline{\nu}(w(t))$ at some point during training (after the dynamics have entered EOS), and continues to match $\overline{\nu}(w(t))$ thereafter, even as this value evolves. Indeed, Figure 20 illustrates how $\nu(t)$ converges to $\overline{\nu}(w(t))$ both in cosine similarity (left) and coordinate-wise (right). See Figures 42(a) to 43(b) for more settings.[50]

The stationarity of ν will allow us to reason about RMSProp's preconditioning strategy with relative ease, i.e. without needing to account for the history of ν. At any weights w, we can view the corresponding *stationary preconditioner*

[50] Since the speed of the ν dynamics in eq. (34) is controlled by the β_2 hyperparameter, one might suspect that $\nu(t)$ will converge quicker to $\overline{\nu}(w(t))$ when β_2 is smaller, and we confirm this in Figure 44. Nevertheless, we emphasize that the quasistationarity of ν w.r.t w empirically holds even when β_2 is relatively large (e.g. 0.99). In keeping with our attitude throughout this paper, we do not claim to have an explanation.

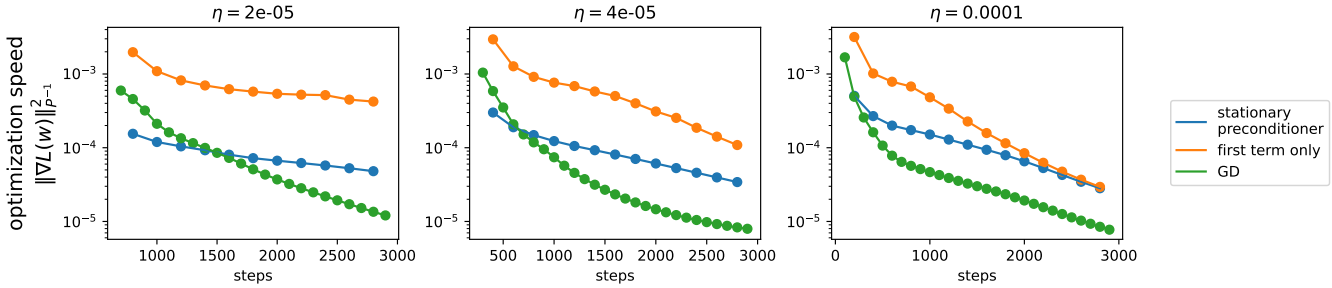


Figure 21: **Optimization speeds for various preconditioners.** Along the RMSProp central flow for various learning rates (columns), we assess the efficacy of three different preconditioners P: the RMSProp stationary preconditioner eq. (35), in blue; a variant eq. (36) with only the first term, in orange; and the preconditioner corresponding to vanilla gradient descent with the largest locally stable learning rate, i.e. $P^{-1} = (2/S(w))\, I$, in green. We assess each preconditioner P by reporting $\|\nabla L(w)\|^2_{P^{-1}} := \nabla L(w)^T P^{-1} \nabla L(w)$, the instantaneous rate of loss decrease under preconditioned gradient flow with preconditioner P. Observe that the "first term only" preconditioner (orange) is much better than the vanilla GD (green) preconditioner, and is also better than the actual stationary preconditioner (blue). The actual stationary preconditioner (blue) is usually better than vanilla GD (green), but not always, especially when η is smaller. See Figures 45(a) and 45(b) for more experimental settings.

$\overline{P}(w) := \mathrm{diag}(\sqrt{\overline{\nu}(w)}/\eta)$ as "the RMSProp preconditioner" that is implicitly used by RMSProp at weights w. We will now interpret this preconditioner to gain insight into RMSProp's preconditioning strategy.

Interpreting the stationary preconditioner In Proposition 7, we show that this stationary preconditioner $\overline{P}(w) := \mathrm{diag}(\sqrt{\overline{\nu}(w)}/\eta)$ is, remarkably, the optimal solution to a convex optimization problem over preconditioners:

$$\overline{P}(w) \quad := \underset{P \text{ diagonal},\, P \succeq 0}{\arg\min} \quad \mathrm{tr}(P) + \tfrac{1}{\eta^2} \underbrace{\|\nabla L(w)\|^2_{P^{-1}}}_{\text{optimization speed}} \quad \text{such that} \quad \underbrace{H(w) \preceq 2P}_{\text{local stability}}. \tag{35}$$

That is, **RMSProp implicitly solves the convex program eq. (35) to compute its preconditioner**.[51] This is the precise sense in which RMSProp "adapts" its preconditioner to the local loss landscape.

We can now understand RMSProp's preconditioning strategy by interpreting the optimization problem eq. (35). The constraint $H(w) \preceq 2P$ is equivalent to $S^{\text{eff}} \leq 2$ and hence stipulates that the preconditioner P should keep RMSProp locally stable. The first term of the objective, $\mathrm{tr}(P)$, is the sum of the inverse effective step sizes. If this were the only term in the objective, RMSProp's preconditioning strategy could be simply summarized as maximizing the *harmonic mean* of the effective step sizes while maintaining local stability — a sensible preconditioning strategy. Indeed, consider a variant of eq. (35) with only the first term:

$$\hat{P}(w) := \underset{P \text{ diagonal},\, P \succeq 0}{\arg\min} \quad \mathrm{tr}(P) \quad \text{such that} \quad H(w) \preceq 2P. \tag{36}$$

Figure 21 demonstrates that this preconditioner is a substantial improvement over vanilla gradient descent. (We describe in Section A.4.1 how we numerically solve eq. (35) and eq. (36).) Interestingly, if the diagonal constraint in eq. (36) were removed, and if $H(w)$ were PSD, then the optimization problem eq. (36) would have the closed-form solution $\hat{P}(w) = \frac{1}{2}H(w)$. That is, the preconditioner P would be a scaling of the Hessian, and preconditioned gradient descent would move in the same direction as Newton's method.[52]

However, matters are complicated by the presence of the second term in the eq. (35) objective. The quantity $\|\nabla L(w)\|^2_{P^{-1}}$ is the instantaneous rate of loss decrease under preconditioned gradient flow with preconditioner P.

[51] Interestingly, this SDP is the dual to the max-cut SDP: $\max_{\Sigma \succeq 0} \langle \Sigma, H \rangle$ such that $\Sigma_{ii} = 1$ for all i. Thus, this preconditioning strategy could be described as solving the max-cut SDP with the Hessian as the weight matrix, and using the resulting dual variable as its preconditioner.

[52] Even if $H(w)$ were not PSD, a similar point would hold: the optimization problem eq. (36) would have the closed-form solution $\hat{P} = \frac{1}{2}\Pi_{\mathbb{S}_+} H(w)$, where $\Pi_{\mathbb{S}_+}$ denotes projection onto the cone of positive semidefinite matrices.

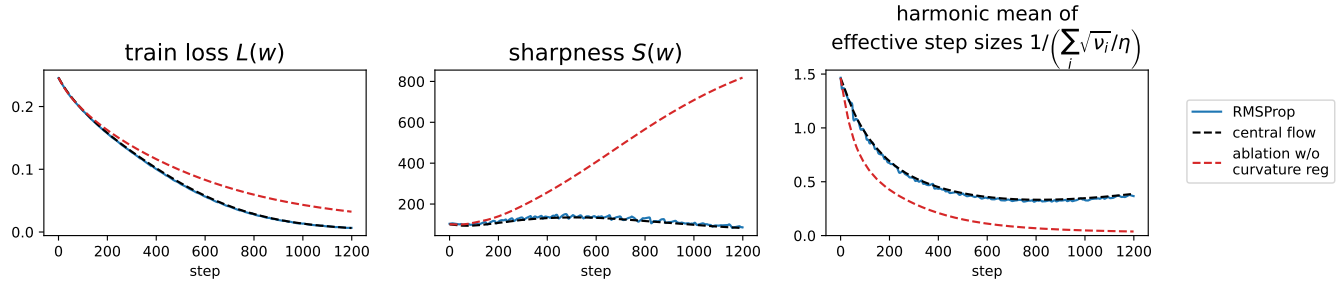


Figure 22: **Implicit curvature regularization accelerates optimization for RMSProp**. Starting from the same initial point, we compare RMSProp (blue) and its central flow (black) to an ablated flow where the implicit curvature regularization is disabled (red). Relative to this ablated flow, the central flow takes a lower-curvature trajectory (middle), in which it takes larger steps (right) and optimizes faster (left). The setting is the same as Figure 17.[55]

Minimizing this term necessarily acts to *slow down* optimization.[53] Indeed, Figure 21 shows that the stationary preconditioner eq. (35) underperforms the variant eq. (36) with only the first term.

Since the second term in eq. (35) is proportional to $\frac{1}{\eta^2}$, its influence diminishes as the learning rate hyperparameter η grows. Indeed, it can be seen in Figure 21 that the performance of the stationary preconditioner tends closer to that of eq. (36) as the learning rate hyperparameter η is made larger. In the limit of large η, the second term vanishes entirely, and the stationary preconditioner reduces completely to eq. (36). Interestingly, in this limit, the stationary preconditioner ceases to depend on η: for example, doubling η will cause $\overline{\nu}$ to quadruple in scale (due to larger oscillations), while keeping the effective step sizes $\eta/\sqrt{\overline{\nu}}$ unchanged. This parallels the situation for Scalar RMSProp in Section 4, where the effective step size at EOS was $2/S(w)$, independent of η.

The stationary flow Substituting $\overline{P}$ into the central flow, we can obtain a *stationary flow* over w alone:

$$\frac{dw}{dt} = -\underbrace{\overline{P}(w)^{-1}}_{\substack{\text{stationary} \\ \text{preconditioner}}} \Big[\nabla L(w) + \underbrace{\tfrac{1}{2}\nabla_w \langle \Sigma, H(w) \rangle}_{\text{implicit curvature penalty}} \Big], \tag{37}$$

where $\Sigma = \Sigma(w; \eta; \beta_2)$ is defined as the solution to a certain semidefinite complementarity problem (Section A.4.1, Definition 10). This model assumes that the ν dynamics (preconditioner adaptation) happen *infinitely fast* relative to the w dynamics (optimization), so that we can treat the preconditioner P as always being fixed at its current stationary value $\overline{P}(w)$ (eq. 35). The appeal of this characterization is that it eliminates ν from the picture entirely, and expresses the time-averaged dynamics of RMSProp as a closed system in w alone. Namely, it suggests that the time-averaged trajectory of RMSProp is equivalent to that of the following simpler-to-understand algorithm:

> At each iteration, compute the preconditioner $\overline{P}(w)$ using eq. (35) and then take a preconditioned gradient step using this preconditioner on a curvature-penalized objective.

Empirically, we find that the stationary flow eq. (37) is often a reasonable model for the RMSProp trajectory. For example, Figures 46(a) and 46(b) show that the stationary flow can accurately predict the instantaneous rate of loss decrease at various points along the central flow trajectory, even though it only has access to $w(t)$ and not $\nu(t)$. Meanwhile, Figures 47(a) and 47(b) show that the stationary flow can tolerably predict the trajectory of RMSProp over moderate timescales, although we note that its accuracy is not as high as the full central flow.[54]

[53] That is, for any w, the optimization speed $\|\nabla L(w)\|^2_{P^{-1}}$ must necessarily be smaller (worse) for eq. (35) than for eq. (36).

[53] In this figure, for RMSProp, we show the train loss at the second-order midpoints between iterates (see Section B.1).

[54] As one might expect, the stationary flow tends to only be an accurate model for RMSProp once ν has reached stationarity.

5.3.2 Acceleration via regularization

As with Scalar RMSProp, we find that RMSProp's implicit curvature regularization enables it to optimize faster. In Figure 22, we show that when the curvature regularization is disabled, the RMSProp central flow navigates into increasingly sharp regions, where it takes smaller steps, and optimizes slower (see Figure 41 for more settings).[56]

Establishing this claim theoretically is more difficult for RMSProp than Scalar RMSProp.[57] However, in the limit of large η and small β_2, it can be argued (Section A.4.1) that the stationary flow eq. (37) reduces to:

$$\frac{dw}{dt} = -\hat{P}(w)^{-1}\left[\nabla L(w) + \frac{\eta^2}{4}\nabla \operatorname{tr} \hat{P}(w)\right], \tag{38}$$

where $\hat{P}(w)$ was defined in eq. (36). This model says that RMSProp implicitly picks the diagonal preconditioner with minimal trace (equivalently, the preconditioner P where the effective learning rates P^{-1} have maximal harmonic mean), and also implicitly moves in a direction in which the trace of this preconditioner will become even smaller. The strength of the latter effect is controlled by η, and in fact, this is the only means by which the learning rate hyperparameter η affects the trajectory, since the preconditioner $\hat{P}(w)$ is independent of η. Thus, as with Scalar RMSProp, larger learning rates translate to larger steps, but only via this indirect mechanism.

[56] To run this ablated flow, we manually set $\nabla H(w) = 0$ both in the expression for β and in the expression for $\frac{dw}{dt}$ (see Section A.4).

[57] Partly, the difficulty of analysis is due to the independent ν dynamics. However, this analysis is also not easy under the stationary flow, because the second term in eq. (35) causes the effective step sizes to depend not just on the current Hessian but also on the current gradient.

6 Experiments

The goal of our experiments is to establish that each central flow accurately approximates the trajectory of its corresponding optimizer in a variety of deep learning settings, and to understand the circumstances under which this approximation breaks down. Because it is computationally costly to discretize central flows, we experiment on small-scale networks and datasets. Note that there is no evidence that scale itself fundamentally affects the dynamics of optimization in deep learning; for example, EOS dynamics have been observed at both smaller (e.g. CIFAR-10) and larger (e.g. ImageNet or WMT) scales, without noticeable differences. Therefore, we expect that the central flow approximation would similarly hold true at larger scales, if such experiments were computationally feasible.

We emphasize that the central flow is a theoretical tool for understanding optimizer behavior, not a practical optimization method. In practice, maintaining an exponential moving average of the iterates (e.g., Morales-Brotons et al., 2024) is likely a computational feasible way to estimate the optimizer's time-averaged trajectory.

Architectures We experiment on a diverse set of six architectures: a convolutional neural network (CNN), a ResNet (He et al., 2016), a Vision Transformer (ViT) (Dosovitskiy et al., 2021), an LSTM (Hochreiter and Schmidhuber, 1997), a (sequence) Transformer (Vaswani et al., 2017), and a Mamba sequence model (Gu and Dao, 2024). Architectural details can be found in Section B.2.

Datasets We test the vision architectures (CNN, ResNet, ViT) on a subset of CIFAR-10 (Krizhevsky, 2009), and the sequence architectures (LSTM, Transformer, Mamba) on a synthetic sorting task (Karpathy, 2020). Further details on these datasets can be found in Section B.3. For each architecture and each dataset, we test both cross-entropy loss and MSE loss. As discussed below, the central flow tends to be somewhat more accurate with MSE loss.

Implementation Discretizing the central flows is somewhat nontrivial, as the flows are non-smooth at points where there is a change in the number of unstable eigenvalues (e.g. going from 0 to 1). We describe our solution in Section A.2.4 for gradient descent and in Section A.5.1 for a generic (potentially) adaptive optimizer. The time complexity of each discretization step scales quadratically with the number of eigenvalues that are at the edge of stability. Most of the computational cost arises from the need to continually re-estimate the top eigenvectors and eigenvalues of the (effective) Hessian, and to compute the necessary third derivatives (gradients of these eigenvalues). Full implementation details can be found in Section B.1.

Our code can be found at: `https://github.com/centralflows/centralflows`.

6.1 Experimental Results

To assess the accuracy of the central flow approximation, we run both the discrete optimizer and the central flow simultaneously, starting from the same initialization. As a baseline, we also run the corresponding stable flow (e.g. for gradient descent, the gradient flow), which we expect to poorly approximate the discrete optimizer when the latter is at the edge of stability.

Our full experimental results, which can be found in Section E, make clear that the central flow can accurately approximate the long-term optimization trajectory in a variety of deep learning settings. We find that the weight-space distance between the discrete optimizer and the central flow generally stays small over time, and is much smaller than the distance between the discrete optimizer and the stable flow baseline. Meanwhile, the network's predictions under the central flow generally match those of the discrete optimizer, whereas the stable flow takes a different path through function space. The central flow can also accurately predict the time-averaged train loss curve and squared gradient norm curve, as well as the covariance of the discrete optimizer's oscillations around the central flow.

That said, the central flow approximation can break down in certain circumstances, which we now describe. An interesting direction for future work would be to rigorously characterize the conditions under which the central flow does or does not approximate the real optimizer trajectory.

Sufficiently large learning rates For all three optimizers studied in this paper, we reliably observe that as the learning rate hyperparameter is made increasingly large, the real optimization trajectory tends to deviate more from

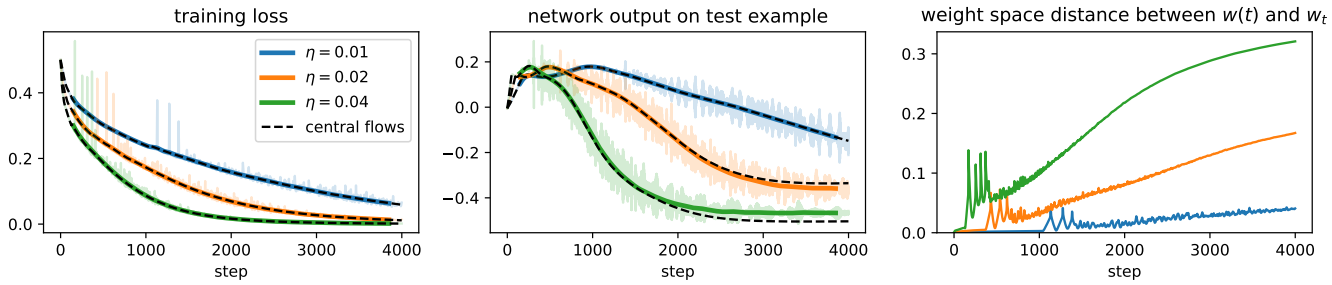


Figure 23: **Central flow approximation is less accurate at larger learning rates**. We run both gradient descent and its central flow at three learning rates (colors). The larger the learning rate, the faster the growth in the accumulated approximation error (right). Indeed, at larger learning rates, the network's output on an arbitrary test example can be visually seen to be slightly different between the central flow and gradient descent (middle). Nevertheless, the central flow approximation is still accurate enough here to accurately capture the train loss curves (left). *Details*: a CNN is trained on CIFAR-10 using MSE loss.

the central flow, as illustrated in Figure 23. (As an extreme example, even if the optimizer is initialized stably, very large learning rates sometimes cause the real optimizer to explosively diverge in the middle of training, whereas this never happens under the central flow.) We do not know whether some corrected version of the central flow would be more successful at capturing the real optimization trajectory in these scenarios, or whether the real trajectory is simply too chaotic to be captured by any flow.

Higher-order terms Sometimes, the local curvature is not well-modeled by the cubic Taylor approximation, as is assumed by our theory. This leads the central flow to mispredict $\Sigma(t)$, causing error to accumulate over the long run. We elaborate on this failure mode in Section C.2.

Smoothness of architecture The smoothness of the architecture seems to affect the accuracy of the central flow approximation; non-smooth components such as ReLU or max pooling often cause the quality of the approximation to break down. For example, in Figure 38, we show that as a network's activation function is interpolated from GeLU (smooth) to ReLU (non-smooth), the accuracy of the central flow approximation degrades, both in weight space and function space. Note that it is not clear how best to precisely quantify "smoothness" in this context.

Large spikes When the EOS dynamics lead to extremely large spikes (e.g. in the gradient norm), we have found such spikes can cause the real trajectory to deviate from the central flow, as illustrated in Figure 39. This may be related to the large learning rate issue described above.

Interactions with loss criterion We have empirically found that the higher-order terms issue and the large spike issue are more common with cross-entropy loss than with mean squared error loss (although we do not have a satisfactory explanation for these observations). Consequently, the central flow approximation is often more accurate under the MSE loss than the cross-entropy loss.

Overall, despite these limitations, we argue that the central flow describes the behavior of the corresponding optimizer "to a first approximation." Even in cases where the central flow is not a perfect quantitative match to the discrete trajectory, it may still capture the important qualitative trends.

7 Discussion

7.1 Modeling decisions

Deterministic setting Our analysis is restricted to the simple setting of deterministic (i.e. full-batch) training, whereas practical deep learning generally involves minibatch training. We study the full-batch setting because, as the simplest special case, understanding full-batch training is a necessary prerequisite for understanding minibatch training. However, we also believe that understanding full-batch training might suffice for some practical purposes, such as designing optimizers. For example, Kunstner et al. (2024) showed that the advantage of adaptive methods over SGD grows larger with larger batch sizes, suggesting that the relevant algorithmic principles can be best understood in the deterministic setting.

An interesting direction for future research is to try to extend our central flows methodology to the stochastic setting. Like deterministic optimizers, stochastic optimizers are known to implicitly regularize the curvature along their trajectories, and in fact this effect is *stronger* in the stochastic setting (Keskar et al., 2017; Jastrzębski et al., 2020, 2021; Lee and Jang, 2023; Andreyev and Beneventano, 2024). However, extending the central flows methodology to the stochastic setting may be nontrivial; due to the randomness, it is not clear whether there exists a deterministic differential equation around which SGD oscillates. An interesting question is whether there exists a differential equation that can predict derived metrics such as network predictions or training loss curves, even if it cannot model the weight-space trajectory of SGD. Finally, while our analysis in this paper sheds light on adaptive optimizers in the deterministic setting, these optimizers could exhibit substantially different behavior in the stochastic setting.

Black-box model of the loss Our analysis treats the loss function as a black box, and never uses that the optimization problem at hand involves training a neural network. The advantage of this approach is its generality: we expect our analysis to apply to generic deep learning architectures and learning problems, including those that do not yet exist. The disadvantage, however, is that the predictions made by our theory are at the abstraction level of the *loss landscape*, and would need to be further translated in order to make concrete claims about the network architecture or learning problem. For example, our theory tells us that the learning rate hyperparameter modulates the strength of an implicit sharpness penalty, but does not tell us how this sharpness penalty affects learning. Nor does our theory shed light on how different layers of the neural network are mechanistically implicated in progressive sharpening or sharpness reduction.

On the one hand, the loss landscape level of abstraction is in some sense "natural" — the overall path that optimizers follow really does intrinsically depend on the (effective) sharpness. But on the other hand, understanding many important aspects of optimization in deep learning will likely require cracking open the black box a bit more.

7.2 Takeaways from our analysis

The unreasonable effectiveness of time-averaging Prior works on EOS show that it is challenging to analyze the oscillatory EOS dynamics in fine-grained detail. Our work shows that, perhaps surprisingly, simple heuristics allow us to analyze the *time-averaged* trajectory with excellent numerical accuracy. Interestingly, the success of this time-averaging approach seems to imply that the oscillations only affect the macroscopic trajectory in an ergodic sense, i.e. via their *covariance* rather than via their fine-grained details. An promising direction for future work is to identify realistic conditions under which our heuristic time-averaging arguments can be made rigorous.

Necessity of third-order Taylor expansions While optimization theory generally relies on second-order Taylor expansions of the loss, Damian et al. (2023) showed that a *third-order* Taylor expansion is necessary for understanding the convergence of gradient descent; such a Taylor expansion reveals that oscillations implicitly trigger curvature reduction, a form of negative feedback which stabilizes optimization. In this work, we have shown that a third-order Taylor expansion is similarly necessary for understanding the *acceleration via mechanism* which underlies the success of adaptive optimizers. Thus, our work further underscores the necessity of a third-order Taylor expansion when analyzing optimization in deep learning.

Oscillatory first-order methods are implicitly second-order methods Over the last decade, optimizers that explicitly use Hessian information have failed to outperform first-order adaptive optimizers which employ only gradient information. Our work demystifies this observation. We have shown that that when first-order optimizers oscillate, they implicitly leverage second order information. Thus, even though RMSProp is a first-order optimizer, it implicitly employs a second-order preconditioning strategy, detailed in Section 5.3.1. Further, this preconditioning strategy is efficient, requiring no more gradient queries than gradient descent does. An exciting direction for future work is to *intentionally* design first-order adaptive methods with such implicit preconditioners in mind.

Adapting to curvature is not enough Traditional optimization theory views the curvature of the loss as a pre-existing feature of the optimization problem, and views the job of an optimizer as *adapting* to this pre-existing curvature. We have shown that the adaptive optimizers that we study do not merely passively adapt to the curvature; they also actively *shape* the curvature along their trajectory, by steering away from high-curvature regions where they would need to take small steps. Further, we have shown that this effect is crucial for their optimization efficacy. Thus, our work suggests that *acceleration via regularization* is a vital design principle for adaptive optimizers.

8 Limitations

Before concluding, we review the limitations of our approach. First, our analysis is currently nonrigorous and is ultimately supported by experiments rather than mathematical proof. Second, our approach is currently restricted to the setting of deterministic (i.e. full-batch) training and does not yet apply to stochastic training or to momentum. Third, it is quite computationally expensive to discretize the central flows, and doing so is only feasible for small networks on small datasets. Fourth, the central flows tend to degrade in accuracy as the learning rate hyperparameter is made increasingly large, and are generally more accurate for MSE loss than cross-entropy loss. Fifth, the central flows may not work well on networks with ReLU or other non-smooth architectural components. Despite these limitations, we believe that central flows are the best available tool for reasoning about the dynamics of optimization in deep learning.

9 Conclusion

In this paper, we have developed a methodology for analyzing deep learning optimizers. To analyze an optimization algorithm, we derive a *central flow* which models the optimizer's time-averaged trajectory, rendering explicit what was previously implicit in the optimizer's oscillatory dynamics. We have empirically shown that these central flows can accurately predict long-term optimization trajectories of neural networks, and by interpreting these flows we have obtained new insights about optimizers' behavior.

These advances are made possible by the fact that we adopt different goals from most works in optimization. Rather than try to characterize global convergence rates, we set ourselves the more modest goal of characterizing the *local* optimization dynamics throughout training. The local dynamics are important, they are more interesting than may have been assumed (even vanilla gradient descent gives rise to rich, complex dynamics), and they are empirically consistent across different deep learning settings, which suggests that a general theory is feasible. We believe that similar analyses can be fruitfully conducted for other optimizers, and we hope to inspire work in that direction.

Acknowledgements

AD acknowledges support from an NSF Graduate Research Fellowship and a Jane Street Graduate Research Fellowship. JDL acknowledges support of Open Philanthropy, NSF IIS 2107304, NSF CCF 2212262, NSF CAREER Award 2144994, and NSF CCF 2019844. AT acknowledges support from National Science Foundation grants IIS1705121, IIS1838017, IIS2046613, IIS2112471, and funding from Meta, Morgan Stanley, Amazon, Google, and Scribe. JC would like to thank Jim and Marilyn Simons for their support of basic research via the Flatiron Institute. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of these funding agencies.

The authors are grateful for feedback from Nikhil Ghosh, Yiding Jiang, Sam Sokota, and Zhili Feng.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

A. Agarwala, F. Pedregosa, and J. Pennington. Second-order regression models exhibit progressive sharpening to the edge of stability. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.

K. Ahn, S. Bubeck, S. Chewi, Y. T. Lee, F. Suarez, and Y. Zhang. Learning threshold neurons via edge of stability. *Advances in Neural Information Processing Systems*, 36, 2024.

A. Andreyev and P. Beneventano. Edge of stochastic stability: Revisiting the edge of stability for sgd. *arXiv preprint arXiv:2412.20553*, 2024.

S. Arora, Z. Li, and A. Panigrahi. Understanding gradient descent on the edge of stability in deep learning. In *International Conference on Machine Learning*, pages 948–1024. PMLR, 2022.

Z. Bai, Z. Zhou, J. Zhao, X. Li, Z. Li, F. Xiong, H. Yang, Y. Zhang, and Z.-Q. J. Xu. Adaptive preconditioners trigger loss spikes in adam. *arXiv preprint arXiv:2506.04805*, 2025.

D. Barrett and B. Dherin. Implicit gradient regularization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=3q5IqUrkcF`.

L. Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

G. Blanc, N. Gupta, G. Valiant, and P. Valiant. Implicit regularization for deep neural networks driven by an ornstein-uhlenbeck like process. In *Annual Conference Computational Learning Theory*, 2019. URL `https://api.semanticscholar.org/CorpusID:125944013`.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

S. Burer and R. D. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical programming*, 103(3):427–444, 2005.

M. D. Cattaneo, J. M. Klusowski, and B. Shigida. On the implicit bias of Adam. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

C. Chen, L. Shen, F. Zou, and W. Liu. Towards practical adam: Non-convexity, convergence theory, and mini-batch acceleration. *Journal of Machine Learning Research*, 23(229):1–47, 2022.

L. Chen and J. Bruna. Beyond the edge of stability via two-step gradient updates. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

X. Chen, S. Liu, R. Sun, and M. Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. In *International Conference on Learning Representations*, 2019a. URL `https://openreview.net/forum?id=H1x-x309tm`.

Z. Chen, Z. Yuan, J. Yi, B. Zhou, E. Chen, and T. Yang. Universal stagewise learning for non-convex problems with convergence on averaged solutions. In *International Conference on Learning Representations*, 2019b. URL `https://openreview.net/forum?id=Syx5V2CcFm`.

J. Cohen, S. Kaur, Y. Li, J. Z. Kolter, and A. Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jh-rTtvkGeM.

J. M. Cohen, B. Ghorbani, S. Krishnan, N. Agarwal, S. Medapati, M. Badura, D. Suo, D. Cardoze, Z. Nado, G. E. Dahl, and J. Gilmer. Adaptive gradient methods at the edge of stability. *arXiv preprint arXiv:2207.14484*, 2022.

E. M. Compagnoni, L. Biggio, A. Orvieto, F. N. Proske, H. Kersting, and A. Lucchi. An sde for modeling sam: Theory and insights. In *International Conference on Machine Learning*, pages 25209–25253. PMLR, 2023.

E. M. Compagnoni, T. Liu, R. Islamov, F. N. Proske, A. Orvieto, and A. Lucchi. Adaptive methods through the lens of SDEs: Theoretical insights on the role of noise. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ww3CLRhF1v.

B. Cornet. Existence of slow solutions for a class of differential inclusions. *Journal of Mathematical Analysis and Applications*, 96(1):130–147, Oct. 1983. ISSN 0022-247X. doi:10.1016/0022-247X(83)90032-X.

M. Crawshaw, M. Liu, F. Orabona, W. Zhang, and Z. Zhuang. Robustness to unbounded smoothness of generalized signsgd. *Advances in Neural Information Processing Systems*, 35:9955–9968, 2022.

A. Damian, T. Ma, and J. D. Lee. Label noise sgd provably prefers flat global minimizers. *Advances in Neural Information Processing Systems*, 34:27449–27461, 2021.

A. Damian, E. Nichani, and J. D. Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=nhKHA59gXz.

Y. Dauphin, A. Agarwala, and H. Mobahi. How hessian structure explains mysteries in sharpness regularization. *Advances in Neural Information Processing Systems*, 37, 2024.

M. K. de Carli Silva and L. Tunçel. Strict complementarity in maxcut sdp, 2018. URL https://arxiv.org/abs/1806.01173.

A. Défossez, L. Bottou, F. Bach, and N. Usunier. A simple convergence proof of adam and adagrad. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=ZPQhzTSWA7.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

O. Elkabetz and N. Cohen. Continuous vs. discrete optimization of deep neural networks. *Advances in Neural Information Processing Systems*, 34:4947–4960, 2021.

M. Even, S. Pesme, S. Gunasekar, and N. Flammarion. (s)gd over diagonal linear networks: implicit bias, large stepsizes and edge of stability. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, 2024.

J. Geiping, M. Goldblum, P. Pope, M. Moeller, and T. Goldstein. Stochastic training is not necessary for generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZBESeIUB5k.

A. Ghosh, H. Lyu, X. Zhang, and R. Wang. Implicit regularization in heavy-ball momentum accelerated stochastic gradient descent. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ZzdBhtEH9yB.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer-Verlag, 1983.

Z. Guo, Y. Xu, W. Yin, R. Jin, and T. Yang. A novel convergence analysis for algorithms of the adam family. *arXiv preprint arXiv:2112.03459*, 2021.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

Y. Hong and J. Lin. On convergence of adam for stochastic optimization under relaxed assumptions. *Advances in Neural Information Processing Systems*, 37:10827–10877, 2024.

F. Hübler, J. Yang, X. Li, and N. He. Parameter-agnostic optimization under relaxed smoothness. In *International Conference on Artificial Intelligence and Statistics*, pages 4861–4869. PMLR, 2024.

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

S. Jastrzębski, Z. Kenton, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. On the relation between the sharpest directions of DNN loss and the SGD step length. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkgEaj05t7.

S. Jastrzębski, M. Szymczak, S. Fort, D. Arpit, J. Tabor, K. Cho*, and K. Geras*. The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1g87C4KwB.

S. Jastrzębski, D. Arpit, O. Astrand, G. B. Kerg, H. Wang, C. Xiong, R. Socher, K. Cho, and K. J. Geras. Catastrophic fisher explosion: Early phase fisher matrix impacts generalization. In *International Conference on Machine Learning*, pages 4772–4784. PMLR, 2021.

A. Karpathy. mingpt - demo.ipynb. https://github.com/karpathy/minGPT/blob/master/demo.ipynb, 2020.

N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

A. Khaled, K. Mishchenko, and C. Jin. Dowg unleashed: An efficient universal parameter-free gradient descent method. *Advances in Neural Information Processing Systems*, 36:6748–6769, 2023.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *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/1412.6980.

A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

I. Kreisler, M. S. Nacson, D. Soudry, and Y. Carmon. Gradient descent monotonically decreases the sharpness of gradient flow solutions in scalar networks and beyond. In *International Conference on Machine Learning*, pages 17684–17744. PMLR, 2023.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

F. Kunstner, P. Hennig, and L. Balles. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in Neural Information Processing Systems*, 32, 2019.

F. Kunstner, R. Yadav, A. Milligan, M. Schmidt, and A. Bietti. Heavy-tailed class imbalance and why adam outperforms gradient descent on language models. In *Neural Information Processing Systems*, 2024.

S. Lee and C. Jang. A new characterization of the edge of stability based on a sharpness measure aware of batch gradient distribution. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=bH-kCY6LdKg.

A. Lewkowycz, Y. Bahri, E. Dyer, J. Sohl-Dickstein, and G. Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

H. Li and Z. Lin. On the $\mathrm{O}(\sqrt{d}/t^{1/4})$ convergence rate of RMSProp and its momentum extension measured by l_1 norm: Better dependence on the dimension. *arXiv preprint arXiv:2402.00389*, 2024.

H. Li, A. Rakhlin, and A. Jadbabaie. Convergence of adam under relaxed assumptions. *Advances in Neural Information Processing Systems*, 36, 2024.

Q. Li, C. Tai, and W. E. Stochastic modified equations and adaptive stochastic gradient algorithms. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

X. Li, Z.-Q. J. Xu, and Z. Zhang. Loss spike in training neural networks. *arXiv preprint arXiv:2305.12133*, 2023.

Y. Li, C. Wei, and T. Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

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

Z. Li, T. Wang, and S. Arora. What happens after SGD reaches zero loss? –a mathematical framework. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=siCt4xZn5Ve.

Z. Li, Z. Wang, and J. Li. Analyzing sharpness along gd trajectory: Progressive sharpening and edge of stability. In *Neural Information Processing Systems*, 2022b.

Y. Liu, Z. Liu, and J. Gore. Focus: First order concentrated updating scheme. *arXiv preprint arXiv:2501.12243*, 2025a.

Z. Liu, Y. Liu, J. Gore, and M. Tegmark. Neural thermodynamic laws for large language model training, 2025b. URL https://arxiv.org/abs/2505.10559.

LucidRains. Vision transformer - pytorch. https://github.com/lucidrains/vit-pytorch, 2024.

K. Lyu, Z. Li, and S. Arora. Understanding the generalization benefit of normalization layers: Sharpness reduction. *Advances in Neural Information Processing Systems*, 35:34689–34708, 2022.

C. Ma, L. Wu, and E. Weinan. A qualitative study of the dynamic behavior for adaptive gradient algorithms. In *Mathematical and Scientific Machine Learning*, pages 671–692. PMLR, 2022.

S. Malladi, K. Lyu, A. Panigrahi, and S. Arora. On the sdes and scaling rules for adaptive gradient algorithms. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 7697–7711. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/32ac710102f0620d0f28d5d05a44fe08-Paper-Conference.pdf.

J. Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21 (146):1–76, 2020. URL http://jmlr.org/papers/v21/17-678.html.

A. Mishkin, A. Khaled, Y. Wang, A. Defazio, and R. M. Gower. Directional smoothness and gradient methods: Convergence and adaptivity. *Advances in Neural Information Processing Systems*, 2024.

D. Morales-Brotons, T. Vogels, and H. Hendrikx. Exponential moving average of weights in deep learning: Dynamics and benefits. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=2M9CUnYnBA.

J.-J. Moreau. Décomposition orthogonale d'un espace hilbertien selon deux cones mutuellement polaires. *Comptes Rendus de l'Académie des Sciences*, 255:238–240, 1962.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

F. Orabona. Neural networks (maybe) evolved to make adam the best optimizer. https://parameterfree.com/2020/12/06/neural-network-maybe-evolved-to-make-adam-the-best-optimizer/, 2020. Accessed: October 17, 2024.

S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

M. Rosca, Y. Wu, C. Qin, and B. Dherin. On a continuous time model of gradient descent dynamics and instability in deep learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=EYrRzKPinA.

E. Rosenfeld and A. Risteski. Outliers with opposing signals have an outsized effect on neural network optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=kIZ3S3tel6.

V. Roulet, A. Agarwala, J.-B. Grill, G. M. Swirszcz, M. Blondel, and F. Pedregosa. Stepping on the edge: Curvature aware learning rate tuners. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=SEflLHIhhJ.

M. Sandler, A. Zhmoginov, M. Vladymyrov, and N. Miller. Training trajectories, mini-batch losses and the curious role of the learning rate, 2023. URL https://arxiv.org/abs/2301.02312.

N. Shi, D. Li, M. Hong, and R. Sun. RMSprop converges with proper hyper-parameter. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=3UDSdyIcBDA.

S. L. Smith, B. Dherin, D. Barrett, and S. De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rq_Qr0c1Hyo.

M. Song and C. Yun. Trajectory alignment: Understanding the edge of stability phenomenon via bifurcation theory. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=PnJaA0A8Lr.

D. E. Stewart. *Dynamics with Inequalities*. Society for Industrial and Applied Mathematics, 2011. doi:10.1137/1.9781611970715. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611970715.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

A. Torres-Leguet. mamba.py. https://github.com/alxndrTL/mamba.py, 2024.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

B. Wang, J. Fu, H. Zhang, N. Zheng, and W. Chen. Closing the gap between the upper bound and lower bound of adam's iteration complexity. *Advances in Neural Information Processing Systems*, 36, 2024a.

B. Wang, H. Zhang, Q. Meng, R. Sun, Z.-M. Ma, and W. Chen. On the convergence of adam under non-uniform smoothness: Separability from sgdm and beyond. *arXiv preprint arXiv:2403.15146*, 2024b.

B. Wang, Y. Zhang, H. Zhang, Q. Meng, R. Sun, Z.-M. Ma, T.-Y. Liu, Z.-Q. Luo, and W. Chen. Provable adaptivity of adam under non-uniform smoothness. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2024c. ISBN 9798400704901.

M. Wang, J. Wang, H. He, Z. Wang, G. Huang, F. Xiong, Z. li, W. E, and L. Wu. Improving generalization and convergence by enhancing implicit regularization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024d. URL https://openreview.net/forum?id=cjM2bhLOiC.

K. Wen, Z. Li, J. S. Wang, D. L. W. Hall, P. Liang, and T. Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape view. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m51BgoqvbP.

J. Wu, V. Braverman, and J. D. Lee. Implicit bias of gradient descent for logistic regression at the edge of stability. *Advances in Neural Information Processing Systems*, 36, 2024.

L. Wu, C. Ma, and W. E. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/6651526b6fb8f29a00507de6a49ce30f-Paper.pdf.

Y. Wu and K. He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

C. Xing, D. Arpit, C. Tsirigotis, and Y. Bengio. A walk with sgd. *arXiv preprint arXiv:1802.08770*, 2018.

J. Yang, X. Li, I. Fatkhullin, and N. He. Two sides of one coin: the limits of untuned sgd and the power of adaptive methods. *Advances in Neural Information Processing Systems*, 36, 2024.

M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar. Adaptive methods for nonconvex optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Q. Zhang, Y. Zhou, and S. Zou. Convergence guarantees for RMSProp and adam in generalized-smooth non-convex optimization with affine noise variance. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=QIzRdjIWnS.

Y. Zhang, C. Chen, N. Shi, R. Sun, and Z.-Q. Luo. Adam can converge without any modification on update rules. *Advances in Neural Information Processing Systems*, 35:28386–28399, 2022.

X. Zhu, Z. Wang, X. Wang, M. Zhou, and R. Ge. Understanding edge-of-stability training dynamics with a minimalist example. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=p7EagBsMAEO.

F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11127–11135, 2019.