On a continuous time model of gradient descent dynamics and instability in deep learning

Anonymous authors
Paper under double-blind review

Abstract

The recipe behind the success of deep learning has been the combination of neural networks and gradient-based optimization. Understanding the behavior of gradient descent however, and particularly its instability, has lagged behind its empirical success. To add to the theoretical tools available to study gradient descent we propose *the principal flow* (PF), a continuous time flow that approximates gradient descent dynamics. To our knowledge, the PF is the only continuous flow that captures the divergent and oscillatory behaviors of gradient descent, including escaping local minima and saddle points. Through its dependence on the eigendecomposition of the Hessian the PF sheds light on the recently observed edge of stability phenomena in deep learning. Using our new understanding of instability we propose a learning rate adaptation method which enables us to control the trade-off between training stability and test set evaluation performance.

1 Introduction

Our goal is to use continuous time models to understand the behavior of gradient descent. Using continuous dynamics to understand discrete time systems opens up tools from dynamical systems such as stability analysis, and has a long history in optimization and machine learning (Glendinning, 1994; Saxe et al., 2013; Nagarajan and Kolter, 2017; Lampinen and Ganguli, 2018; Arora et al., 2018; Advani et al., 2020; Elkabetz and Cohen, 2021; Vardi and Shamir, 2021; Franca et al., 2020; Barrett and Dherin, 2021; Smith et al., 2021). Most theoretical analysis of gradient descent using continuous time systems uses the negative gradient flow, but this has well known limitations such as not being able to explain any behavior contingent on the learning rate. To mitigate these limitations we find a new continuous time flow which reveals important new roles of the Hessian in gradient descent training. To do so, we use backward error analysis (BEA), a method with a long history in the numerical integration community (Hairer et al., 2006) that has only recently been used in the deep learning context (Barrett and Dherin, 2021; Smith et al., 2021).

We find that the proposed flow sheds new light on gradient descent stability, including but not limited to divergent and oscillatory behavior around a fixed point. Instability — areas of training where the loss consistently increases — and edge of stability behaviors (Cohen et al., 2021) —areas of training where the loss does not behave monotonically but decreases over long time periods — are pervasive in deep learning and occur for all learning rates and architectures Cohen et al. (2021); Gur-Ari et al. (2018); Gilmer et al. (2021); Lewkowycz et al. (2020). We use our novel insights to understand and mitigate these instabilities.

The structure of the presented work is as follows:

- We discuss the advantages of a continuous time approach in Section 2, where we also highlight the limitations of existing continuous-time flows.
- We introduce **the principal flow** (the PF), a flow in complex space defined by the eigendecomposition of the Hessian (Section 3). To our knowledge the PF is the first continuous time flow that captures that gradient descent can diverge around local minima and saddle points. We show that using a complex flow is crucial in understanding instabilities in gradient descent.
- We show the PF is better than existing flows at modelling neural network training dynamics in Section 4. In Section 5 we use the PF to shed new light on edge of stability behaviors in deep

learning by connecting changes in the loss and Hessian eigenvalues with core quantities exposed by the PF and the NN landscape explored through the continuous-time behaviour of gradient flows.

- Through a continuous time perspective we demonstrate empirically how to control the trade-off between stability in performance in deep learning in Section 6. We do so using DAL (Drift Adjusted Learning rate), an approach to setting the learning rate dynamically based on insights on instability derived from the PF.
- We end by showcasing the potential of integrating our continuous time approach with other optimization schemes and highlighting how the PF can be used as a tool for existing continuous time analyses in Section 7.

Notation: We denote as E the loss function, θ the parameter vector of dimension D, $\nabla^2_\theta E$ the loss Hessian and λ_i and u_i the i'th largest eigenvalue and eigenvector respectively of $\nabla^2_\theta E$. As a convention, since if u_i is an eigenvector of $\nabla^2_\theta E$ so is $-u_i$, we always use u_i such that $Re[\nabla_\theta E^T u_i] \geq 0$. In the context of a continuous time flow $\theta(h)$ refers to the solution of the flow at time h.

Experiments: A list of figures with details on how to reproduce each of them is provided in the Appendix.

2 Continuous time models of gradient descent

The aim of this work is to understand the dynamics of gradient descent updates with learning rate h

$$\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) \tag{1}$$

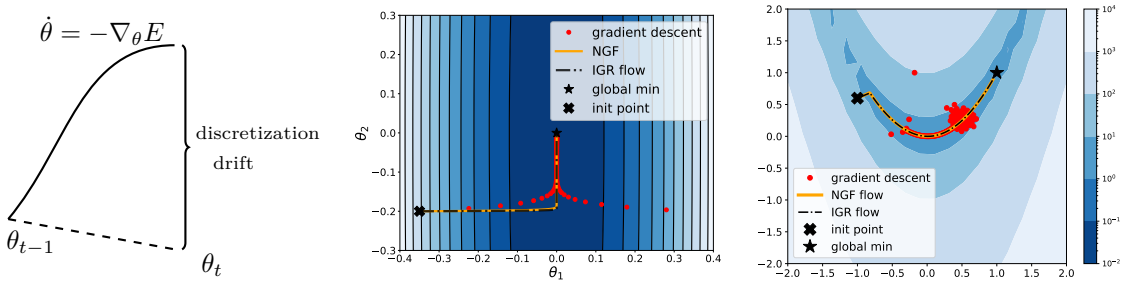
from the perspective of continuous dynamics. When using continuous time dynamics to understand gradient descent it is most common to use *the negative gradient flow* (NGF)

$$\dot{\theta} = -\nabla_\theta E \tag{2}$$

Gradient descent can be obtained from the NGF through Euler numerical integration, with an error of $\mathcal{O}(h^2)$ after one gradient descent step. Studying gradient descent and its behavior around equilibria and beyond has thus taken two main approaches: directly studying the discrete updates of Eq 1 (Bartlett et al., 2018a;b; Mescheder et al., 2017; Gunasekar et al., 2018; Du et al., 2019; Allen-Zhu et al., 2019; Du and Hu, 2019; Ziyin et al., 2021a; Liu et al., 2021), or the continuous time NGF of Eq 2 (Glendinning, 1994; Saxe et al., 2013; Nagarajan and Kolter, 2017; Lampinen and Ganguli, 2018; Arora et al., 2018; Advani et al., 2020; Elkabetz and Cohen, 2021; Vardi and Shamir, 2021; Franca et al., 2020; Balduzzi et al., 2018). The appeal of continuous time systems lies in their connection with dynamical systems and the plethora tools that thus become available, such as stability analysis, or the simplicity by which conserved quantities can be obtained (Du et al., 2018; Franca et al., 2020) and analogies that can be constructed through similarities with physical systems(Franca et al., 2020). Because of the availability of tools for the analysis of continuous time systems, it has been previously noted that discrete time approaches are often more challenging and proofs are inspired from continuous time ones (May, 1976; Elkabetz and Cohen, 2021). We use an example to showcase the ease of continuous time analyses: when following the NGF the loss E decreases since $\frac{dE}{dt} = \frac{dE}{d\theta}^T \frac{d\theta}{dt} = -||\nabla_\theta E||^2$. Showing that and *when* following the discrete time gradient descent update in Eq 1 is more challenging and requires adapting the analysis on the form of the loss function E. Classical convergence guarantees associated with other optimization approaches such as natural gradient are also derived in continuous time (Amari, 1998; Ollivier, 2015a;b). By analyzing the properties of continuous time systems one can also determine whether optimizers should more closely follow the underlying continuous time flow (Song et al., 2018; Qin et al., 2020), what regularizers should be constructed to ensure convergence or stability (Nagarajan and Kolter, 2017; Balduzzi et al., 2018; Rosca et al., 2021), construct converge guarantees in functional space for infinitely wide networks (Jacot et al., 2018; Lee et al., 2019).

2.1 Limitations of existing continuous time flows

The well-known discrepancy between Euler integration and the NGF, often called *discretization error* or *discretization drift* (Figure 1(a)) leads to certain limitations when using the NGF to describe gradient



(a) Discretisation drift. (b) 2D convex case. (c) Banana function.

Figure 1: **Motivation**. Using continuous time flows to understand gradient descent is limited by the gap between the discrete and continuous dynamics. In the case of the negative gradient flow, we call this gap *discretization drift*. Other flows have been introduced to capture part of the drift, but they also fail to capture the oscillatory or unstable behavior of gradient descent.

descent, namely: the NGF cannot explain divergence around a local minima for high learning rates or convergence to flat minima as often seen in the training of neural networks. Critically, since the NGF does not depend on the learning rate, it cannot explain any learning rate dependent behavior.

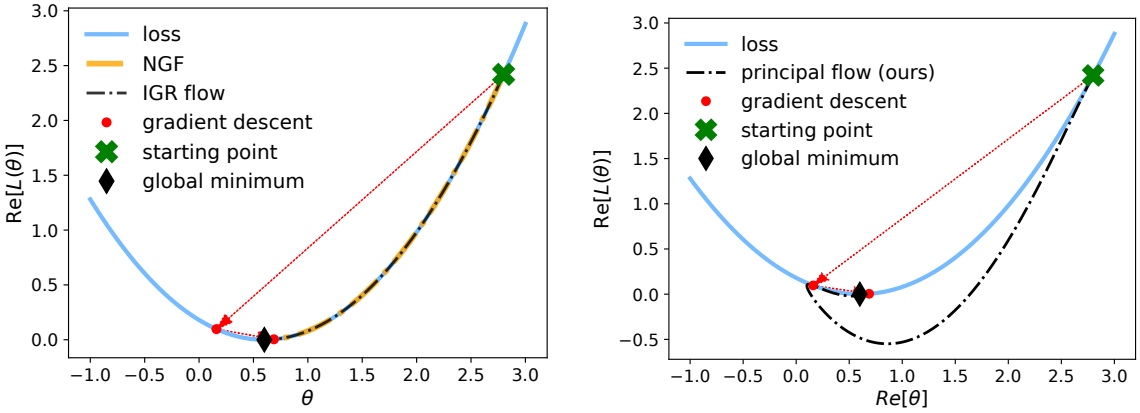
The appeal of continuous time methods together with the limitations of the NGF have inspired the machine learning community to look for other continuous time systems which may better approximate the gradient descent trajectory. One approach to constructing continuous time flows approximating gradient descent that takes into account the learning rate is backward error analysis (BEA). Using this approach, Barrett and Dherin (2021) introduce the Implicit Gradient Regularization flow (IGR flow):

$$\dot{\theta} = -\nabla_\theta E - \frac{h}{2}\nabla_\theta^2 E \nabla_\theta E \tag{3}$$

which tracks the dynamics of the gradient descent step $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ with an error of $\mathcal{O}(h^3)$, thus reducing the order of the error compared to the NGF. Unlike the NGF flow, the IGR flow depends on the learning rate h. This dependence explains certain properties of gradient descent, such as avoiding paths with high gradient norm towards a local minima; the authors connect this behavior to convergence to flat minima.

Like the NGF flow however, the IGR flow does not explain the instabilities of gradient descent, as we illustrate in Figure 1. Indeed, Barrett and Dherin (2021) (their Remark 3.4) show that performing stability analysis around local minima using the IGR flow does not lead to qualitatively different conclusions from those using the NGF: both NGF and the IGR flow predict gradient descent to be always locally attractive around a local minima, contradicting the empirically observed behavior of gradient descent (proofs in Section A.4.1). To understand why both the NFG and the IGR flow cannot capture oscillations and divergence around a local minima, we note that stationary points $\nabla_\theta E = 0$ are fixed points for both flows. We visualize an example in Figure 2(a): since to go from the initial point to the gradient descent iterates requires passing through the local minimum, both flows would stop at the local minimum. In the case of neural networks we show in Figure 31 in the Appendix that while the IGR flow is better than the NGF at describing gradient descent, a substantial gap remains.

The lack of ability of existing continuous time flows to model instabilities empirically observed in gradient descent such as those shown in Figure 1 has been used as a motivation to use discrete-time methods instead (Yaida, 2018; Liu et al., 2021). The goal of our work is to overcome this issue by introducing a novel continuous-time flow which captures instabilities observed in gradient descent. To do so, we follow the footsteps of Barrett and Dherin (2021) and use Backward Error Analysis. By using a continuous time flow we can leverage the tools and advantages of continuous-time methods discussed earlier in this section; by incorporating discretization drift into our model of gradient descent we can increase their applicability to explain unstable training behavior. Indeed, we show in Figure 2(b) that the flow we propose captures the training instabilities; a key reason why is that, unlike existing flows, it operates in complex space. In Section



(a) Real flows.

(b) Complex flow.

Figure 2: **Complex flows capture oscillations and divergence around local minima**. In the real space, the trajectory going from the starting point to the second gradient descent iterate goes through the global minima, and real flows stop there. In complex space however, that need not be the case.

3 we show the importance of operating in complex space in order to understand oscillatory and instability behaviors of gradient descent.

2.2 Backward error analysis

Backward error analysis (BEA) is a tool in numerical analysis developed to understand the discretization error of numerical integrators. We now present an overview of how to use it in the context of gradient descent; for a general overview see Hairer et al. (2006). BEA provides a modified vector field:

$$\tilde{f}_n(\theta) = -\nabla_\theta E + hf_1(\theta) + \cdots + h^n f_n(\theta), \tag{4}$$

by finding functions f_1, ... f_n such that the solution of the modified ODE at order n, that is,

$$\dot{\tilde{\theta}} = -\nabla_\theta E + hf_1(\theta) + \cdots + h^n f_n(\theta) \tag{5}$$

follows the discrete dynamics of the gradient descent update with an error $\|\theta_t - \tilde{\theta}(h)\|$ of order $\mathcal{O}(h^{n+2})$, where $\tilde{\theta}(h)$ is the solution of the modified equation truncated at order n at time h, with $\tilde{\theta}(0) = \theta_{t-1}$. The full modified vector field with all orders ($n \to \infty$)

$$\tilde{f}(\theta) = -\nabla_\theta E + hf_1(\theta) + \cdots + h^n f_n(\theta) + \cdots, \tag{6}$$

is usually divergent and only forms an asymptotic expansion. What BEA provides is the Taylor expansion in h of an unknown h-dependent vector field $f_h(\theta)$ developed at $h = 0$:

$$\tilde{f}(\theta) = \text{Taylor}_{h=0} f_h(\theta). \tag{7}$$

Thus a strategy for finding f_h is to find a series of the form in Eq 6 via BEA and then find the function f_h such that its Taylor expansion in h at 0 results in the found series. Using this approach we can find the flow $\dot{\tilde{\theta}} = f_h(\tilde{\theta})$ which exactly describes the gradient descent step $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$.

While flows obtained using BEA are constructed to approximate one gradient descent step, the same flows can be used over multiple gradient descent steps as shown in Section A.8 in the Appendix.

BEA proofs. The general structure of BEA proofs is as follows: start with a Taylor expansion in h of the modified flow in Equation 5; write each term in the Taylor expansion as a function of $\nabla_\theta E$ and the desired f_i (this often requires applying the chain rule repeatedly); group together terms of the same order in h in

the expansion; and identify f_i such that all terms of $\mathcal{O}(h^p)$ are 0 for $p \geq 2$, as is the case in the gradient descent update. A formal overview of BEA proofs can be found in Section A.1 in the Appendix.

We now exemplify how to use BEA to find the IGR flow (Eq 3) (Barrett and Dherin, 2021). Since we are only looking for the first correction term, we only need to find f_1. We perform a Taylor expansion to find the value of $\tilde{\theta}(h)$ up to order $\mathcal{O}(h^3)$ and then identify f_1 from that expression such that the error $\|\theta_t - \tilde{\theta}(h)\|$ is of order $\mathcal{O}(h^3)$. We have: $\tilde{\theta}(h) = \theta_{t-1} + h\tilde{\theta}^{(1)}(\theta_{t-1}) + \frac{h^2}{2}\tilde{\theta}^{(2)}(\theta_{t-1}) + \mathcal{O}(h^3)$. We know by the definition of the modified vector field (Eq 5) that $\tilde{\theta}^{(1)} = -\nabla_\theta E + hf_1(\tilde{\theta})$. We can then use the chain rule to obtain $\tilde{\theta}^{(2)} = \frac{-\nabla_\theta E + hf_1(\theta)}{dt} = \frac{-\nabla_\theta E}{dt} + \mathcal{O}(h) = \frac{-\nabla_\theta E}{d\theta}\frac{d\theta}{dt} + \mathcal{O}(h) = \nabla^2_\theta E \nabla_\theta E + \mathcal{O}(h)$. Thus $\tilde{\theta}(h) = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) + h^2 f_1(\theta_{t-1}) + \frac{h^2}{2}\nabla^2_\theta E(\theta_{t-1})\nabla_\theta E(\theta_{t-1}) + \mathcal{O}(h^3)$. We can then write $\theta_t - \tilde{\theta}(h) = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) - \Big(\theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) + hf_1(\theta_{t-1}) + \frac{h^2}{2}\nabla^2_\theta E(\theta_{t-1})\nabla_\theta E(\theta_{t-1}) + \mathcal{O}(h^3)\Big)$. After simplifying we obtain $\theta_t - \tilde{\theta}(h) = h^2 f_1(\theta_{t-1}) + \frac{h^2}{2}\nabla^2_\theta E(\theta_{t-1})\nabla_\theta E(\theta_{t-1}) + \mathcal{O}(h^3)$. For the error to be of order $\mathcal{O}(h^3)$ the terms of order $\mathcal{O}(h^2)$ have to be 0. This entails $f_1 = -\frac{1}{2}\nabla^2_\theta E \nabla_\theta E$ leading to Eq 3.

3 The principal flow

In the previous section we have seen how BEA can be used to define continuous time flows which capture the dynamics of gradient descent up to a certain order in learning rate. We have also explored the limitations of these flows, including the lack of ability to explain oscillations observed empirically when using gradient descent. To further expand our understanding of gradient descent via continuous time methods, we would like to get an intuition for the structure of higher order modified vector fields provided by BEA. We start with the following modified vector field, which we will call *the third order flow* (proof in Section A.2):

$$\dot{\theta} = -\nabla_\theta E - \frac{h}{2}\nabla^2_\theta E \nabla_\theta E - h^2\left(\frac{1}{3}(\nabla^2_\theta E)^2\nabla_\theta E + \frac{1}{12}\nabla_\theta E^T(\nabla^3_\theta E)\nabla_\theta E\right) \tag{8}$$

The third order flow tracks the dynamics of the gradient descent step $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ with an error of $\mathcal{O}(h^4)$, thus further reducing the order of the error compared to the IGR flow. Like the IGR flow and the NGF, the third order flow has the property that $\dot{\theta} = 0$ if $\nabla_\theta E = 0$ and thus will exhibit the same limitations observed in Figure 2. We can now spot a pattern: the correction term of order $\mathcal{O}(h^n)$ in the BEA modified flow describing gradient descent contains the term $(\nabla^2_\theta E)^n \nabla_\theta E$ and terms which contain higher order derivatives with respect to parameters, terms which we will denote as $\mathcal{C}(\nabla^3_\theta E)$.

Our approach. We will use the terms of the form $(\nabla^2_\theta E)^n \nabla_\theta E$ to construct a new continuous time flow. We will take a three-step approach. First, for an arbitrary order $\mathcal{O}(h^n)$ we will find the terms containing only first and second order derivatives in the modified vector field given by BEA and show they are of the form $(\nabla^2_\theta E)^n \nabla_\theta E$ (Theorem 3.1). Second, we will use all orders to create a series (Corollary 3.1). Third, we will use the series to find the modified flow given by BEA (Theorem 3.2). All proofs are provided in Section A of the Appendix.

Theorem 3.1 *The modified vector field with an error of order $\mathcal{O}(h^{n+2})$ to the gradient descent update $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ has the form:*

$$\dot{\theta} = \sum_{p=0}^{n} \frac{-1}{p+1} h^p (\nabla^2_\theta E)^p \nabla_\theta E + \mathcal{C}(\nabla^3_\theta E) \tag{9}$$

where $\mathcal{C}(\nabla^3_\theta E)$ denotes the family of functions which can be written as a sum of terms, each term containing a derivative of higher order than 3 with respect to parameters.

The result is proven by induction. The base cases for $n = 1, 2$ and 3 follow from the NGF, IGR and third order flows. For higher order terms, the proof uses induction to find the term in f_i depending on $\nabla^2_\theta E$ and $\nabla_\theta E$ only and follows the structure highlighted in Section 2.2, but Step 3 is modified to not account for terms in $\mathcal{C}(\nabla^3_\theta E)$. From the above, we can obtain the following corollary by using all orders n and the eigen decomposition of $\nabla^2_\theta E$:

Corollary 3.1 *The modified flow capturing gradient descent discrete updates exactly is of the form:*

$$\dot{\theta} = \sum_{p=0}^{\infty} \frac{-1}{p+1} h^p (\nabla_\theta^2 E)^p \nabla_\theta E + \mathcal{C}(\nabla_\theta^3 E) = \sum_{p=0}^{\infty} \frac{-1}{p+1} h^p \left(\sum_i^D \lambda_i^p u_i u_i^T \right) \nabla_\theta E + \mathcal{C}(\nabla_\theta^3 E) \tag{10}$$

$$= \sum_{i=1}^{D} \left(\sum_{p=0}^{\infty} \frac{-1}{p+1} h^p \lambda_i^p \right) (\nabla_\theta E^T u_i) u_i + \mathcal{C}(\nabla_\theta^3 E) \tag{11}$$

where λ_i and u_i are the respective eigenvalues and eigenvectors of the Hessian $\nabla_\theta^2 E$.

If $\lambda_0 > 1/h$ the BEA series above diverges. Generally BEA series are not convergent and approximate the discrete scheme only by truncation Hairer et al. (2006). When the series in Eq 11 diverges truncating it up to any order n however will result in a flow which will not be able to capture instabilities, even in the quadratic case. Such flows (including the IGR flow) will always predict the loss function will decrease for a quadratic loss where a minimum exists, since: $\frac{dE}{dt} = \nabla_\theta E^T \left(\sum_{p=0}^{n} \frac{-1}{p+1} h^p (\nabla_\theta^2 E)^p \nabla_\theta E \right) = -\sum_{p=0}^{n} \frac{1}{p+1} h^p \sum_{i=1}^{D} (\lambda_i^p)(\nabla_\theta E^T u_i)^2$ which is never positive for any quadratic loss where a minimum exists (i.e. when $\lambda_i \geq 0, \forall i$). The above also entails that the flows always predict convergence around a local minimum, which is not the case for gradient descent which can diverge for large learning rates.

To further track instabilities we can use the BEA series to formulate the following complex flow:

Definition 3.1 *We define the **principal flow** (PF) as*

$$\dot{\theta} = \sum_{i}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} (\nabla_\theta E^T u_i) u_i \tag{12}$$

We note that $\lim_{\lambda \to 0} \frac{\log(1-h\lambda)}{h\lambda} = -1$ and thus the PF is well defined when the Hessian $\nabla_\theta^2 E$ is not invertible. Unlike the NGF and the IGR flow, the modified vector field of the PF cannot be always written as the gradient of a loss function in $\mathbb{R}$.

Theorem 3.2 *The Taylor expansion in h at $h = 0$ of the PF vector field coincides with the series coming from the BEA of gradient descent (Eq 11).*

Proof: Using the Taylor expansion $\text{Taylor}_{z=0} \frac{\log(1-z)}{z} = \sum_{p=0}^{\infty} \frac{-1}{p+1} z^p$ we obtain:

$$\text{Taylor}_{h=0} \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} (\nabla_\theta E^T u_i) u_i = \sum_{i=1}^{D} \left(\sum_{p=0}^{\infty} \frac{-1}{p+1} h^p \lambda_i^p \right) (\nabla_\theta E^T u_i) u_i \tag{13}$$

□

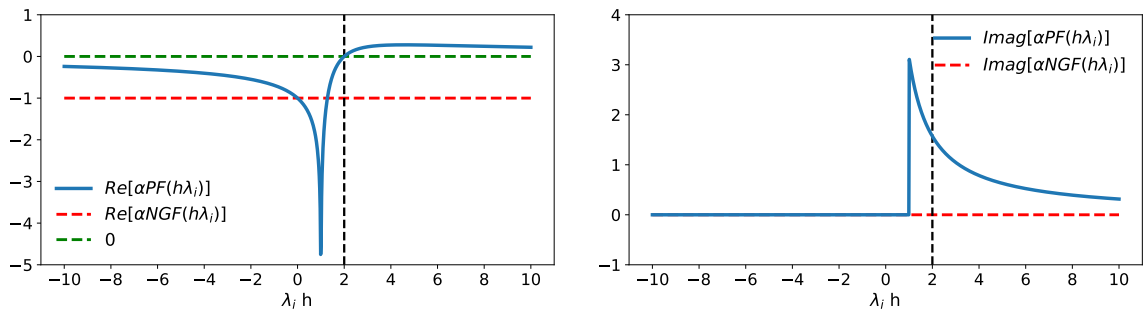
We have used BEA to find the flow that when Taylor expanded at $h = 0$ leads to the series in Eq 11. When the BEA series in Eq 11 converges, namely $\lambda_0 < 1/h$, the PF and the flow given by the BEA series are the same. When $\lambda_0 > 1/h$ however, the PF is complex and the BEA series diverges. While in this case any BEA truncated flow will not be able to track gradient descent closely, we show that for quadratic losses the PF will track gradient descent exactly, and similarly that is is a good model of gradient descent around fixed points. We show examples of the PF tracking gradient descent exactly in the quadratic case in Figures 2(b) and 5.

Remark 3.1 *For quadratic losses of the form $E = \frac{1}{2}\theta^T A\theta + b^T\theta$, the PF captures gradient descent exactly. This case has been proven in Hairer et al. (2006). The solution of the PF can also be computed exactly in terms of the eigenvalues of $\nabla_\theta^2 E$: $\theta(t) = \sum_{i=1}^{D} e^{\frac{\log(1-h\lambda_i)}{h} t} \theta_0^T u_i u_i + t \sum_{i=1}^{D} \frac{\log(1-h\lambda_i)}{h\lambda_i} b^T u_i$.*

Remark 3.2 *In a small enough neighborhood around a critical point (where higher order derivatives can be ignored) the PF can be used to describe gradient descent dynamics closely. We show this also using a linearization argument in Section A.5 in the Appendix.*

Negative Gradient Flow	**IGR Flow**	**Principal Flow**
$\dot{\theta} = \sum_i^D -(\nabla_\theta E^T u_i)u_i$	$\dot{\theta} = \sum_i^D -(1+\frac{h}{2}\lambda_i)(\nabla_\theta E^T u_i)u_i$	$\dot{\theta} = \sum_i^D \frac{\log(1-h\lambda_i)}{h\lambda_i}(\nabla_\theta E^T u_i)u_i$
$\alpha_{NGF}(h\lambda_i) = -1$	$\alpha_{IGR}(h\lambda_i) = -(1+\frac{h}{2}\lambda_i)$	$\alpha_{PF}(h\lambda_i) = \frac{\log(1-h\lambda_i)}{h\lambda_i}$

Table 1: Understanding the differences between the flows discussed in terms of the eigendecomposition of the Hessian. All flows have the form $\dot{\theta} = \sum_{i=1}^D \alpha(h\lambda_i)(\nabla_\theta E^T u_i)u_i$ with different α summarized here.



(a) Real part.

(b) Imaginary part.

Figure 3: Comparing the coefficients α_{NGF} and α_{PF} across the training landscape.

Definition 3.2 *The terms $\mathcal{C}(\nabla_\theta^3 E)$ are called* ***non-principal terms****. The term $\frac{1}{12}\nabla_\theta E^T(\nabla_\theta^3 E)\nabla_\theta E$ in Eq 8 is a non-principal term (we will call this term non-principal third order term).*

Definition 3.3 *We define the* ***principal flow with third order non principal term*** *as*

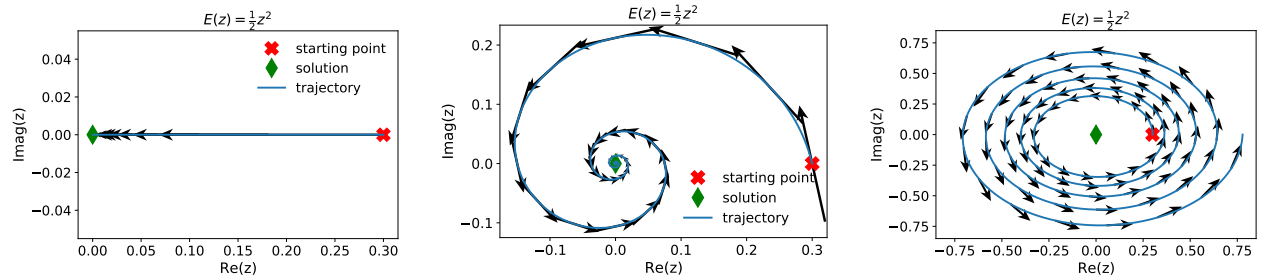
$$\dot{\theta} = \sum_i^D \frac{\log(1-h\lambda_i)}{h\lambda_i}(\nabla_\theta E^T u_i)u_i - \underbrace{\frac{h^2}{12}\nabla_\theta E^T(\nabla_\theta^3 E)\nabla_\theta E}_{\textit{third order non principal term}} \tag{14}$$

General theoretical bounds on the error between the continuous time flows and gradient descent are challenging to construct in the case of a general parametrised $E(\theta)$ as the error will be determined by the shape of E. We know the conditions which determine when certain flows follow gradient descent exactly: the NGF and gradient descent will follow the same trajectory when E has a constant gradient and thus is a linear function[1], while the PF generalises that and will follow the same trajectory as gradient descent when E is quadratic. Informally, we can state that the closer we are to these exact conditions, the more likely the flows to capture the dynamics of gradient descent. Formally, bounds on the error between GD and NGF can be provided by the Fundamental Theorem (Theorem 10.6 in Wanner and Hairer (1996)) which has recently been adapted to a neural network parametrisation by Elkabetz and Cohen (2021); this bound depends on the magnitude of the smallest Hessian eigenvalue along the NGF trajectory. We hope that future work can expand the Fundamental Theorem such that error bounds between the PF and gradient descent can be constructed for deep neural networks. Here we take an empirical approach and show that although not exact outside the quadratic case the PF captures key features of the gradient descent dynamics in stable or unstable regions of training, around and outside critical points, for small examples or large neural networks.

3.1 The principal flow and the eigen decomposition of the Hessian

All flows considered here have the form form $\dot{\theta} = \sum_{i=1}^D \alpha(h\lambda_i)(\nabla_\theta E^T u_i)u_i$, where α is a function computing the corresponding coefficient; we will denote the one associated with each flow as α_{NGF}, α_{PF} and α_{IGR} respectively. For a side-by-side comparison between the NGF, IGR flow and the PF as functions of the Hessian eigendecomposition see Table 1. Since $\nabla_\theta E^T u_i \geq 0$, the α function determines the sign of a

[1]This follows from Theorem 6.1 since $\nabla_\theta^2 E \nabla_\theta E = \nabla_\theta \frac{1}{2}\|\nabla_\theta E\|^2 = 0 \implies \|\nabla_\theta E\|^2 = c$. Which leads to $\frac{d}{d\theta_i}\|\nabla_\theta E\|^2 = 2\frac{d}{d\theta_i}E\frac{d^2}{d\theta_i}E = 0 \implies \frac{d}{d\theta_i}E = c_i \implies E(\theta) = \sum_i c_i\theta_i + d$.



(a) $\lambda_0 < 1/h, \quad \alpha_{PF}(\lambda_0 h) < 0$ (b) $1/h < \lambda_0 < 2/h, \;\; Re(\alpha_{PF}(\lambda_0 h)) < 0$ (c) $2/h < \lambda_0, \;\; Re(\alpha_{PF}(\lambda_0 h)) > 0$

Figure 4: The behavior of PF on $E(z) = \frac{1}{2}z^2$ with solution $z(t) = e^{\log(1-h)/h}z(0)$. When $\lambda_0 < 1/h$, $z(t) = (1-h)^{t/h}z(0)$ which is in real space and converges to the equilibrium. When $\lambda_0 > 1/h$, $z(t) = (h-1)^{t/h}\left(\cos(\pi t/h) + i\sin(\pi t/h)\right)z(0)$. This exhibits oscillatory behavior, and when $\lambda_0 > 2/h$, diverges.

modified vector field in the direction u_i. For brevity, it will be useful to define the coefficient of u_i in the vector field of the PF:

Definition 3.4 *We call* $sc_i = \frac{\log(1-h\lambda_i)}{h\lambda_i}(\nabla_\theta E^T u_i) = \alpha_{PF}(h\lambda_i)\nabla_\theta E^T u_i$ *the* ***stability coefficient*** *for eigendirection* i*.* $sign(sc_i) = sign(\alpha_{PF}(h\lambda_i))$*.*

In order to understand the PF and how it is different to the NGF, we explore the change in each eigendirection u_i and we perform case analysis on the relative value of the eigenvalues λ_i and the learning rate h. To do so, we will compare α_{NGF} and α_{PF} based on the value of $h\lambda_i$; comparing with the NGF is especially insightful, since that is the direction in which the function E is minimized. Since our goal is to understand the behavior of gradient descent, we perform the case by case analysis of what happens at the start of a gradient descent iteration and thus use real values for λ_i and u_i, even when the PF is complex. We visualize α_{NGF} and α_{PF} in Figure 3 and we use Figure 4 to show examples of each case using a simple function.

Real stable case: $\lambda_i < 1/h$. $\text{sign}(\alpha_{NGF}) = \text{sign}(\alpha_{PF}) = -1$.

$\alpha_{NGF} = -1$ and $\alpha_{PF}(h\lambda_i) = \frac{\log(1-h\lambda_i)}{h\lambda_i} < 0$. The coefficients of both the NGF and PF in eigendirection u_i are both negative and real. The case is exemplified in Figure 4(a)..

Complex stable case: $1/h < \lambda_i < 2/h$. $\text{sign}(\alpha_{NGF}) = \text{sign}(Re[\alpha_{PF}]) = -1$. $\alpha_{PF} \in \mathbb{C}$.

$\alpha_{NGF} = -1$ and $\alpha_{PF}(h\lambda_i) = \frac{\log(1-h\lambda_i)}{h\lambda_i} = \frac{\log(-1+h\lambda_i)+i\pi}{h\lambda_i} \in \mathbb{C}$ and $Re[\alpha_{PF}] = \frac{\log(-1+h\lambda_i)}{h\lambda_i} < 0$. The real part of the coefficient of both the NGF and PF in eigendirection u_i are both negative. However, the imaginary part of α_{PF} can still introduce instability and oscillations, as we show in Figure 4(b).

Unstable complex case: $2/h < \lambda_i$. $\text{sign}(\alpha_{NGF}) \neq \text{sign}(Re[\alpha_{PF}])$. $\alpha_{PF} \in \mathbb{C}$.

$\alpha_{NGF} = -1$ and $\alpha_{PF}(h\lambda_i) = \frac{\log(1-h\lambda_i)}{h\lambda_i} = \frac{\log(-1+h\lambda_i)+i\pi}{h\lambda_i} \in \mathbb{C}$ and $Re[\alpha_{PF}] = \frac{\log(-1+h\lambda_i)}{h\lambda_i} > 0$. The real part of the coefficient the of NGF in eigendirection u_i is negative, while the real part of the coefficient of the PF is positive. Since the direction minimising the function E is given by the NGF and is negative, the change in sign given by the PF can cause instabilities. The complex component can still introduce oscillations, however, the higher $\lambda_i h$, the smaller the complex component is. We show a simple example in Figure 4(c).

The importance of the largest eigenvalue λ_0. The largest eigenvalue λ_0 plays an important part in the PF. Since $h\lambda_0 \geq h\lambda_i \quad \forall i$, λ_0 determines where in the above cases the PF is situated and thus whether there are oscillations and unstable behavior in training. For all flows of the form we consider we can write:

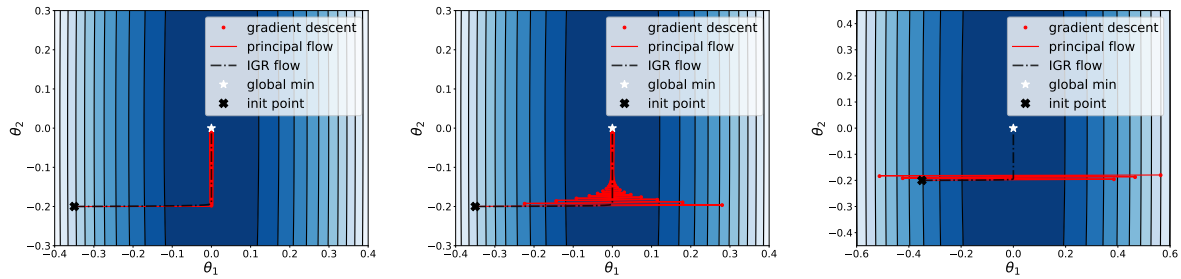
$$\frac{dE(\theta)}{dt} = \frac{dE(\theta)}{d\theta}^T \frac{d\theta}{dt} = \nabla_\theta E^T \sum_{i=1}^{D} \alpha(h\lambda_i)\nabla_\theta E^T u_i u_i = \sum_{i=1}^{D} \alpha(h\lambda_i)(\nabla_\theta E^T u_i)^2 \tag{15}$$

and thus if $\alpha(h\lambda_i) \in \mathbb{R}$ and $\alpha(h\lambda_i) < 0 \; \forall i$ then $\frac{dE(\theta)}{dt} \leq 0$ and following the corresponding flow minimises E. In the case of the PF this gets determined by λ_0. If $\lambda_0 < \frac{1}{h}$ then $\alpha_{PF}(h\lambda_i) < 0 \; \forall i$ (real stable case above)

and the PF minimises E. If $1/h < \lambda_0 < \frac{2}{h}$ then $Re[\alpha_{PF}(h\lambda_i)] < 0 \; \forall i$ (complex stable case above) close to a gradient descent iteration $\lambda_i, u_i \in \mathbb{R}$ we can write that $\frac{dRe[E(\theta)]}{dt} = \sum_{i=1}^{D} Re[\alpha_{PF}(h\lambda_i)](\nabla_\theta E^T u_i)^2$ and thus the real part of the loss function decreases. If $\lambda_0 > \frac{2}{h}$ then $Re[\alpha_{PF}(h\lambda_0)] > 0$ (unstable complex case above) and if $(\nabla_\theta E^T u_0)^2$ is sufficiently large we can no longer ascertain the behavior of E. We present a discrete time argument for this observation in Section A.7.1.

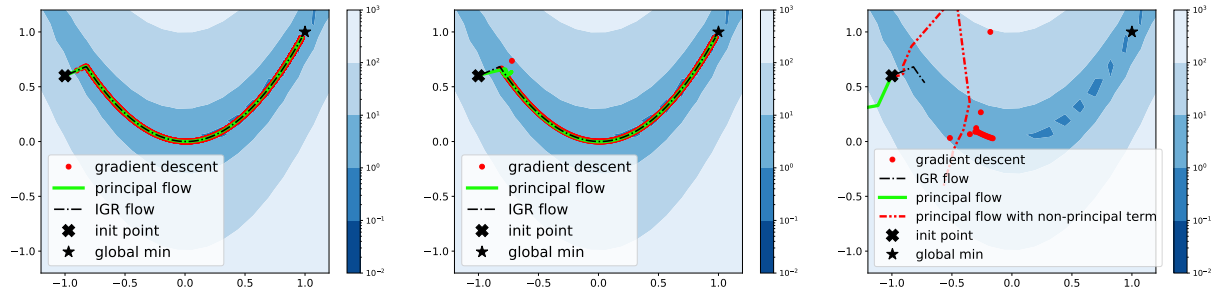
Building intuition. For quadratic objective $E(\theta) = \frac{1}{2}\theta^T A\theta$ the PF describes gradient descent exactly. We show examples Figures 2 and 5. Unlike the NGF or the IGR flow, the PF captures the oscillatory and divergent behavior of gradient decent. Importantly, to capture the unstable behavior which occurs when $\lambda_0 > 1/h$ the imaginary part of the PF is needed. To expand intuition outside the quadratic case, we show the PF for the banana function (Rosenbrock, 1960) in Figure 6 and an additional example in 1D with a non-quadratic function (Figure 30 in the Appendix). In this case, the PF no longer follows the gradient descent trajectory exactly, but we still we observe the importance of the PF in capturing instabilities of gradient descent; we also observe that adding non-principal terms can restabilize the trajectory.

Remark 3.3 *For the banana function, the principal terms have a destabilizing effect when $h > 2/\lambda_0$ while the non principal terms can have a stabilizing effect.*



(a) $\lambda_0 < 1/h$ (stability) (b) $1/h < \lambda_0 < 2/h$ (oscillations) (c) $\lambda_0 > 2/h$ (divergence)

Figure 5: **Quadratic losses in 2 dimensions**. The PF captures the behavior of gradient descent exactly for quadratic losses, including oscillatory behavior and divergence.



(a) $\lambda_0 < 1/h$ (b) $\lambda_0 < 2/h (\lambda_0 \approx 1.9/h)$ (c) $\lambda_0 >> 2/h (\lambda_0 \approx 5/h)$

Figure 6: **Banana function**. The PF can capture instability and the gradient descent trajectory over many iterations when λ_0 is close to $2/h$. When $\lambda_0 >> 2/h$ (right) the PF does not track the GD trajectory over many gradient descent steps, but when including a non-principal term the flow is able to capture the general trajectory of gradient descent and unstable behavior of gradient descent.

3.2 The stability analysis of the principal flow

We now perform stability analysis on the PF, to understand how it can be used to predict certain behaviors of gradient descent around critical points of the loss function E. Consider θ^* such a critical point, i.e $\nabla_\theta E(\theta^*) = 0$. For a critical point θ^* to be locally attractive, all eigenvalues of the Jacobian evaluated at θ^* need to have negative real part.

The principal flow has the following Jacobian (proof in Section A.4 in the Appendix):

$$J_{PF}(\theta^*) = \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i^*)}{h} u_i^* u_i^{*T} \tag{16}$$

where λ_i^*, u_i^* are the eigenvalues and eigenvectors of the Hessian $\nabla_\theta^2 E(\theta^*)$. We thus have that the eigenvalues of the Jacobian $J_{PF}(\theta^*)$ at the critical point θ^* are $\frac{1}{h}\log(1 - h\lambda_i^*)$ for $i = 1, \ldots, D$.

Local minima. Suppose that θ^* is a local minimum. Then all Hessian eigenvalues are positive $\lambda_i^* \geq 0$. We perform the stability analysis in cases given by the value of λ_i^*, corresponding to the cases in Section 3.1:

$h < 1/\lambda_i^*$. The corresponding eigenvalue of the Jacobian $\frac{1}{h}\log(1 - h\lambda_i^*)$ is negative, since $0 < 1 - h\lambda_i^* < 1$. The principal vector field is attractive in the corresponding eigenvector direction.

$h \in [1/\lambda_i^*, 2/\lambda_i^*]$. The corresponding eigenvalue of the Jacobian $\frac{1}{h}\log(1 - h\lambda_i^*) = \frac{1}{h}\log(h\lambda_i^* - 1) + i\frac{\pi}{h}$ is complex, with negative real part, since since $h\lambda_i^* - 1 < 1$. The principal vector field is attractive in the corresponding eigenvector direction.

$h > 2/\lambda_i^*$. The corresponding eigenvalue of the Jacobian $\frac{1}{h}\log(1-h\lambda_i^*) = \frac{1}{h}\log(h\lambda_i^*-1)+i\frac{\pi}{h}$ is complex, with positive real part, since since $h\lambda_i^* - 1 > 1$. The principal vector field is not attractive in the corresponding eigenvector direction.

The last case tells us that the PF is not always attracted to local minima, as it is not attractive in eigendrections where $h > 2/\lambda_i^*$. Thus **like gradient descent, the PF can be repelled around local minima for large learning rates**. This is in contrast to the NGF and the IGR flow, which always predict convergence around a local minima: the eigenvalues of the NGF Jacobian are $-\lambda_i^*$, and for the IGR flow the eigenvalues are $-\lambda_i^* - \frac{h^2}{2}\lambda_i^{*2}$, both are negative when λ_i^* is positive. For derivations see Section A.4.1 in the Appendix.

Remark 3.4 *For quadratic losses, where the PF is exact, the results above recover the classical gradient descent result for quadratic losses namely that gradient descent convergences if $\lambda_0 < 2/h$, otherwise diverges.*

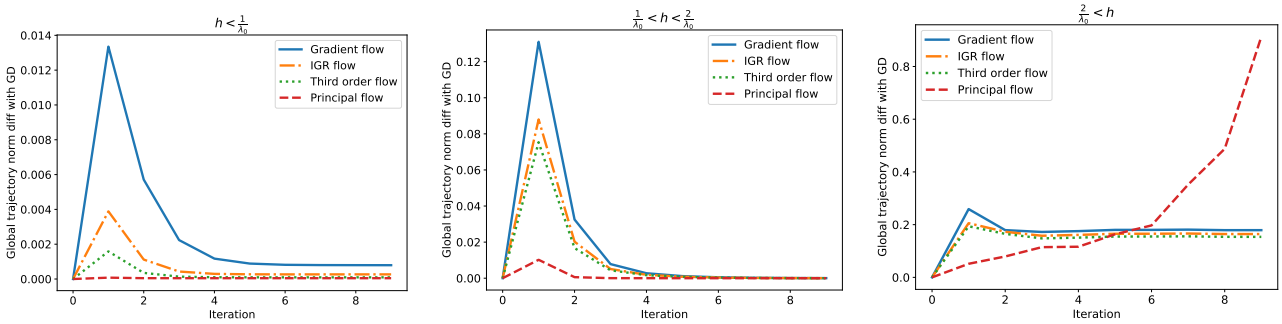
Saddle points. Suppose that θ^* is a saddle point. In this case there exists λ_s^* such that $\lambda_s^* < 0$. We want to analyse the behavior of the PF in the direction of the corresponding eigenvector u_s^*. In that case, $\log(1 - h\lambda_s^*) > 0$ which entails that the PF is repelled in the eigendirections of saddle points. Note that this is also the case for the NGF since the corresponding eigenvalues of the Jacobian of the NGF would be $-\lambda_s^*$, also positive. Unlike the NGF however, the subspace of eigendirections that the PF is repelled by can be larger since it includes also eigendirections where $\lambda_i^* > 2/h > 0$.

4 Predicting neural network gradient descent dynamics with the principal flow

Computing the PF on large neural networks during training is computationally prohibitive, as it requires finding all eigenvalues of the Hessian matrix once for each step of the flow simulation, corresponding to many eigen-decompositions per gradient descent step. To build intuition about the PF for neural networks, we start with a small MLP for a 2 dimensional input regression problem, with random inputs and labels. Here we can understand the behavior of the PF since we can compute its modified vector field exactly and compare it with the behavior of gradient descent. We show results in Figure 7, where we visualize the norm of the difference between gradient descent parameters at each iteration and the parameters produced by the continuous time flows we compare with. We observe that *short term the principal flow is better than all other flows at tracking the behavior of gradient descent.* As the number of iterations increases however, the PF accumulates error in the case of $\lambda_0 > 2/h$; this is likely due to the fact that while gradient descent parameters are real, this is not the case for the PF, as discussed in Remark 4.1. Since we are primarily concerned with using the PF to understand gradient descent for a small number of iterations this will be less of a concern in our experimental settings. Additional results which confirm the PF is better than the other flows at tracking gradient descent on a bigger network trained the UCI breast cancer dataset are shown in Figure 32 in the Appendix.

Remark 4.1 ***On the multiple iteration behavior of the principal flow***. *We note that while gradient descent parameters are real for any iteration θ_t, θ_{t+1}, ... θ_{t+n}, when we approximate the behavior of gradient*

descent by initializing $\theta(0) = \theta_t$ and running the PF for time nh, there is nothing enforcing that $\theta(h)$, ... $\theta(nh)$ will be real when the PF is complex ($\lambda_0 > 1/h$). We also note that in that case the hermitian Hessian is not symmetric and the eigenvalues and eigenvector of the Hessian will not be real. For long term trajectories (larger n), this can have an effect on long term error between gradient descent and PF trajectories, through an accumulating effect of the imaginary part in the PF. This can be mitigated by using the PF to understand the short term behavior of gradient descent (small n).



(a) $\lambda_0 < 1/h$ (b) $1/h < \lambda_0 < 2/h$ (c) $\lambda_0 > 2/h$

Figure 7: Error between gradient descent parameters and parameters obtained following continuous time flows for multiple iterations: $\|\theta_n - \theta(nh)\|$ with $\theta(0) = \theta_0$. For small n, the PF is better at capturing the behavior of gradient descent across all cases.

4.1 Predicting $\nabla_\theta E^T u_0$ using the principal flow

For large neural networks, instead of computing the entire PF describing how the entire parameter vector changes in time we can use the PF to approximate changes in time in one eigendirection only. This will allow us to compare the predictions of the PF against the predictions of the NGF and IGR flow on realistic settings. To do so, we first have to compute how the gradient changes in time:

Corollary 4.1 *If θ follows the PF, then:* $(\dot{\nabla_\theta E}) = \sum_i^D \frac{\log(1-h\lambda_i)}{h}(\nabla_\theta E^T u_i)u_i$.

This follows from applying the chain rule and using the definition of the PF. We contrast this with how the gradient evolves if the parameters follow the NGF:

Corollary 4.2 *If θ follows the NGF, then:* $(\dot{\nabla_\theta E}) = \sum_i^D -\lambda_i(\nabla_\theta E^T u_i)u_i$

Corollary 4.3 *If θ follows the IGR flow, then:* $(\dot{\nabla_\theta E}) = \sum_i^D -\left(\lambda_i + \frac{h}{2}\lambda_i^2\right)(\nabla_\theta E^T u_i)u_i$

We would like to use the above to assess how $\nabla_\theta E^T u_i$ changes in time under the above flows and check their predictions empirically against results obtained when training neural networks with gradient descent. Since u_i is an eigenvector of the Hessian it also changes in time according to the changes given by the corresponding flow, making $(\nabla_\theta \dot{E}^T u_i)$ difficult to calculate. Even when if we wrote an exact flow for $(\nabla_\theta \dot{E}^T u_i)$, it would be computationally challenging to simulate it since finding the new values of u_i would depend on the full Hessian and would lead to the same computational issues we are trying to avoid in the case of large neural networks. In order to mitigate these concerns, we will make the additional approximation that λ_i and u_i do not change inside an iteration which will allow us to approximate changes to $\nabla_\theta E^T u_i$ and compare them against empirical observations. We note that we will not use this approximation for any other results.

Remark 4.2 *If we assume that λ_i, u_i do not change between iterations, if θ follows the PF then* $(\nabla_\theta \dot{E}^T u_i) = \frac{\log(1-h\lambda_i)}{h}\nabla_\theta E^T u_i$.

Remark 4.3 *If we assume that λ_i, u_i do not change between iterations, if θ follows the NGF we can write* $(\nabla_\theta \dot{E}^T u_i) = -\lambda_i \nabla_\theta E^T u_i$.

Remark 4.4 *If we assume that λ_i, u_i do not change between iterations, if θ follows the IGR flow we can write* $(\nabla_\theta \dot{E}^T u_i) = -\left(\lambda_i + \frac{h}{2}\lambda_i^2\right)\nabla_\theta E^T u_i$.

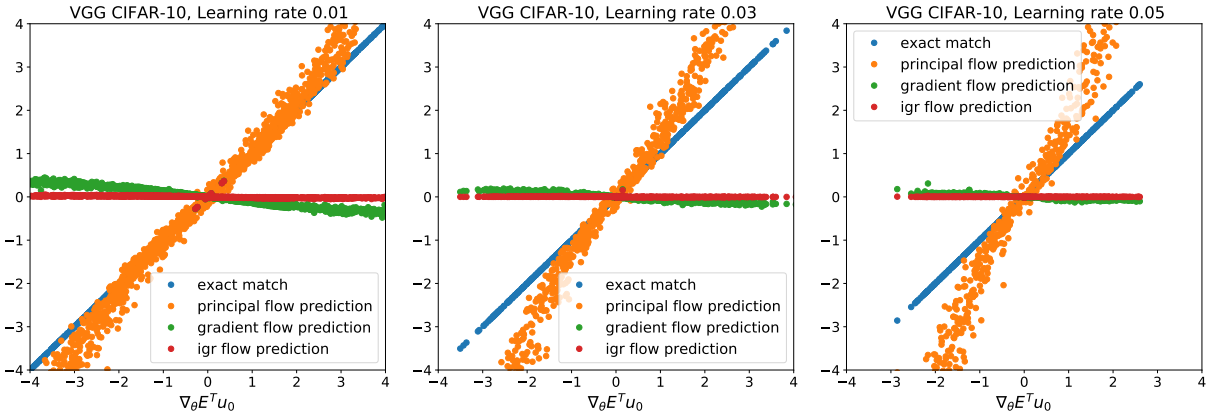


Figure 8: Predictions of $\nabla_\theta E^T u_0$ according to the NGF, IGR flow and the PF. On the x axis we plot the value of $\nabla_\theta E^T u_0$ as measured empirically in training, and on the y axis we plot the corresponding prediction according to the flows from the value of the dot product at the previous iteration. The 'exact match' line indicates a perfect prediction, the upper bound of performance. The PF performs best from all the compared flows, however for higher learning rates its performance degrades when $\nabla_\theta E^T u_0$ is large; this is due to the fact that the higher the learning rate and the higher the gradient norm, the more likely it is that the additional assumption we used that λ_i, u_i do not change does not hold.

The above flows have the form $\dot{x} = cx$, with solution $x(t) = x(0)e^{ct}$. We can thus test these solutions empirically by training neural networks with gradient descent with learning rate h and at each step compute $\nabla_\theta E(\theta_t)^T (u_i)_{t-1}$ and compare it with the prediction $x(h)$ obtained from the solution from each flow initialized at the previous iteration, i.e. $x(0) = \nabla_\theta E(\theta_{t-1})^T (u_i)_{t-1}$. We show results with a VGG model trained on CIFAR-10 in Figure 8. The results show that the PF is substantially better than the NGF and IGR flows at predicting the behavior of $\nabla_\theta E^T u_0$. Since the NGF and the IGR flow solutions scale the initial value by the inverse of an exponential of magnitude given by λ_0 for large eigenvalues this leads to a small prediction, which is not aligned with what is observed empirically. We also note that the higher the value of $\nabla_\theta E^T u_0$, the worse the prediction of the PF; these are the areas where the approximations made in the above remarks are likely not to hold due to large gradient norms.

5 The principal flow, stability coefficients and edge of stability results

We now show the PF can be used to explain phenomena observed empirically when training neural networks using full-batch gradient descent.

Edge of stability results. Cohen et al. (2021) did a thorough empirical study to show that when training deep neural networks with full batch gradient descent the largest eigenvalue of the Hessian λ_0 keeps growing until reaching approximately $2/h$ (a phase of training they call *progressive sharpening*), after which it remains in that area; for mean squared losses this continues indefinitely while for cross entropy losses they show it further decreases later in training. They also show that instabilities in training occur when $\lambda_0 > 2/h$. Their empirical study spans neural architectures, data modalities and loss functions. We visualize the edge of stability behavior they observe in Figure 9: training is stable until the $2/h$ threshold is achieved, and that eigenvalues keep growing until reaching the threshold; since we use a cross entropy loss, the eigenvalues later decrease in training. We also visualize that iterations where the loss increases compared to the previous iteration overwhelmingly occur when $\lambda_0 > 2/h$. Cohen et al. (2021) also empirically observe that $\theta^T u_0$ has oscillatory behavior in the edge of stability area but is 0 or small outside it.

Continuous-time models of gradient descent at edge of stability. To investigate if existing continuous time flows and the PF capture gradient descent behavior at the edge of stability we train a 5 layer MLP on the toy UCI Iris dataset; this simple setting allows for the computation of the full eigenspectrum of the Hessian. We show results in Figure 10: the NGF and IGR flows have a larger error compared to the PF when predicting the parameters at the next gradient descent iteration in the edge of stability regime; the NGF and IGR flows predict the loss will decrease, while the PF captures the loss increase observed when following

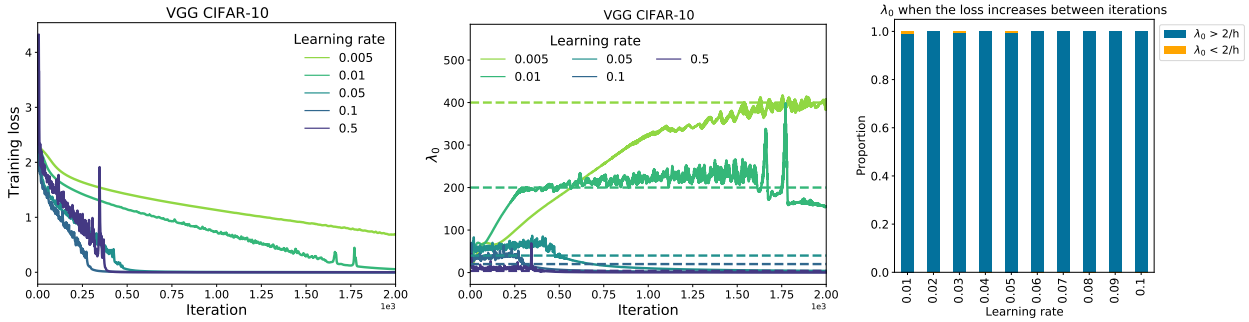


Figure 9: Edge of stability in neural networks (Cohen et al., 2021): instability occurs when $\lambda_0 > 2/h$.

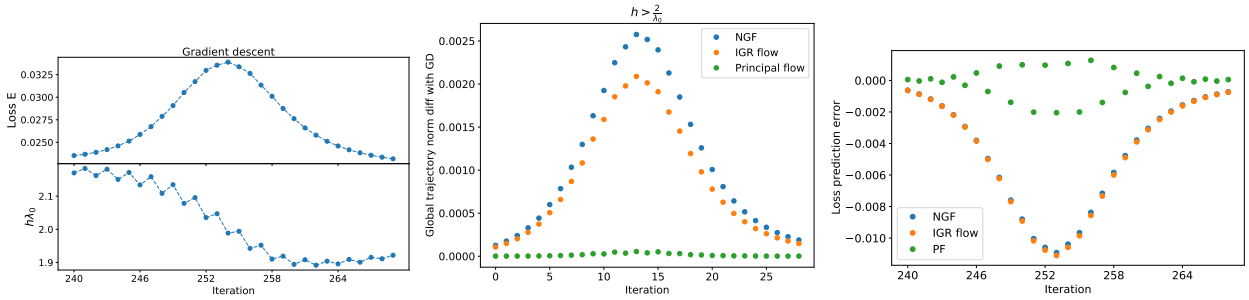


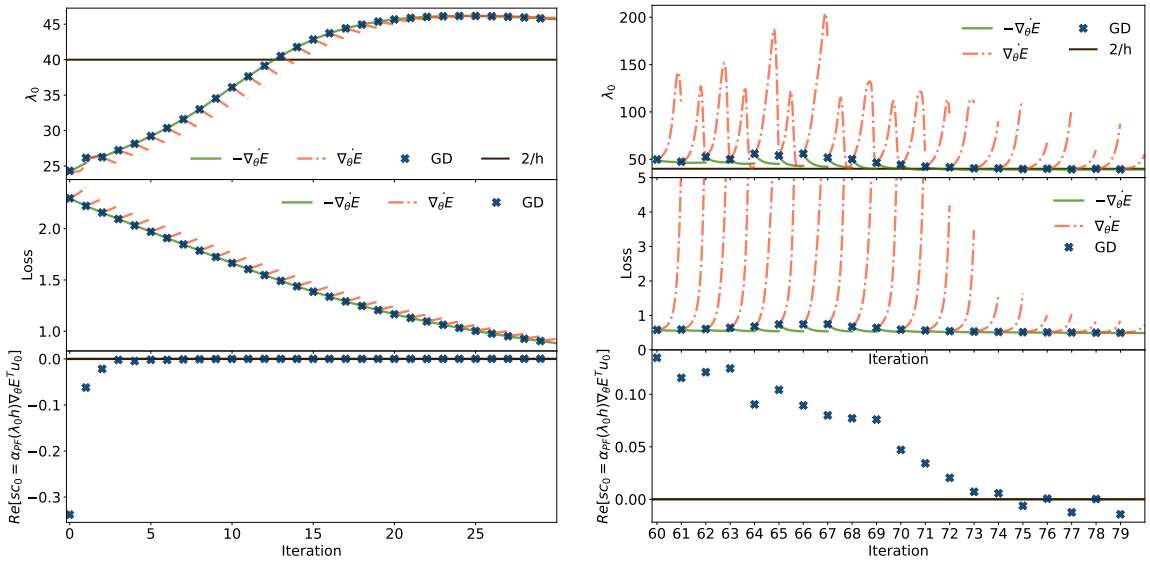
Figure 10: Comparing different continuous time models of gradient descent at the edge of stability area on a small 5 layer MLP, with 10 units per layer. We show the local parameter prediction error $\|\theta_t - \theta(h; \theta_{t-1})\|$ for the NGF, IGR and PF flows (middle), as well as $E(\theta_t) - E(\theta(h; \theta_{t-1}))$ (right).

gradient descent. As we remarked in Section 2, the NGF and the IGR flow do not capture instabilities when the eigenvalues of the Hessian are positive, which has been remarked to be largely the case in the neural network setting through empirical studies (Sagun et al., 2017; Ghorbani et al., 2019; Papyan, 2018) and we observe here (Figure 33 in the Appendix). We spend the rest of the section focusing on how we can use the PF to understand and model edge of stability phenomena using a continuous time approach.

Connection with the principal flow: stability coefficients. The PF contains some of the key quantities observed in the edge of stability phenomenon: the eigenvalues of the Hessian λ_i, the dot product $\nabla_\theta E^T u_i$ and the threshold $2/h$. All these quantities appear in the PF via the stability coefficient $sc_i = \frac{\log(1-h\lambda_i)}{h\lambda_i}\nabla_\theta E^T u_i = \alpha_{PF}(\lambda_i h)\nabla_\theta E^T u_i$ of eigendirection u_i. Through the PF, by connecting the case analysis in Section 3.1 with existing and new empirical observations, we can shed light on the edge of stability behavior in deep learning.

First phase of training (progressive sharpening): $\lambda_0 < 2/h$. This entails $Re[sc_i] = Re[\alpha_{PF}(h\lambda_i)] \leq 0, \forall i$ (Real stable and complex stable cases of the analysis in Section 3.1). $\text{sign}(\alpha_{NGF}) = \text{sign}(\alpha_{PF}) = -1$ and following the PF minimises E or its real part (Eq 15). To understand the behavior of λ_0, we now have to make use of empirical observations about the behavior of the NGF early in the training of neural networks. It has been empirically observed that in early areas of training, λ_0 increases here when following the NGF (Cohen et al., 2021); we further show this in Figure 49 in the Appendix. Since in this part of training gradient descent follows closely the NGF, it exhibits similar behavior and λ_0 increases. We show this case in Figure 11(a).

Second phase of training (edge of stability) $\lambda_0 \geq 2/h$. This entails $Re[sc_0(\theta)] = Re[\alpha_{PF}(h\lambda_i)] \geq 0$. (Unstable complex case of the analysis in Section 3.1). We can no longer say that following the PF minimizes E. $\text{sign}(\alpha_{NGF}(h\lambda_0)) \neq \text{sign}(Re[(\alpha_{PF}(h\lambda_0)])$, since $\alpha_{NGF}(h\lambda_0) = -1$ and $\text{sign}(Re[(\alpha_{PF}(h\lambda_0)]) > 0$ meaning that in that direction gradient descent resembles the positive gradient flow $\dot{\theta} = \nabla_\theta E$ rather than the NGF. The positive gradient flow component can cause instabilities, and the strength of the instabilities depends on the stability coefficient $sc_0 = \alpha_{PF}(h\lambda_0)\nabla_\theta E^T u_0$. We show in Figures 11(b) and 13 how the behavior of the loss and λ_0 are affected by the behavior of the positive gradient flow when $\lambda_0 > 2/h$.



(a) Early training

(b) Edge of stability.

Figure 11: Understanding the edge of stability results using the PF on a 4 layer MLP: we plot the behavior of the NGF $\dot{\theta} = -\nabla_\theta E$ and the positive gradient flow $\dot{\theta} = \nabla_\theta E$ initialized at each gradient descent iteration parameters, and see that the behavior of gradient descent is connected to the behavior of the respective flow through the stability coefficient.

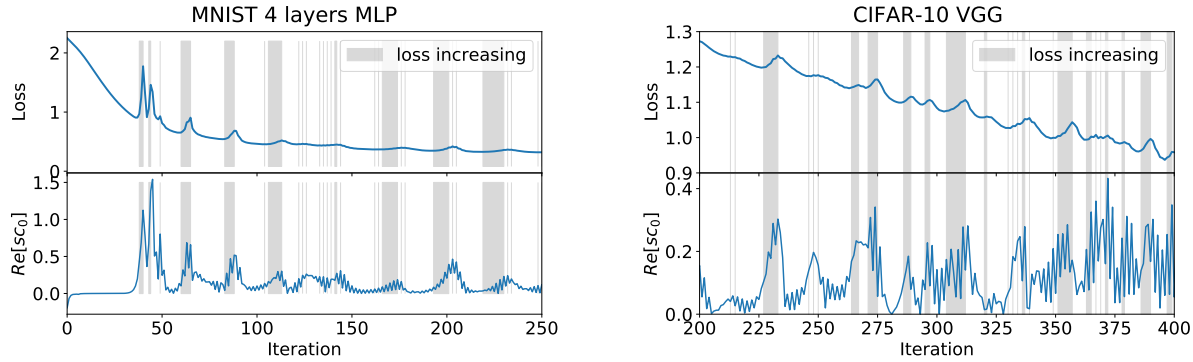


Figure 12: Understanding changes to the loss using the PF: areas where the loss increases corresponds to areas where the sc_0 is large. The highlighted areas correspond to regions where the loss increases.

More than λ_0: the importance of stability coefficients. While the sign of the real part of the stability coefficient sc_0 is determined by λ_0, its magnitude is modulated by the dot product $\nabla_\theta E^T u_0$, since $sc_0 = \alpha_{PF}(h\lambda_0)\nabla_\theta E^T u_0$. The magnitude of $\nabla_\theta E^T u_0$ plays an important role, since if λ_0 is the only eigenvalue greater than $2/h$ training is stable if $\nabla_\theta E^T u_0 = 0$, as we see in Figure 11. *To understand instabilities, we have to look at stability coefficients, not only eigenvalues.* We show in Figure 12 how the instabilities in training can be related with the stability coefficient sc_0: the increases in loss occur when the corresponding $Re[sc_0]$ is positive and large. In Figure 13 we show the equivalent results in the behavior of λ_0: λ_0 increases or decreases based on the behavior of the corresponding flow and the strength of the stability coefficient and that gets reflected in instabilities in the loss function; specifically when $\lambda_0 > 2/h$, the increase both in loss value and λ_0 of gradient descent are proportional to the increase of the positive gradient flow in that area. We show additional results in Figures 37 and 38 in the Appendix.

Is one eigendirection enough to cause instability? One question that arises from the PF is whether the leading eigendirection u_0 can be sufficient to cause instabilities, especially in the context of deep networks

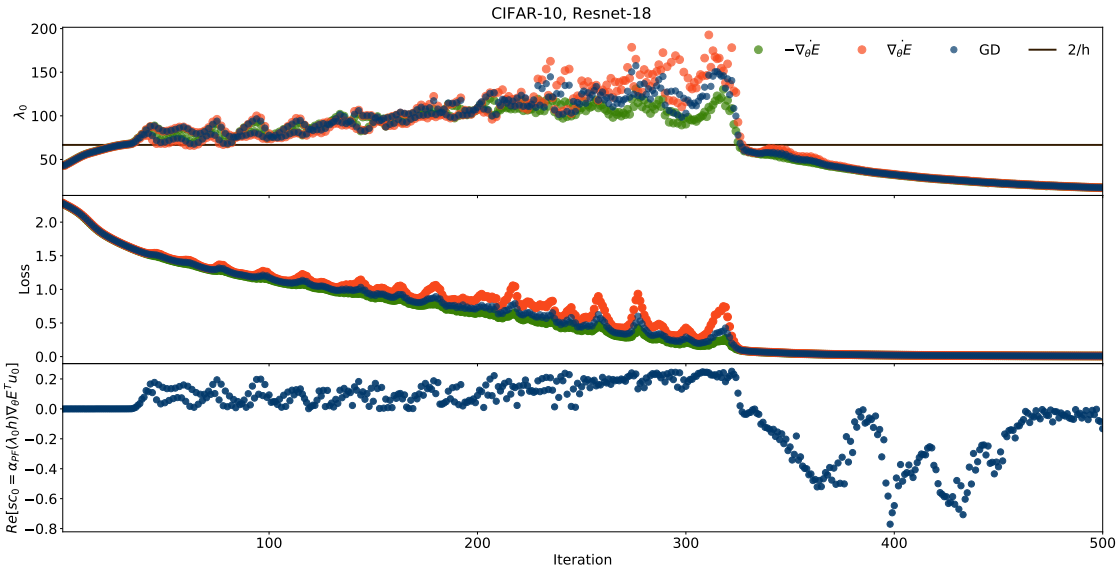


Figure 13: Understanding changes to the loss using the PF: areas where the loss increases corresponds with areas where the sc_0 is large. Together with the behavior of gradient descent, we also plot the behavior of the NGF and positive gradient flow initialized at θ_t and simulated for time h for each iteration t. As expected, the NGF decreases the loss around a gradient descent iteration, while the positive gradient flow increases it. Interestingly, when $\lambda_0 > 2/h$, the increases in loss value of gradient descent are proportional to the increase of the positive gradient flow in that area (can be seen best between iterations 200 and 350). The same behavior can be seen in relation to the eigenvalue λ_0.

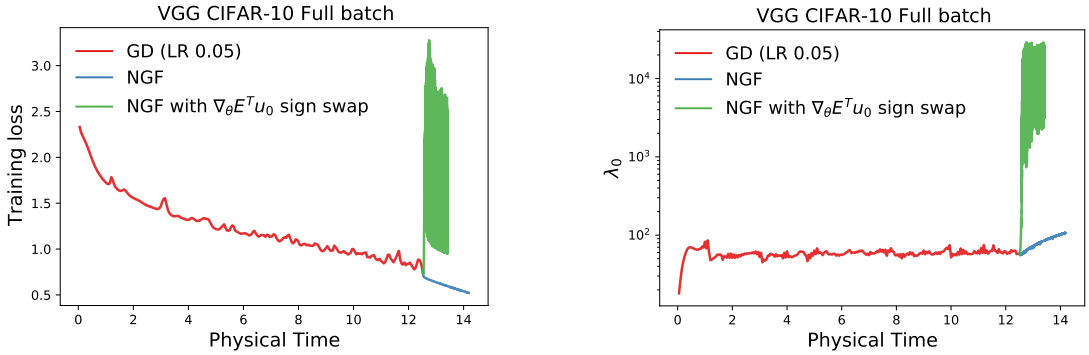


Figure 14: One eigendirection is sufficient to lead to instabilities. We construct a flow given by the NGF in all eigendirections but u_0; in the direction of u_0, we change the sign of the flow. This leads to the flow $\dot{\theta} = \nabla_\theta E^T u_0 u_0 + \sum_{i=1}^{D-1} -\nabla_\theta E^T u_i u_i$. We show this flow can be very unstable when initialised in an edge of stability area.

with millions of parameters. To assess this we train a model with gradient descent until it reaches the edge of stability ($\lambda_0 \approx 2/h$), after which we approximate the continuous flow $\dot{\theta} = \nabla_\theta E^T u_0 u_0 + \sum_{i=1}^{D-1} -\nabla_\theta E^T u_i u_i$. The coefficients of the modified vector field of this flow are negative for all eigendirections except from u_0, which is positive; this is also the case for the PF when λ_0 is the only eigenvalue greater than $2/h$. In Figure 14 we show the results: a positive coefficient for u_0 can be responsible for an increase in loss value and a significant change in λ_0.

Decreasing the learning rate. Cohen et al. (2021) show that if the edge of stability behavior is reached and the learning rate is decreased, the training stabilizes and λ_0 keeps increasing (Figure 34 in the Appendix). The PF tells us that decreasing the learning rate entails going from $Re[sc_0] \geq 0$ to $Re[sc_0] \leq 0$ since $\lambda_0 < 2/h$

after the learning rate change. Since all stability coefficients are now negative, this reduces instability. The increase in λ_0 is likely due to the behavior of the NGF in that area (as can be seen in Figure 14 when changing from gradient descent training to the NGF in an edge of stability area leads to an increase of λ_0).

The behavior of $\nabla_\theta E^T u_0$. The PF also allows us to explain the unstable behavior of $\nabla_\theta E^T u_0$ around edge of stability areas. As done in Section 4.1, we assume that λ_i, u_i do not change substantially between iterations and write $\nabla_\theta \dot{E}^T u_i = \frac{\log(1-h\lambda_i)}{h} \nabla_\theta E^T u_i$ under the PF, with solution $(\nabla_\theta E^T u_i)(t) = (\nabla_\theta E^T u_i)(0) e^{\frac{\log(1-h\lambda_i)}{h} t}$. This solution has different behavior depending on the value of λ_0 relative to $2/h$: decreasing below $2/h$ and increasing above $2/h$. We show this theoretically predicted behavior in Figure 15, alongside empirical behavior showcasing the fluctuation of $\nabla_\theta E^T u_0$ in the edge of stability area, which confirms the theoretical prediction. We also compute the prediction error of the proposed flow and show it can capture the dynamics of $\nabla_\theta E^T u_0$ closely in this setting. We present a discrete time argument for this observation in Section A.7.2.

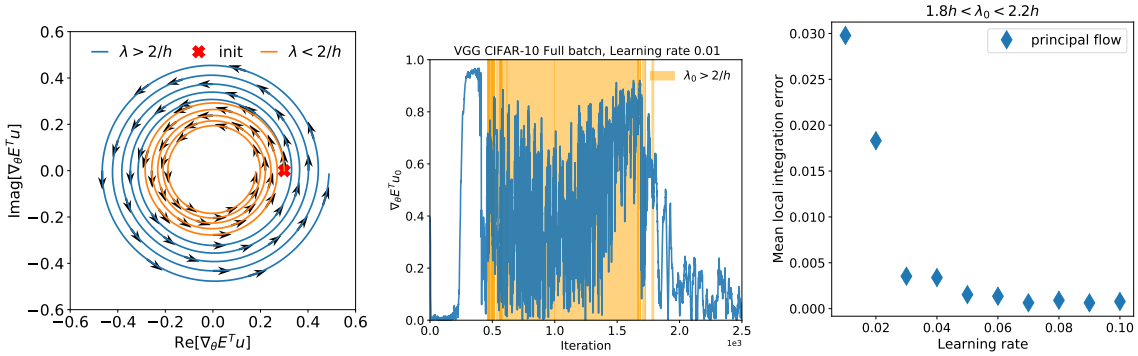


Figure 15: The unstable dynamics of $\nabla_\theta E^T u$ in the edge of stability area ($\lambda \approx 2/h$). Left: the predicted behavior of $\nabla_\theta E^T u$ under $\nabla_\theta \dot{E}^T u_i = \frac{\log(1-h\lambda_i)}{h} \nabla_\theta E^T u_i$, with an inflection point at $\lambda = 2/h$. Middle: empirical behavior of $\nabla_\theta E^T u$ for a model shows instabilities in the edge of stability area (highlighted). Right: the approximation made to derive the flow is suitable around $\lambda \approx 2/h$.

In this section we have shown that despite not capturing non-principal terms, the PF closely predicts the behavior of gradient descent in neural network training. When $\lambda_0 < 2/h$ the PF predicts stable behavior (Eq 15), which is consistent with empirical observations of gradient descent (RHS of Figure 9); this suggests that when $\lambda_0 < 2/h$ non-principal terms do not have a strong unstable presence (at least not sufficiently strong to overcome the stability coming from the PF). When $\lambda_0 > 2/h$, the PF predicts instability, which is what is observed in practice with gradient descent. While we do not know all non-principal terms and their behavior, in Section B in the Appendix we provide a justification for why the non-principal term we do know (Eq 14) can have a stabilizing effect. This evidence suggests that the PF plays a major role in capturing instability in deep learning, perhaps due to the specific structure of neural network models. We note that we used the PF to measure changes in the loss given a local Hessian; to model changes in the Hessian eigenspectrum we relied on empirical evidence from the NGF and other continuous time flows and contrasted them with gradient descent. While we focus on a continuous time approach, a discrete time approach can be used to motivate some of our observations and results (Section A.7); this is complementary to our approach which focuses on continuous time but nonetheless related, since it also ignores higher order derivatives of the loss and further suggests the strength of a quadratic approximation of the loss in the case of neural networks, as observed by Cohen et al. (2021).

6 Stabilizing training by adjusting discretization drift

The PF allows us to understand not only how gradient descent differs from the trajectory given by the NGF, but also when do they follow each other very closely. Understanding when gradient descent behaves like the NGF flow is important, since it reveals when the existing analyses of gradient descent using the NGF discussed in Section 2 are valid. It also has practical implications, since in areas where gradient descent follows the NGF closely training can be sped up by increasing the learning rate. Prior works have empirically observed that gradient descent follows the NGF early in neural network training (Cohen et al., 2021) and this observation can be used to explain why decaying learning rates (Loshchilov and Hutter,

2016) or learning rate warm up (He et al., 2019) are successful when training neural networks: having a high learning rate in areas where the drift is small will not cause instabilities and can speed up training and decaying the learning rate avoids instabilities later in training when the drift is larger.

6.1 $\nabla_\theta^2 E \nabla_\theta E$ in determines discretization drift

In previous sections we have seen that the Hessian plays an important role in defining the PF and in training instabilities. We now want to quantify the difference between the NGF and the PF in order to understand when the NGF can be used as a model of gradient descent. We find that:

Remark 6.1 *In a region of the space where $\nabla_\theta^2 E \nabla_\theta E = 0$ the PF is the same as the NGF.*

To see why, we can expand

$$\nabla_\theta^2 E \nabla_\theta E = \sum_{i=1}^{D} \lambda_i \nabla_\theta E^T u_i u_i. \tag{17}$$

By multiplying the above with u_j we have that $\lambda_j \nabla_\theta E^T u_j = 0$. For each dimension j either $\lambda_j = 0$ or $\nabla_\theta E^T u_j = 0$. If $\lambda_j = 0$ then $\alpha_{NGF}(h\lambda_j) = \alpha_{PF}(h\lambda_j) = -1$. We thus get that $\dot{\theta} = \sum_{i=1}^{D} \alpha_{PF}(h\lambda_i)(\nabla_\theta E^T u_i)u_i = \sum_{i=1}^{D} \alpha_{NGF}(h\lambda_i)(\nabla_\theta E^T u_i)u_i$.

Thus comparing the PF with the NGF reveals an important quantity: $\nabla_\theta^2 E \nabla_\theta E$. Further investigating this quantity reveals it has a connection with the total drift, since:

Theorem 6.1 *The discretization drift (error between gradient descent and the NGF) after 1 iteration of gradient descent $\theta_t = \theta_t - h\nabla_\theta E(\theta_{t-1})$ is $\frac{h^2}{2}\nabla_\theta^2 E(\theta')\nabla_\theta E(\theta')$ for a set of parameters θ' in the neighborhood of θ_{t-1}.*

This follows from the Taylor reminder theorem of the NGF in mean value form (proof in Section A.9). From here we have:

Corollary 6.1 *In a region of space where $\nabla_\theta^2 E \nabla_\theta E = 0$ gradient descent follows the NGF.*

Thus the PF revealed $\nabla_\theta^2 E \nabla_\theta E$ as a core quantity in the discretisation drift of gradient descent. To further see the connection between with the PF consider that $\left\|\nabla_\theta^2 E \nabla_\theta E\right\|^2 = \left\|\sum_{i=1}^{D} \lambda_i \nabla_\theta E^T u_i u_i\right\|^2 = \sum_{i=1}^{D} \left\|\lambda_i \nabla_\theta E^T u_i\right\|^2$; the higher each term in the sum, the higher the difference between the NGF and the PF. To measure the connection between per iteration drift and $\nabla_\theta^2 E \nabla_\theta E$ in the neural network setting we approximate it via $\left\|\theta_t - \widetilde{NGF}(\theta_{t-1}, h)\right\|$ where $\widetilde{NGF}$ is the numerical approximation to the NGF initialised at θ_{t-1}. We show results in Figures 16 and 17, which show the strong correlation between per iteration drift and $\left\|\nabla_\theta^2 E \nabla_\theta E\right\|$ throughout training and across learning rates. Since Theorem 6.1 tells us the form of the drift but not the exact value of θ', we have used θ_{t-1} instead to evaluate $\left\|\nabla_\theta^2 E \nabla_\theta E\right\|$ and thus some error exists.

Understanding this connection is advantageous since computing discretization drift is computationally expensive as it requires approximating the continuous time NGF but computing $\left\|\nabla_\theta^2 E \nabla_\theta E\right\|$ via Hessian-vector products is cheaper and approximations are available, such as $\nabla_\theta^2 E \nabla_\theta E = \frac{1}{2}\nabla_\theta \left\|\nabla_\theta E\right\|^2 \approx \frac{E(\theta + \epsilon \nabla_\theta E) - E(\theta)}{\epsilon}$ which only requires an additional backward pass Geiping et al. (2021).

6.2 Drift adjusted learning rate (DAL)

A natural question to ask is how to use the correlation between $\left\|\nabla_\theta^2 E \nabla_\theta E\right\|$ and the iteration drift to improve training stability; $\left\|\nabla_\theta^2 E \nabla_\theta E\right\|$ captures all the quantities we have shown to be relevant to instability highlighted by the PF: λ_i and $\nabla_\theta E^T u_i$ (Eq. 17). One way to use this information is to adapt the learning rate of the gradient descent update, such as using $\frac{2}{\left\|\nabla_\theta^2 E \nabla_\theta E\right\|}$ as the learning rate. This learning rate slows down training when the drift is large — areas where instabilities are likely to occur — and it speeds up training in regions of low drift — areas where instabilities are unlikely to occur. Computing the norm of the update

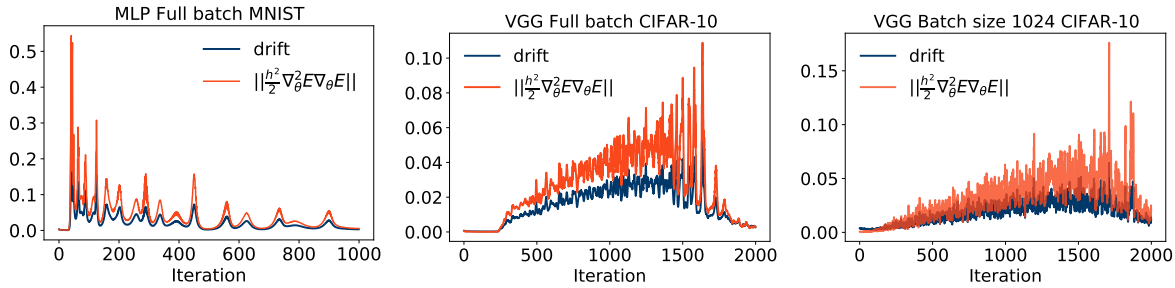


Figure 16: Connection between $||\nabla^2_\theta E \nabla_\theta E||$ and the per iteration drift as measured during training.

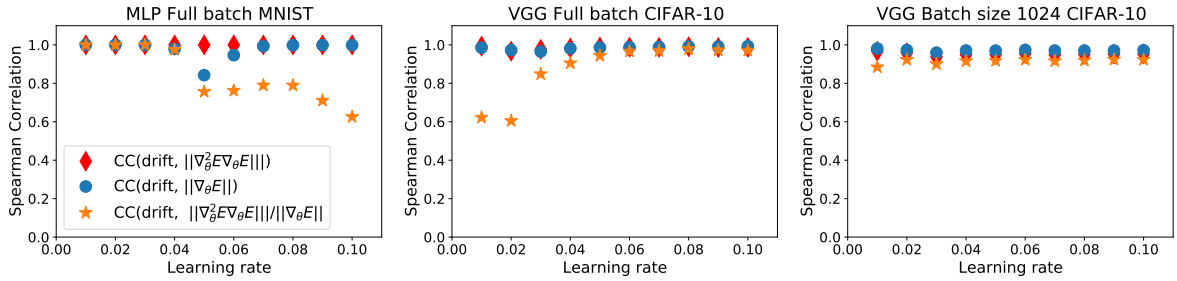


Figure 17: Correlation between $||\nabla^2_\theta E \nabla_\theta E||$ and the per iteration drift. Since $||\nabla^2_\theta E \nabla_\theta E|| = \left(||\nabla^2_\theta E \nabla_\theta E||\right) / ||\nabla_\theta E|| ||\nabla_\theta E||$, we plot the correlation with the individual terms as well.

provided by this learning rate shows a challenge however since $2/\left\|\nabla^2_\theta E \nabla_\theta E\right\| \geq \frac{2}{\lambda_0 \|\nabla_\theta E\|}$; this implies that when using this learning rate the norm of the gradient descent update will never be 0 and thus training will not result in convergence. Furthermore, the magnitude of the gradient update will be independent of the gradient norm. To reinstate the gradient norm, we propose using the learning rate

$$h(\theta) = \frac{2}{\|\nabla^2_\theta E \nabla_\theta E\| \,/\, \|\nabla_\theta E\|} = \frac{2}{\|\nabla^2_\theta E \hat{g}(\theta)\|} \tag{18}$$

where $\hat{g}(\theta)$ is the unit normalised gradient $\nabla_\theta E / \|\nabla_\theta E\|$. We will call this learning rate **DAL** (Drift Adjusted Learning rate). As shown in Figure 16, $\left\|\nabla^2_\theta E \hat{g}(\theta)\right\|$ has a strong correlation with the per iteration drift.

We use DAL to set the learning rate and show results across architectures, models and datasets in Figures 18 (with additional results in Figure 41 in the Appendix). *Despite not requiring a learning rate sweep, DAL is stable compared to using fixed learning rates.* To provide intuition about DAL, we show the learning rate and the update norm in Figure 19: for DAL the learning rate decreases in training after which it slowly increases when reaching areas with low drift. Compared to larger learning static learning rates where the update norm can increase in the edge of stability area with DAL the update norm steadily decreases in training.

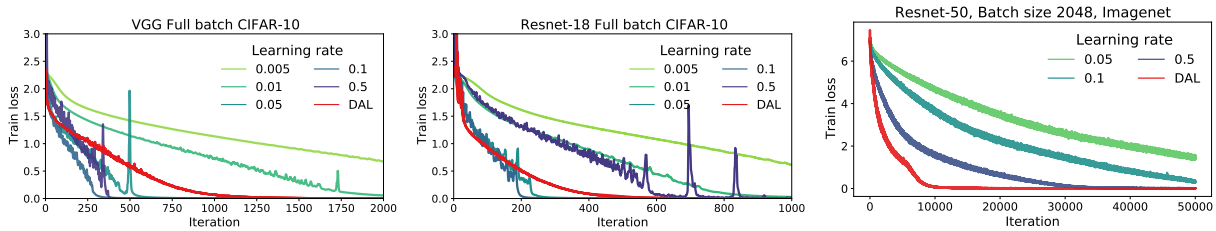


Figure 18: DAL: using the learning rate $\frac{2}{\left\|\nabla^2_\theta E \hat{g}(\theta)\right\|}$ results in improved stability without requiring a hyper-parameter sweep.

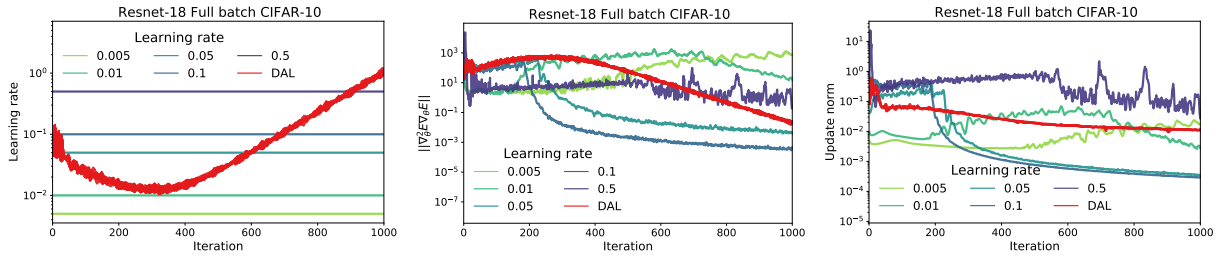


Figure 19: Key quantities in DAL versus fixed learning rate training: learning rate, and update norms.

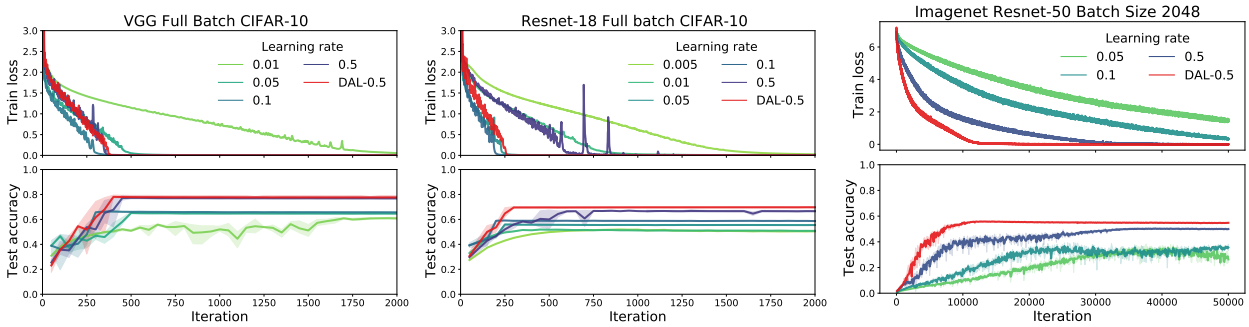


Figure 20: DAL-0.5: increased training speed and generalization compared to a sweep of fixed learning rates.

6.3 The trade-off between stability and performance

Since we are interested in understanding the dynamics of training gradient descent, we have so far focused on training performance. We now try to move our attention to test performance and generalization. Previous works (Li et al., 2019; Barrett and Dherin, 2021; Jastrzebski et al., 2019) have shown that the higher the learning rate the better the generalization performance. We now try to further correlate this information with the per iteration drift and the principal flow. To do so, we use learning rates with various degrees of sensitivity to iteration drift using DAL-p:

$$h_p(\theta) = \frac{2}{(\|\nabla_\theta^2 E\hat{g}(\theta)\|)^p} \tag{19}$$

The higher p, the slower the training and less drift there is; the lower p, there is more drift. We start with extensive experiments with $p = 0.5$, which we show in Figure 20, and show more results in Figure 42. Compared to $p = 1$ (DAL), there is faster training but at times also more instability. Performance on the test set shows that DAL-0.5 performs as well or better than when using fixed learning rates.

Remark 6.2 *We find that across datasets and batch sizes, DAL-0.5 performs best in terms of the stability generalization trade-off and in these settings can be used as a drop in replacement for a learning rate sweep.*

To further investigate the connection between drift and test set performance, we perform a set of sweeps over the power p and show results in Figure 21. These results show that the higher the drift (the smaller p), the more generalization. We also show in Figure 22 the correlation between mean per iteration drift and test accuracy both for learning rate and DAL-p sweeps. The results consistently show that the lower the mean iterations drift, the higher the test accuracy. We also show that the mean iteration drift has a connection to the largest eigenvalue λ_0: the lower the drift, the smaller λ_0. These results add further evidence to the idea that discretization drift is beneficial for generalization performance in the deep learning setting. We also notice that DAL-p with smaller values of p leads to a small λ_0 compared to vanilla gradient descent even when large learning rates are used for the latter; this could suggest its generalisation capabilities as lower sharpness has been connected to generalisation in previous works (Keskar et al., 2016; Jastrzębski et al., 2018; Foret et al., 2020).

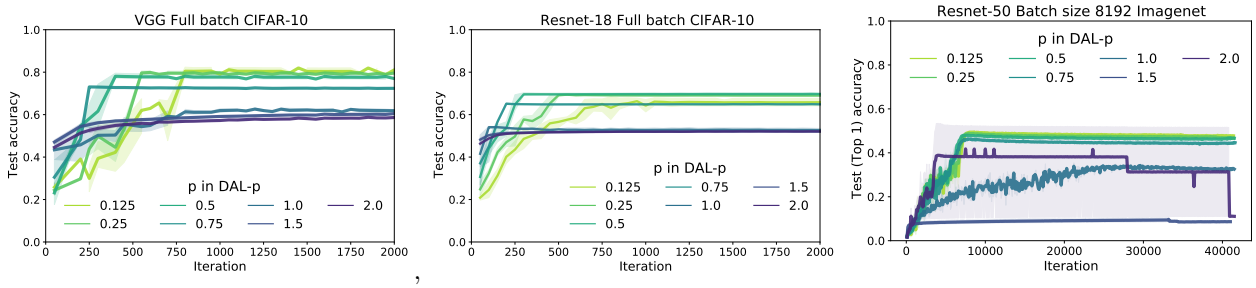
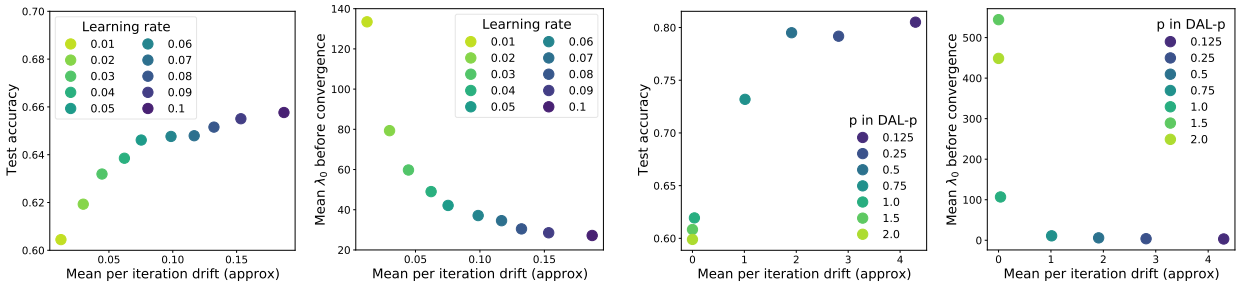


Figure 21: DAL-p sweep: discretization drift helps test performance at the cost of stability. Corresponding training curves and loss functions are present in the Figure 43 in the Appendix.



(a) Fixed learning rate sweep.

(b) DAL-p sweep.

Figure 22: The correlation between drift, test set performance and λ_0 in full batch training on CIFAR-10. The same pattern can be seen in SGD results in Figure 47.

7 Future work

Beyond gradient descent. In this work we focused on understanding vanilla gradient descent. Understanding discretization drift via the PF can be beneficial for improving other gradient based optimization algorithms as well, as we briefly illustrate for momentum updates with decay m and learning rate h:

$$v_t = mv_{t-1} - h\nabla_\theta E(\theta_{t-1}); \qquad \theta_t = \theta_{t-1} + v_t \tag{20}$$

We can scale $\nabla_\theta E(\theta_{t-1})$ in the above not by a fixed learning rate h, but by the approximation to the drift. This has two advantages: it removes the need for a learning rate sweep and it uses local information in adapting the moving average, such that in areas of large drift the contribution is decreased, while it is increased in areas where the drift is small (a more formal justification is provided in Section A.9). This leads to the following updates:

$$v_t = mv_{t-1} - \frac{1}{2||\nabla^2_\theta E(\theta_{t-1})\hat{g}(\theta)(\theta_{t-1})||}\nabla_\theta E(\theta_{t-1}) \qquad \theta_t = \theta_{t-1} + v_t \tag{21}$$

As with DAL-p, we can use powers to control the stability performance trade-off: the lower p, the more the current update contribution is reduced in high drift (instability) areas. We tested this approach on Imagenet and show results in Figure 23. The results show that integrating drift information improves the speed of convergence compared to standard gradient descent (Figure 21), and leads to more stable training compared to using a fixed learning rate. We present additional experimental results in the Appendix.

Just as momentum is a common staple of optimization algorithms, so are adaptive schemes such as Adam (Kingma and Ba, 2015) and Adagrad (Duchi et al., 2011), which adjust the step taken for each parameter independently. We can also use the knowledge from the PF to set a per parameter learning rate: instead of using $\left\|\nabla^2_\theta E(\theta_{t-1})\nabla_\theta E(\theta_{t-1})\right\|$ to set a global learning rate, we can use the per parameter information provided by $\nabla^2_\theta E(\theta_{t-1})\nabla_\theta E(\theta_{t-1})$ to adapt the learning rate of each parameter. We present preliminary

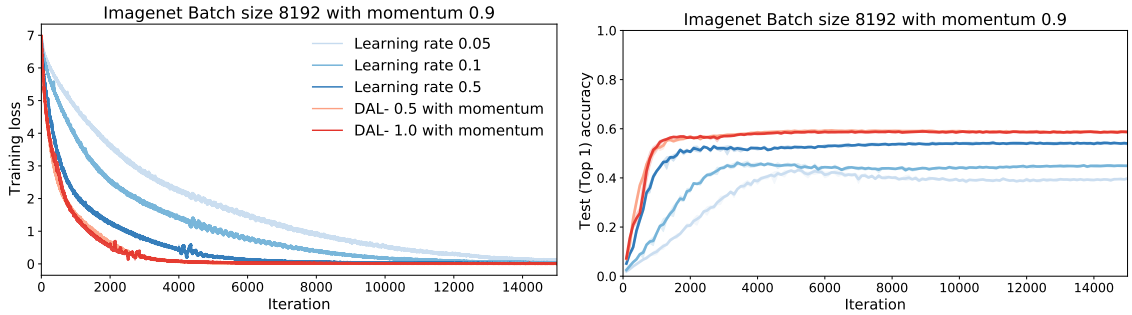


Figure 23: DAL with momentum: integrating drift information results in faster and more stable training compared to a fixed learning rate sweep. Compared to vanilla gradient descent there is also a significant performance and convergence speed boost.

results in the Appendix (Figures 45 and 46). The above two approaches (momentum and per-parameter learning rate adaptation) can be combined, bringing us closer to the most commonly used deep learning optimization algorithms. While we do not explore this avenue here, we are hopeful that this understanding of discretization drift can be leveraged further to stabilize and improve deep learning optimization.

Non-principal terms. This work focuses on understanding the effects of the principal flow on the behavior of gradient descent. The principal terms however are not the only terms in the discretization drift: we have found one non-principal term (Eq 8) and have seen that it can have a stabilising effect (Figure 6). We provide a preliminary explanation for the stabilising effect of this non-principal term together with results measuring its value in neural network training in Section B in the Appendix; we hope that future work will enhance our understanding of non-principal terms and their effects on training stability and generalization.

Neural network theory. Many theoretical works studying at gradient descent in the neural network context use the NGF (Du et al., 2018; Elkabetz and Cohen, 2021; Kunin et al., 2021; Jacot et al., 2018). We posit that replacing NGF in these theoretical contexts with PF may yield interesting results. In contrast to the NGF, the PF allows the incorporation of the learning rate into the analysis, and unlike existing continuous time models of gradient descent, it can model unstable behaviours observed in the discrete case. An example can be seen using the Neural Tangent Kernel: Jacot et al. (2018) model gradient descent using the NGF to show that in the infinite wide limit gradient descent for neural networks follows kernel gradient descent. The PF can be incorporated in this analysis either by replacing the NGF with the PF as a model of gradient descent, or by studying the difference in the PF for infinitely wide and finite width networks, since discretisation drift could be responsible for the observed gap between finite and infinite networks in the large learning rate case (Lee et al., 2020).

8 Related work

Modified flows for deep learning optimization. Barrett and Dherin (2021) found the first order correction modified flow for gradient descent using BEA and uncovered its regularization effects; they were the first to show the power of BEA in the deep learning context. Smith et al. (2021) find the first order error correction term in expectation during one epoch of stochastic gradient descent. Modified flows have also been used for other optimizers than vanilla gradient descent: Franca et al. (2020); Shi et al. (2021) compare momentum and Nesterov accelerated momentum; Kunin et al. (2021) study the symmetries of deep neural networks and use modified vector fields to show commonly used discrete updates break conservation laws present when using the NGF (for gradient descent they use the IGR flow while for momentum and weight decay they introduce different flows); Kovachki and Stuart (2021) use modified flows to understand the behavior of momentum by approximating Hamiltionian systems; França et al. (2021) construct optimizers controlling their stability and convergence rates while Li et al. (2017) construct optimizers with adaptive learning rates in the context of stochastic differential equations. In the context of two-player games, Rosca et al. (2021) compute the first order BEA correction terms while Chavdarova et al. (2021) use high-resolution differential equations to shed light on the properties of different saddle point optimizers.

Edge of stability and the importance of the Hessian. There have been a number of empirical studies on the Hessian in gradient descent. Cohen et al. (2021) observed the edge of stability behavior and performed an extensive study which led to many empirical observations used in this work. Jastrzębski et al. (2018) performed a similar study in the context of stochastic gradient descent. Sagun et al. (2017); Ghorbani et al. (2019); Papyan (2018) approximate the entire spectrum of the Hessian, and show that there are only a few negative eigenvalues, plenty of eigenvalues centered around 0, and a few positive eigenvalues with large magnitude. Similarly, Gur-Ari et al. (2018) discuss how gradient descent operates in a small subspace. Lewkowycz et al. (2020) discuss the large learning rate catapult in deep learning when the largest eigenvalue exceeds $2/h$. Gilmer et al. (2021) assess the effects of the largest Hessian eigenvalue in a large number of empirical settings.

There have been a series of concurrent works aimed at theoretically explaining the empirical results above. Ahn et al. (2022) connect the edge of stability behavior with what they coin as the 'relative progress ratio': $\frac{E(\theta-h\nabla_\theta E)-E(\theta)}{h\|\nabla_\theta E\|^2}$, which they empirically show is 0 in stable areas of training and 1 in the edge of stability areas. To see the connection between the relative progress ratio and the quantities discussed in this paper, one can perform a Taylor expansion on $\frac{E(\theta-h\nabla_\theta E)-E(\theta)}{h\|\nabla_\theta E\|^2} \approx \frac{-h\nabla_\theta E^T\nabla_\theta E+h^2/2\nabla_\theta E^T\nabla^2_\theta E\nabla_\theta E}{h\|\nabla_\theta E\|^2} = -1+h/2\frac{\nabla_\theta E^T\nabla^2_\theta E\nabla_\theta E}{\|\nabla_\theta E\|^2}$. While this ratio is related to the quantities we discuss, we also note significant differences: it is a scalar, and not a parameter length vector and thus does not capture per eigendirection behavior as we see with the stability coefficients (Section 5). Arora et al. (2022) prove the edge of stability result occurs under certain conditions either on the learning rate or on the loss function. Ma et al. (2022) empirically observe the multi-scale structure of the loss landscape in neural networks and use it to theoretically explain the edge of stability behavior of gradient descent. Chen and Bruna (2022) use low dimensional theoretical insights around a local minima to understand the edge of stability behavior. These important works are complementary to our own work; they do not use continuous time approaches and tackle primarily the edge of stability problem or its subcases, while we focus on understanding gradient descent and applying that understanding broadly, including but not limited to the edge of stability phenomenon.

Discrete models of gradient descent. The desire to understand learning rate specific behavior in gradient descent has been a motivation in the construction of discrete time analyses. These analyses have provided great insights, from studying noise in the stochastic gradient descent setting (Liu et al., 2021; Ziyin et al., 2021b), the study of overparametrized neural models and their convergence (Gunasekar et al., 2018; Du et al., 2019; Allen-Zhu et al., 2019), providing examples when gradient descent can converge to local maxima (Ziyin et al., 2021a), the importance of width for proving convergence in deep linear networks (Du and Hu, 2019). We differ from these studies both in motivation and execution: we are looking for a continuous time flow which will increase the applicability of continuous-time analysis of gradient descent. We do so by capturing discretisation drift using BEA and showing that the resulting flow is a useful model of gradient descent, which captures instabilities and escape of local minima and saddle points.

Understanding the difference between the negative gradient flow and discrete gradient descent. Elkabetz and Cohen (2021) recently examined the differences between gradient descent and the NGF in the deep learning context; their work examines the importance of the Hessian in determining when gradient descent follows the NGF. Their theoretical results show that neural networks are roughly convex and thus for reasonably sized learning rates one can expect that gradient descent follows the NGF flow closely. Their results complement ours and their approach might be extended to help us understand why the PF is sufficient to shed light on many instability observations in the neural network training.

Second-order optimization. By using second order information (or approximations thereof) to set the learning rate, DAL is related to second-order approaches used in deep learning. Many second-order methdos can be seen as approximates of Newton's method $\theta_t = \theta_{t-1} - \nabla^2_\theta E^{-1}(\theta_{t-1})\nabla_\theta E(\theta_{t-1})$. Since computing the inverse of Hessian can be prohibitively expensive for large models, many practical methods approximate it with tractable alternatives (Martens and Grosse, 2015). Foret et al. (2020) propose an optimisation scheme directly aimed at minimising sharpness, and show this can improve generalisation.

Connection between drift and generalization. We have made the connection between increased drift and increased generalization. This connection was first made by Barrett and Dherin (2021) through the IGR flow. Generalization has also been connected to the largest eigenvalue λ_0(Hochreiter and Schmidhuber, 1997; Keskar et al., 2016; Jastrzębski et al., 2018); recently Kaur et al. (2022) however showed a more complex

picture, primarily in the context of stochastic gradient descent. The largest eigenvalue could be a confounder to the drift as we have observed in Section 6.3; we hope that future work can deepen these connections.

9 Conclusion

We have expanded on previous works which used Backward Error Analysis in deep learning to find a new continuous time flow, called **the Principal Flow**, to analyze the behavior of gradient descent. Unlike existing flows, the the principal flow operates in complex space which enables it to better capture the behavior of gradient descent compared to existing flows, including but not limited to instability and oscillatory behavior. We use the form of the principal flow to find new quantities relevant to the stability of gradient descent, and shed light on newly observed empirical phenomena, such as the edge of stability results. After understanding the core quantities connected to instabilities in deep learning we devised an automatic learning rate schedule, DAL, which exhibits stable training. We concluded by cementing the connection between large discretization drift and increased generalization performance. We ended by highlighting future work avenues including incorporating the principal flow in existing theoretical analyses of gradient descent which use the negative gradient flow, incorporating our understanding of the drift of gradient descent in other optimization approaches and specializing the PF for neural network function approximators.

References

Paul Glendinning. *Stability, instability and chaos: an introduction to the theory of nonlinear differential equations*. Cambridge university press, 1994.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Vaishnavh Nagarajan and J Zico Kolter. Gradient descent gan optimization is locally stable. In *Advances in neural information processing systems*, pages 5585–5595, 2017.

Andrew K Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. *arXiv preprint arXiv:1809.10374*, 2018.

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pages 244–253. PMLR, 2018.

Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020.

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

Gal Vardi and Ohad Shamir. Implicit regularization in relu networks with the square loss. In *Conference on Learning Theory*, pages 4224–4258. PMLR, 2021.

Guilherme Franca, Jeremias Sulam, Daniel Robinson, and Rene Vidal. Conformal symplectic and relativistic optimization. In *Conference on Neural Information Processing Systems (NeurIPS 2020)*. 2020.

David GT Barrett and Benoit Dherin. Implicit gradient regularization. In *International Conference on Learning Representations*, 2021.

Samuel L Smith, Benoit Dherin, David GT Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021.

Ernst Hairer, Marlis Hochbruck, Arieh Iserles, and Christian Lubich. Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882, 2006.

Jeremy M Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. *arXiv preprint arXiv:2103.00065*, 2021.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Justin Gilmer, Behrooz Ghorbani, Ankush Garg, Sneha Kudugunta, Behnam Neyshabur, David Cardoze, George Dahl, Zachary Nado, and Orhan Firat. A loss curvature perspective on training instability in deep learning. *arXiv preprint arXiv:2110.04369*, 2021.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

Peter Bartlett, Dave Helmbold, and Philip Long. Gradient descent with identity initialization efficiently learns positive definite linear transformations by deep residual networks. In *International conference on machine learning*, pages 521–530. PMLR, 2018a.

Peter L Bartlett, Steven N Evans, and Philip M Long. Representing smooth functions as compositions of near-identity functions with implications for deep network optimization. *arXiv preprint arXiv:1804.05012*, 2018b.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In *Advances in Neural Information Processing Systems*, pages 1825–1835, 2017.

Suriya Gunasekar, Jason D Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. *Advances in Neural Information Processing Systems*, 31, 2018.

Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pages 1675–1685. PMLR, 2019.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

Simon Du and Wei Hu. Width provably matters in optimization for deep linear neural networks. In *International Conference on Machine Learning*, pages 1655–1664. PMLR, 2019.

Liu Ziyin, Botao Li, James B Simon, and Masahito Ueda. Sgd can converge to local maxima. In *International Conference on Learning Representations*, 2021a.

Kangqiao Liu, Liu Ziyin, and Masahito Ueda. Noise and fluctuation of finite learning rate stochastic gradient descent. In *International Conference on Machine Learning*, pages 7045–7056. PMLR, 2021.

David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, pages 354–363. PMLR, 2018.

Simon S Du, Wei Hu, and Jason D Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. *Advances in Neural Information Processing Systems*, 31, 2018.

Robert M May. Simple mathematical models with very complicated dynamics. *Nature*, 261:459, 1976.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Yann Ollivier. Riemannian metrics for neural networks i: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153, 2015a.

Yann Ollivier. Riemannian metrics for neural networks ii: recurrent networks and learning symbolic data sequences. *Information and Inference: A Journal of the IMA*, 4(2):154–193, 2015b.

Yang Song, Jiaming Song, and Stefano Ermon. Accelerating natural gradient with higher-order invariance. In *International Conference on Machine Learning*, pages 4713–4722. PMLR, 2018.

Chongli Qin, Yan Wu, Jost Tobias Springenberg, Andrew Brock, Jeff Donahue, Timothy P Lillicrap, and Pushmeet Kohli. Training generative adversarial networks by solving ordinary differential equations. 2020.

Mihaela C Rosca, Yan Wu, Benoit Dherin, and David Barrett. Discretization drift in two-player games. In *International Conference on Machine Learning*, pages 9064–9074. PMLR, 2021.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Sho Yaida. Fluctuation-dissipation relations for stochastic gradient descent. In *International Conference on Learning Representations*, 2018.

Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg New York, 1996.

HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The computer journal*, 3(3):175–184, 1960.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

Vardan Papyan. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*, 2018.

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

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.

Jonas Geiping, Micah Goldblum, Phillip E Pope, Michael Moeller, and Tom Goldstein. Stochastic training is not necessary for generalization. *arXiv preprint arXiv:2109.14119*, 2021.

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

Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations*, 2019.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the relation between the sharpest directions of dnn loss and the sgd step length. *arXiv preprint arXiv:1807.05031*, 2018.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015.

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

Daniel Kunin, Javier Sagastuy-Brena, and Hidenori Tanaka Ganguli, Surya Daniel L.K. Yamins. Symmetry, conservation laws, and learning dynamics in neural networks. In *International Conference on Learning Representations*, 2021.

Jaehoon Lee, Samuel Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *Advances in Neural Information Processing Systems*, 33:15156–15172, 2020.

Bin Shi, Simon S Du, Michael I Jordan, and Weijie J Su. Understanding the acceleration phenomenon via high-resolution differential equations. *Mathematical Programming*, pages 1–70, 2021.

Nikola B Kovachki and Andrew M Stuart. Continuous time analysis of momentum methods. *Journal of Machine Learning Research*, 22(17):1–40, 2021.

Guilherme França, Michael I Jordan, and René Vidal. On dissipative symplectic integration with applications to gradient-based optimization. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(4):043402, 2021.

Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and adaptive stochastic gradient algorithms. In *International Conference on Machine Learning*, volume 70, pages 2101–2110, 2017.

Tatjana Chavdarova, Michael I Jordan, and Manolis Zampetakis. Last-iterate convergence of saddle point optimizers via high-resolution differential equations. *arXiv preprint arXiv:2112.13826*, 2021.

Kwangjun Ahn, Jingzhao Zhang, and Suvrit Sra. Understanding the unstable convergence of gradient descent. *arXiv preprint arXiv:2204.01050*, 2022.

Sanjeev Arora, Zhiyuan Li, and Abhishek Panigrahi. Understanding gradient descent on edge of stability in deep learning. *arXiv preprint arXiv:2205.09745*, 2022.

Chao Ma, Lei Wu, and Lexing Ying. The multiscale structure of neural network loss functions: The effect on optimization and origin. *arXiv preprint arXiv:2204.11326*, 2022.

Lei Chen and Joan Bruna. On gradient descent convergence beyond the edge of stability. *arXiv preprint arXiv:2206.04172*, 2022.

Liu Ziyin, Kangqiao Liu, Takashi Mori, and Masahito Ueda. Strength of minibatch noise in sgd. *arXiv preprint arXiv:2102.05375*, 2021b.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

Simran Kaur, Jeremy Cohen, and Zachary C Lipton. On the maximum hessian eigenvalue and generalization. *arXiv preprint arXiv:2206.10654*, 2022.

Yann LeCun, Lawrence D Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261 (276):2, 1995.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Appendices

A Proofs **28**

A.1 BEA proof structure 28

A.2 Third order flow 29

A.3 Higher order terms of the form $(\nabla_\theta f)^n f$ or $\left(\nabla_\theta^2 E\right)^n \nabla_\theta E$ 30

A.4 The Jacobian of the principal flow at critical points 37

A.5 Linearization results around critical points 38

A.6 The solution of the principal flow for quadratic losses 39

A.7 Comparison with a discrete time approach 40

A.8 Multiple gradient descent steps 40

A.9 Approximations to per iteration drift for gradient descent and momentum 41

B Non-principal terms **42**

C Experimental details **42**

D Additional experimental results **45**

A Proofs

A.1 BEA proof structure

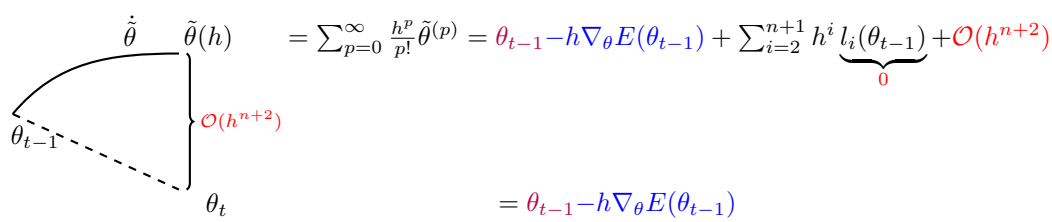


Figure 24: BEA finds continuous modified flows which describe the gradient descent update with learning rate h with an error of $\mathcal{O}(h^{n+2})$. We identify $f_1, \cdots f_n$ so that terms of order $\mathcal{O}(h^p), 2 \leq p \leq n+1$ in $\tilde{\theta}(h)$ are 0.

General structure. The goal of BEA is to find the functions f_1, ... f_n such that the flow

$$\dot{\tilde{\theta}} = -\nabla_\theta E + hf_1(\theta) + \cdots + h^n f_n(\theta) \tag{22}$$

has an error $\|\theta_t - \tilde{\theta}(h)\|$ of order $\mathcal{O}(h^{n+2})$ after 1 gradient descent step of learning rate h. To do so requires multiple steps (visualized in Figure 24):

1. Expand $\tilde{\theta}(h)$ via a Taylor expansion in h: $\tilde{\theta}(h) = \sum_{p=0}^{\infty} \frac{h^p}{p!} \tilde{\theta}^{(p)}$;

2. Expand each $\tilde{\theta}^{(p)}$ up to order $\mathcal{O}(h^{n+2-p})$ as a function of f_1, ... f_n via the chain rule;

3. Group together terms of the same order in h in the expansion, up to order $n+1$.

$$\tilde{\theta}(h) = \sum_{i=0}^{n} h^i l_i(\theta) + \mathcal{O}(h^{n+1}) = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) + \sum_{i=2}^{n+1} h^i l_i(\theta_{t-1}) + \mathcal{O}(h^{n+2}) \tag{23}$$

4. Compare the above update with the gradient descent update $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ and conclude that $l_i = 0$, $\forall i \in \{2, n\}$. Use this to identify f_1, ... f_n.

Notation and context: all proofs below apply to general Euler updates not only gradient descent. We thus assume an update function f with the Euler step $\theta_t = \theta_{t-1} + hf(\theta_{t-1})$. We can then use BEA to find the higher order correction terms describing the Euler discrete update up to a certain order, and replace $f = -\nabla_\theta E$ to obtain the corresponding results for gradient descent. When we perform a Taylor expansion in h we often drop in notation the evaluation at $h = 0$, and we will make that implicit.

A.2 Third order flow

Theorem A.1 *The modified flow*

$$\dot{\theta} = f - h\frac{1}{2}\nabla_\theta f f + h^2\left(\frac{1}{3}(\nabla_\theta f)^2 f + \frac{1}{12} f^T(\nabla_\theta^2 f) f\right) \tag{24}$$

with $\theta(0) = \theta_{t-1}$ follows an Euler update $\theta_t = \theta_{t-1} + hf(\theta_{t-1})$ with an error of $\mathcal{O}(h^4)$ after 1 gradient descent step.

Proof: Since we are using BEA, we wil be looking for functions f_1 and f_2 such that the modified flow:

$$\dot{\theta} = f + hf_1 + h^2 f_2 \tag{25}$$

follow the steps of GD with an error up to $\mathcal{O}(h^4)$. We now perform a Taylor expansion of step size h of the above modified flow to be able to see the displacement in that time up to order $\mathcal{O}(h^4)$.

We obtain (all function evaluations of f and f_i are at θ_{t-1} which we omit for simplicity, and annotate proof steps, CR denotes Chain Rule):

$$\theta(h) = \theta(0) + h\dot{\theta}(0) + \frac{1}{2}h^2\ddot{\theta}(0) + \frac{1}{6}h^3\dddot{\theta}(0) + \mathcal{O}(h^4) \tag{26}$$

$$= \theta_{t-1} + h\left[(f + hf_1 + h^2 f_2)\right] + \frac{1}{2}h^2\frac{d}{dt}(f + hf_1 + h^2 f_2) + \frac{1}{6}h^3\dddot{\theta} + \mathcal{O}(h^4) \qquad \text{(using Eq 25, 27)}$$

$$= \theta_{t-1} + hf + h^2 f_1 + h^3 f_2 + \frac{1}{2}h^2(\nabla_\theta(f + hf_1 + h^2 f_2))(f + hf_1 + h^2 f_2) + \frac{1}{6}h^3\dddot{\theta} + \mathcal{O}(h^4) \qquad \text{(CR, 28)}$$

$$= \theta_{t-1} + hf + h^2 f_1 + h^3 f_2 + \frac{1}{2}h^2(\nabla_\theta f f + h\nabla_\theta f f_1 + h\nabla_\theta f_1 f) + \frac{1}{6}h^3\dddot{\theta} + \mathcal{O}(h^4) \tag{29}$$

$$= \theta_{t-1} + hf + h^2 f_1 + h^3 f_2 + \frac{1}{2}h^2(\nabla_\theta f f + h\nabla_\theta f f_1 + h\nabla_\theta f_1 f) + \frac{h^3}{6}\frac{d}{dt}\left[\nabla_\theta f f + h\nabla_\theta f f_1 + h\nabla_\theta f_1 f\right] + \mathcal{O}(h^4) \tag{30}$$

$$= \theta_{t-1} + hf + h^2 f_1 + h^3 f_2 + \frac{1}{2}h^2(\nabla_\theta f f + h\nabla_\theta f f_1 + h\nabla_\theta f_1 f) + \frac{1}{6}h^3\frac{d}{dt}\left[\nabla_\theta f f\right] + \mathcal{O}(h^4) \tag{31}$$

$$= \theta_{t-1} + hf + h^2 f_1 + h^3 f_2 + \frac{1}{2}h^2(\nabla_\theta f f + h\nabla_\theta f f_1 + h\nabla_\theta f_1 f) + \frac{1}{6}h^3\nabla_\theta\left[\nabla_\theta f f\right](f + hf_1 + h^2 f_2) + \mathcal{O}(h^4) \qquad \text{(CR, 32)}$$

$$= \theta_{t-1} + hf + h^2 f_1 + h^3 f_2 + \frac{1}{2}h^2(\nabla_\theta f f + h\nabla_\theta f f_1 + h\nabla_\theta f_1 f) + \frac{1}{6}h^3\nabla_\theta\left[\nabla_\theta f f\right] f + \mathcal{O}(h^4) \tag{33}$$

If we match the terms with the Euler update, we obtain that:

$$f_1 = -\frac{1}{2}\nabla_\theta f f \tag{34}$$

and

$$f_2 = -\frac{1}{2}\left[\nabla_\theta f f_1 + \nabla_\theta f_1 f\right] - \frac{1}{6}\nabla_\theta\left[\nabla_\theta f f\right] f \tag{35}$$

$$f_2 = -\frac{1}{2}\left[\nabla_\theta f(-\frac{1}{2}\nabla_\theta f f) + \nabla_\theta(-\frac{1}{2}\nabla_\theta f f) f\right] - \frac{1}{6}\nabla_\theta\left[\nabla_\theta f f\right] f \tag{36}$$

$$f_2 = \frac{1}{4}(\nabla_\theta f)^2 f + \frac{1}{4}\nabla_\theta(\nabla_\theta f f) f - \frac{1}{6}\nabla_\theta\left[\nabla_\theta f f\right] f \tag{37}$$

$$f_2 = \frac{1}{4}(\nabla_\theta f)^2 f + \frac{1}{12}\nabla_\theta(\nabla_\theta f f) f \tag{38}$$

$$f_2 = \frac{1}{4}(\nabla_\theta f)^2 f + \frac{1}{12}\left((\nabla_\theta f)^2 f + f^T \nabla_\theta^2 f\right) \tag{39}$$

$$f_2 = \frac{1}{3}(\nabla_\theta f)^2 f + \frac{1}{12} f^T (\nabla_\theta^2 f) f \tag{40}$$

Replacing f_1 and f_2 in Equation 25 leads to the desired Equation 24.

□

In the case of gradient descent, substituing $f = -\nabla_\theta E$ in f_1 and f_2, we obtain (Equation 8):

$$\dot{\theta} = -\nabla_\theta E - \frac{h}{2}\nabla_\theta^2 E \nabla_\theta E - h^2\left(\frac{1}{3}(\nabla_\theta^2 E)^2 \nabla_\theta E + \frac{1}{12}\nabla_\theta E^T (\nabla_\theta^3 E)\nabla_\theta E\right) \tag{41}$$

A.3 Higher order terms of the form $(\nabla_\theta f)^n f$ or $\left(\nabla_\theta^2 E\right)^n \nabla_\theta E$

Theorem A.2 *The modified vector field with an error of order $\mathcal{O}(h^{p+1})$ to the Euler update $\theta_t = \theta_{t-1} + f(\theta_{t-1})$ has the form:*

$$\dot{\theta} = \sum_{n=0}^{p} \frac{(-1)^n}{n+1}(\nabla_\theta f)^n f + \mathcal{C}(\nabla_\theta^2 f) \tag{42}$$

where $\mathcal{C}(\nabla_\theta^2 f)$ denotes a class of functions defined as a sum of individual terms each containing higher than first order derivatives applied to f.

Proof: Consider the modified ODE given by BAE:

$$\dot{\theta} = \tilde{f} = f + h f_1 + h^2 f_2 + ... + h^n f_n + ... \tag{43}$$

For the proof we need to show that the only terms not involving higher order derivatives are of the form $(\nabla_\theta f)^n f$ and we need to find the coefficients of $(\nabla_\theta f)^n f$ in f_n, which we will call c_n. We already know that $c_0 = 1$, $c_1 = -\frac{1}{2}$ (Eq. 34) and $c_3 = \frac{1}{3}$ (Eq. 40). We use the general structure provided in Section A.1.

We start by expanding $\theta(h)$ via Taylor expansions:

$$\theta(h) = \theta(0) + h\dot{\theta} + \sum_{k=2}^{\infty}\frac{1}{k!}\theta^{(k)} = \theta_{t-1} + h f(\theta_{t-1}) + \sum_{n=1}^{\infty} h^{n+1} f_n + \sum_{k=2}^{\infty}\frac{h^k}{k!}\theta^{(k)} \tag{44}$$

We now express in $\theta^{(k)}$ in terms of the wanted quantities f_n via Lemma A.3:

$$\theta^{(k)} = \sum_{m\geq 0} h^m \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i \quad k \geq 2 \tag{45}$$

By replacing Eq 45 in Eq 44 we obtain:

$$\theta(h) = \theta_{t-1} + hf(\theta_{t-1}) + \sum_{n=1}^{\infty} h^{n+1} f_n(\theta_{t-1}) + \sum_{k=2}^{\infty} \frac{h^k}{k!} \sum_{m\geq 0} h^m \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j(\theta_{t-1}) f_i(\theta_{t-1}) \tag{46}$$

$$= \theta_{t-1} + hf(\theta_{t-1}) + \sum_{n=1}^{\infty} h^{n+1} f_n(\theta_{t-1}) + \sum_{m\geq 0} \sum_{k=2}^{\infty} \frac{h^{k+m}}{k!} \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j(\theta_{t-1}) f_i(\theta_{t-1}) \tag{47}$$

We note that due to the derivative wrt to t and a dependence of h in the modified vector field, there will be a dependence of h in $\frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i$. To highlight this dependence on h we introduce the notation:

$$\frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i = \sum_p h^p A^p_{k,m} \tag{48}$$

leading to:

$$\theta(h) = \theta_{t-1} + hf(\theta_{t-1}) + \sum_{n=1}^{\infty} h^{n+1} f_n(\theta_{t-1}) + \sum_{m\geq 0} \sum_{k=2}^{\infty} \frac{h^{k+m}}{k!} \sum_p h^p A^p_{k,m}(\theta_{t-1}) \tag{49}$$

Since the Euler update $\theta_t = \theta_{t-1} + hf(\theta_{t-1})$ does not involve terms of order higher than h the f_n terms are obtained by cancelling the terms of order $\mathcal{O}(h^{n+1})$ in $\theta(h)$, we have that for $k + m + p = n + 1$:

$$f_n = -\sum_{m\geq 0} \sum_{k=2}^{\infty} \frac{1}{k!} A^{n+1-(k+m)}_{k,m} \tag{50}$$

Since $A^{n+1-(k+m)}_{k,m}$ depends on the functions f_n, we have found a recursive relation for f_n. We now use induction to show that

$$f_n = c_n (\nabla_\theta f)^n f + \mathcal{C}(\nabla^2_\theta f) \tag{51}$$

where $\mathcal{C}(\nabla^2_\theta f)$ denotes a class of functions defined as a sum of individual terms each containing higher than first order derivatives applied to f (Lemma A.4). The same proof shows the recurrence relation for c_n:

$$c_n = -\sum_{k=2}^{n+1} \frac{1}{k!} \sum_{l_1+l_2+\ldots+l_k=n-k+1} c_{l_1} c_{l_2} \ldots c_{l_k} \tag{52}$$

with solution $c_n = \frac{(-1)^n}{n+1}$ (Lemma A.5)

We have found that each f_n can be written as $\frac{(-1)^n}{n+1} (\nabla_\theta f)^n f + \mathcal{C}(\nabla^2_\theta f)$. Replacing this in Eq.43 concludes the proof.

□

In the case of gradient descent, where $f = -\nabla_\theta E$ we have that the modified flow has the form

$$\dot{\theta} = \sum_{n=0}^{p} \frac{(-1)^n}{n+1} (-\nabla_\theta f)^n (-\nabla_\theta E) + \mathcal{C}(\nabla_\theta^2 f) \tag{53}$$

$$= \sum_{n=0}^{p} \frac{(-1)^n}{n+1} (-\nabla_\theta^2 E)^n (-\nabla_\theta E) + \mathcal{C}(\nabla_\theta^2 f) \tag{54}$$

$$= \sum_{n=0}^{p} \frac{-1}{n+1} (\nabla_\theta^2 E)^n (\nabla_\theta E) + \mathcal{C}(\nabla_\theta^2 f) \tag{55}$$

$$\tag{56}$$

which is the statement of Theorem 3.1 in the main paper.

Lemma A.3

$$\theta^{(k)} = \sum_{m \geq 0} h^m \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i \quad k \geq 2 \tag{57}$$

Proof:

$$\dot{\theta} = \tilde{f} \tag{58}$$

$$\ddot{\theta} = \nabla_\theta \tilde{f} \tilde{f} \tag{59}$$

$$\theta^{(k)} = \frac{d^{k-2}}{dt^{k-2}} \dot{\theta} = \frac{d^{k-2}}{dt^{k-2}} \nabla_\theta \tilde{f} \tilde{f} \tag{60}$$

We have that

$$\nabla_\theta \tilde{f} \tilde{f} = \left(\sum_i h^i \nabla_\theta f_i \right) \left(\sum_j h^j f_j \right) = \sum_{m \geq 0} h^m \sum_{i+j=m} \nabla_\theta f_i f_j \tag{61}$$

which concludes the proof. We note that this is an adaptation of Lemma A.2 from (Barrett and Dherin, 2021). We need this adaptation to avoid assuming that f is a symmetric vector field - i.e that $\tilde{f} \nabla_\theta \tilde{f} = \frac{1}{2} \nabla_\theta \left\| \tilde{f} \right\|^2$. □

Lemma A.4 *By induction we have that:*

$$f_n = c_n (\nabla_\theta f)^n f + \mathcal{C}(\nabla_\theta^2 f) \tag{62}$$

with

$$c_n = -\sum_{k=2}^{n+1} \frac{1}{k!} \sum_{l_1 + l_2 + \ldots + l_k = n-k+1} c_{l_1} c_{l_2} \ldots c_{l_k} \tag{63}$$

Proof: **Base step**: This is true for $n = 1, 2$ and 3, for which we computed all terms in the BEA expansion.

Induction step: We know that (Equation 48):

$$f_n = -\sum_{m \geq 0} \sum_{k=2}^{\infty} \frac{1}{k!} A_{k,m}^{n+1-(k+m)} \tag{64}$$

where $A^p_{k,m}$ is defined as the coefficient of h^p in:

$$\frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i = \sum_p h^p A^p_{k,m} \tag{65}$$

Under the induction hypothesis we can use that

$$f_i = c_i (\nabla_\theta f)^i f + \mathcal{C}(\nabla^2_\theta f) \qquad \forall i < n \tag{66}$$

And thus that the flow is of the form:

$$\dot{\theta} = f + \sum_i^{n-1} c_i (\nabla_\theta f)^i f + \mathcal{C}(\nabla^2_\theta f) + \mathcal{O}(h^n) \tag{67}$$

To find f_n (Eq 64) we need to find $A^{n+1-(k+m)}_{k,m}$ with $k \geq 2$ and $m \geq 0$. We thus need to find $A^p_{k,m}$ with $p \leq n-1$ (since $p \leq (n+1) - (k+m)$ and $k \geq 2, m \geq 0 \implies p \leq n-1$). We do so expanding $\frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i$ under the induction step. Since $i+j=m$ and $k+m \leq n+1$ and $k \geq 2$, we have that $m \leq n-1$ and thus $i, j \leq n-1$ and thus we can apply the induction hypothesis for f_i and f_j.

We annotate the steps with IH (Induction Hypothesis, often by using the form in Eq. 67) and CR (Chain Rule) and S (Simplifying or grouping terms of $\mathcal{C}(\nabla^2_\theta f)$) by using that products and sum of terms in this function class also belong in this function class).

$$\frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \nabla_\theta f_j f_i \tag{68}$$

$$= \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \left(\nabla_\theta (c_j (\nabla_\theta f)^j f + \mathcal{C}(\nabla_\theta^2 f)\right) \left(c_i (\nabla_\theta f)^i f + \mathcal{C}(\nabla_\theta^2 f)\right) \tag{IH, 69}$$

$$= \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \left(c_j (\nabla_\theta f)^{j+1} + c_j f^T \nabla_\theta (\nabla_\theta f)^j + \mathcal{C}(\nabla_\theta^2 f)\right) \left(c_i (\nabla_\theta f)^i f + \mathcal{C}(\nabla_\theta^2 f)\right) \tag{CR, 70}$$

$$= \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} \left(c_j (\nabla_\theta f)^{j+1} + \mathcal{C}(\nabla_\theta^2 f)\right) \left(c_i (\nabla_\theta f)^i f + \mathcal{C}(\nabla_\theta^2 f)\right) \tag{S, 71}$$

$$= \frac{d^{k-2}}{dt^{k-2}} \sum_{i+j=m} c_i c_j (\nabla_\theta f)^{m+1} f + \mathcal{C}(\nabla_\theta^2 f) \tag{S, 72}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} \frac{d}{dt} c_i c_j (\nabla_\theta f)^{m+1} f + \mathcal{C}(\nabla_\theta^2 f) \tag{73}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} \frac{d\theta}{dt} \frac{d}{d\theta} \left(c_i c_j (\nabla_\theta f)^{m+1} f + \mathcal{C}(\nabla_\theta^2 f)\right) \tag{CR, 74}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} f_{l_1} + \mathcal{O}(h^n)) \frac{d}{d\theta} \left(c_i c_j (\nabla_\theta f)^{m+1} f + \mathcal{C}(\nabla_\theta^2 f)\right) \tag{Eq. 67, 75}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} f_{l_1}) \frac{d}{d\theta} \left(c_i c_j (\nabla_\theta f)^{m+1} f + \mathcal{C}(\nabla_\theta^2 f)\right) + \mathcal{O}(h^n) \tag{76}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} f_{l_1}) \left(c_i c_j (\nabla_\theta f)^{m+2} + \mathcal{C}(\nabla_\theta^2 f)\right) + \mathcal{O}(h^n) \tag{CR,S, 77}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} c_{l_1} (\nabla_\theta f)^{l_1} f + \mathcal{C}(\nabla_\theta^2 f) \left(c_i c_j (\nabla_\theta f)^{m+2} + \mathcal{C}(\nabla_\theta^2 f)\right) + \mathcal{O}(h^n) \tag{IH, 78}$$

$$= \frac{d^{k-3}}{dt^{k-3}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} c_i c_j c_{l_1} (\nabla_\theta f)^{l_1+m+2} f + \mathcal{C}(\nabla_\theta^2 f)) + \mathcal{O}(h^n) \tag{S, 79}$$

$$= \frac{d^{k-4}}{dt^{k-4}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} c_i c_j c_{l_1} (\sum_{l_2=0}^{n-1} h^{l_2} f_{l_2}) \frac{d}{d\theta} \left((\nabla_\theta f)^{l_1+m+2} f + \mathcal{C}(\nabla_\theta^2 f))\right) + \mathcal{O}(h^n) \tag{CR, Eq. 67, 80}$$

$$= \frac{d^{k-4}}{dt^{k-4}} \sum_{i+j=m} (\sum_{l_1=0}^{n-1} h^{l_1} c_i c_j c_{l_1} (\sum_{l_2=0}^{n-1} h^{l_2} c_{l_2} (\nabla_\theta f)^{l_2} f + \mathcal{C}(\nabla_\theta^2 f)) \left((\nabla_\theta f)^{l_1+m+3} + \mathcal{C}(\nabla_\theta^2 f)\right) + \mathcal{O}(h^n) \tag{IH, 81}$$

$$= \frac{d^{k-4}}{dt^{k-4}} \sum_{i+j=m} \sum_{l_1=0}^{n-1} \sum_{l_2=0}^{n-1} h^{l_1+l_2} c_i c_j c_{l_1} c_{l_2} (\nabla_\theta f)^{l_1+l_2+m+3} f + \mathcal{C}(\nabla_\theta^2 f) + \mathcal{O}(h^n) \tag{S, 82}$$

$$= \sum_{i+j=m} \sum_{l_1=0}^{n-1} \sum_{l_2=0}^{n-1} \dots \sum_{l_{k-2}=0}^{n-1} h^{l_1+l_2+\dots+l_{k-2}} c_i c_j c_{l_1} c_{l_2} \dots c_{l_k-2} (\nabla_\theta f)^{l_1+l_2+\dots+l_{k-2}+m+k-1} f + \mathcal{C}(\nabla_\theta^2 f) + \mathcal{O}(h^n) \tag{83}$$

$$= \sum_{i+j=m} \sum_{\substack{0 \le l_1, \dots, l_{k-2} \le n-1, \\ l_1+l_2+\dots l_{k-2}=p}} h^p c_i c_j c_{l_1} c_{l_2} \dots c_{l_k-2} (\nabla_\theta f)^{p+m+k-1} f + \mathcal{C}(\nabla_\theta^2 f) + \mathcal{O}(h^n) \tag{84}$$

By extracting the terms of $\mathcal{O}(h^p)$ with $p \leq n-1$ from the above we can now conclude that (note that since we are now only concerned with $p \leq n-1$ we drop the $l_i \leq n-1$ since it is implied from $l_1 + l_2 + ... l_{k-2} = p$):

$$A^p_{k,m} = \sum_{i+j=m} \sum_{\substack{l_1,...,l_{k-2} \geq 0, \\ l_1+l_2+...l_{k-2}=p}} c_i c_j c_{l_1} c_{l_2} ... c_{l_k-2} (\nabla_\theta f)^{p+m+k-1} f + \mathcal{C}(\nabla^2_\theta f) \qquad \forall p \leq n-1 \tag{85}$$

Replacing this in f_n:

$$f_n = -\sum_{m \geq 0} \sum_{k=2}^{\infty} \frac{1}{k!} A^{n+1-(k+m)}_{k,m} \tag{86}$$

$$= -\sum_{m \geq 0} \sum_{k=2}^{\infty} \frac{1}{k!} \sum_{i+j=m} \sum_{\substack{l_1,...,l_{k-2} \geq 0, \\ l_1+l_2+...l_{k-2}=n+1-(k+m)}} c_i c_j c_{l_1} c_{l_2} ... c_{l_k-2} (\nabla_\theta f)^{n+1-(k+m)+k+m-1} f + \mathcal{C}(\nabla^2_\theta f) \tag{87}$$

$$= -\sum_{m \geq 0} \sum_{k=2}^{\infty} \frac{1}{k!} \sum_{i+j=m} \sum_{\substack{l_1,...,l_{k-2} \geq 0, \\ l_1+l_2+...l_{k-2}=n+1-(k+m)}} c_i c_j c_{l_1} c_{l_2} ... c_{l_k-2} (\nabla_\theta f)^{n} f + \mathcal{C}(\nabla^2_\theta f) \tag{88}$$

$$= -\sum_{k=2}^{\infty} \frac{1}{k!} \sum_{m \geq 0} \sum_{\substack{l_1,...,l_k \geq 0, \\ l_1+l_2+...l_k=n+1-k}} c_{l_1} c_{l_2} ... c_{l_k} (\nabla_\theta f)^{n} f + \mathcal{C}(\nabla^2_\theta f) \tag{89}$$

We can now say that $f_n = c_n (\nabla_\theta f)^n f + \mathcal{C}(\nabla^2_\theta f)$. Not only that, we have now found the recurrence relation we were seeking:

$$c_n = -\sum_{k=2}^{n+1} \frac{1}{k!} \sum_{l_1+l_2+...+l_k=n-k+1} c_{l_1} c_{l_2} ... c_{l_k} \tag{90}$$

□

Lemma A.5 *The solution to the recurrence relation*

$$c_n = -\sum_{k=2}^{n+1} \frac{1}{k!} \sum_{l_1+l_2+...+l_k=n-k+1} c_{l_1} c_{l_2} ... c_{l_k} \tag{91}$$

with $c_0 = 1$ *is*

$$c_n = \frac{(-1)^n}{n+1} \tag{92}$$

Proof: We will use generating functions to solve the recurrence. Let $c(x)$ be the generating function of c_n:

$$c(x) = c_0 + \sum_{n=1}^{\infty} c_n x^n \tag{93}$$

We have that:

$$c(x) = 1 - \sum_{n=1}^{\infty} x^n \left(\sum_{k=2}^{n+1} \frac{1}{k!} \sum_{l_1+l_2+\ldots+l_k=n-k+1} c_{l_1} c_{l_2} \ldots c_{l_k} \right) \tag{94}$$

$$= 1 - \frac{1}{x} \sum_{n=1}^{\infty} x^{n+1} \left(\sum_{k=2}^{n+1} \frac{1}{k!} \sum_{l_1+l_2+\ldots+l_k=n-k+1} c_{l_1} c_{l_2} \ldots c_{l_k} \right) \tag{95}$$

$$= 1 - \frac{1}{x} \sum_{n=1}^{\infty} \left(\sum_{k=2}^{n+1} \frac{x^k}{k!} x^{n+1-k} \sum_{l_1+l_2+\ldots+l_k=n-k+1} c_{l_1} c_{l_2} \ldots c_{l_k} \right) \tag{96}$$

$$= 1 - \frac{1}{x} \sum_{n=1}^{\infty} \left(\sum_{k=2}^{n+1} \frac{x^k}{k!} \sum_{l_1+l_2+\ldots+l_k=n-k+1} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \right) \tag{97}$$

$$= 1 - \frac{1}{x} \sum_{k=2}^{\infty} \frac{x^k}{k!} \left(\sum_{l_1=0}^{\infty} \sum_{l_2=0}^{\infty} \sum_{l_k=0}^{\infty} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \right) \tag{98}$$

$$= 1 - \frac{1}{x} \sum_{k=2}^{\infty} \frac{x^k}{k!} \left(\sum_{l_1=0}^{\infty} c_{l_1} x^{l_1} \sum_{l_2=0}^{\infty} c_{l_2} x^{l_2} \sum_{l_k=0}^{\infty} c_{l_k} x^{l_k} \right) \tag{99}$$

$$= 1 - \frac{1}{x} \sum_{k=2}^{\infty} \frac{x^k}{k!} c(x)^k \tag{100}$$

$$= 1 - \frac{1}{x} (e^{xc(x)} - 1 - xc(x)) \tag{101}$$

Where Eq 98 can be obtained via the following reasoning:

$$\sum_{k=2}^{\infty} \frac{x^k}{k!} \left(\sum_{l_1=0}^{\infty} \sum_{l_2=0}^{\infty} \sum_{l_k=0}^{\infty} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \right) \tag{102}$$

$$= \sum_{k=2}^{\infty} \frac{x^k}{k!} \left(\sum_{m=0}^{\infty} \sum_{l_1+l_2+\ldots l_k=m} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \right) \tag{103}$$

$$= \sum_{k=2}^{\infty} \frac{x^k}{k!} \left(\sum_{n=k-1}^{\infty} \sum_{l_1+l_2+\ldots l_k=n+1-k} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \right) \tag{104}$$

$$= \sum_{k=2}^{\infty} \sum_{n=k-1}^{\infty} \frac{x^k}{k!} \sum_{l_1+l_2+\ldots l_k=n+1-k} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \tag{105}$$

$$= \sum_{n=1}^{\infty} \sum_{k=2}^{n+1} \frac{x^k}{k!} \sum_{l_1+l_2+\ldots l_k=n+1-k} c_{l_1} x^{l_1} c_{l_2} x^{l_2} \ldots c_{l_k} x^{l_k} \tag{106}$$

Solving for $c(x)$ we obtain:

$$c(x) = 1 - \frac{1}{x} (e^{xc(x)} - 1 - xc(x)) \tag{107}$$

$$c(x) = 1 - \frac{1}{x} e^{xc(x)} + \frac{1}{x} + c(x)) \tag{108}$$

$$0 = 1 - \frac{1}{x} e^{xc(x)} + \frac{1}{x} \tag{109}$$

$$e^{xc(x)} = x + 1 \tag{110}$$

$$xc(x) = \log(x+1) \tag{111}$$

$$c(x) = \frac{\log(x+1)}{x} \tag{112}$$

We now perform the series expansion for $c(x)$:

$$c(x) = \frac{\log(x+1)}{x} = \frac{\sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} x^n}{x} = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} x^{n-1} = \sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} x^n \tag{113}$$

We thus have that $c_n = \frac{(-1)^n}{n+1}$ which finishes the proof.

□

A.4 The Jacobian of the principal flow at critical points

Under the principal flow, we have that the Jacobian at a critical point θ^* with the eigenvalues and eigenvector of $\nabla_\theta^2 E(\theta^*)$ denoted as λ_i^* and u_i^*, respectively. We will use that $\nabla_\theta E(\theta^*) = 0$.

$$\begin{aligned}
J_{PF}(\theta^*) &= \sum_{i=1}^{D} \frac{d(\nabla_\theta E^T u_i)\frac{\log(1-h\lambda_i)}{h\lambda_i} u_i}{d\theta}(\theta^*) && (114)\\
&= \sum_{i=1}^{D} \frac{d(\nabla_\theta E^T u_i)\frac{\log(1-h\lambda_i)}{h\lambda_i}}{d\theta}(\theta^*) u_i^{*T} + \sum_{i=1}^{D} \underbrace{\nabla_\theta E(\theta^*)^T}_{0} u_i^* \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \frac{du_i}{d\theta} && (115)\\
&= \sum_{i=1}^{D} \frac{d(\nabla_\theta E^T u_i)\frac{\log(1-h\lambda_i)}{h\lambda_i}}{d\theta}(\theta^*) u_i^{*T} && (116)\\
&= \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \frac{d(\nabla_\theta E^T u_i)}{d\theta}(\theta^*) u_i^{*T} + \sum_{i=1}^{D} (\underbrace{\nabla_\theta E(\theta^*)^T}_{0} u_i^*) \frac{d\frac{\log(1-h\lambda_i)}{h\lambda_i}}{d\theta} u_i^{*T} && (117)\\
&= \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \frac{d(\nabla_\theta E^T u_i)}{d\theta}(\theta^*) u_i^{*T} && (118)\\
&= \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \left(\frac{d\nabla_\theta E}{d\theta}(\theta^*)\right)^T u_i^* u_i^{*T} + \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \left(\frac{du_i}{d\theta}(\theta^*)\right)^T \underbrace{\nabla_\theta E(\theta^*)}_{0} u_i^{*T} && (119)\\
&= \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \nabla_\theta^2 E(\theta^*) u_i^* u_i^{*T} && (120)\\
&= \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h\lambda_i^*} \lambda_i^* u_i^* u_i^{*T} && (121)\\
&= \sum_{i=1}^{D} \frac{\log(1-h\lambda_i^*)}{h} u_i^* u_i^{*T} && (122)
\end{aligned}$$

This shows that the eigenvalues of the Jacobian for the PF are $\frac{\log(1-h\lambda_i^*)}{h}$. Thus thus local minima where $\lambda_i^* \geq 0$ are only attractive if $h\lambda_i^* < 2$.

A.4.1 Comparison with the Jacobian of other flows

We show here the Jacobian of the NGF:

$$J_{NGF}(\theta^*) = -\nabla_\theta^2 E(\theta^*) = -\sum_{i=1}^{D} \lambda_i^* u_i^* u_i^{*T} \tag{123}$$

and thus the eigenvalues of the Jacobian for the NGF are $-\lambda_i^*$. Thus local minima where $\lambda_i^* \geq 0 \implies -\lambda_i^* \leq 0$ are attractive for the NGF.

For the IGR flow, we have:

$$J_{IGR}(\theta^*) = -\frac{d\left(\nabla_\theta E + \frac{h^2}{2}\nabla_\theta^2 E \nabla_\theta E\right)}{d\theta}(\theta^*) \tag{124}$$

$$= -\frac{d\nabla_\theta E}{d\theta}(\theta^*) - \frac{h^2}{2}\frac{d\nabla_\theta^2 E \nabla_\theta E}{d\theta}(\theta^*) \tag{125}$$

$$= -\sum_{i=1}^{D} \lambda_i^* u_i^* u_i^{*T} - \frac{h^2}{2}\left(\nabla_\theta E^T \nabla_\theta^3 E + \nabla_\theta^2 E \frac{d\nabla_\theta E}{d\theta}\right)(\theta^*) \tag{126}$$

$$= -\sum_{i=1}^{D} \lambda_i^* u_i^* u_i^{*T} - \frac{h^2}{2}\underbrace{\nabla_\theta E(\theta^*)^T}_{0}\nabla_\theta^3 E(\theta^*) - \frac{h^2}{2}\nabla_\theta^2 E\left(\theta^*\right)\nabla_\theta^2 E\left(\theta^*\right) \tag{127}$$

$$= -\sum_{i=1}^{D} \lambda_i^* u_i^* u_i^{*T} - \frac{h^2}{2}\nabla_\theta^2 E\left(\theta^*\right)\nabla_\theta^2 E\left(\theta^*\right) \tag{128}$$

$$= -\sum_{i=1}^{D} \lambda_i^* u_i^* u_i^{*T} - \sum_{i=1}^{D}\frac{h^2}{2}\lambda_i^{*2} u_i^* u_i^{*T} \tag{129}$$

$$= -\sum_{i=1}^{D}\left(\lambda_i^* + \frac{h^2}{2}\lambda_i^{*2}\right) u_i^* u_i^{*T} \tag{130}$$

and thus the eigenvalues of the Jacobian for the IGR flow are $-\left(\lambda_i^* + \frac{h^2}{2}\lambda_i^{*2}\right)$. Thus local minima where $\lambda_i^* \geq 0 \implies -\left(\lambda_i + \frac{h^2}{2}\lambda_i^{*2}\right) \leq 0$ are attractive for the IGR flow.

A.5 Linearization results around critical points

Assume we are around a critical point θ^*, i.e. $\nabla_\theta E(\theta^*) = 0$. We can write:

$$\nabla_\theta E \approx \nabla_\theta E(\theta^*) + \nabla_\theta^2 E(\theta^*)(\theta - \theta^*) = \nabla_\theta^2 E(\theta^*)(\theta - \theta^*) \tag{131}$$

Replacing this in the gradient descent updates:

$$\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) \implies \theta_t - \theta^* \approx (I - h\nabla_\theta^2 E(\theta^*))(\theta_{t-1} - \theta^*) \approx (I - h\nabla_\theta^2 E(\theta^*))^t(\theta_0 - \theta^*) \tag{132}$$

Since $\left(I - h\nabla_\theta^2 E(\theta^*)\right)^t(\theta_0 - \theta^*) = \sum_{i=1}^{D}(1 - h\lambda_i^*)^t u_i^* u_i^{*T}(\theta_0 - \theta^*)$, gradient descent converges in the limit of $t \to \infty$ when $|1 - h\lambda_i^*| < 1$, i.e. $\lambda_i < 2/h$. Thus, we obtain the same conclusion as obtained with the PF around a local minima.

We now also show how we can use these results to derived a specialized version of the PF, one that is only valid around critical points and requires the additional assumption that the Hessian of the critical point is invertible.

We start with:

$$\theta_t - \theta^* \approx (I - h\nabla_\theta^2 E(\theta^*))^t(\theta_0 - \theta^*) = \sum_{i=1}^{D}(1 - h\lambda_i^*)^t u_i^* u_i^{*T}(\theta_0 - \theta^*) \tag{133}$$

Under the above approximation we have:

$$\theta_t = \theta^* + \sum_{i=1}^{D}(1 - h\lambda_i^*)^t u_i^* u_i^{*T}(\theta_0 - \theta^*) \tag{134}$$

which we can write in continuous form as (we use that iteration n is at time $t = nh$):

$$\theta(t) = \theta^* + \sum_{i=1}^{D} (1 - h\lambda_i^*)^{t/h} u_i^* u_i^{*T} (\theta_0 - \theta^*) \tag{135}$$

$$= \theta^* + \sum_{i=1}^{D} e^{\log(1-h\lambda_i^*)t/h} u_i^* u_i^{*T} (\theta_0 - \theta^*) \tag{136}$$

and taking the derivative wrt to t:

$$\dot{\theta} = \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i^*)}{h} \underbrace{e^{\log(1-h\lambda_i^*)t/h} u_i^* u_i^{*T} (\theta_0 - \theta^*)}_{Eq.134} = \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i^*)}{h} (\theta - \theta^*) \tag{137}$$

From here, if we assume the Hessian at the critical point is invertible we can write from the approximation above (Eq. 131) that $\theta - \theta^* = (\nabla_\theta^2 E)^{-1} \nabla_\theta E = \sum_{i=1}^{D} \frac{1}{\lambda_i^*} \nabla_\theta E^T u_i^* u_i^{*T}$ and replacing this in the above we obtain:

$$\dot{\theta} = \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i^*)}{h\lambda_i^*} \nabla_\theta E^T u_i^* u_i^{*T} \tag{138}$$

which is the principal flow.

Differences with the BEA. The above analysis only applies around a critical point. What is crucial about the BEA derivation is that it allows us to specify the form of the modified flow for *any iteration of gradient descent*, something we leveraged intensely in our experiments.

A.6 The solution of the principal flow for quadratic losses

We derive the solution of the PF in the case of quadratic losses - Remark 3.1 in the main paper. We have $E(\theta) = \frac{1}{2}\theta^t A\theta + b^T\theta$ with A symmetric. $\nabla_\theta E = A\theta + b$ and $\nabla_\theta^2 E = A$. Thus λ_i and u_i are the eigenvalues and eigenvectors of A and do not change based on θ. Replacing this in the PF leads to:

$$\dot{\theta} = \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} (A\theta + b)^T u_i u_i \tag{139}$$

$$= \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} (A\theta)^T u_i u_i + \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} b^T u_i u_i \tag{140}$$

$$= \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} \theta^T A u_i u_i + \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} b^T u_i u_i \tag{141}$$

$$= \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} \lambda_i \theta^T u_i u_i + \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} b^T u_i u_i \tag{142}$$

$$= \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h} \theta^T u_i u_i + \sum_{i=1}^{D} \frac{\log(1 - h\lambda_i)}{h\lambda_i} b^T u_i u_i \tag{143}$$

$$\tag{144}$$

From here

$$(\dot{\theta^T u_i}) = \frac{\log(1 - h\lambda_i)}{h} \theta^T u_i + \frac{\log(1 - h\lambda_i)}{h\lambda_i} b^T u_i \tag{145}$$

The solution of this ODE is $(\theta^T u_i)(t) = e^{\frac{\log(1-h\lambda_i)}{h}t}\theta_0^T u_i + t\frac{\log(1-h\lambda_i)}{h\lambda_i}b^T u_i$.

Since u_i form a basis of $\mathbb{R}^D$, we can now write

$$\theta(t) = \sum_{i=1}^{D} \theta(t)^T u_i u_i \tag{146}$$

$$= \sum_{i=1}^{D} \left(e^{\frac{\log(1-h\lambda_i)}{h}t}\theta_0^T u_i + t\frac{\log(1-h\lambda_i)}{h\lambda_i}b^T u_i \right) u_i \tag{147}$$

A.7 Comparison with a discrete time approach

A.7.1 Changes in loss function

We aim to obtain similar intuition to what we have obtained from the PF by discretising the NGF using Euler steps. We have:

$$E(\theta_{t+1}) \approx E(\theta_t) - h\nabla_\theta E(\theta_t)^T \nabla_\theta E(\theta_t) + h^2/2 \nabla_\theta E(\theta_t)^T \nabla_\theta^2 E(\theta_t) \nabla_\theta E(\theta_t) \tag{148}$$

$$\approx E(\theta_t) - h\nabla_\theta E(\theta_t)^T \nabla_\theta E(\theta_t) + h^2/2 \sum_i \lambda_i \left(\nabla_\theta E(\theta_t)^T u_i\right)^2 \tag{149}$$

$$\approx E(\theta_t) - h\sum_i \left(\nabla_\theta E(\theta_t)^T u_i\right)^2 + h^2/2 \sum_i \lambda_i \left(\nabla_\theta E(\theta_t)^T u_i\right)^2 \tag{150}$$

$$\approx E(\theta_t) + \sum_i (1 - h/(2\lambda_i)) \left(\nabla_\theta E(\theta_t)^T u_i\right)^2 \tag{151}$$

thus under this approximation loss to decrease between iterations one requires $1 - h/(2\lambda_i) \leq 0$. This is consistent with the observations of the loss obtained using the PF in Equation 15.

A.7.2 The dynamics of $\nabla_\theta E^T u_i$

We now show to obtain the approximated dynamics of $\nabla_\theta E^T u_i$ obtained from the PF in Section 5 using a discrete time approach.

We have that:

$$\nabla_\theta E(\theta_{t+1}) \approx \nabla_\theta E\left(\theta_t - h\nabla_\theta E(\theta_t)\right) \tag{152}$$

$$\approx \nabla_\theta E(\theta_t) - h\nabla_\theta^2 E(\theta_t)\nabla_\theta E(\theta_t) \tag{153}$$

If we assume that the Hessian eigenvectors do not change between iterations, we have:

$$\nabla_\theta E(\theta_{t+1})^T (u_i)_{t+1} \approx \nabla_\theta E(\theta_t)^T (u_i)_{t+1} - h\nabla_\theta^2 E(\theta_t)\nabla_\theta E(\theta_t)(u_i)_{t+1} = (1 - h\lambda_i)\nabla_\theta E(\theta_t)^T (u_i)_{t+1} \tag{154}$$

and obtain the same behavior predicted in Section 5.

A.8 Multiple gradient descent steps

We now show that if we use BEA to construct a flow that has an error of order $\mathcal{O}(h^p)$ after 1 gradient descent step, then the error will be of the same order after 2 steps. The same argument can be applied to multiple steps. We have $\dot{\theta}$ with $\theta(0) = \theta_{t-1}$ tracks $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ with an error $\mathcal{O}(h^p)$ (i.e $\|\theta(h) - \theta_t\|$ is $\mathcal{O}(h^p)$), then $\dot{\theta}$ tracks two steps of gradient descent $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ and $\theta_{t+1} = \theta_t - h\nabla_\theta E(\theta_t)$ with an error of the same order $\mathcal{O}(h^p)$ (i.e. $\|\theta(2h) - \theta_{t+1}\|$ is $\mathcal{O}(h^p)$). We prove this below. For the purpose of this proof we use the following notation $\theta(h; \theta')$ is the value of the flow at time h with initial condition θ'. Making the initial condition explicit is necessary for the proof.

$$\|\theta(2h;\theta_{t-1}) - \theta_{t+1}\| = \|\theta(2h;\theta_{t-1}) - \theta(h;\theta_t) + \theta(h;\theta_t) - \theta_{t+1}\| \tag{155}$$
$$\leq \|\theta(2h;\theta_{t-1}) - \theta(h;\theta_t)\| + \|\theta(h;\theta_t) - \theta_{t+1}\| \tag{156}$$
$$\leq \|\theta(2h;\theta_{t-1}) - \theta(h;\theta_t)\| + \mathcal{O}(h^p) \tag{157}$$
$$= \|\theta(h;\theta(h;\theta_{t-1})) - \theta(h;\theta_t)\| + \mathcal{O}(h^p) \tag{158}$$

We thus have to bound how the flow changes after time h when starting with two different initial conditions $\theta(h;\theta_{t-1})$ and $\theta_t)$. We also know that $\|\theta(h;\theta_{t-1}) - \theta_t\|$ is $\mathcal{O}(h^p)$. Thus by expanding the Taylor series and the mean value theorem we obtain:

$$\|\theta(h;\theta(h;\theta_{t-1})) - \theta(h;\theta_t)\| = \left\| \sum_{i=0}^{\infty} \frac{h^i}{i!}\theta^{(i)}\left(\theta(h;\theta_{t-1})\right) - \sum_{i=0}^{\infty} \frac{h^i}{i!}\theta^{(i)}(\theta_t)) \right\| \tag{159}$$
$$= \left\| \sum_{i=0}^{\infty} \frac{h^i}{i!}\left(\theta^{(i)}\left(\theta(h;\theta_{t-1})\right) - \theta^{(i)}(\theta_t)\right) \right\| \tag{160}$$
$$= \left\| \sum_{i=0}^{\infty} \frac{h^i}{i!}\frac{d}{d\theta}\theta^{(i)}(\theta')\left(\theta(h;\theta_{t-1}) - \theta_t\right) \right\| \tag{161}$$
$$\leq \sum_{i=0}^{\infty} \frac{h^i}{i!}\left|\frac{d}{d\theta}\theta^{(i)}(\theta')\right| \underbrace{\|(\theta(h;\theta_{t-1})) - \theta_t\|}_{\mathcal{O}(h^p)} = \mathcal{O}(h^p) \tag{162}$$

This tells us that we can construct a bound:

$$\|\theta(2h;\theta_{t-1}) - \theta_{t+1}\| \leq \mathcal{O}(h^p) + \mathcal{O}(h^p) = \mathcal{O}(h^p) \tag{163}$$

Dependence on the number of steps. While the order in learning rate is the same, we note that as the number of steps increases the errors are likely to accumulate with the number of steps (for n discrete steps we will sum n terms of order $\mathcal{O}(h^p)$ in the above bound). For example when taking the number of steps $n \to \infty$ the above no longer provides a bound.

A comment on stochastic gradient descent. Using BEA we can construct flows which approximate one step of a discrete update. The same flows will approximate multiple discrete steps with the same level of error. In the case of deep learning however, it is common to use stochastic gradient descent where individual discrete gradient updates minimize different losses depending on the data batch: $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}; x_{t-1})$ and $\theta_{t+1} = \theta_t - h\nabla_\theta E(\theta_t; x_t)$. Thus if we perform BEA on each stochastic gradient descent update separately, we obtain different modified flows. To use BEA in the case of stochastic gradient descent we have to choose one of the following options: expand the two discrete gradient descent updates into one update and perform BEA on the combined discrete update; work in expectation over batches in an epoch as done in Smith et al. (2021); or focus on each gradient descent update independently, and try to understand the per iteration behavior of gradient descent. In this work, we took the latter approach and thus for simplicity we excluded the dependence of the function E on the data batch.

A.9 Approximations to per iteration drift for gradient descent and momentum

We now prove Thm 6.1, on the per iteration drift of gradient descent. We apply the Taylor reminder theorem on the NGF initialised at gradient descent iteration parameters θ_{t-1}. We have that there exists h' such that

$$\theta(h) = \theta(0) + h\dot{\theta}(0) + \frac{h^2}{2}\ddot{\theta}(h') \tag{164}$$
$$= \theta(0) - h\nabla_\theta E\theta(0) + \frac{h^2}{2}\nabla_\theta^2 E(\theta(h'))\nabla_\theta E(\theta(h')) \tag{165}$$
$$= \theta_{t-1} - h\nabla_\theta E(\theta_{t-1}) + \frac{h^2}{2}\nabla_\theta^2 E(\theta')\nabla_\theta E(\theta') \tag{166}$$

The above proof measures the drift of gradient descent, which we then used to construct DAL. In Section 7, we also used DAL for momentum, and used an intuitive justification. We now show a more theoretical justification, by measuring the total per iteration drift of momentum. To do so, we use the following flow to describe momentum, provided by Kunin et al. (2021):

$$\dot{\theta} = -\frac{1}{1-m}\nabla_\theta E \tag{167}$$

and we measure the discretization drift of a momentum update compared to this flow.

If we apply the same strategy as above, we have that

$$\theta(h) = \theta(0) + h\dot{\theta}(0) + \frac{h^2}{2}\dot{\dot{\theta}}(h') \tag{168}$$

$$= \theta(0) - \frac{h}{1-m}\nabla_\theta E\theta(0) + \frac{h^2}{2(1-m)^2}\nabla_\theta^2 E(\theta(h'))\nabla_\theta E(\theta(h')) \tag{169}$$

$$= \theta_{t-1} - \frac{h}{1-m}\nabla_\theta E(\theta_{t-1}) + \frac{h^2}{2(1-m)^2}\nabla_\theta^2 E(\theta')\nabla_\theta E(\theta') \tag{170}$$

When we compare this with the discrete update

$$\theta_t = \theta_{t-1} + mv_{t-1} - h\nabla_\theta E(\theta_{t-1}) \tag{171}$$

and obtain that the per iteration drift is

$$\left(\theta_{t-1} - \frac{h}{1-m}\nabla_\theta E(\theta_{t-1}) + \frac{h^2}{2(1-m)^2}\nabla_\theta^2 E(\theta')\nabla_\theta E(\theta')\right) - (\theta_{t-1} + mv_{t-1} - h\nabla_\theta E(\theta_{t-1})) \tag{172}$$

$$= \frac{h^2}{2(1-m)^2}\nabla_\theta^2 E(\theta')\nabla_\theta E(\theta') + mv_{t-1} + h\nabla_\theta E(\theta_{t-1}) - \frac{h}{1-m}\nabla_\theta E(\theta_{t-1}) \tag{173}$$

$$= \frac{h^2}{2(1-m)^2}\nabla_\theta^2 E(\theta')\nabla_\theta E(\theta') + mv_{t-1} - \frac{hm}{1-m}\nabla_\theta E(\theta_{t-1}) \tag{174}$$

Thus there is one part of the drift that comes from the same source as gradient descent and another part of the drift which comes from the alignment of the current gradient and the moving average obtained using momentum. Thus when using DAL-p with momentum, we only focus on one of the sources of drift (the first term).

B Non-principal terms

This work focuses on understanding the effects of the principal flow on the behavior of gradient descent. The principal terms however are not the only terms in the discretization drift: we have found one non-principal term of the form $\nabla_\theta E^T(\nabla_\theta^2\nabla_\theta E)\nabla_\theta E$ (Eq 8) and have seen that it can have a stabilising effect (Figure 6). To provide some intuition, while $\nabla_\theta E^T(\nabla_\theta^2\nabla_\theta E)\nabla_\theta E$ cannot be written as a gradient operator, locally we can approximate it with $\nabla_\theta\left[\nabla_\theta E(\theta_{t-1})^T(\nabla_\theta^2 E)\nabla_\theta E(\theta_{t-1})\right]$. Under this approximation, $\nabla_\theta E(\theta_{t-1})^T(\nabla_\theta^2 E)\nabla_\theta E(\theta_{t-1}) = \sum_i \lambda_i(\nabla_\theta E(\theta_{t-1})^T u_i)^2$ is being minimized close to an iteration of gradient descent, i.e. λ_i is minimized in areas where $(\nabla_\theta E(\theta_0)^T u_i)^2$ is large, which can be a stabilizing effect by ensuring the model eigenvalues do not increase drastically. We plot the value of this term in neural network training in Figure 25. These results show that the non-principal third order term is very small outside the edge of stability area, but has a larger magnitude around the edge of stability, where the largest eigenvalues stop increasing but the magnitude of $(\nabla_\theta E(\theta_0)^T u_i)^2$ can be large (Figure 15). This observation is consistent with the above interpretation, but more theoretical and empirical work is needed to understand the effects of non-principal terms on training stability and generalization.

C Experimental details

Estimating continuous time trajectories of flows. To estimate the continuous time trajectories we use Euler integration with a small learning rate 5×10^{-5}. We reached this learning rate through experimentation:

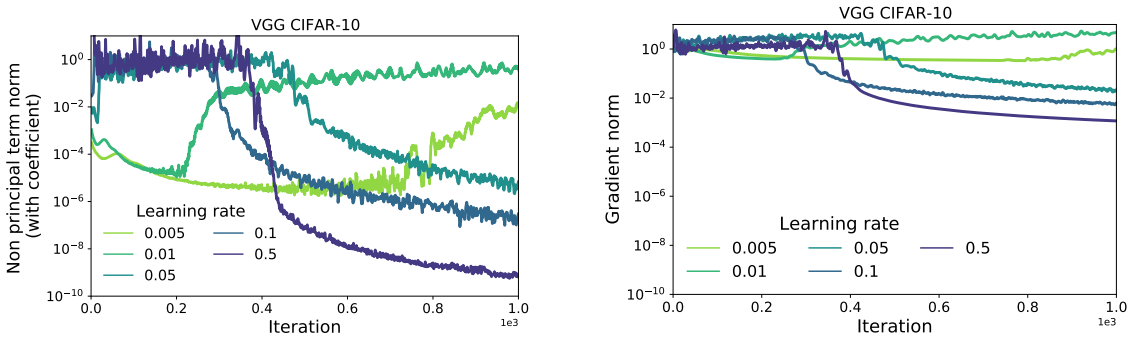


Figure 25: The value of the non-principal third order term in training: outside the edge of stability areas (best seen for learning rates 0.005 and 0.01), the non-principal third order term is very small compared to the gradient. Inside the edge of stability areas, the non-principal term has a higher magnitude. Corresponding loss and λ_0 plots are provided in Figure 9.

further decreasing it did not change the obtained results. We note that this approach can be computationally expensive: to estimate the trajectory of the NGF of time give by the gradient descent learning rate 10^{-2}, we need to do 5000 gradient steps. It is common in the literature to use RungeKutta4 to approximate continuous time flows, but we noticed that approximating a flow for time h using RungeKutta4 with learning rate h still introduced significant drift: if the learning rate was further reduced and multiple steps were taken the results were significantly different. Thus RungeKutta4 also needs multiple steps to estimate continuous trajectories. Given that one RungeKutta4 update requires computing 4 gradients, we found that using Euler integration with small learning rates is both sufficient and more efficient in practice.

Datasets. When training neural networks with primarily used three standard datasets: MNIST (LeCun et al., 1995), CIFAR-10 (Krizhevsky et al., 2009) and Imagenet (Deng et al., 2009). On the small NN example in Section 3, we used a dataset of 5 examples, where the input is 2D sampled from a Gaussian distribution and the regression targets are also sample randomly. We used the UCI breast cancer dataset (Asuncion and Newman, 2007) in Figure 32.

Architectures. We use standard architectures: MLPs for MNIST, VGG (Simonyan and Zisserman, 2014) or Resnet-18 (Version 1) (He et al., 2016) for CIFAR-10, Resnet-50 for Imagenet (Version 1). We do not use any form of regularisation or early stopping. We use the Elu activation function (Clevert et al., 2015) to ensure that the theoretical setting we discussed applies directly (we thus avoid discontinuities caused by Relus(Agarap, 2018)).

Losses. Unless otherwise specified we used a cross entropy loss.

Computing eigenvalues and eigenvectors. We use the Lanczos algorithm to compute λ_i and u_i.

Computing $\left\|\nabla^2_\theta E \nabla_\theta E\right\|$. We use Hessian vector products to compute $\left\|\nabla^2_\theta E \nabla_\theta E\right\|$. We experimented with using the approximation $\nabla^2_\theta E \nabla_\theta E = \frac{1}{2}\nabla_\theta \left\|\nabla_\theta E\right\|^2 \approx \frac{E(\theta+\epsilon\nabla_\theta E)-E(\theta)}{\epsilon}$ with $\epsilon = 0.01$ and saw no decrease in performance in the full batch setting when using it in DAL, but a slight decrease in performance when using it in stochastic gradient descent. We show experimental results in Figures 26, 27, 28 and 29. More experimentation is needed to see how to efficiently leverage this approximation, for example whether other values of ϵ are more suited for large networks. Alternatively, even when small batches are used for training a larger batch can be used to approximate $\left\|\nabla^2_\theta E \nabla_\theta E\right\|$. We note however that all the conclusions we observed in the paper regarding the relative performance of p values of DAL still hold, with and without the approximation and that indeed we see less of a difference when higher values of p are used.

Seeds. All test accuracies are shown averaged from 3 seeds. For training curves, we compare models across individual training runs to be able to observe the behavior of gradient descent. We did not observe variability across seeds with any of the behaviors reported in this paper.

DAL. When using DAL we set a maximum learning rate of 5 to avoid any potential instabilities.

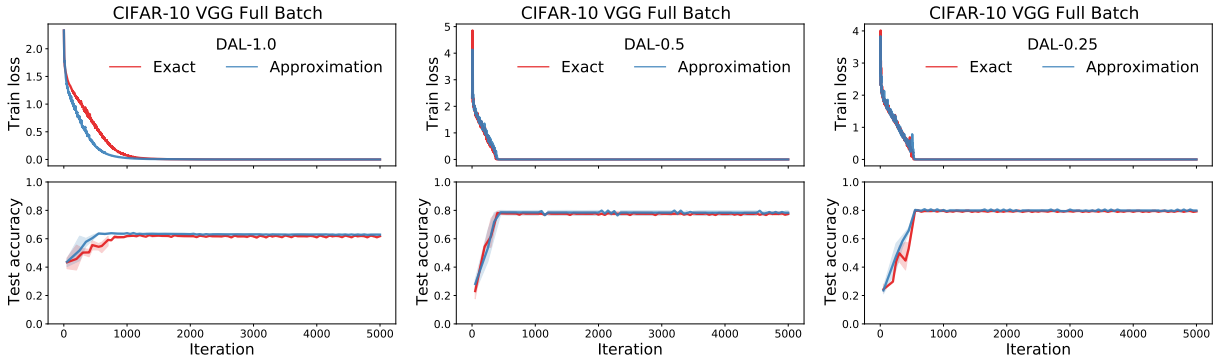


Figure 26: CIFAR-10 DAL results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation.

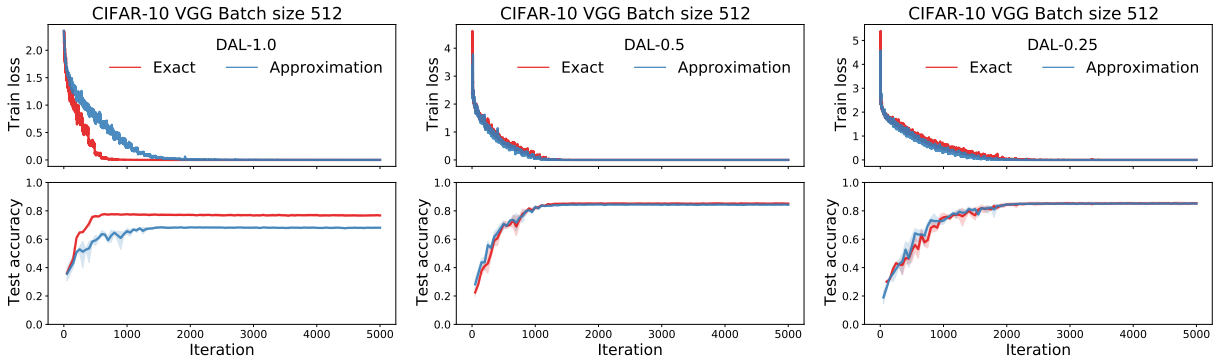


Figure 27: CIFAR-10 DAL results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation.

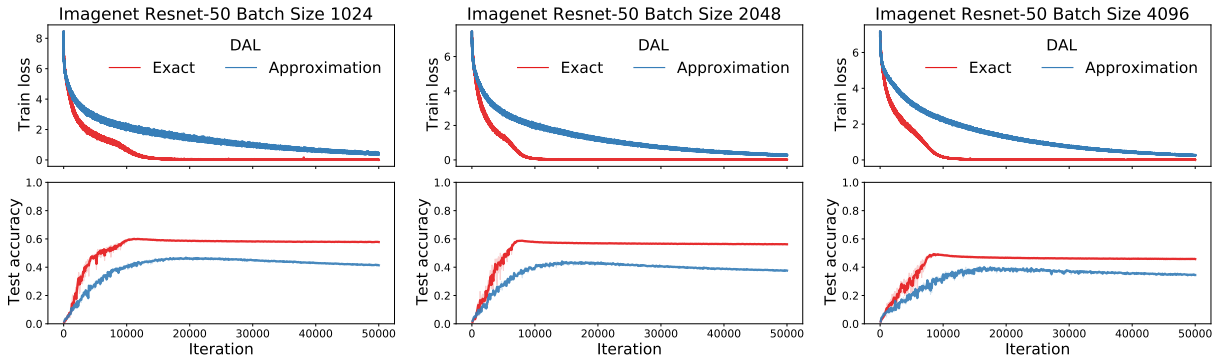


Figure 28: Imagenet DAL results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation.

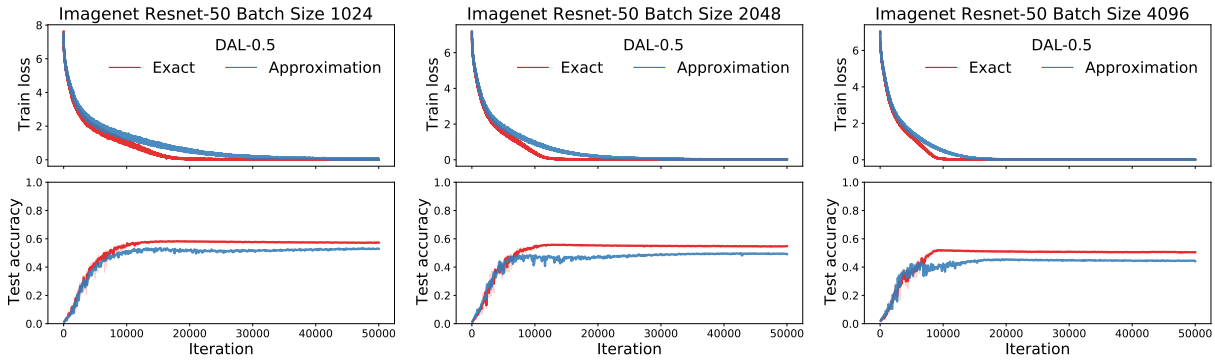


Figure 29: Imagenet DAL-0.5 results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation.

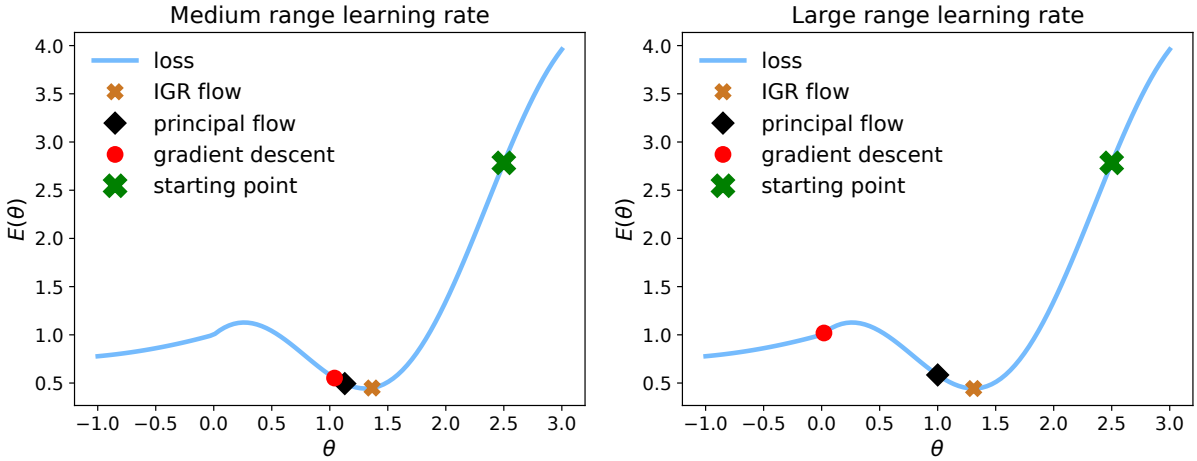


Figure 30: Results with a non quadratic function, including cos and sin. The function used is: $E(\theta) = \cos(\theta) + \theta, \text{if } \theta < 0; (\theta/3)^2 2 + 1 + \theta/3)$ otherwise. The principal flow and the IGR flow stay on the cosine branch. The main reason for using this function was to show the escape of gradient descent into a flatter valley and to assess what the corresponding flows do. Learning late on the left: 0.5. On the right: 0.85.

D Additional experimental results

Oscillations for non linear functions. We show that in the case of non-linear functions in Figure 30, using cos. Here too, the PF is better at describing the behavior of GD than the NGF and IGR flow, which stop at local minima.

Per iteration drift for neural networks with IGR and third order flows. We show results estimating the per iteration error between GD and the IGR flow and the third order flow in Figure 31. While the IGR and third order flows reduce error compared to the NGF, a large gap remains.

Global error for the UCI breast cancer dataset. We show in Figure 32 that the PF is better at tracking the behavior of GD in the case of NNs both in the stability and edge of stability areas.

Additional edge of stability experiments. Results showing the decrease in learning rate in an edge of stability area stabilizes training are shown in Figure 34. We show that one dimension is sufficient to cause instability in training on MNIST in Figure 35 (as we have done in Figure 14 for CIFAR-10). We show the connection between the stability coefficient sc_0 and areas where the loss increases in Figure 36: this shows that when the loss increases, it is in an area where sc_0 is also increasing. We note that to get a full picture of the behavior of the PF we would need to use continuous time computation, but since that is computationally prohibitive we use the incomplete but easily available discrete data. Figure 37 zooms in the behavior of λ_0

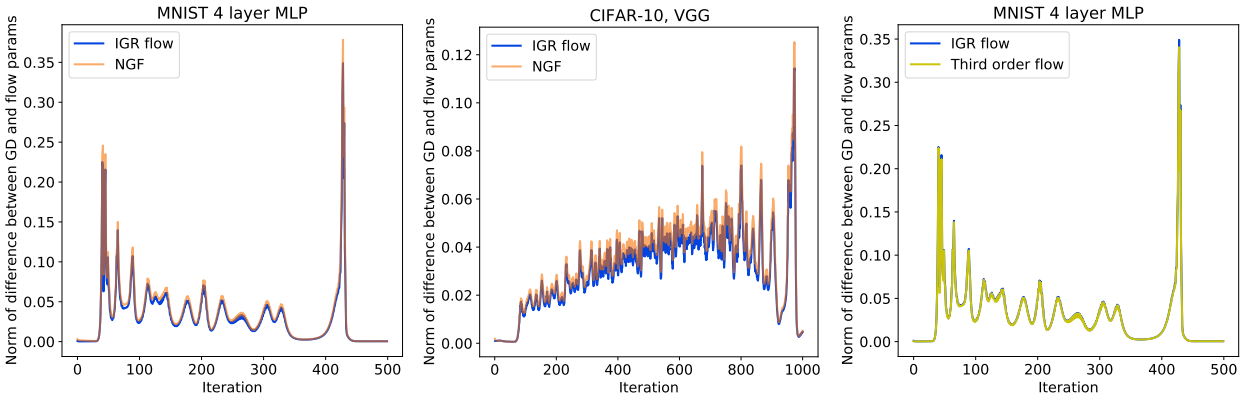
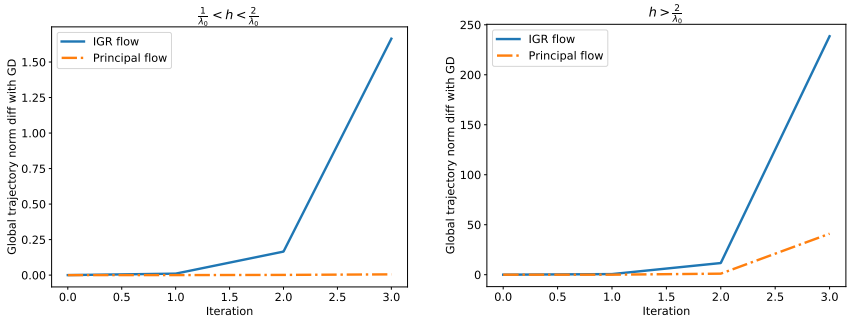


Figure 31: **Motivation**. While modified flows reduce the per iteration discretization error for neural networks trained with gradient descent, there is still a large gap. We obtain the results by training a model with gradient descent $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ and at each iteration t of gradient descent approximating the respective flows for time h and computing the difference in norms between the resulting flow parameters and the gradient descent parameters at the next iteration: $\|\theta_t - \theta(h)\|$ with $\theta(h) = \theta_{t-1}$.



(a) Stability

(b) Edge of stability

Figure 32: Global parameter error for IGR ad PF flows compared on an MLP trained on the UCI breast cancer dataset. We initialize both gradient descent and the IGR and PF flows at a set of initial parameters, and run them for time step $2h$ and compare the error after each time step h. The PF is better at tracking the trajectory of GD than the IGR flow.

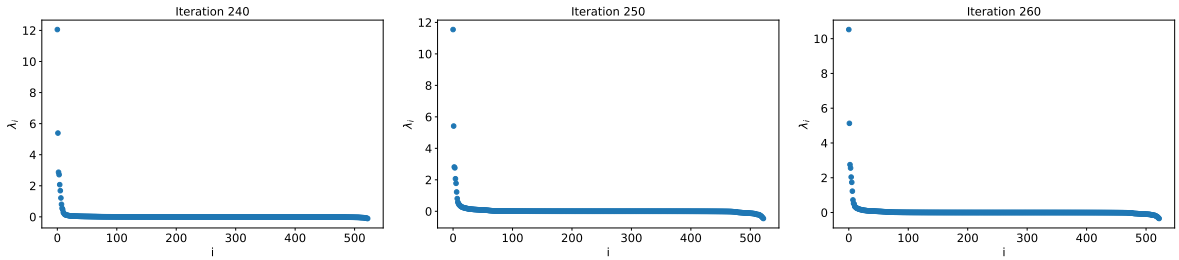


Figure 33: The eigenspectrum obtained along the gradient descent path of a 5 layer small MLP trained on the UCI Iris dataset. The Hessian eigenvalues are largely positive. The result accompanies Figure 10 in the main paper.

as dependent of the behavior of the corresponding flows on MNIST and CIFAR-10 and Figure 38 shows additional Resnet-18 results.

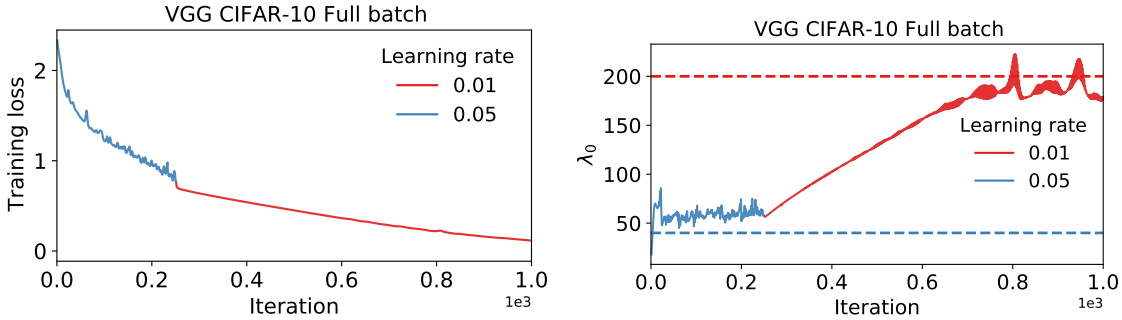


Figure 34: The effect of learning rate changes: as discovered by (Cohen et al., 2021), reducing the learning rate in training leads to increased stability while the largest eigenvalue increases. The lack of instability with a smaller learning rate is explained by the principal flow, and the increase in the largest eigenvalue can be thought of through the lens of the principal flow arguments.

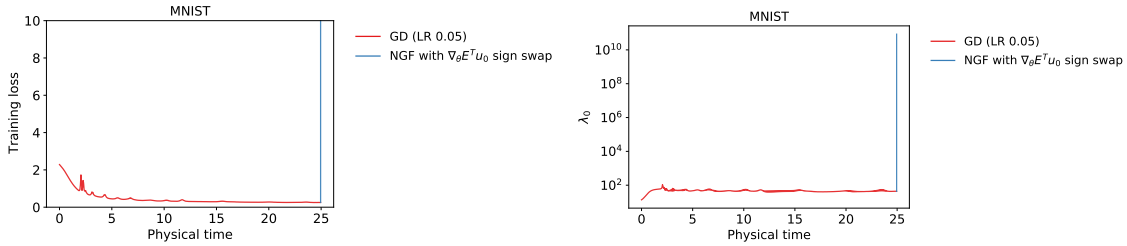


Figure 35: MNIST results when changing one direction in continuous time: one eigendirection is sufficient to cause instability. The model was trained with learning rate 0.05 until the edge of stability area is reached after which the flow $\dot{\theta} = \nabla_\theta E^T u_0 u_0 + \sum_{i=1}^{D-1} -\nabla_\theta E^T u_i u_i$ is used.

DAL. We use gradient descent training with a fixed learning rate to measure the quantities we would like to use as a learning rate in DAL, to see if they have a reasonable range and show results in Figure 39. We show sweeps across batch sizes Figures 40, 41, 42. We show DAL-p sweeps with effective learning rates, training losses and test accuracies in FIgure 43. We show results with DAL on a mean square loss in Figure 44.

Learning rate per parameter. We show preliminary results of how to use a DAL like learning rate to use a per-parameter learning rate in Figure 45 and 46. We note that here we found that we can directly use $\frac{1}{\left(\nabla_\theta^2 E \nabla_\theta E\right)_i}$ as the learning rate for parameter i. When using the per parameter learning rate update, the argument regarding the norm of the gradient used for DAL is no longer necessary.

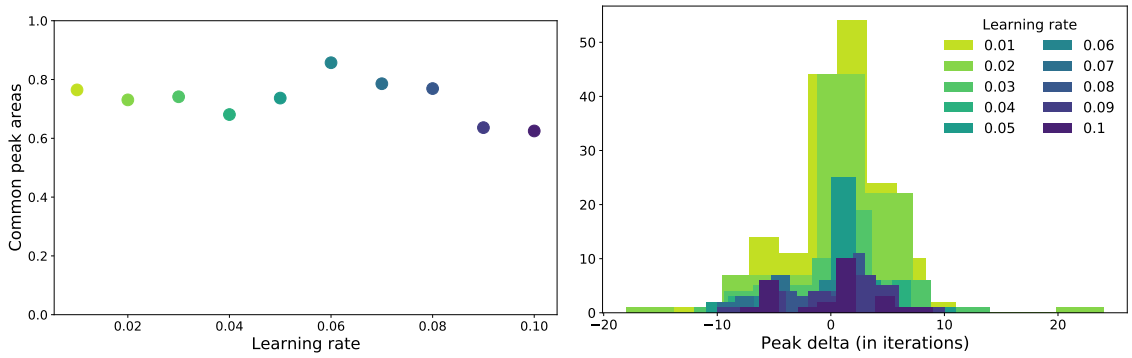
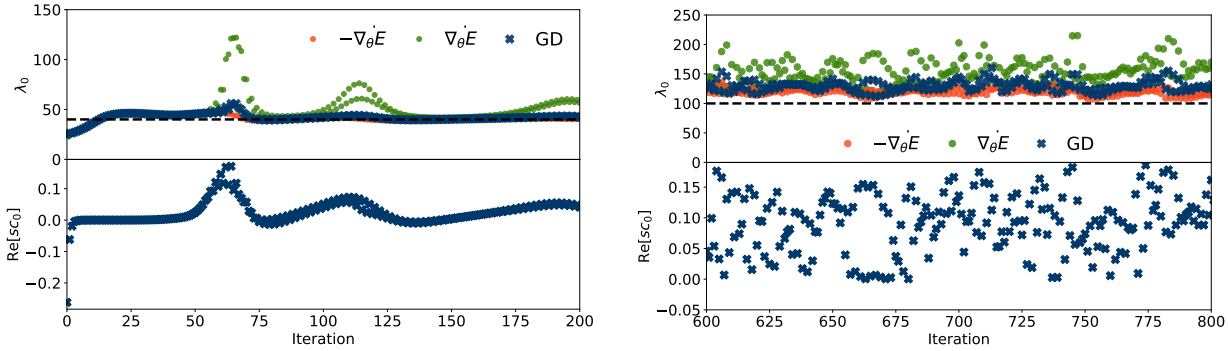


Figure 36: Peak areas of loss increase and the coefficient for the largest eigendirection of the principal flow. Here we normalise the coefficient by the gradient norm to remove the gradient norm as a potential confounder. Results on CIFAR-10.



(a) MNIST MLP

(b) CIFAR-10 VGG

Figure 37: Understanding changes to λ_0 using the principal flow: increases in λ_0 above $2/h$ correspond to increases in the flow $\dot{\theta} = \nabla_\theta E$.

Momentum. We show additional results with learning rate adaptation and momentum in Figure 48.

Global discretization error. We show the global error in trajectory between the NGF and gradient descent in Figure 49. As previously observed (Cohen et al., 2021), gradient descent follows the NGF early in training, and the eigenvalues of the trajectory following the NGF keeps growing.

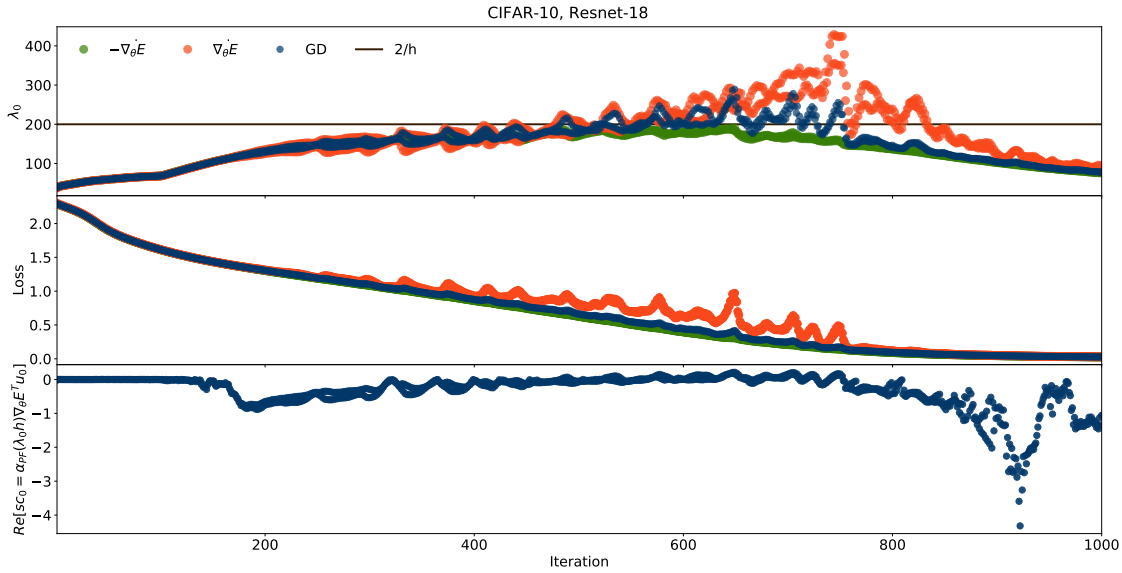


Figure 38: Learning rate: 0.01. Edge of stability results: the connection between the principal flow, λ_0 and loss instabilities with Resnet-18.

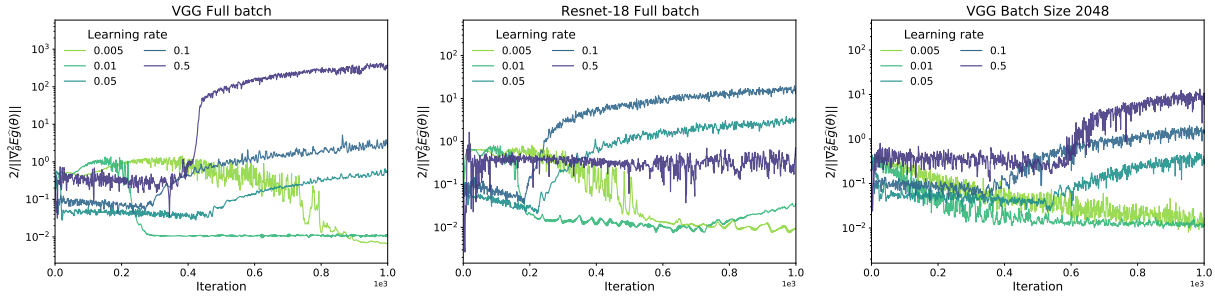


Figure 39: Evaluating the value of the DAL learning rate for a vanilla (fixed) learning rate sweep to assess whether it would have reasonable magnitudes.

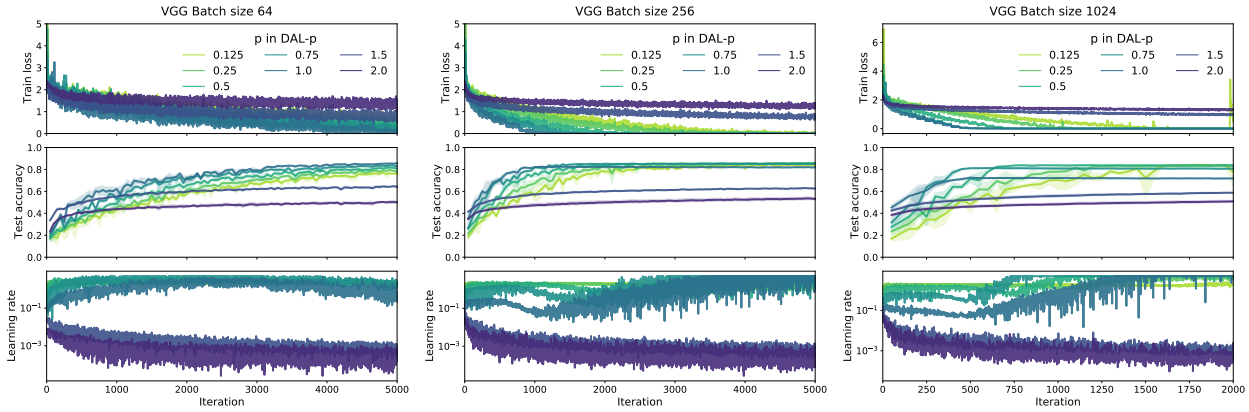


Figure 40: DAL-p training a VGG model on CIFAR-10. Sweep across batch sizes: discretization drift helps test performance, but at the cost of stability. We also show the effective learning rate and train losses and test accuracies.

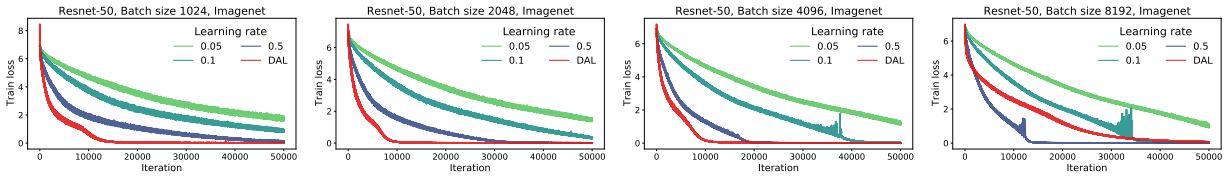


Figure 41: DAL: Imagenet results across batch sizes.

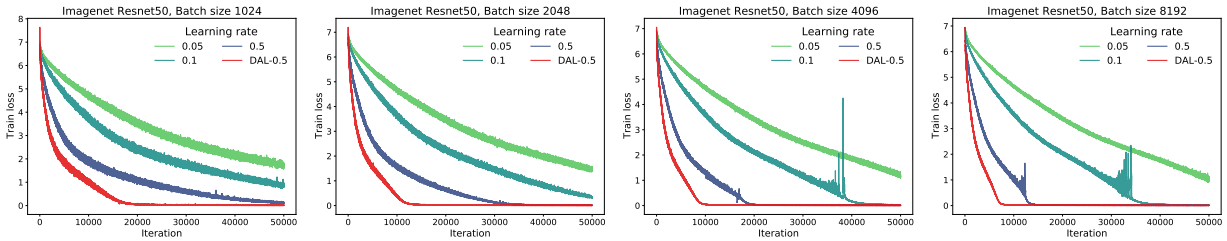


Figure 42: DAL-0.5: Imagenet results across batch sizes.

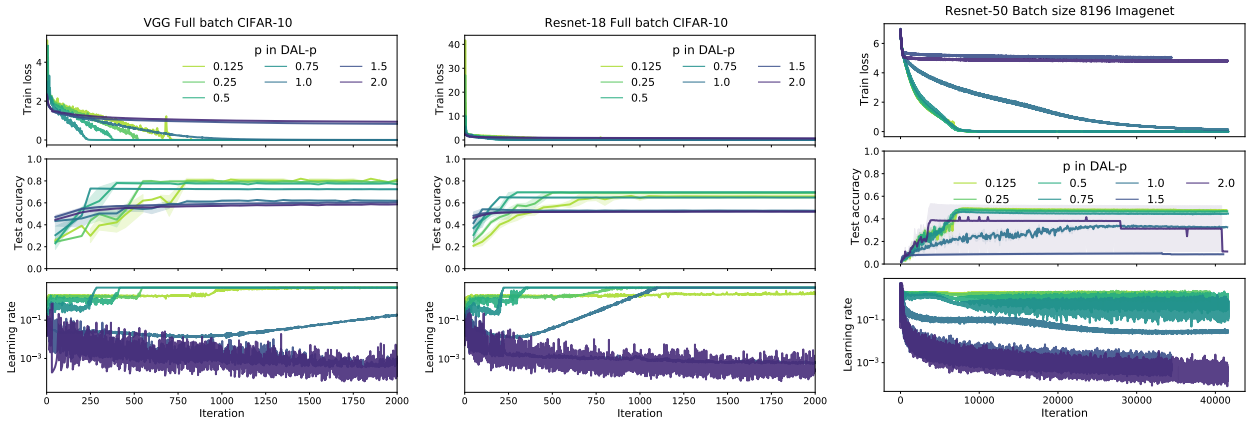


Figure 43: DAL-p sweep: discretization drift helps test performance, but at the cost of stability. We also show the effective learning rate and train losses and test accuracies.

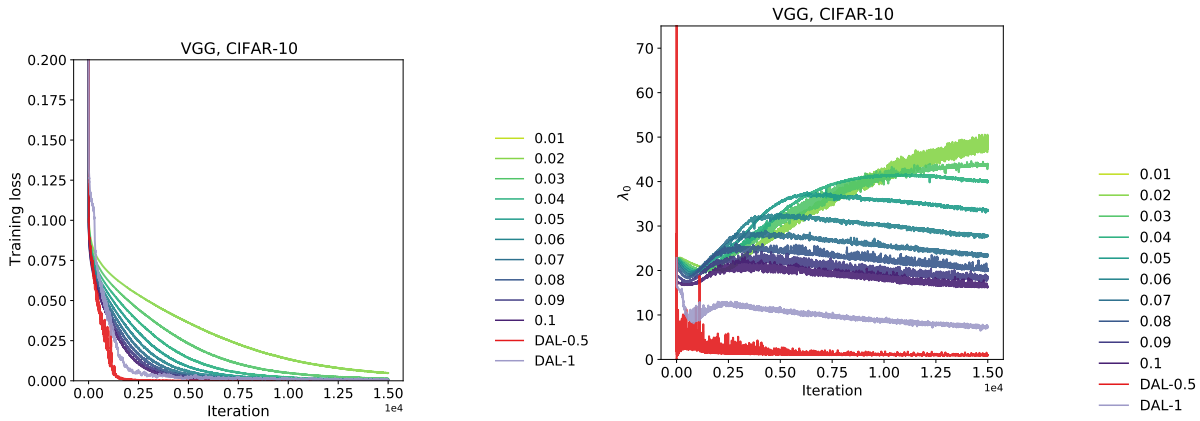


Figure 44: Results with a least square loss. DAL and DAL-0.5 lead to quicker training in this setting as well.

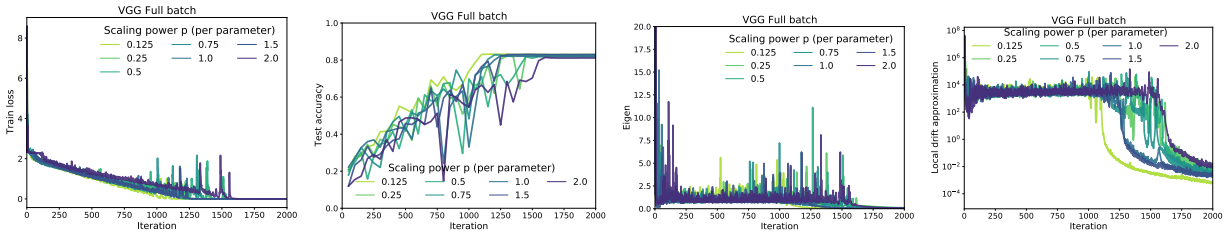


Figure 45: VGG results with learning rate scaling per parameter, but instead of using the global learning rate $2/||\nabla_\theta E^2 \widehat{g}(\theta)||^p$, using the per parameter θ_i learning rate $2/(\nabla_\theta E^2 \nabla_\theta E)_i^p$.

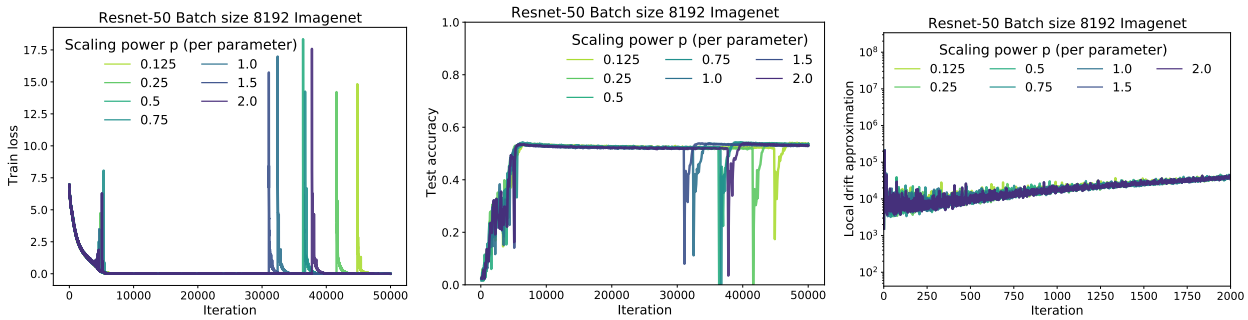
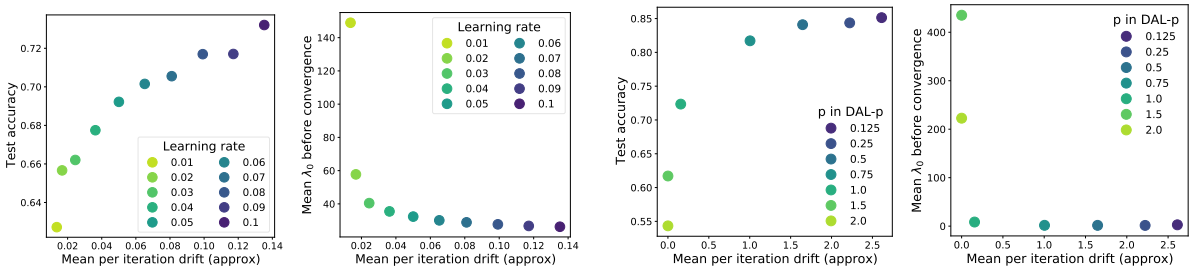


Figure 46: Imagenet results when using a DAL like method of scaling per parameter, but instead of using the global learning rate $2/||\nabla_\theta E^2 \widehat{g}(\theta)||$, using the per parameter θ_i learning rate $2/(\nabla_\theta E^2 \nabla_\theta E)_i^p$.



(a) Fixed learning rate sweep.

(b) DAL-p sweep.

Figure 47: VGG CIFAR-10 with batch size 1024: connection between drift, test error and λ_0. We observe the same patterns for other batch sizes.

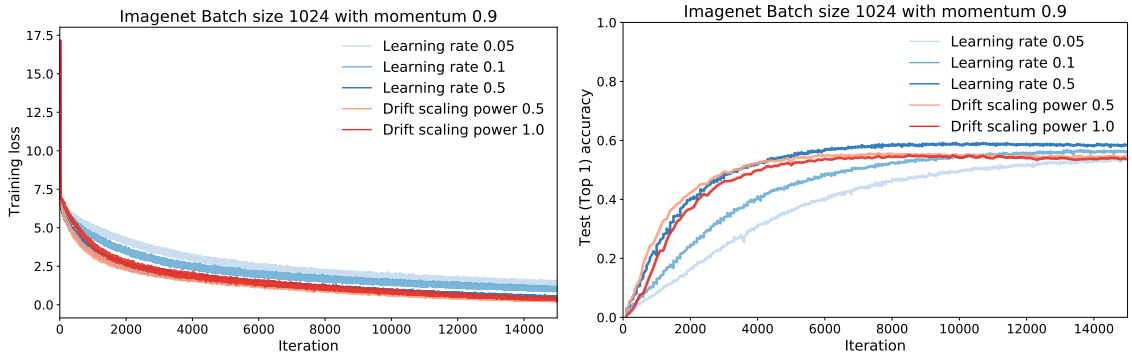


Figure 48: Integrating information on drift with momentum: with smaller batch sizes, we observe a smaller empirical gain. Since these are preliminary results, more investigation is needed into why that is. These results are consistent with the observation obtained with vanilla gradient descent, where we say larger gains with larger batch sizes on Imagenet.

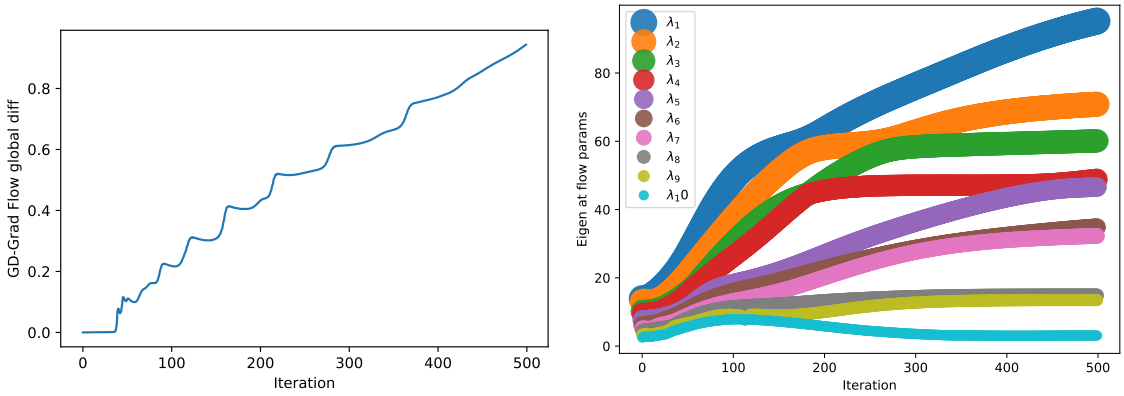


Figure 49: The global error between the NGF and gradient descent (left), as well as the behavior of λ_0 of the NGF.

Reproduction details for each figure

1 2D convex plot. The loss function is $E = \theta^T A\theta$ with $A = ((1, 0.0), (0.0, 0.01))$. The learning rate is 0.9. . . . 3

2 1D example. $E = \frac{1}{2}(\theta - 0.6)^2$ and $h = 1.2$ . . . 4

3 Comparing the coefficients α_{NGF} and α_{PF} across the training landscape. . . . 7

4 $E(z) = \frac{1}{2}z^2$. Learning rates are 0.8, 1.5 and 2.1. . . . 8

5 Quadratic losses in 2 dimensions. $E = \theta^T A\theta$ with $A = ((1, 0.0), (0.0, 0.01))$, with learning rates 0.5, 0.9 and 1.05. . . . 9

6 Banana function. Learning rates 0.0006, 0.0017 0.005. . . . 9

7 Error between gradient descent parameters and parameters obtained following continuous time flows for multiple iterations on a small MLP: $\|\theta_{t+n} - \theta(nh)\|$. Input size 2, MLP of with output units $10, 1$. Learning rates 0.1, 0.2 and 0.25. We use a dataset with 5 examples where the input is generated from a Gaussian distribution with 2 dimensions and the targets are samples from a Gaussian distribution with 1 dimension. A mean square error loss is used. . . 11

8 Predictions of $\nabla_\theta E^T u_0$ using the principal flow on neural networks. Full batch training with a VGG network on CIFAR-10. Learning rates 0.01, 0.03 and 0.05. . . . 12

9 Edge of stability on VGG networks on CIFAR-10. Full batch training. . . . 13

10 Edge of stability on a small 5 layer MLP with 10 units per layer on the UCI Iris dataset. Full batch training. . . . 13

11 Edge of stability results. MNIST MLP with 4 layers. Learning rate 0.05. For each model we approximate the NGF and the positive gradient flow initialized at the current gradient descent parameters and measure the value od the loss function and λ_0. . . . 14

12 Using the PF and stability coefficient to understand instabilities in deep learning. MNIST results use a 4 layer MLP and a 0.05 learning rate. CIFAR-10 MLP use a VGG network and a 0.02 learning rate. . . . 14

13 Understanding the behavior of the loss through the PF and the behavior of λ_0. Using a Resnet-18 trained on CIFAR-10 using a learning rate of 0.03. For each model we approximate the NGF and the positive gradient flow initialized at the current gradient descent parameters and measure the value of the loss function and λ_0. . . . 15

14 Assessing whether 1 dimension is enough to cause instability in a continuous time flow. We train a model with gradient descent until it reaches the edge of stability ($\lambda_0 \approx 2/h$), after which we use a approximate the continuous flow $\dot{\theta} = \nabla_\theta E^T u_0 u_0 + \sum_{i=1}^{D-1} -\nabla_\theta E^T u_i u_i$. The model is a VGG model trained on CIFAR-10 with learning rate 0.05 and after the edge of stability is reached the flow is approximated using Euler steps of size $5e-5$. . . . 15

15 The unstable dynamics of $\nabla_\theta E^T u$ in the edge of stability area. The model results are obtained from a VGG model trained on CIFAR-10 with a learning rate of 0.01. . . . 16

16 $||\nabla^2_\theta E \nabla_\theta E||$ and the per iteration drift as measured during training. The plots are obtained on an MNIST MLP with 4 layers and 100 units per layer, and a VGG model trained on CIFAR-10. The learning rates used are 0.05, 0.01 and 0.01 respectively. . . . 18

17 Correlation between $||\nabla^2_\theta E \nabla_\theta E||$ and the iteration drift. The plots are obtained on an MNIST MLP with 4 layers and 100 units per layer, and a VGG model trained on CIFAR-10. Results are obtained over a learning rate sweep of 10 learning rates 0.01, 0.02... 0.1. . . . 18

18 Models trained using a learning rate sweep or DAL on CIFAR-10 and Imagenet. Models are VGG, Resnet-18 and Resnet-50 respectively. . . . 18

19 Key quantities in DAL versus fixed learning rate training: learning rate, and update norms. Results obtained on a Resnet-18 model trained on CIFAR-10. . . . 19

20 DAL-0.5 results on a CIFAR-10 and Imagenet using VGG, Resnet-18 and Resnet-50 model respectively. . . . 19

21 DAL-p sweep on full batch training on CIFAR-10 with VGG and Resnet-18 models and a Resnet-50 model trained on Imagenet. . . . 20

22 VGG model trained on CIFAR-10 using full batch gradient descent, either with a fixed learning rate in a sweep or a DAL-p sweep. The drift per iteration is approximated using $h^2/2\nabla_\theta E^2 \nabla_\theta E$ or $h(\theta)^2/2\nabla_\theta E^2 \nabla_\theta E$ (the adaptive learning rate) in case of DAL-p. . . . 20

23 Resnet-50 results with 0.9 momentum and a learning rate sweep or DAL. Models trained on Imagenet. . . . 21

24 BEA finds continuous modified flows which describe the gradient descent update with learning rate h with an error of $\mathcal{O}(h^{n+2})$. We identify $f_1, \cdots f_n$ so that terms of order $\mathcal{O}(h^p), 2 \leq p \leq n+1$ in $\tilde{\theta}(h)$ are 0. . . . 28

25 Learning rate sweep showing the value of the non-principal third order term in training, on a Resnet-18 model trained on CIFAR-10. The coefficients used for the non-principal term are those in equation 8. . . . 43

26 CIFAR-10 DAL results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation. Full batch training. . . . 44

27 CIFAR-10 DAL results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation. Batch size 512. . . . 44

28 Imagenet DAL results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation. . . . 44

29 Imagenet DAL-0.5 results with a Hessian vector product computation of $\nabla_\theta E \nabla_\theta E$ compared to an approximation. We used $\epsilon = 0.01$ in the approximation. . . . 45

30 Results with a non quadratic function, including cos and sin. The function used is: $E(\theta) = \cos(\theta) + \theta$, if $\theta < 0$; $(\theta/3)^2 2 + 1 + \theta/3)$ otherwise. The principal flow and the IGR flow stay on the cosine branch. The main reason for using this function was to show the escape of gradient descent into a flatter valley and to assess what the corresponding flows do. Learning late on the left: 0.5. On the right: 0.85. . . . 45

31 Per iteration error between the NGF, IGR and PF and gradient descent. MNIST learning rate 0.05. CIFAR learning rate 0.02. We obtain the results by training a model with gradient descent $\theta_t = \theta_{t-1} - h\nabla_\theta E(\theta_{t-1})$ and at each iteration t of gradient descent approximating the respective flows for time h and computing the difference in norms between the resulting flow parameters and the gradient descent parameters at the next iteration: $\|\theta_t - \theta(h)\|$ with $\theta(h) = \theta_{t-1}$. Results obtained on full batch training. . . . 46

32 Global parameter error for IGR ad PF flows compared on an MLP trained on the UCI breast cancer dataset. MLP with 10 hidden units and 1 output unit. Learning rates 1.5 and 5. respectively. . . . 46

33 The eigenspectrum obtained along the gradient descent path of a 5 layer small MLP trained on the UCI Iris dataset. The Hessian eigenvalues are largely positive. The result accompanies Figure 10 in the main paper.. . . . 47

34 Training a VGG model on CIFAR-10 and training with learning rate 0.05 after which switching to 0.01. . . . 47

35 MNIST results when changing one direction in continuous time. The model is a 4 layer MLP with 100 hidden units per layer. The model was trained with learning rate 0.05 until the edge of stability area is reached after which the flow $\dot{\theta} = \nabla_\theta E^T u_0 u_0 + \sum_{i=1}^{D-1} -\nabla_\theta E^T u_i u_i$ is used. . 47

36 Peak areas of loss increase and the coefficient for the largest eigendirection of the PF. Results using a VGG model trained with full batch gradient descent on CIFAR-10 across a learning rate sweep. . . . 48

37 Understanding changes to λ_0 using the principal flow. MLP with 4 layers with 100 units each trained on MNIST, and VGG model trained on CIFAR-10. Learning rate 0.05 and 0.01 respectively. 48

38 The connection between the principal flow, λ_0 and loss instabilities with Resnet-18 trained on CIFAR-10 with a learning rate 0.01. 49

39 Evaluating the value of the DAL learning rate for a vanilla (fixed) learning rate sweep to assess whether it would have reasonable magnitudes. Models trained on CIFAR-10 using a learning rate sweep. 49

40 DAL-p sweep with a VGG model trained on CIFAR-10 for small and large batch sizes. 49

41 DAL on Imagenet, a learning rate sweep across batch sizes. 50

42 DAL-0.5: on Imagenet, a learning rate sweep across batch sizes. 50

43 DAL-p sweep on VGG, Resnet-18 and Imagenet. 50

44 DAL results with a least square loss. Full batch training of a VGG model on CIFAR-10. . . . 50

45 VGG CIFAR-10 full batch training with a learning rate per parameter. 51

46 Resnet-50 Imagenet training with a learning rate per parameter. Batch size 8192. 51

47 VGG trained on CIFAR-10 with batch size 1024: connection between drift, test error and λ_0. 51

48 DAL-p with momentum 0.9 on Imagenet. The model is Resnet-50 trained with batch size 1024. 51

49 MNIST results with a 4 layer MLP with 100 hidden units. The learning rate used is 0.05. . . 52