Gradients are Not All You Need

Luke Metz[*]   **C. Daniel Freeman**[*]   **Samuel S. Schoenholz**
Google Research, Brain Team
{lmetz, cdfreeman, schsam}@google.com

Tal Kachman
Radboud University
Donders Institute for Brain, Cognition and Behaviour
tal.kachman@donders.ru.nl

Abstract

Differentiable programming techniques are widely used in the community and are responsible for the machine learning renaissance of the past several decades. While these methods are powerful, they have limits. In this short report, we discuss a common chaos based failure mode which appears in a variety of differentiable circumstances, ranging from recurrent neural networks and numerical physics simulation to training learned optimizers. We trace this failure to the spectrum of the Jacobian of the system under study, and provide criteria for when a practitioner might expect this failure to spoil their differentiation based optimization algorithms.

1   Introduction

Owing to the overwhelming success of deep learning techniques in providing fast function approximations to almost every problem practitioners care to look at, it has become popular to try to make differentiable implementations of different systems—the logic being, that by taking the tried-and-true suite of techniques leveraging derivatives when optimizing neural networks for a task, one need only take their task of interest, make it differentiable, and place it in the appropriate place in the pipeline and train "end to end". This has lead to a plethora of differentiable software packages, ranging across rigid body physics [Heiden et al., 2021, Hu et al., 2019, Werling et al., 2021, Degrave et al., 2019, de Avila Belbute-Peres et al., 2018, Gradu et al., 2021, Freeman et al., 2021], graphics [Li et al., 2018, Kato et al., 2020], molecular dynamics [Schoenholz and Cubuk, 2020, Hinsen, 2000], differentiating though optimization procedures [Maclaurin, 2016], weather simulators [Bischof et al., 1996], and nuclear fusion simulators [McGreivy et al., 2021].

Automatic differentiation provides a conceptually straightforward handle for computing derivatives though these systems, and often can be applied with limited compute and memory overhead [Paszke et al., 2017, Ablin et al., 2020, Margossian, 2019, Bischof et al., 1991, Corliss et al., 2013]. The resulting gradients however, while formally "correct" in the sense that they are exactly the desired mathematical object [2], might not be algorithmically useful—especially when used to optimize certain functions of system dynamics. In this work, we discuss one potential issue that arises when working with iterative differentiable systems: chaos.

Chaotic dynamics and difficulties differentiating through them are not a discovery of this work. To our knowledge, discussion first appeared in climate modeling [Lea et al., 2000, Köhl and Willebrand,

---

[*]Equal contribution

[2]Up to numerical precision, though see [Chow and Palmer, 1992, Kachman et al., 2017] for cases where the gradients will be close.

Table 1: Different kinds of machine learning techniques often resemble differentiating through some iterative system. f is the iteratively applied function, s is an input, θ stands for parameters and l is the optimization objective

Domain	f	s	θ	l
Neural Network Training	A layer transformation of a neural network	The inputs to that layer	The weight matrix and bias vector for that layer	cross entropy mean squared error, l2 regularization, etc.
Reinforcement Learning	The step function of an environment	The state data of the environment and agent	The parameters of a policy	The reward function for the environment
Learned Optimization	The application of an optimizer	The parameters in a network being optimized	The tunable parameters for the optimizer, e.g., learning rate	The performance of the network being optimized on a task after some number of steps of optimization

2002], but have expanded to a varity of different domains ranging from neural network initialization and activation design [Yang and Schoenholz, 2017, Hayou et al., 2019], model based control [Parmas et al., 2018, Parmas], meta-learning [Metz et al., 2019], fluid simulation [Ni and Wang, 2017, Kochkov et al., 2021], and learning protein structures [Ingraham et al., 2018].

2 Preliminaries: Iterated Dynamical Systems

Chaos emerges naturally in iterated maps [Bischof et al., 1991, Ruelle, 2009]. Consider the following discrete matrix equation:

$$s_{k+1} = A_k s_k \qquad (1)$$

where A_k is some possibly state-dependent matrix describing how the state information, s_k, transforms during a step. It is not difficult to show (see App. A), under appropriate assumptions, that the trajectories of x_k depend on the eigenspectrum of the family of transformations A_k. Crucially, if the largest eigenvalue of the A_k is typically larger than 1, then trajectories will tend to diverge exponentially like the largest eigenvalue λ_{max}^k. Contrariwise, if the largest eigenvalue is less than one, trajectories will tend to vanish.

Of course, dynamical systems encountered in the wild are usually more complicated, so suppose instead that we have a transition function, f which depends on state data, s, and control variables θ:

$$s_{t+1} = f(s_t, \theta) \qquad (2)$$

We're typically concerned with functions of our control variables, evaluated over a trajectory, for example, consider some loss function which sums losses (l_t) computed up to a finite number of steps N:

$$l(\theta) = \sum_{t=0}^{N} l_t(s_t, \theta). \qquad (3)$$

This formalism is extremely general, and equation 3 encompasses essentially the entire modern practice of machine learning. For several common examples of $f, s, \theta,$ and l, see Table 1.

Solving our problem usually amounts to either maximizing or minimizing our loss function, l, and a differentiably minded practitioner is usually concerned with the derivative of l. Consider the derivatives of the first few terms of this sum:

$$\frac{\mathrm{d}l_0}{\mathrm{d}\theta} = \frac{\partial l_0}{\partial s_0}\frac{\partial s_0}{\partial \theta} + \frac{\partial l_0}{\partial \theta} \qquad (4)$$

$$\frac{\mathrm{d}l_1}{\mathrm{d}\theta} = \frac{\partial l_1}{\partial s_1}\frac{\partial s_1}{\partial s_0}\frac{\partial s_0}{\partial \theta} + \frac{\partial l_1}{\partial s_1}\frac{\partial s_1}{\partial \theta} + \frac{\partial l_1}{\partial \theta} \qquad (5)$$

$$\frac{\mathrm{d}l_2}{\mathrm{d}\theta} = \frac{\partial l_2}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial s_0}\frac{\partial s_0}{\partial \theta} + \frac{\partial l_2}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial \theta} + \frac{\partial l_2}{\partial s_2}\frac{\partial s_2}{\partial \theta} + \frac{\partial l_2}{\partial \theta} \qquad (6)$$

Here, the pattern is conceptually clear, and then for an arbitrary t:

$$\frac{\mathrm{d}l_t}{\mathrm{d}\theta} = \frac{\partial l_t}{\partial \theta} + \sum_{k=1}^{t} \frac{\partial l_t}{\partial s_t} \left(\prod_{i=k}^{t} \frac{\partial s_i}{\partial s_{i-1}}\right) \frac{\partial s_k}{\partial \theta} \tag{7}$$

This leaves us with a total loss:

$$\frac{\mathrm{d}l}{\mathrm{d}\theta} = \frac{1}{N} \sum_{t=0}^{N} \left[\frac{\partial l_t}{\partial \theta} + \sum_{k=1}^{t} \frac{\partial l_t}{\partial s_t} \left(\prod_{i=k}^{t} \frac{\partial s_i}{\partial s_{i-1}}\right) \frac{\partial s_k}{\partial \theta}\right] \tag{8}$$

Note the product $\left(\prod_{i=k}^{t} \frac{\partial s_i}{\partial s_{i-1}}\right)$ appearing on the right hand side of equation 8. The matrix of partial derivatives $\frac{\partial s_i}{\partial s_{i-1}}$ is exactly the Jacobian of the dynamical system (f), and this has precisely the iterated structure discussed in the beginning of this section. Thus, one might not be surprised to find that the gradients of loss functions of dynamical systems depend intimately on the spectra of Jacobians.

At the risk of belaboring the point: As N grows, the number of products in the sum also grows. If $\frac{\partial s_i}{\partial s_{i-1}}$ is a constant then the gradient will either exponentially grow, or shrink in N leading to exploding or vanishing gradients. When the magnitude of all eigenvalues of $\frac{\partial s_i}{\partial s_{i-1}}$ are less than one, the system is stable and the resulting product will be well behaved. If some, or all, of the eigenvectors are above one, the dynamics can diverge, and could even be chaotic [Bollt, 2000]. If the underlying system is known to be chaotic, namely small changes in initial conditions result in diverging states – e.g. rigid body physics simulation in the presence of contacts [Coluci et al., 2005] – this product will diverge. This concept is often colloquially called the butterfly effect Lorenz [1963].

Thus far, we have made the assumption that the system is deterministic. For many systems we care about the function f is modulated by some, potentially stochastic, procedure. In the case of neural network training and learned optimization, this randomness could come from different minibatches of data, in the case of reinforcement learning this might come from the environment, or randomness from a stochastic control policy. In physics simulation, it can even come from floating point noise in how engine calculations are handled on an accelerator. To compute gradients a combination of the reparameterization trick [Kingma and Welling, 2013] and Monte Carlo estimates are usually employed, averaging the gradients computed by backpropagating through the non-stochastic computations [Schulman et al., 2015]. The resulting loss surface of such stochastic systems are thus "smoothed" by this stochasticity and, depending on the type of randomness, this could result in better behaved
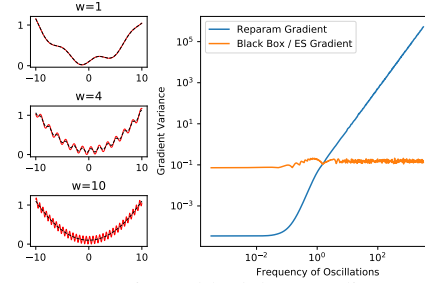

Figure 1: Sometimes, black box gradient estimates can result in lower variance gradient estimates. On the left, we plot $l(x) = 0.1\sin(xw/(\pi)) + (x/10)^2 + 0.1$ for different values of w in red, as well as the loss smoothed by convolving with a 0.3 std Gaussian. On the figure to the right we show the max gradient variance computed over all $x \in [-10, 10]$. When the frequency of oscillations grows higher, the reparameterization gradient variance also grows while the back box gradient remains constant.

loss functions (e.g. smoother, meaning the true gradient has a smaller norm). In chaotic systems, this notion of smoothing is related to the "chaotic hypothesis" which loosely states that time averages in ergodic system are well behaved even if individual trajectories are not [Ruelle et al., 1980, Gallavotti and Cohen, 1995]. Due to the nature of how the reparameterized gradients are estimated – taking products of sequences of state Jacobian matrices – they can result in extremely high gradient norms. In some cases, throwing out the fact that the underlying system is differentiable and using black box methods to estimate *the same* gradient can result in a better estimate of the exact same gradient [Parmas et al., 2018, Metz et al., 2019]! Even when the underlying objective is smooth, but chaotic dynamics and noise give rise to exploding gradients, these black box methods are known to provide low variance estimates, as pointed out by [Parmas et al., 2018].

As a simple example of this, consider some recurrent system with loss $\hat{l}(\theta)$ made stochastic by sampling parameter noise from a Gaussian between $\mathcal{N}(0, I\sigma^2)$ where σ^2 is the standard deviation of

the smoothing. If the underlying loss is bounded in some way, then the smoothed loss's gradients will also be bounded, controlled by the amount of smoothing (σ^2). If using the reparameterization trick and backprop to estimate gradients of this this, however, depend on the unsmoothed loss $\hat{l}$, and could have extremely large gradients even growing exponentially and thus exponentially many samples will be needed to obtain accurate estimates. One can instead employ a black box estimate similar to evolutionary strategies [Rechenberg, 1973, Schwefel, 1977, Wierstra et al., 2008, Schulman et al., 2015] or variational optimization [Staines and Barber, 2012] to compute a gradient estimate. Because this just works with the function evaluations of the un-smoothed loss the gradient estimates again become better behaved. This comes at the cost, however, of poor performance with increased dimensionality.

The fact that black box gradients can have better variance properties is counter intuitive. As a pictorial demonstration, consider figure 1. Instead of using a recurrent system, we simply plot $\hat{l}(x) = 0.1\sin(xw/(\pi))+(x/10)^2+0.1$ and vary w as a proxy for potentially exploding gradients. We assume a Gaussian smoothing of this unsmoothed loss ($\sigma = 0.3$). We find for lower frequency oscillations the reparameterization gradient results in lower variance gradient estimates where as for higher frequency oscillations the black box estimate has lower variance gradients and the reparameterization variance continues to grow. A similar set of experiments comparing among gradient estimators has also been shown in Gal [2016], Mohamed et al. [2020]. This example, and many others explored in this paper are multiscale in that they have a high curvature local structure, with some other global structure. Kong and Tao [2020] showed that even even when performing noise free gradient descent, with a large enough learning rate, optimization resembles stochastic gradient descent converging to distributions of minima.

3   Chaotic loss across a variety of domains

In this section we will demonstrate chaotic dynamics which result in poorly behaved gradients in a variety of different systems. Note that these systems are not chaotic over the entire state space. It's often possible to find restrictions to the problem that restore stability. For example, in a rigid body simulator, simulating in a region without contacts. Or in the case of optimization trajectories, simulating with an extremely small learning rate. However, in many cases, these regions of state space are not the most "interesting", and running optimization most generally moves one towards regions of instability [Xiao et al., 2020]. Chaotic dynamics in iterative systems have been demonstrated in physical systems in Parmas et al. [2018], in optimization in [Pearlmutter, 1996, Maclaurin et al., 2015] and learned optimization in [Metz et al., 2019].

3.1   Rigid Body Physics

First, we consider differentiating through physics simulations. We make use of the recently introduced Brax [Freeman et al., 2021] physics package and look at policy optimization of some stochastic policy parameterized via a neural network. We test this using the default Ant environment and default MLP policy. For all computations, we use double precision floating point numbers.

For each environment, we first fix randomness and evaluate the loss along a fixed, random direction for different numbers of simulation steps (figure 2a). For small number of steps, the resulting loss appears smooth, but as more and more steps are taken the loss surface becomes highly sensitive to the parameters of the dynamical system (θ). Next, we show the loss surface, with the same shifts, averaged over a number of random seeds(figure 2b). This randomness controls the sampling done by the policy. This averaged surface is considerably better behaved, similar to what was shown in Parmas et al. [2018]. Finally, we look at the variance of the gradients computed over different random seeds as a function of the unroll length(figure 2c). We test 4 different locations to compute gradients – using the same shift direction used in the 1D loss slices. Despite the smoothed loss due to averaging over randomness, we find an exponential growth in gradient variance as unroll length increases and great sensitivity to where gradients are being computed. When averaging over gradient samples we can reduce variance as $1/\sqrt{N}$ where N is the number of samples, but this quickly computationally infeasible as gradients norm can grow exponentially!
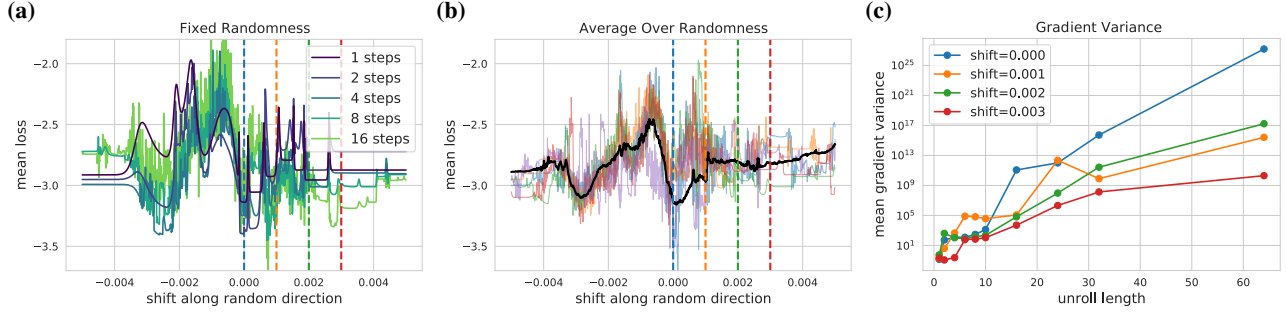
Figure 2: Loss surface and gradient variance for a stochastic policy on a a robotics control task – the Ant environment from Brax. **(a)**: We show a 1D projection of the loss surface along a random direction. All randomness in this plot is fixed. Color denotes different lengths of unroll when computing the loss. For small numbers of iterations the loss is smooth. For higher numbers of steps the underlying loss becomes highly curved. **(b)**: Instead of fixing randomness as done in the left plot, we average over multiple random samples for the 8 step unroll (average is in black, samples are in colors). We find that averaging greatly smooths the underlying loss surface. **(c)**: We look at gradient variance of gradients computed over multiple random samples from the stochastic policy. We show three different parameter values (shifts corresponding to the x-axis in the first two plots and are denoted with the same color vertical dashed lines). Despite having a seemingly smooth loss surface, the gradient variance explodes in exponential growth.

3.2 Meta-learning: Backpropagation through learned optimization

Next we explore instabilities arising from backpropagating through unrolled optimization in a meta-learning context. We take the per parameter, MLP based, learned optimizer architecture used in [Metz et al., 2019] and use this to train 2 layer, 32 hidden unit MLP's on MNIST [LeCun, 1998]. To compute gradients with respect to the learned optimizer parameters, we iteratively apply the learned optimizer using inner-gradients computed on a fixed batch of data. Analogous to the sum of rewards in the previous section, we use the average of the log loss over the entire unroll as our meta-objective – or the objective we seek to optimize the weights of the learned optimizer against.

In figure 3a, we show this loss computed with different length unrolls (shown in color). We can see the same "noisy" loss surfaces with increased unroll length as before despite again having no sources of randomness. Not all parameter values of learned optimizer parameter are sensitive to small changes in value. Many randomly chosen directions resulted in flat, well behaved loss landscapes. For this figure, we selected an initialization and direction out of 10 candidates to highlight this instability. In figure 3b, we numerically compute the average loss smoothed around the current learned optimizer parameter value by a Gaussian with a standard deviation of 0.01 similar to what is done in [Metz et al., 2019]. We see that this smoothed loss surface appears to be well behaved – namely low curvature. Finally in figure 3c we compute the variance of the meta-gradient (gradient with respect to learned optimizer weights) over the loss smoothed by the normal distribution. As before, we compute this gradient at different parameter values. For some parameter values the gradient variance grows modestly. In others, such as the 0.008 shifted value, the parameter space is firmly in the unstable regime and the gradient variance explodes.

3.3 Molecular Dynamics

Finally, we tune the properties of a simple material by differentiating through a molecular dynamics trajectory. In particular, we consider a variant of the widely studied "packing problem" [Lodi et al., 2002] in which disks are randomly packed into a box in two-dimension [O'Hern et al., 2003]. We take a bi-disperse system composed of disks of two different diameters in a box of side-length L; the smaller disks are taken to have diameter D, while the larger disks have a fixed diameter of one. We fix the side-length so that the volume of space taken up by the disks, called the packing fraction, is constant (set to $\phi = 0.98$) as we vary D. It is well-known that different choices of D lead to significantly different material properties: when D is close to one the system forms a hexagonal crystal that packs nicely into the space; when $D \ll 1$ the smaller disks fit into the interstices of the larger disks that once again form a hexagonal crystal; however, when $D \sim 0.8$ the packing
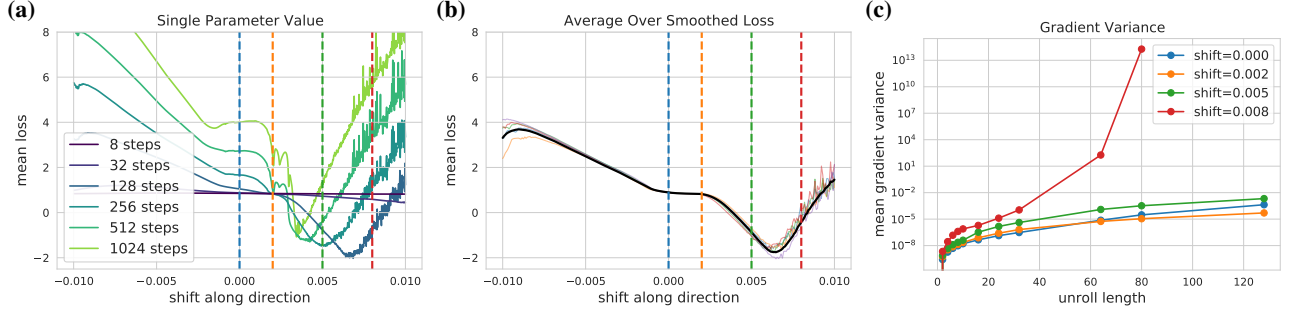
5

(a) Single Parameter Value
(b) Average Over Smoothed Loss
(c) Gradient Variance

Figure 3: Loss surface and gradient variance calculations for meta-learning an optimizer. **(a)**: We show a 1D projection of the meta-loss surfaces (loss with respect to learned optimizer parameters) for different length unrolls – in this case, different numbers of application of the learned optimizer. For small numbers of steps, we find a smooth loss surfaces, but for higher numbers of steps we see a mix of smooth, and high curvature regions. **(b)**: We show an average of the meta-loss over Gaussian perturbed learned optimizer weights. The average is shown in black, and the losses averaged over are shown in color. We find this averaged loss is smooth and appears well behaved. **(c)**: We plot gradient variance over the different perturbations of the learned optimizer weights. These perturbations are shifts corresponding to the x-axis in the first two figures and are marked there with colored dashed vertical lines. For some settings of the learned optimizer weights (corresponding to the x-axis of the first 2 figures) we find well behaved gradient variance. For others, e.g. red, we find exponential growth in variance

becomes disordered and the disks are not able to pack as tightly. To generate packings, we use JAX MD [Schoenholz and Cubuk, 2020] to produce some initial configuration of disks and then use a momentum-based optimizer called FIRE [Bitzek et al., 2006] to quench the configuration to the nearest minimum.

Previous work showed that D could be tuned to find the maximally disordered point by differentiating through optimization [Schoenholz and Cubuk, 2020] provided the system was initialized close to a stable packing. In this regime, the optimization procedure was not chaotic since small changes to the initial configuration of disks will lead to the same final packing. On the other hand, if the disks are randomly initialized then the dynamics become chaotic, since small changes to the initial configuration will lead to significantly different packings. To demonstrate that these chaotic dynamics spoil gradient estimates computed using automatic differentiation, we randomly initialize disks varying D and the random seed; we compute the derivative of the final energy with respect to D by differentiating through optimization.

In figure 4 we show the results of differentiating the energy through optimization with respect to the diameter. In figure 4a we see the energy for a number of different random seeds. As the number of steps of optimization grows, the energy decreases but the variance across seeds increases. In figure 4b we see the energy for a number of different random seeds after 256 steps of optimization along with the energy averaged over seeds. We see that, especially for small diameters, the variance is extremely large although the average energy is well-behaved. Finally in figure 4c, we see the variance of the gradients as a function of optimization steps for several different diameters. We see that the variance grows quickly, especially for the smallest particle diameters.

3.4 Connecting gradient explosion to spectrum of the recurrent Jacobian

To better understand what causes these gradient explosions, we look to measuring statistics of the recurrent Jacobian ($\frac{\partial s_i}{\partial s_{i-1}}$) as well as the product of recurrent Jacobians ($\prod_{i=0}^{t} \frac{\partial s_i}{\partial s_{i-1}}$). We do this experimentally on the Ant environment with two different parameter values with which to measure at. First, a random initialization of the NN policy (init1), which is poorly behaved and results in exploding gradient norms, and an initialization which shrinks this initialization by multiplying by 0.01. This second initialization was picked so that gradient would not explode. With these two initializations, we plot the spectrum of the Jacobian, the max eigenvalue of the Jacobian for each
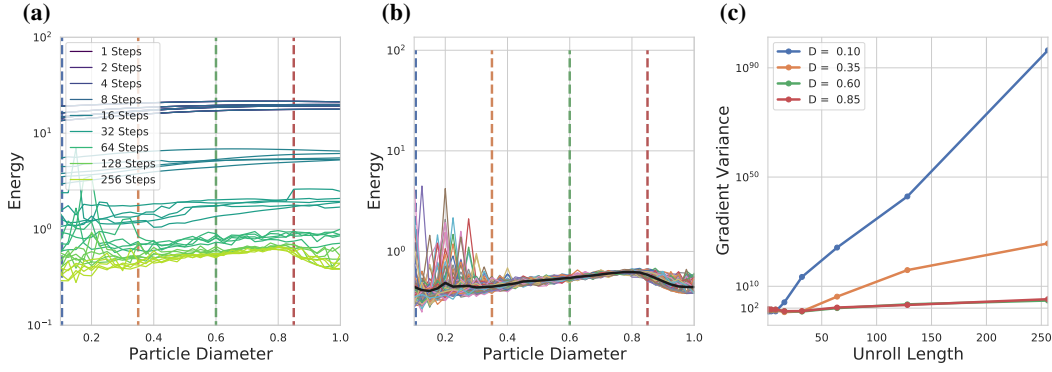
6

Figure 4: Energy for packings of bi-disperse disks varying the diameter of the small disk, D, and the number of optimization steps. **(a)**: The energy of the system as a function of D for different numbers of optimization steps. We see that the energy decreases with more steps of optimization. **(b)**: The energy for the maximum number of optimization steps considered (256). Each individual curve is the energy for one random configuration and the black line indicates the energy averaged over many random seeds. **(c)**: The variance of the gradient estimate for different values of D as a function of the number of steps of optimization.
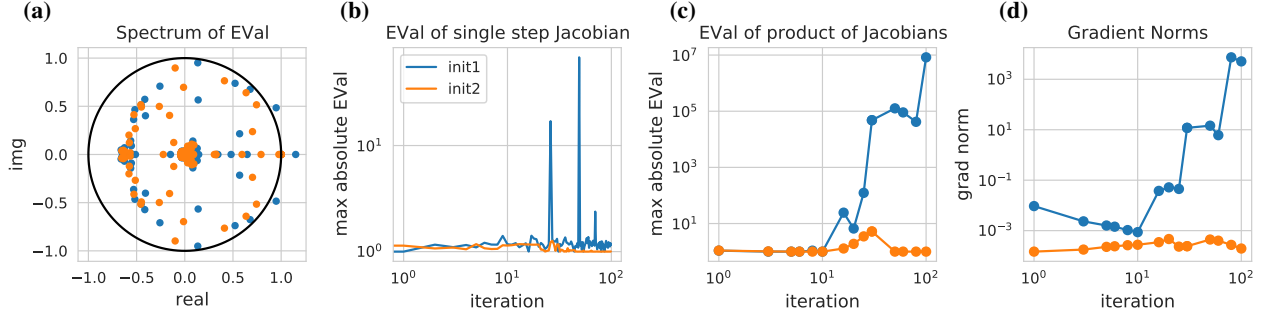

Figure 5: Exploration into the eigenspectrum of the recurrent jacobians of the Brax Ant experiment. We show two parameter values: init1 which is initialized in an unstable regime, and init2 which is in stable regime. **(a)**: we show the spectrum of the recurrent jacobain taken from the 90th iteration ($\frac{\partial s_{90}}{\partial s_{89}}$). **(b)**: We plot the length of the maximum norm eigenvalue of each recurrent jacobian along the sequence ($\frac{\partial s_i}{\partial s_{i-1}}$ for each i). **(c)**: We plot the length of the max eigenvalue of the ($\frac{\partial s_i}{\partial s_0}$ for each i) and find that the unstable initialization grows exponentially. **(d)**: We plot the gradient norms of each initialization and find exploding gradients in the unstable initialization.

iteration (i.e. $\frac{\partial s_i}{\partial s_{i-1}}$ for all i), the cumultive max absolute of the product of jacobians (i.e. $\frac{\partial s_i}{\partial s_0}$ for all i), and the gradient norm for a given unroll length. Results in figure 5.

For the unstable initialization (init1) we find many eigenvalues with norm greater than length 1 (figure 5ab blue), and thus find the cumulative max eigen value grows exponentially (figure 5c blue) and thus gradient norms grow(figure 5d blue). For the stable initialization (init2), we find many eigenvalues close to 1 (figure 5ab orange), resulting in little to no growth in the max eigenvalue (figure 5c orange) and thus controlled gradient norms(figure 5d orange).

4   What can be done?

If we cannot naively use gradients to optimize these systems what can we use? In this section we discuss a couple of options often explored by existing work.

7

4.1 Pick well behaved systems

If exploding gradients emerge from chaotic dynamics of the underlying system, one way to "solve" this problems is to change systems. Whether or not this is really a "solution" is, perhaps, a matter of perspective. For example, empirically it seems that many molecular physics systems naturally have dynamics that are not particularly chaotic and gradient-based optimization can be successfully employed (for example [Goodrich et al., 2021, Kaymak et al., 2021]). For many types of systems, e.g. language modeling, the end goal is to find a high performance model, so it doesn't matter if one particular type of architecture has difficult to optimize, chaotic dynamics—we're free to simply pick a more easily optimize-able architecture. We discuss these modifications and why they work in 4.1.1. For other systems, such as rigid-body physics, changing the system in this way will be biased, and such bias could effect one's downstream performance. For example, in rigid-body physics simulations we want to simulate physics, not a non-chaotic version of physics. We discuss modifications that can be done in 4.1.2.

4.1.1 Recurrent neural networks

One general dynamical system explored in deep learning which also has these types of exploding or vanishing gradients are recurrent neural networks (RNN). The gradient of a vanilla RNN exhibits exactly the same exponentially sensitive dynamics described by Eq. 8, with vanishing/exploding gradients depending on the jacobian of the hidden state parameters.[Pascanu et al., 2013].

Of the many solutions discussed, we will highlight 2 which overcome this issue: different initializations, and different recurrent structure.

Change the initialization: IRNN[Le et al., 2015] work around this problem by initializing the RNN near the identity. At initialization this means the recurrent Jacobian will have eigenvalues near 1 and thus be able to be unrolled longer before encountering issues. After training progresses and weights update, however, the Jacobian drifts, eventually resulting in vanishing/exploding gradients late enough in training.

Change recurrent structure: A second solution is to change the problem entirely. In the case of an RNN this is feasible by simply changing the neural architecture. LSTM's [Hochreiter and Schmidhuber, 1997], GRU [Chung et al., 2014], UGRNN [Collins et al., 2016] are such modifications. As shown in [Bayer, 2015], the recurrent jacobian of an LSTM was specifically designed to avoid this exponential sensitivity to the hidden state, and is thus significantly more robust than a vanilla RNN. While changing architecture does increase stability, it comes at the cost of no-longer being able to model chaotic relationships. Monfared et al. [2021] discuss this, and the relationship chaos and the gradients of the loss.

4.1.2 Rigid Body Physics

Physics simulation involving contact is notoriously difficult to treat differentiably. The problem arises from sharp changes in object velocity before and after a contact occurs (e.g., a ball bouncing off of a wall). Various methods have been developed recently to circumvent this issue. Contact "softening" has proven particularly fruitful, where contact forces are blurred over a characteristic lengthscale, instead of being enforced at a sharp boundary [Huang et al., 2021]. Others have explicitly chunked the process of trajectory optimization into a sequence of mini-optimizations demarcated by moments of contact [Cleac'h et al., 2021].

While these methods have proven fruitful in several domains, they introduce significant algorithmic complexity. This may be a necessary cost—i.e., difficult problems requiring difficult solutions—but we have also found black box methods to extremely reliably solve these problems without needing to introduce any task-specific algorithmic tuning (Section 4.6). It's surprising, then, that gradients seem to *introduce* complexity to the task of trajectory optimization in robotics.

4.1.3 Well-behaved proxy objectives

In some cases, especially in statistical physics systems, features of the energy landscape that govern properties of interest are known. These features can serve as proxy objectives that can be optimized using automatic differentiation without differentiating through long simulation trajectories. In atomistic systems, for example: the eigenvalues of the hessian near minima – called phonons –
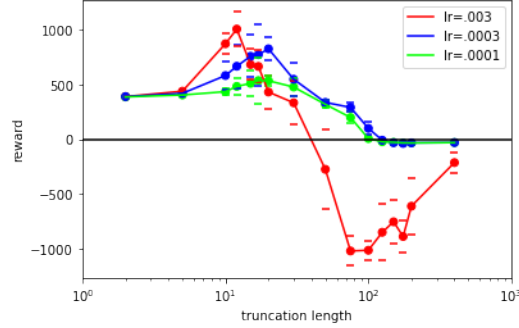
Figure 6: Reward on a modified version of the ant locomotion task in Brax. In this task, we backpropagate the task reward directly to the policy parameters after 400 steps in the environment. For truncation length t, a stop_gradient op was inserted every t steps of the 400 step trajectory. Short truncations typically optimize a lunging policy that results in the ant moving a short distance to the right, but staying on its feet. Long truncations uniformly fail to optimize. Truncations around length 10 result in successful locomotive policies.

control properties ranging from heat transport to stiffness [Ashcroft et al., 1976], the height of saddle points in the landscape compared to the minima control the rate at which the system moves between minima [Truhlar et al., 1996]. These properties have been successfully optimized using automatic differentiation [Schoenholz and Cubuk, 2020, Blondel et al., 2021, Goodrich et al., 2021] and do not suffer from bias as in the case of truncated gradients. Of course, this approach can only be exploited when such proxy objectives are known *a priori*.

In cases where the property of the landscape that we are trying to optimize is known and can be phrased in terms of the fixed point of some dynamical system, there can often be significant benefit to computing derivatives implicitly rather than unrolling optimization explicitly Rajeswaran et al. [2019], Bai et al. [2019], Blondel et al. [2021]. Using implicit differentiation in this way removes conditioning issues due to unrolling and improves memory cost of computing derivatives. However, these methods are only well-defined when the iterated map converges deterministically to a single fixed point. In the case of gradient descent, this often corresponds to initializing the solver in a convex region around minima of the landscape. As such, implicit gradients do not fundamentally resolve the issue of chaos in the general case.

It is also possible to modify the underlying loss surface to better regularize it. Ingraham et al. [2018] regularize their dynamics function to be approximarly 1-Lipshitz. If exact this would imply chaos is impossible. In climate sciences, "nudging" has been proposed [Abarbanel et al., 2010, Köhl and Willebrand, 2002]. These methods modify the underlying loss with a term that "nudges" the state to stay near some reference trajectory. By regularizing this way, gradients over many iterations grow at a slower rate. While they have been used to tune parameters of weather simulators, we are not aware of any applications in machine learning.

4.2 Truncated backpropogation

Another common technique used to control these issues is called truncated backpropogation through time [Werbos, 1990, Tallec and Ollivier, 2017]. This has been used to great effect in training of language models [Sutskever, 2013], learned optimizers [Andrychowicz et al., 2016, Wichrowska et al., 2017, Lv et al., 2017, Metz et al., 2019], and fluid simulation [Kochkov et al., 2021]. This type of solution comes at the cost of bias, however, as the gradients computed are missing terms – in particular, the longer products of Jacobians from equation 8. Depending on the underlying task this bias can be severe [Wu et al., 2016], while for other tasks, empirically, this matters less.

To demonstrate the effects of truncation we attempt to train a policy for the Ant environment in Brax using batched stochastic gradient descent, visualized in Fig. 6. Learning via the raw gradient signal backpropagated through 400 environment steps catastrophically fails to find any useful policy despite an extensive hyper parameter search. However, with truncated gradient updates, there exists a narrow band of truncation lengths at which the Ant policy is able to successfully learn a locomotive gait.

9

Truncated backpropogation operates by stopping backpropogated signal. Ingraham et al. [2018] soften this approximation by decaying the gradient signal each iteration and thus lowering the eigenvalues of the recurrent Jacobian.

4.3 Gradient clipping

Another common technique to train in the precesses of exploding gradients is gradient clipping Pascanu et al. [2013]. Instead of taking the true gradients, one can train using gradients clipped in some way. This has proven to be of use in a variety of domains[Kaiser and Sutskever, 2015, Merity et al., 2017, Gehring et al., 2017] but will not fix all problems. As before, this calculation of the gradient is biased. To demonstrate this, we took the same Ant policy and sweep learning rate and gradient clipping strength. We found no setting which results in positive performance and thus omitted the plot.

4.4 Methods for ergodic systems

In some cases, the underlying dynamical system is ergodic which enable more sophisticated gradient estimation techniques. Loosely speaking, ergodicity here means this means that over sufficiently long times, the dynamical system will uniformly sample from a distribution over states in a time-independent manner. Examples of this often arise in physical systems composed of many constituents such as fluids, materials, or models of the climate.

4.4.1 Least Squares Shadowing

Shadowing methods tackle the problem of computing derivatives through infinite length time averages. To do this they take advantage of the shadowing lemma[Pilyugin, 2006]. The shadowing lemma tells us that for every trajectory computed with some numerical error (say rounding errors) stays uniformly close to some true trajectory. Least squares shadowing (LSS) methods make use of this to find a better behaved nearby trajectory and then use this trajectory to compute gradients. To our knowledge, these methods are most often discussed in the continuous time regime requiring differential equation solvers instead of discrete simulation steps, and tangent solvers instead of simple back propogation.

This idea was first introduced in [Wang et al., 2014] with further which construct a least squares problem to find these nearby trajectories, linearize then solves it and uses this solution trajectory to estimate gradients. Non-intrusive LSS (NILSS) [Ni and Wang, 2017] extend this by reducing computation only to the unstable subspace and Finite-Difference NILSS (FD-NILSS) [Ni et al., 2019] leverages finite difference as opposed to a tangent solver. For an in-depth discussion see Ni [2021]. These methods were applied to sensitivity analysis for fluids in turbulent systems Blonigan et al. [2014] and large-eddy simulation [Blonigan et al., 2017].

4.4.2 Inverting the Shadow Operator

Wang [2013] propose another approach to compute derivatives through chaotic systems which also leverages the shadowing lemma and assumes infinite time averages. A "Shadowing Operator" is constructed which which maps between perturbed and true coordinates and inverted to compute gradients. Wang [2013] include both a forward mode differentiation and reverse mode (adjoint method).

4.4.3 Probabilistic approaches

In these approaches instead of working with samples, one works with the distribution of states directly. This idea was explored in climate modeling leveraging the Fokker-Planck equation [Thuburn, 2005]. Other methods make use of the Fluctuation-Dissipation Theorem with various diferent assumptions [Abramov and Majda, 2007, Cooper and Haynes, 2011]. Gutiérrez and Lucarini [2019] cast the evolution of the state space distribution as Markov Chains enabling gradient calculation. Chandramoorthy and Wang [2020]

Another family of methods to compute time averaged gradient involve computing averages over an ensemble of finite horizon evaluations. This concept was first proposed in [Lea et al., 2000] and improved upon by the space-split sensitivity algorithm (S3) [Chandramoorthy and Wang, 2021] to

have better convergence rates. See [Chandramoorthy, 2021] for a more in depth discussion of these, and other methods for differentiating through ergodic systems.

4.5 Learned Models

While differentiating through chaotic dynamics is challenging, it is often much simpler to learn to approximate these functions, and then use the approximation for some other task. Learning models like this is commonplace across many domains: fluid simulation [Ladický et al., 2015, Tompson et al., 2017, Kochkov et al., 2021], control [Ha and Schmidhuber, 2018], and reinforcement learning of games[Hafner et al., 2019, Ozair et al., 2021]. These methods "collapse" an otherwise complex iterative process with a more shallow computation (e.g. a forward pass through a neural network). In addition to being better behaved, these approximate function are often faster to compute than the original simulation. These approximate functions can then be used in place of, or used in combination with the original dynamics.

To our knowledge, the main use of this family of method is in model based reinforcement learning where one learns a model of an environment and uses this model for control or planning. Instead of differentiating through this learned model, however, many methods instead learn a value function [3] which determines performance in the future then differentiates though a handful of steps before truncating the trajectory with this learned value [Feinberg et al., 2018, Buckman et al., 2018, Kurutach et al., 2018, Clavera et al., 2020].

4.6 Just use black box gradients

One somewhat naive sounding solution is to throw away all gradients and resort a some black box method to estimate gradients. In some cases this also forms an unbiased estimate of the exact same gradient of interest. For example REINFORCE with no discounting Williams [1992] can be used to estimate the same gradient as computed in the Brax experiments, and evolutionary strategies [Rechenberg, 1973, Schwefel, 1977, Wierstra et al., 2008, Salimans et al., 2017] can be used to estimate the gradient for the the learned optimizers.

By resorting to a black box method, we theoretically lose a factor of dimensionality of efficiency when estimating gradients. In practice, however, and given that gradient variance can grow exponentially, computing gradients in this way can lead to lower variance estimates. We demonstrated a sketch of this in figure 1. Instead of picking one or the other estimator, one can instead combine the two gradient estimates to produce an even lower variance estimate of the gradients. In the context of continuous control this has been explored in depth as a solution to high variance gradients arising from chaos in Parmas et al. [2018], and in the context of learned optimizers in Metz et al. [2019].

In addition to unbiased methods, there are a host of other methods with varying bias/variance properties that can also be used – most of which coming from the Deep RL community. For example, PPO [Schulman et al., 2017] easily outperforms all of our experiments training the Ant policy with gradients we performed.

5 Discussion

In this paper we dive into chaos as a potential issue when computing gradients through dynamical systems. We believe this is one particularly sinister issue that is often ignored and hence the focus of this work. Many other potential issues also exist. Numerical precision for these unrolled systems is known to be a problem and addressed to some extent by Maclaurin et al. [2015]. Memory requirements are often cited as an issue for backprop as naively it requires N times as much memory as a forward pass where N is the length of the sequence, or $\sqrt{N}$ memory with gradient checkpointing[Griewank and Walther, 2000, Chen et al., 2016]. Finally, many systems of interest could have extremely flat, or hard to explore loss landscapes making gradient based learning extremely tricky. Part of the reason gradient descent works in neural networks is due to over parameterization [Kawaguchi, 2016] and known weights prior/initialization, which is often not possible in simulated systems have.

Despite the large number of issues computing gradients through recurrent processes, they have shown many wonderful results! We hope this paper sheds light into when gradients can be used –

---

[3]or Q function

11

when the recurrent Jacobian has small eigenvalues. In the other cases, when gradients do not work, we encourage readers to try black box methods – they estimate the same quantity and with less pathological variance properties, especially when it's possible to calculate a smoothed proxy for the loss function of interest. In summary, gradients are not all you need. Just because you can take a gradient doesn't mean you always should.

Acknowledgments and Disclosure of Funding

References

Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. NeuralSim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL `https://github.com/google-research/tiny-differentiable-simulator`.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021.

Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019.

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31:7178–7189, 2018.

Paula Gradu, John Hallman, Daniel Suo, Alex Yu, Naman Agarwal, Udaya Ghai, Karan Singh, Cyril Zhang, Anirudha Majumdar, and Elad Hazan. Deluca–a differentiable control library: Environments, methods, and benchmarking. *arXiv preprint arXiv:2102.09968*, 2021.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax-a differentiable physics engine for large scale rigid body simulation. 2021.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

Samuel S. Schoenholz and Ekin D. Cubuk. Jax m.d. a framework for differentiable physics. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020. URL `https://papers.nips.cc/paper/2020/file/83d3d4b6c9579515e1679aca8cbc8033-Paper.pdf`.

Konrad Hinsen. The molecular modeling toolkit: a new approach to molecular simulations. *Journal of Computational Chemistry*, 21(2):79–85, 2000.

Dougal Maclaurin. *Modeling, inference and optimization with composable differentiable procedures*. PhD thesis, 2016.

Christian H Bischof, Gordon D Pusch, and Ralf Knoesel. Sensitivity analysis of the mm5 weather model using automatic differentiation. *Computers in Physics*, 10(6):605–612, 1996.

Nick McGreivy, Stuart R Hudson, and Caoxiang Zhu. Optimized finite-build stellarator coils using automatic differentiation. *Nuclear Fusion*, 61(2):026020, 2021.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Pierre Ablin, Gabriel Peyré, and Thomas Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. In *International Conference on Machine Learning*, pages 32–41. PMLR, 2020.

Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 9(4):e1305, 2019.

Christian Bischof, Andreas Griewank, and David Juedes. Exploiting parallelism in automatic differentiation. In *Proceedings of the 5th international conference on Supercomputing*, pages 146–153, 1991.

George Corliss, Christele Faure, Andreas Griewank, Laurent Hascoet, and Uwe Naumann. *Automatic differentiation of algorithms: from simulation to optimization*. Springer Science & Business Media, 2013.

Shui-Nee Chow and Kenneth J Palmer. On the numerical computation of orbits of dynamical systems: the higher dimensional case. *Journal of Complexity*, 8(4):398–423, 1992.

Tal Kachman, Shmuel Fishman, and Avy Soffer. Numerical implementation of the multiscale and averaging methods for quasi periodic systems. *Computer Physics Communications*, 221:235–245, 2017.

Daniel J Lea, Myles R Allen, and Thomas WN Haine. Sensitivity analysis of the climate of a chaotic system. *Tellus A: Dynamic Meteorology and Oceanography*, 52(5):523–532, 2000.

Armin Köhl and Jürgen Willebrand. An adjoint method for the assimilation of statistical characteristics into eddy-resolving ocean models. *Tellus A: Dynamic meteorology and oceanography*, 54(4): 406–425, 2002.

Greg Yang and Samuel S Schoenholz. Mean field residual networks: On the edge of chaos. *arXiv preprint arXiv:1712.08969*, 2017.

Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In *International conference on machine learning*, pages 2672–2680. PMLR, 2019.

Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4062–4071, 2018.

Paavo Parmas. Total stochastic gradient algorithms and applications to model-based reinforcement learning.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565, 2019.

Angxiu Ni and Qiqi Wang. Sensitivity analysis on chaotic dynamical systems by non-intrusive least squares shadowing (nilss). *Journal of Computational Physics*, 347:56–77, 2017.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

John Ingraham, Adam Riesselman, Chris Sander, and Debora Marks. Learning protein structure with a differentiable simulator. In *International Conference on Learning Representations*, 2018.

David Ruelle. A review of linear response theory for general differentiable dynamical systems. *Nonlinearity*, 22(4):855, 2009.

Erik M Bollt. Controlling chaos and the inverse frobenius–perron problem: global stabilization of arbitrary invariant measures. *International Journal of Bifurcation and Chaos*, 10(05):1033–1050, 2000.

VR Coluci, SB Legoas, MAM De Aguiar, and DS Galvao. Chaotic signature in the motion of coupled carbon nanotube oscillators. *Nanotechnology*, 16(4):583, 2005.

Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *arXiv preprint arXiv:1506.05254*, 2015.

David Ruelle et al. Measures describing a turbulent flow. *Annals of the New York Academy of Sciences*, page 357, 1980.

Giovanni Gallavotti and Ezechiel Godert David Cohen. Dynamical ensembles in stationary states. *Journal of Statistical Physics*, 80(5):931–970, 1995.

Ingo Rechenberg. Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution. 1973.

Hans-Paul Schwefel. Evolutionsstrategien für die numerische optimierung. In *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, pages 123–176. Springer, 1977.

Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3381–3387. IEEE, 2008.

Joe Staines and David Barber. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

Yarin Gal. Uncertainty in deep learning. 2016.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132):1–62, 2020.

Lingkai Kong and Molei Tao. Stochasticity of deterministic gradient descent: Large learning rate for multiscale objective function. *arXiv preprint arXiv:2002.06189*, 2020.

Lechao Xiao, Jeffrey Pennington, and Samuel Schoenholz. Disentangling trainability and generalization in deep neural networks. In *International Conference on Machine Learning*, pages 10462–10472. PMLR, 2020.

Barak Pearlmutter. *An investigation of the gradient descent process in neural networks*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 1996.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(02)00123-6. URL https://www.sciencedirect.com/science/article/pii/S0377221702001236.

Corey S. O'Hern, Leonardo E. Silbert, Andrea J. Liu, and Sidney R. Nagel. Jamming at zero temperature and zero applied stress: The epitome of disorder. *Phys. Rev. E*, 68:011306, Jul 2003. doi: 10.1103/PhysRevE.68.011306. URL https://link.aps.org/doi/10.1103/PhysRevE.68.011306.

Erik Bitzek, Pekka Koskinen, Franz Gähler, Michael Moseler, and Peter Gumbsch. Structural relaxation made simple. *Physical review letters*, 97(17):170201, 2006.

Carl P Goodrich, Ella M King, Samuel S Schoenholz, Ekin D Cubuk, and Michael P Brenner. Designing self-assembling kinetics with differentiable statistical physics models. *Proceedings of the National Academy of Sciences*, 118(10), 2021.

Mehmet Cagri Kaymak, Ali Rahnamoun, Kurt A O'Hearn, Adri CT van Duin, Kenneth M Merz Jr, and Hasan Metin Aktulga. Jax-reaxff: A gradient based framework for extremely fast optimization of reactive force fields. 2021.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

Justin Simon Bayer. *Learning sequence representations*. PhD thesis, Technische Universität München, 2015.

Zahra Monfared, Jonas M Mikhaeil, and Daniel Durstewitz. How to train rnns on chaotic data? *arXiv preprint arXiv:2110.07238*, 2021.

Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. 2021.

Simon Le Cleac'h, Taylor Howell, Mac Schwager, and Zachary Manchester. Fast contact-implicit model-predictive control. 2021.

Neil W Ashcroft, N David Mermin, et al. Solid state physics, 1976.

Donald G Truhlar, Bruce C Garrett, and Stephen J Klippenstein. Current status of transition-state theory. *The Journal of physical chemistry*, 100(31):12771–12800, 1996.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation, 2021.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. 2019.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Henry DI Abarbanel, Mark Kostuk, and William Whartenby. Data assimilation with regularized nonlinear instabilities. *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography*, 136(648):769–783, 2010.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *International Conference on Machine Learning*, 2017.

Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. *arXiv preprint arXiv:1703.03633*, 2017.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger B Grosse. Understanding short-horizon bias in stochastic meta-optimization. pages 478–487, 2016.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR, 2017.

Sergei Yu Pilyugin. *Shadowing in dynamical systems*. Springer, 2006.

Qiqi Wang, Rui Hu, and Patrick Blonigan. Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations. *Journal of Computational Physics*, 267:210–224, 2014.

Angxiu Ni, Qiqi Wang, Pablo Fernandez, and Chaitanya Talnikar. Sensitivity analysis on chaotic dynamical systems by finite difference non-intrusive least squares shadowing (fd-nilss). *Journal of Computational Physics*, 394:615–631, 2019.

Angxiu Ni. *Numerical Differentiation of Stationary Measures of Chaos*. PhD thesis, University of California, Berkeley, 2021.

Patrick J Blonigan, Steven A Gomez, and Qiqi Wang. Least squares shadowing for sensitivity analysis of turbulent fluid flows. In *52nd Aerospace Sciences Meeting*, page 1426, 2014.

Patrick J Blonigan, Pablo Fernandez, Scott M Murman, Qiqi Wang, Georgios Rigas, and Luca Magri. Toward a chaotic adjoint for les. *arXiv preprint arXiv:1702.06809*, 2017.

Qiqi Wang. Forward and adjoint sensitivity computation of chaotic dynamical systems. *Journal of Computational Physics*, 235:1–13, 2013.

J Thuburn. Climate sensitivities via a fokker–planck adjoint approach. *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography*, 131(605):73–92, 2005.

Rafail V Abramov and Andrew J Majda. Blended response algorithms for linear fluctuation-dissipation for complex nonlinear dynamical systems. *Nonlinearity*, 20(12):2793, 2007.

Fenwick C Cooper and Peter H Haynes. Climate sensitivity via a nonparametric fluctuation–dissipation theorem. *Journal of the Atmospheric Sciences*, 68(5):937–953, 2011.

Manuel Santos Gutiérrez and Valerio Lucarini. Response and sensitivity using markov chains. *arXiv preprint arXiv:1907.12881*, 2019.

Nisha Chandramoorthy and Qiqi Wang. A computable realization of ruelle's formula for linear response of statistics in chaotic systems. *arXiv preprint arXiv:2002.04117*, 2020.

Nisha Chandramoorthy and Qiqi Wang. Efficient computation of linear response of chaotic attractors with one-dimensional unstable manifolds. *arXiv preprint arXiv:2103.08816*, 2021.

Nisha Chandramoorthy. *An efficient algorithm for sensitivity analysis of chaotic systems*. PhD thesis, Massachusetts Institute of Technology, 2021. URL https://web.mit.edu/nishac/www/papers/PhD_Thesis-compressed.pdf.

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aäron van den Oord, and Oriol Vinyals. Vector quantized models for planning. *arXiv preprint arXiv:2106.04615*, 2021.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Ignasi Clavera, Violet Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. *arXiv preprint arXiv:2005.08068*, 2020.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of check-pointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Kenji Kawaguchi. Deep learning without poor local minima. *arXiv preprint arXiv:1605.07110*, 2016.

Vladislav Kargin. Products of random matrices: Dimension and growth in norm. *The Annals of Applied Probability*, 20(3):890–906, 2010.

Dragana Bajovic. *Large Deviations Rates for Distributed Inference*. PhD thesis, Instituto Superior Técnico Lisbon, Portugal, 2013.

A Progression of dynamical systems

Dynamical progression of training systems are very subtle. In it's core is an iterative algebraic process where one defines a canonical transformation to take a state vector from time t to $t+1$. Albiet for long range temporal processes there could be a cascade of confounder that effect the convergence properties or the generated dynamics. In the context of automatic differentiation this can become even more critical since the typical number of dynamical steps spans vast orders of magnitude. In what follow we will overview some of the convergence, or lack of, properties for different scenarios of the dynamical progression.

A.1 Deterministic single transformation

We start off by considering the iterative map A as a propagator of dynamics i.e

$$x^1 = Ax^0$$

where the superscript denotes at time step. If the transformation is always the same then the state at point k is simply

$$x^k = A^k x_0 \tag{9}$$

we note, that this will only converge for every initial state x_0 iif the eigenvalues $\lambda \neq 1$ have an amplitude $|\lambda| < 1$ and more harshly if the spectrum contains $\lambda^* = 1$ then it has to have a rank that fulfills $\phi(\lambda) = \det(\lambda I - A)$. To show this, let us examine thee Jordan canonical form of the matrix

$$A = XDX^{-1}$$

where X is a matrix who's columns are the eigenvalues and D is a diagonal matrix who's diagonal are the eigenvalues. The dynamical progression 9 under the Jordan form becomes

$$x^k = A^k x_0 = \left(XDX^{-1}\right)^k x_0 = X \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{pmatrix} X^{-1} x_0$$

we see that any eigenvalues smaller then 1 will decay with larger maps, and eigenvalues which are bigger then one will diverge. Interestingly, if the eigenvalue of $\lambda = 1$ does exist, such eigenvalues may yield periodic or nearly periodic orbits or even orbits that may diverge to infinity.

A.2 Deterministic multiple transformation

Last section all of the time dynamics was exactly the same, we can relax this assumption by looking at transformation functions that are inhomoginues over time, i.e.,

$$A^i \neq A^{i+1}$$

and thus the spectrum also changes over time. For this scenario we can write down the dynamics in the following way

$$x^k = \prod_{i=0}^{k} A^i x_0$$

and using the Jordan form

$$x^k = X^0 \prod_{i=0}^{k} D^i \left(X^i\right)^{-1} x_0$$

to simplify this lets look at first consider the case of small increments between two steps $X^i D^i \left(X^i\right)^{-1} X^{i+1} D^{i+1} \left(X^{i+1}\right)^{-1}$. For a slowly varying dynamics or matrices then we can approximate

$$\left(X^i\right)^{-1} X^{i+1} \approx I + \epsilon$$

Where we can numerically justify the small variation angle by a simple numerical check as shown in figure 7. With this the transformation now becomes
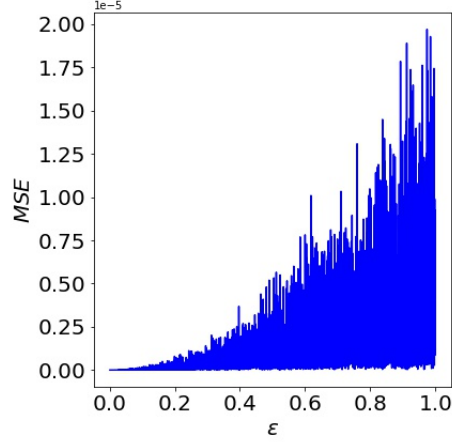
Figure 7: MSE as a function of a small perturbation $\epsilon \in \mathbb{R}^{N \times N}$ where $X^{i+1} = X^i + \epsilon$

$$x^k = X^0 \prod_{i=0}^{k} D^i \left(X^i\right)^{-1} x_0 = X^0 \prod_{i=0}^{k} D^0 \left(I + \epsilon\right) D^1 \left(I + \epsilon\right) D^2 ...... \left(X^0\right)^{-1} x_0$$

$$x^k \approx X^0 \left(\prod_{i=0}^{k} D^I\right) \left(X^0\right)^{-1} x_0$$

$$x^k = X \begin{pmatrix} \prod_{i=0}^{k} \lambda_1^k & & & \\ & \prod_{i=0}^{k} \lambda_2^k & & \\ & & \ddots & \\ & & & \prod_{i=0}^{k} \lambda_n^k \end{pmatrix} X^{-1} x_0$$

such we see the same result, for eigenvalues which are bigger then 1 we will have divergence, for smaller then one the dynamic will halt. More interestingly for this case, we don't just need a single eigenvalue of 1 but rather a cascade of same scale eigenvalues for the dynamics to converge.

A.3 Stochastic transformations

More interestingly one can also consider the case of a random transformation where A is in fact a stochastic random matrix. For this case, let us look at two different cases.

A.3.1 Gaussian randomness

For this case, it is easier to look at the product of eigenvalues i.e the determinant of

$$x^k = \prod_{i=0}^{k} A^i x_0$$

and defining

$$d_k = \det \left(\prod_{i=0}^{k} A^i\right) = \prod_{i=0}^{k} \det A^i$$

now if $a_{ij}\left(i\right) \sim \mathcal{N}\left(0, \sigma^2\right)$ then the expectation is

$$\mathbb{E}\left[d_k\right] = 0$$

which we also mark as

$$\mathbb{E}\left[d_k\right] = \mu_k$$

20

now be denoting the symmetric and non symmetric parts of the minor as

$$\prod_{i,j} a_{ij}(i) = \begin{cases} S_1(i) & i = j \\ S_2(i) & i \neq j \end{cases}$$

Where S are the chi-squared distribution, then we can write the Variance as

$$\mathbb{V}[d_k] = \mathbb{E}[d_k^2] = \sigma^{4k} \prod_{i=0}^{k} \mathbb{E}\left[(S_1(i) - S_2(i))^2\right]$$

This is important since the difference of two chi squared distribution is a generalized Laplace distribution

$$S_1(i) - S_2(i) \sim \Gamma_\nu \left(\mu = 0, \alpha = \frac{1}{2}, \beta = 0, \lambda = \frac{1}{2}, \gamma = \frac{1}{2}\right)$$

which also tells us that

$$\mathbb{V}[S_1(i) - S_2(i)] = \frac{2\lambda}{\gamma^2}\left(1 + \frac{2\beta^2}{\gamma^2}\right) = 4$$

such

$$\mathbb{V}[d_k] = 4^k \sigma^{4k}$$

Similar to what we have seen before, the convergence of this depends on the variance.

The problem of bounding non Gaussian noise has been extensively discussed in the literature [Kargin, 2010, Bajovic, 2013]. While it is an interesting case, an in depth evaluation of this type bound is out of the scope of this paper and is the basis for future exploration.

A.3.2 On the special case of discrete laplacian

We have discussed gradients and maps with their dynamical progression. It's worthwhile to look at one special case, where the dynamical progression is dependent on the hessian i.e a mapping of the following form

$$u_t = \alpha u_{xx}$$

under some boundary conditions

$$\begin{cases} x \in [a, b] \\ u(x, 0) = f(x) \\ u(a, t) = u(b, t) = 0 \end{cases}$$

under a fixed grid discretization of $\Delta x = \frac{b-a}{n}$ the dynamical process actually becomes

$$\begin{pmatrix} u_1^{j+1} \\ u_2^{j+1} \\ \vdots \\ \vdots \\ u_{n-1}^{j+1} \end{pmatrix} = \begin{pmatrix} 1 - 2\frac{\alpha\Delta t}{\Delta x^2} & \frac{\alpha\Delta t}{\Delta x^2} & & & \\ \frac{\alpha\Delta t}{\Delta x^2} & 1 - 2\frac{\alpha\Delta t}{\Delta x^2} & & & \\ & & \ddots & & \\ & & & \ddots & \frac{\alpha\Delta t}{\Delta x^2} \\ & & & \frac{\alpha\Delta t}{\Delta x^2} & 1 - 2\frac{\alpha\Delta t}{\Delta x^2} \end{pmatrix} \begin{pmatrix} u_1^j \\ u_2^j \\ \vdots \\ \vdots \\ u_{n-1}^j \end{pmatrix}$$

$$u^j = A^j u^0$$

If $\frac{\alpha\Delta t}{\Delta x^2} < 0.5$ then the map matrix A is non negative and thus the forbeniues perron theorem tells us that there is a dominant eigenvalue $\lambda > 0$ with a non negative eigenvector v. Building on this we can write the eigenvalue equation (in coordinate to coordinate transformation)

$$\lambda v = Av$$
$$\lambda v_i = \frac{\alpha\Delta t}{\Delta x^2} v_{i-1} + \left(1 - 2\frac{\alpha\Delta t}{\Delta x^2}\right) v_i + h v_{i+1} \leq 1$$

more so, if i index corresponds to $v_i = 1$ then $\lambda < 1$. This actually lets us state Gerschgorin's theorem:

Each eigenvalue λ of the real or complex matrix $A = (a_{ij})$ lies in at least one of the close disks

$$|\lambda - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|$$

A.3.3 On the special case of discrete drift diffusion

As a further escalation, let us now look at the drift diffusion equation

$$u_t = \alpha u_{xx} + \beta u_x$$

under some boundary conditions

$$\begin{cases} x \in [a, b] \\ u(x, 0) = f(x) \\ u(a, t) = u(b, t) = 0 \end{cases}$$

under a fixed grid discretization of $\Delta x = \frac{b-a}{n}$ the dynamical process actually becomes

$$u_i^{j+1} = u_i^j + \frac{\alpha \Delta t}{\Delta x^2}\left(u_{i+1}^j - 2u_i^j + u_{i-1}^j\right) + \frac{\beta \Delta t}{\Delta x}\left(u_{i+1}^j \text{-} u_i^j\right)$$

$$\begin{pmatrix} u_1^{j+1} \\ u_2^{j+1} \\ \vdots \\ \vdots \\ u_{n-1}^{j+1} \end{pmatrix} = \left[\begin{pmatrix} 1 - 2\frac{\alpha \Delta t}{\Delta x^2} & \frac{\alpha \Delta t}{\Delta x^2} & & & \\ \frac{\alpha \Delta t}{\Delta x^2} & 1 - 2\frac{\alpha \Delta t}{\Delta x^2} & \frac{\alpha \Delta t}{\Delta x^2} & & \\ & & \ddots & & \\ & & & \ddots & \frac{\alpha \Delta t}{\Delta x^2} \\ & & & \frac{\alpha \Delta t}{\Delta x^2} & 1 - 2\frac{\alpha \Delta t}{\Delta x^2} \end{pmatrix} \right. $$
$$\left. \begin{pmatrix} -\frac{\beta \Delta t}{\Delta x} & \frac{\beta \Delta t}{\Delta x} & & & \\ 0 & -\frac{\beta \Delta t}{\Delta x} & \frac{\beta \Delta t}{\Delta x} & & \\ & & \ddots & & \\ & & & \ddots & \frac{\alpha \Delta t}{\Delta x^2} \\ & & & & \frac{\beta \Delta t}{\Delta x} \end{pmatrix} \right] \begin{pmatrix} u_1^j \\ u_2^j \\ \vdots \\ \vdots \\ u_{n-1}^j \end{pmatrix} + $$

In this case for the map to converge we have to require

$$1 - 2\frac{\alpha \Delta t}{\Delta x^2} - \frac{\beta \Delta t}{\Delta x} \geq 0$$

$$\Delta x^2 \frac{1}{\Delta t} - \Delta x \frac{\beta}{\Delta t} - \frac{2\alpha}{\Delta t} \geq 0$$

$$\left(\Delta x - \frac{\beta}{\Delta t} \pm \sqrt{\beta^2 + 8\alpha}\right)\left(\Delta x + \frac{\beta}{\Delta t} \pm \sqrt{\beta^2 + 8\alpha}\right) \geq 0$$