Chaotic Dynamics are Intrinsic to Neural Network Training with SGD

Anonymous Author(s)
Affiliation
Address
email

Abstract

With the advent of deep learning over the last decade, a considerable amount of effort has gone into better understanding and enhancing Stochastic Gradient Descent so as to improve the performance and stability of artificial neural network training. Active research fields in this area include exploiting second order information of the loss landscape and improving the understanding of chaotic dynamics in optimization. This paper exploits the theoretical connection between the curvature of the loss landscape and chaotic dynamics in neural network training to propose a modified SGD ensuring non-chaotic training dynamics to study the importance thereof in NN training. Building on this, we present empirical evidence suggesting that the negative eigenspectrum - and thus directions of local chaos - cannot be removed from SGD without hurting training performance. Extending our empirical analysis to long-term chaos dynamics, we challenge the widespread understanding of convergence against a confined region in parameter space. Our results show that although chaotic network behavior is mostly confined to the initial training phase, models perturbed upon initialization do diverge at a slow pace even after reaching top training performance, and that their divergence can be modelled through a composition of a random walk and a linear divergence. The tools and insights developed as part of our work contribute to improving the understanding of neural network training dynamics and provide a basis for future improvements of optimization methods.

1 Introduction

In the last decade, the advent of Deep Learning has led to an explosive development of powerful machine learning models capable of solving ever more complex problems. However, while numerous different architectures have been developed, the underlying optimization procedure - Stochastic Gradient Descent (SGD) (Robbins, 2007) and its descendants (Duchi et al., 2011; Kingma and Ba, 2017) - has largely remained the same, with many aspects of its nature not being fully understood yet. The role of second order dynamics is one of those aspects, as are chaotic dynamics of Artificial Neural Network (ANN) training, both being notoriously difficult to investigate due to the high computational cost involved. In this paper, we establish a connection between the two aspects and generate new insights by studying the intersection of these two through a series of experiments conducted on small-sized models trained on natural datasets. Our main four contributions are as follows:

1. By modeling ANN training with SGD as a time-discrete dynamical system, we propose a modified SGD algorithm ensuring non-chaotic training dynamics to study the importance of chaos in ANN training.
2. We find empirical evidence suggesting that directions of negative curvature - and thus local chaos - cannot be removed without hurting the training performance of ANNs.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

3. We show empirically that the network dynamics start out diverging exponentially at the beginning of the training but transition asymptotically against polynomial behaviour as the model performance converges.
4. Elaborating on the previous aspect, we show that even as the model training converges, the distance between similarly initialized models continues to grow at a small pace, and that this behaviour can be modelled as a sum of a linear divergence and a random walk.

2 Theoretical preliminaries

2.1 Definition of chaos

A popular saying characterizing the notion of chaos states that 'The flapping of a butterfly's wings can cause a storm on the other side of the world.', sometimes also briefly referenced as the butterfly effect. Formally, a dynamical system is chaotic if it is (1) **sensitive to initial conditions**, (2) topologically transitive and (3) has periodic points of the system which are dense in state space (Skokos, 2009, p.3, Def.1). Although a complete analysis of chaos dynamics of a dynamical system would encompass quantifying all three aforementioned properties, property (1) is the most relevant property for model optimization: Given a small deviation in the parameter state of a model at any point of the optimization, one should hope to obtain a similar solution at the end of the optimization procedure, and by extension a solution that performs similarly well both on training and validation data. Therefore, we use chaotic to refer to the sensitivity to initial conditions.

2.2 Recap: Lyapunov Exponents

A common tool for characterizing chaos are Lyapunov exponents, which measure the evolution of displacement vectors over time. Consider a dynamical system $^{(t+1)} = f(\ ^{(t)})$, where $^{(t)}$ is the system state and f the dynamics function which determines how the system evolves over time. Given an initial, infinitesimally small displacement vector $_0$ s.t. $^{0}_{0} = \ _0 + \ _0$, the evolution of the displacement vector over time is given by the so-called tangent map, typically denoted $d_w\ ^{(t)}$ or $Y^{(t)}$ in matrix notation, satisfying

$$^{(t)} = Y^{(t)}\ ^{(0)} = \frac{@\ ^{(t)}}{@\ ^{(0)}}\ ^{(0)}; \tag{1}$$

where the matrix $Y^{(t)}$ is the Jacobian of the dynamics function at time t w.r.t. to the state at the initial time step. When the state $^{(t)}$ depends only on the state at the previous time point $^{(t\ \ 1)}$ the action of the tangent map can be decomposed through application of the chain rule to

$$Y^{(t)} = J_f(\ ^{(t)})Y^{(t\ \ 1)} = \frac{@\ ^{(t)}}{@\ ^{(t\ \ 1)}}Y^{(t\ \ 1)}; \tag{2}$$

with J_f being the Jacobian of the dynamics function at time step t. The Lyapunov exponents are defined as the eigenvalues (the Lyapunov spectrum) of the matrix

$$^{(1)} = \lim_{t\ !\ 1} \frac{1}{2t} \ln Y^{(t)T} Y^{(t)}; \tag{3}$$

which we will refer to as *Lyapunov matrix* from now on. With this definition, we expect initial system perturbations $_0$ to evolve according to

$$(t)\ /\ e^{\ _1 t}\ \ _0; \tag{4}$$

where $_1$ is the maximum eigenvalue of the Lyapunov matrix, also called **maximum Lyapunov Characteristic Exponent** (mLCE). The respective eigenvector gives us the direction of maximum expansion of an initial displacement as the dynamic system evolves. Thus, the mLCE can be used to characterize three different types of system dynamics:

1. $_1 > 0$: Nearby trajectories **diverge exponentially** (chaotic).
2. $_1 = 0$: Nearby trajectories **diverge polynomially** (edge-chaotic).
3. $_1 < 0$: The distance of nearby trajectories is **upper-bounded**.

Although the mLCE is sufficient to characterize the overall system dynamics, a more fine-grained analysis can be obtained in a natural manner by considering the full eigenvalue spectrum of the Lyapunov matrix, with the biggest eigenpair $(\ _1; v_1)$ giving us the "most chaotic" direction of the system evolution, $(\ _2; v_2)$ the second-most chaotic, and so on assuming $_1 \quad _2 \quad \cdots \quad _N$. Thus, for a small initial perturbation $^{\theta(0)} = \ ^{(0)} + \varepsilon v_i$, we expect the perturbation to evolve approximately as dictated by the respective eigenpair as $\quad (t) \ / \ e^{\ _i t} \varepsilon v_i$. Along any direction, we can expect the bound

$$e^{\ _N(t)t}\varepsilon \qquad (t) \quad e^{\ _1(t)t}\varepsilon \tag{5}$$

to hold for sufficiently small ε s.t. the first order approximation

$$^{\theta(t)} \qquad ^{(t)} + Y^{(t)}(\varepsilon w) \tag{6}$$

of the training process at time step t, where w is a unit-length perturbation axis, is accurate.

2.3 Curvature and chaos

A neural network evolving according to SGD with learning rate (without momentum) and trained with data batches $z^{(t)}$ at time step t can be described by the equation

$$^{(t)} = \ ^{(t\ \ 1)} \qquad g(\ ^{(t\ \ 1)}; z^{(t\ \ 1)}); \tag{7}$$

where $g(\ ^{(t\ \ 1)}; z^{(t\ \ 1)})$ is the loss gradient w.r.t. to network parameters $^{(t\ \ 1)}$. We show in Appendix A.1 and A.3 that the evolution of the tangent map without and with momentum can be described as

$$Y^{(t+1)} = (I \qquad H^{(t)})Y^{(t)} \quad \text{and} \quad Y^{(t+1)} = Y^{(t)} \qquad \sum_{s=1}^{t} \ ^{(t\ \ s)} H^{(s)} Y^{(s)} \ ; \tag{8}$$

respectively. This result shows that, discounting the effects of random batch sampling in SGD, the chaotic dynamics depend on the Hessian and thus on the curvature of the loss landscape during training.

2.4 Local chaos and negative Hessian eigenvalue spectrum

Given the connections we established above, one could hope to improve the training performance of SGD by using second-order curvature information to avoid/promote a chaotic evolution of the system, assuming chaos to be detrimental/beneficial for neural network training, e.g. by pruning the directions of maximum chaos from the gradient updates. Unfortunately, the Lyapunov matrix is not practical for this purpose because it considers the chaos over the entire training up to the current time point, so it can only inform us of a chaotic evolution of the system a posteriori. To mitigate this problem, one can instead consider a greedy approach where at any given time step t, one looks at the finite-time Lyapunov matrix after a single time step, i.e.

$$^{(t;t+1)} = \frac{1}{2}\ln(I \qquad H)^T(I \qquad H) = \frac{1}{2}\ln(I \qquad H)^2 \tag{9}$$

The eigenvalues of this matrix are called the **Local Lyapunov Exponents (LLE)** at time step t. Analogously to the LCEs, we say the system is *locally chaotic* in the direction of eigenpair $(\ _i; v_i)$ iff $_i > 0$. Using the fact that $^{(t;t+1)}$ has the same eigenvectors as H, we can relate the eigenspectrum of the Hessian to local chaotic dynamics as follows:

Theorem 2.1. *Given a neural network with N parameters, let H be the Hessian of the loss w.r.t. network parameters, and let $(\ _i; v_i)_{i=1}^N$ be the Hessian's eigenpairs. Furthermore, let us assume that the network is trained using SGD with learning rate and no momentum. Then, the network's LLEs indicate locally chaotic training behaviour in the direction of v_i if*

$$_i < 0 \_ \quad _i > \frac{2}{} \tag{10}$$

For a proof see Appendix A.5. Our result leads to the interesting observation that sufficiently large positive eigenvalues (as determined by the learning rate) imply locally chaotic training behaviour in SGD. More interestingly however, negative eigenvalues automatically lead to locally chaotic training behaviour. We can further extend this observation as shown in Appendix A.7 to arrive at the following theorem

Theorem 2.2. Given a neural network with N parameters, let $H^{(t)}$ be the Hessian of the loss w.r.t. network parameters at time step t. If for all time steps t the Hessian $H^{(t)}$ has at least one negative eigenvalue, then the mLCE of the training satisfies $\lambda_1 > 0$ and thus the global training dynamics are chaotic.

2.5 Pruning chaotic updates

Since the eigenpairs $(s_i; v_i)_{i=1}^{N}$ of $^{(t;t+1)}$ give us the directions of local chaos ordered by magnitude of the LLEs, we can remove the locally chaotic directions from the parameter updates in SGD by projecting the update vector onto non-locally-chaotic eigenvectors of $^{(t;t+1)}$. More speci cally:

Theorem 2.3. Given a neural network with Hessian $H^{(t)}$, eigenpairs $(^{(t)}_i; v_i^{(t)})_{i=1}^{N}$ of $^{t;t+1}$ and gradient updates $^{(t)}$, let there be k N chaotic eigenpairs, i.e. $_1 \quad ::: \quad _k > 0$. Suppose all chaotic components are removed through projection, leading to a modi ed update vector

$$\tilde{}^{(t)} = V^{(t)}(V^{(t)})^T \quad ^{(t)}; \qquad V^{(t)} = \sum_{l>k} v_l^{(t)} v_l^{(t)T}: \tag{11}$$

Then, for training with SGD without momentum, the updated training dynamics are given by

$$^{(t+1)} = \quad ^{(t)} \quad V^{(t)} V^{(t)T} g(\ ^{(t)}); \qquad Y^{(t+1)} = (I \quad V^{(t)} V^{(t)T}) Y^{(t)} \tag{12}$$

and for suf ciently small variations of the initial parameters, the system is guaranteed not to have exponentially diverging orbits.

Proof is provided in Appendix A.8.

3 Related Work

Since the full Lyapunov eigenspectrum is very expensive to calculate, previous work analyzing chaotic dynamics has so far mostly been limited either to small networks or to the largest mLCE, as investigated for instance by Das et al. (2000). Several investigations of chaos in feedforward ANNs such as those by Li (2019); Feng et al. (2019); Zhang et al. (2021) have focused on sensitivity of outputs w.r.t. the inputs rather than w.r.t. the training dynamics, modelling the network function itself as a discrete dynamical system. Recently, Vogt et al. (2020) have calculated approximations of the Lyapunov exponents in RNNs and found correlations between better generalization (through lower validation loss) and a smaller mLCE. To our knowledge, Sasdelli et al. (2021) are the only ones to have investigated chaos dynamics in ANN training by SGD, but interestingly, they model ANN training by SGD as an approximately time-continuous dynamical system and not more accurately as a time-discrete dynamical system, thus arriving at slightly different theoretical results. While their theoretical work also links the biggest Lyapunov exponent to the most negative eigenpair of the network Hessian, positive eigenpairs as indicators of chaos disappear in their considerations, which is not the case for time-discrete SGD without further assumptions. Unfortunately, they do not investigate how the directions of chaotic evolution in general and negative curvature in particular affect the training process.

There is a considerable body of work on the eigenvalue spectra of ANN Hessians that can be tied into our theoretical ndings, including an analysis of the full eigenvalue spectrum of a small 784-2-10 MLP for MNIST digit classi cation by Sagun et al. (2017). Among other ndings, the authors remark a two-phase distribution of eigenvalues with a bulk centered around zero and large positive top eigenvalues (although not suf ciently big to satisfy $\lambda_2 = \ )$, as well as the presence of negative eigenvalues throughout the entire training Further analysis of the negative spectrum by Alain et al. (2019) has indicated that individual negative eigenvalues on average lead to the biggest improvement of the loss in neural networks, but that current optimization algorithms appear not to be good at optimizing in directions of negative curvature. This nding was extended by Gur-Ari et al. (2018), who found that the gradient of the Hessian mostly resides in the space spanned by the top Hessian eigenvectors and does not mix with the bulk, suggesting a marginal importance of the negative Hessian eigenpairs for the training dynamics.

To the best of our knowledge, we are the rst to use time-discrete modelling of training dynamics by SGD to derive predictions of how curvature affects chaotic dynamics, and the rst to explore

extensively - using the full Hessian spectrum at every training step - how directions of different curvature affect the dynamics of the training process. This extensive investigation provides us with empirical evidence allowing us to establish a novel connection between chaotic training dynamics and the training performance.

4 Experiments

Taking into account all aspects discussed so far, we can make the following claims:

1. Due to the presence of negative eigenvalues in all training steps and assuming the MLP for MNIST is a representative model for Feedforward NNs, the LCE of ANNs can be expected to be positive according to Theorem 2.2.
2. Unless the positive top eigenvalues of the Hessian are suf ciently big to satisfy $2 =$ - which was not the case for the MLP for MNIST investigated by Sagun et al. (2017) - the chaotic behaviour of ANN training by SGD is exclusively determined by negative eigenvalues in the bulk.

What remains to be veri ed experimentally is whether the negative eigenspectrum has little or at worst a detrimental effect on the training performance, as suggested by the ndings of Alain et al. (2019) and Gur-Ari et al. (2018), or if it is actually beni cial to the performance of ANN training.

Methods The code for all our experiments is written in Python using the Pytorch framework (Paszke et al., 2017). For the calculation of the Hessians, we use O2Grad (Anonymous, 2022), a package on top of Pytorch that enables faster calculation of the Hessian of small ANNs using 2nd order backpropagation. As datasets, we use USPS (Hull, 1994) and FashionMNIST (Xiao et al., 2017) (with images subsampled to 1616 pixels), since these datasets contain suf ciently low-dimensional, natural data to allow for the calculation of the Hessian and its eigenvalue decomposition at every training step, both for a 784-20-10 MLP and for a small 2D CNN. To modify the parameter updates of the model as proposed in Theorem 2.2, we use the Pytorch implementation of SGD and alter it slightly to implement an own class CGD (Chaos-sensitive Gradient Descent) with the ability to lter parts of the eigenvalue spectrum.

Our experiments were run on single GPU nodes of a cluster with a AMD Ryzen Threadripper 1950X processor, 4x Nvidia GeForce RTX 2080 TI GPUs (Driver version 460.91.03) with 11GB VRAM, and 64GB RAM. The system runs on Debian 5.10.46-4 (2021-08-03) x86_64.

4.1 Local Chaos Investigation

Chaotic direction pruning deteriorates training performance In order to verify the impact of pruning different components on the training performance, we equipped CGD with the ability to lter (1.) chaotic eigenpairs, (2.) only negative eigenpairs and (3.) only positive eigenpairs, and out of those, lter only the k largest by absolute value or prune k at random. As our results for the MLP on USPS in Figure 1 show, models trained without pruning (i.e. regular SGD) reach high/low training accuracies/losses, while the models trained with CGD and pruning of chaotic eigenvalues get stuck at much lower/higher training accuracies/losses. The same phenomenon can be observed for validation losses/accuracies, and on the 2D CNN and FashionMNIST (see Appendix B.1). This is remarkable because it seems to suggest that locally chaotic training behaviour is essential for ANN training in order to quickly achieve reasonable training performance and generalization, and that ANNs trained by SGD are hardly able to learn in the absence of locally chaotic directions. To make sure this is not an effect occurring only in the early stages of training by SGD, we execute several runs where the pruning of locally chaotic directions starts later in the training. As we can see in Figure 2, training loss and accuracy start improving at a slower rate after the onset of pruning, suggesting that the models stop learning as soon as the pruning of chaotic directions is engaged.

Negative Hessian eigenspectrum drives training by SGD Interestingly, the number of positive chaotic eigenvalues of the Hessian is effectively 0 throughout most of the training (see Appendix B.2). This means that the chaotic directions pruned during the training come almost exclusively from the negative eigenpairs of the Hessian, which is in line with our previously formulated expectation. However, the extent to which pruning of the negative eigenvalues hinders the training of the network

(a) (b)

Figure 1: Line plots of the (a) accuracy and (b) loss curves for an MLP (relu activation) trained on the USPS dataset without momentum. The models trained with CGD use pruning of the full chaotic, negative or positive Hessian spectrum at every time step. The smoothed, solid lines are obtained through local averaging (window size 50) of the respective sequential data. Pruning of the full positive spectrum (red) only has a small effect on training performance, while negative and chaotic pruning strongly limits the network's ability to learn.

(a) (b)

Figure 2: Line plots of the (a) accuracy and (b) loss curves for an MLP trained on the USPS dataset without momentum. We use a sigmoid activation because this slows down convergence and makes the splitting of performance metrics better visible. The models trained with CGD use pruning of the full chaotic Hessian spectrum at every time step after the onset of pruning after 2k, 5k and 10k time steps. The smoothed, solid lines are obtained through local averaging (window size 50) of the respective sequential data). The accuracy and loss curves start diverging from the baseline curve when pruning starts.

suggests that the negative eigenspectrum of the Hessian is far more important to the training dynamics of neural networks than previously believed by Alain et al. (2019). To better understand the role of the negative and positive eigenspectrum of the Hessian, we perform several runs where we prune (1.) the entire negative eigenspectrum and (2.) the entire positive eigenspectrum of the Hessian.

As expected, our results in Figure 1 for pruning the full negative eigenspectrum coincide with the results for pruning the full chaotic eigenspectrum of the Hessian: Models trained with pruning of the full negative eigenspectrum of the Hessian reach far lower training performance than models

trained with regular SGD. However, pruning the full positive eigenspectrum of the Hessian only has a small impact on the training performance. Although the accuracy/loss reaches slightly lower/higher levels than for SGD without pruning, the ANNs trained in this way fare much better than their counterparts with negative pruning. We observe this behaviour across different datasets and models (see Appendix B.1).

These empirical results lead us to the following conclusions:

1. The negative eigenvalue spectrum of the Hessian contains important information about the optimization problem since it cannot be discarded without hurting training performance.
2. The positive eigenvalue spectrum of the Hessian contains less important information about the optimization since discarding it has less impact on the training.
3. Since the negative eigenspectrum induces locally chaotic training dynamics, locally chaotic training dynamics are inevitable in SGD without hurting training performance.

These appear to contradict the findings of Gur-Ari et al. (2018), who observe that the overlap of the gradient is biggest with the positive eigenpairs of the Hessian - an observation that we confirm in our experiments (see Appendix B.2). The apparent contradiction can be resolved through a simplified model of ANN training by SGD, in which negative curvature precedes positive curvature: Directions of negative curvature indicate high potential for future loss improvement from moving along those axes, while directions of positive curvature indicate proximity to a local minimum along said axes, and thus reduced potential for loss improvement (see Appendix B.3.1). Removing the negative components of the Hessian from the gradient eliminates possibilities for big future improvements of the loss and may only allow the optimizer to finish converging in the directions where it was moving anyway, hence why the network is still able to decrease its loss even when the negative eigenspectrum is pruned, though at a smaller pace. This also matches our observation that the count of negative/positive Hessian eigenvalues decreases/increases as the training proceeds (see Appendix B.2).

Note that this model does not explain why the gradient overlap with axes of positive curvature is greater than with axes of negative curvature, only why the former and a high importance of the negative eigenvalues can exist at the same time: Negative curvature is important during an initial discovery phase, and positive curvature during a subsequent exploitation phase.

4.2 Global Chaos Investigation

Our experiments so far have relied on a local quantification of chaos via the LLEs. Next, we extend our investigations to global chaos using finite-time Lyapunov exponents to quantify chaos over longer time spans.

Finite-time Lyapunov exponents In order to better understand long-term chaotic dynamics, we calculate the tangent maps from equation (8) at every time step t and apply eigenvalue decomposition on the resulting finite-time Lyapunov matrix $^{(t)}$ to end up with the corresponding finite-time Lyapunov eigenspectrum. The Lyapunov exponents are above 0 at the beginning of the training and converge against 0 as training progresses (see C.1), suggesting the evolution is initially chaotic, but edge-chaotic in the time limits. Note that since the exponents do not drop below 0, we should expect models with similar initializations to continue diverging even after they reach top performance.

Perturbation analysis The divergence behavior can be further explored by utilizing the Lyapunov eigenspace to perturb models along determined directions. As discussed in section 2.2, this space describes the global divergence directions locally around an initial set of parameters. In our perturbation experiments, we train a model A until convergence at time T_{conv} and calculate the Lyapunov eigenpairs. We then use those eigenpairs to apply initial perturbations onto several models $B_1, \ldots, B_n$ initialized with the same random seed as A, and observe how their parameters evolve compared to A along the perturbation axes. The latter are derived from the finite-time subspaces

$$V_{chaotic}(t) := \{ v \in \mathrm{Eig}(\ ^{(t)};\ ) \mid \ > 0 \}; \qquad V_{non\text{-}chaotic}(t) := \{ v \in \mathrm{Eig}(\ ^{(t)};\ ) \mid \ < 0 \}$$

according to 5 perturbation strategies: We take (1) the top eigenvector of $^{(N)}$ as the maximally chaotic direction and (2) the bottom eigenvector as maximally non-chaotic direction. Furthermore we consider a linear combination of vectors sampled randomly from $V_{chaotic}(N)$ and (4)

(a) Empirical perturbation evolution (b) Theoretical modelling

Figure 3: Euclidean distance between a baseline MLP and a perturbed MLP trained with SGD (without momentum). We use 5 different strategies for computing initial perturbations of norm $\varepsilon = 0.1$ using the Lyapunov eigenspace of a model trained beyond convergence of training accuracy. Darker lines represent perturbation directions which we expect to lead to higher model divergence. The observed divergence in (a) on USPS (green) and on FashionMNIST (blue) increases even after convergence of the training accuracy across all perturbation directions. As shown in (b) for the USPS curves, the divergence can be approximated through a function $\sqrt{\alpha^2 + \beta t}$ (red) within the observed time range. The choice of α for the trajectories is in agreement with the theoretical bounds (purple) predicted by the maximum/minimum LCE $\lambda_1(t)$/ $\lambda_N(t)$ calculated at every time step.

$V_{\text{non-chaotic}}(N)$ that describe random chaotic and random non-chaotic directions, respectively. Finally, we also sample (5) a completely random direction for comparison. The initial parameters are perturbed along the respective directions, where we test different magnitudes $\varepsilon_1, \ldots, \varepsilon_k$ for each direction to make sure we consider perturbation magnitudes that are in agreement with the locality required by our theoretical preliminaries (see equation (6)).

Ghorbani et al. (2019) finds that the loss landscape near a local minimum is almost flat at (0) and thus using the finite-time Lyapunov matrix at a time point $t = T_{\text{conv}}$ where the training is sufficiently converged should provide an acceptable approximation. Therefore, the tangent map $Y^{(t)}$ at the next time step $t > T_{\text{conv}} + 1$ and its eigenspace would barely change.

$$Y^{(T_{\text{conv}}+1)} = (I - H^{(T_{\text{conv}})})Y^{(T_{\text{conv}})} - Y^{(T_{\text{conv}})} \tag{13}$$

Although we also see a similar distribution of the Hessian eigenspectrum, we do not observe a full convergence of the Hessian to zero (see Appendix B.2). However, the subspaces of chaotic and non-chaotic eigenvectors do stabilise towards the end of the training (see Appendix C.2), making a finite-time analysis with the above perturbation strategies sound.

Distance results Figure 3 (a) shows the distance evolution of an MLP without momentum on USPS and FashionMNIST. For a comparison between model architectures and analysis of momentum, see Appendix C.3. In general, the maximal divergence (empirically 1 in Figure 3 (a), up to 12 in further experiments in Appendix C.3) lies in range of the expected distance of standard initialized models in Pytorch (LeCun et al., 2012) for the duration of our experiments:

$$E_{\mathcal{U}(-\sqrt{\frac{1}{k_i}}, \sqrt{\frac{1}{k_i}})}(\|\omega_1 - \omega_2\|_2) = 6.38,$$

where k_i are the number of input features of layer i (see Appendix A.9). Still, a surprising result is that the distances do not seem to saturate for the duration of our experiments, which would be expected for a valley of finite size. In fact, we are still reaching a similar performance with perturbations of magnitudes up to $\varepsilon = 7.5$, and only for $\varepsilon = 10$ there seems to be a significant drop in performance (see Appendix C.3).

Theoretically, the observed distance evolution can be explained as composition of linear divergence and a random walk. Suppose the gradients $g_1(t); g_2(t)$ have distance components $w_1(t); w_2(t)$ with constant distance $:= k w_1(t) \quad w_2(t) k_2^2$

$$g_i(t) \quad w_i(t) + \quad (t) \tag{14}$$

and noise components de ned by $_1(t); \quad _2(t) \; 2 \; N \; (0; \quad ^2)$. Then the distance (t) between two sets of parameters $_1^{(t)}$ and $_2^{(t)}$ evolves as follows:

$$(t) \quad E \; h \; P_{s=0}^{t} (w_1(s) \quad w_2(s)) + P_{s=0}^{t} (\; _1(s) \quad _2(s)) \; _2^{i} = p \; \overline{t^2 + \; t;} \tag{15}$$

with $:= 2 D \; ^2$ (see Appendix A.11). For smaller t, this term is still in uenced by the random walk term t whereas for larger t the t^2 term dominates and leads to linear divergence. We give examples of tting this baseline to the data in Figure 3(b).

Unfortunately, we were unable to explore whether the model distances saturate for a suf ciently long time evolution, which should be the case if the models end up in a bounded loss valley, whereas an unbounded loss valley would allow for model divergence to be unbounded as well. A more conclusive analysis on the matter will have to be provided by further studies. In any case, the evolution of training dynamics from chaotic to edge-chaotic in the parameter domain is interesting given that ANNs transition from unchaotic to chaotic in the input domain (Feng et al., 2019), suggesting there may be a complementary relation between the two.

Choice of perturbation length We present the experiments with a perturbation length of $\text{"}=0:1$ as we do not nd that the approximation holds for larger ". Surprisingly, we see a similar behavior for smaller perturbation lengths $\text{"}<0:1$: the smaller the differences get, the more randomly they behave. We argue that this phenomenon is caused by noise, probably induced by a numerical error. For further analysis see Appendix C.4.

Theoretical bounds In Figure 3 (b), we present the theoretical bounds derived from the maximal and minimal LCE (see equation (5)). Both bounds hold for our choice of $\text{"}=0:1$ over the entirety of our experiments where the lower bound converges always to zero (see Appendix C.5). The upper bound, on the other hand, can be utilized as a measure of maximal divergence of a single training run without conducting the full scale of our experiment. Note that for this purpose, it suf ces to calculate a Lyapunov-vector-product which can be realized through a Hessian-vector-product for which faster algorithms exist (Pearlmutter, 1994).

5 Discussion

Although we have done our best to perform the pruning experiments on common datasets and model architectures, the experiments are currently dif cult to extend to higher-dimensional datasets and models due to the high expense of calculating the Hessian at every training step. For similar reasons, we have not studied the effect of BatchNorm (Ioffe and Szegedy, 2015) and skip connections (He et al., 2015) yet, although we intend to do this at a later time. The high computational cost also limits training time used in our analysis and it could be that some aspects don't apply for longer training times, e.g. our model for perturbed network divergence (see 3 (b)). In this context, it would also be interesting to investigate possible connections between the edge-chaotic dynamics later in training and over tting, but we mainly limited the scope of our analysis to training metrics. Another aspect requiring further inquiry is why the model divergence is not always ordered according to our expectations for different perturbation strategies.

In this paper, we have shown that chaotic dynamics of ANN training by SGD are linked to negative curvature. We have found strong evidence that using negative curvature information is essential for training, and we have concluded that chaotic behaviour is inherent to training by SGD. Although globally chaotic dynamics are mostly present at the beginning of the training behaviour and fade towards the end of training, we have shown that models with slightly different initializations continue to diverge linearly in parameter space and provided a model for this behavior. Our work provides a theoretical starting point and tools for further investigations aiming to advance the understanding of chaos in ANN training. In this context, we especially recommend investigating the divergence of trajectories at the end of the training and exploring possible connections to generalization capability (see A.1), as well as the relation between chaos in the input and in the parameter domain.

References

Alain, G., Roux, N. L., and Manzagol, P.-A. (2019). Negative eigenvalues of the hessian in deep neural networks.

Anonymous (2022). O2grad - a pytorch extension for 2nd order backpropagation. Draft attached in the Supplementary Material.

Das, A., Roy, A., and Das, P. (2000). Chaos in a three dimensional neural network. *Applied Mathematical Modelling*, 24(7):511–522.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. 12(null):2121–2159.

Feng, L., Zhang, L., and Lai, C. H. (2019). Optimal machine intelligence at the edge of chaos.

Ghorbani, B., Krishnan, S., and Xiao, Y. (2019). An investigation into neural net optimization via hessian eigenvalue density. *CoRR*, abs/1901.10159.

Gur-Ari, G., Roberts, D. A., and Dyer, E. (2018). Gradient descent happens in a tiny subspace.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg.

Li, H. (2019). Analysis on the nonlinear dynamics of deep neural networks: Topological entropy and chaos.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Pearlmutter, B. A. (1994). Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–160.

Robbins, H. E. (2007). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Sagun, L., Bottou, L., and LeCun, Y. (2017). Eigenvalues of the hessian in deep learning: Singularity and beyond.

Sasdelli, M., Ajanthan, T., Chin, T.-J., and Carneiro, G. (2021). A chaos theory approach to understand neural network optimization. In *2021 Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–10.

Skokos, C. (2009). The lyapunov characteristic exponents and their computation. *Lecture Notes in Physics*, page 63–135.

Vogt, R., Touzel, M. P., Shlizerman, E., and Lajoie, G. (2020). On lyapunov exponents for rnns: Understanding information propagation using dynamical systems tools. *CoRR*, abs/2006.14123.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Zhang, L., Feng, L., Chen, K., and Lai, C. H. (2021). Edge of chaos as a guiding principle for modern neural network training.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default [TODO] to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a justification to your answer , either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section 22
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately re ect the paper's contributions and scope? [Yes] Where to nd mentioned contributions of the abstract in the paper:
      - The paper establishes a theoretical connection between curvature and locally chaotic dynamics
        We show this in Theorem 2.1 in section 2.4
      - Locally chaotic dynamics are mainly determined by the negative eigenvalue spectrum of the Hessian:
        Applying Theorem 2.1 to the results of logging the maximal Hessian eigenvalue $_1$ (see Appendix B.2) veri es that local chaos is rarely induced by the top eigenspectrum of the Hessian as $_1 < 2=$ for most of the training.
      - The directions of local chaos cannot be removed from SGD without hurting the training performance:
        See subsection 4.1 and Figure 1.
      - We challenge the widespread understanding of convergence to a con ned region in parameter space and show models diverge at a slow pace even after reaching top training performance:
        See subsection 4.2 and Figure 3.
      - Their divergence can be modelled through a composition of a random walk and a linear divergence
        See equation 15 and Figure 3.
      - The tools and insights developed as part of our work contribute to improving the understanding of neural network training dynamics and provide a basis for future improvements of optimization:
        See subsection 4.2 § Theoretical distance evolution as well as section 5.
   (b) Did you describe the limitations of your work? [Yes] See discussion section 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] No, the scope of this paper is to improve understanding of SGD and we do not expect any immediate negative societal impacts from this.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs can be found in the proof section in Appendix A except the equations from the preliminaries section 2 which are taken from (Skokos, 2009).
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] Not yet, these will be included in the nal submission including supplementary material by 26th May.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Aside from logging speci cs, all training details required to reproduce our experiments given the code base are speci ed. More detailed explanations for reproducing the experiments will be provided with the code in the supplementary material.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] No, our work does not aim to beat the performance of existing models on certain tasks but merely to improve the understanding of how SGD works. Errors bars would not be purposeful here.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes, since the computations we run involving O2Grad require GPUs with state-of-the-art VRAM size and can thus only be reproduced by people with access to comparably poweful systems.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cite the creators of the main assets used, i.e. Pytorch, USPS and FashionMNIST.

(b) Did you mention the license of the assets? [No] No, the datasets (USPS and FashionMNIST) are released under no license and under an open source license (MIT), respectively, and commonly used in the ML community. As such, mentioning the license in the paper is super uous. We declare the Python packages used for our experiments as part of the code in the supplementary material.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include a self-developed asset (O2Grad) in the supplementary material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] No, the datasets used are publicly available under open source licenses, explicit consent by the creators is not necessary.

(e) Did you discuss whether the data you are using/curating contains personally identi able information or offensive content? [No] No, neither should apply to the USPS or to the FashionMNIST dataset.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proofs

Tangent map of SGD

Theorem A.1. Consider a neural network with N parameters $\theta^{(t)} \in \mathbb{R}^N$ at time step t and training with SGD according to

$$\theta^{(t+1)} = f(\theta^{(t)}) = \theta^{(t)} - \eta\, g(\theta^{(t)}; z^{(t)}); \tag{16}$$

where η is the learning rate and $z^{(t)}$ is a data batch sampled at time step t. Then, the tangent map of the corresponding dynamical system can be interatively calculated as

$$Y^{(t+1)} = (I - \eta H^{(t)}) Y^{(t)}. \tag{17}$$

Proof. The tangent map of a dynamical system at time step which only depends on the previous time step can be obtained as

$$Y^{(t+1)} = J_f(\theta^{(t)}) Y^{(t)} = \frac{\partial \theta^{(t+1)}}{\partial \theta^{(t)}} Y^{(t)}; \tag{18}$$

where $J_f(\theta^{(t)})$ is the Jacobian of the evolution function f of the dynamical system. In this case, this is simply

$$\frac{\partial \theta_i^{(t+1)}}{\partial \theta_j^{(t)}} = \frac{\partial \theta_i^{(t)}}{\partial \theta_j^{(t)}} - \eta \frac{\partial}{\partial \theta_j^{(t)}} g_i(\theta^{(t)}) = \delta_{ij} - \eta H_{ij}(\theta^{(t)}); \tag{19}$$

where δ_{ij} is the Kronecker delta and H denotes the Hessian of the loss w.r.t. network parameters and we are done. □

Tangent map of SGD with momentum

Theorem A.2. Consider a neural network evolving according to SGD with learning rate η and momentum α as

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t)}; \qquad v^{(t)} = \alpha\, g(\theta^{(t)}; z^{(t)}) + \alpha\, v^{(t-1)}; \tag{20}$$

The parameters at time step $t+1$ depend on the gradients at all previous time steps and the dependency can be written as

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t)}; \qquad v^{(t)} = \sum_{s=0}^{t} \alpha^{(t-s)} g(\theta^{(s)}; z^{(s)}). \tag{21}$$

Proof. For the sake of convenience, we omit writing down the dependency on the batch $z^{(t)}$. Induction start for $t = 0$, where we assume $v^{(0)} = 0$:

$$\theta^{(1)} = \theta^{(0)} - \eta\, g(\theta^{(0)}) = \theta^{(0)} - \eta\, \alpha^0 g(\theta^{(0)}) \tag{22}$$

Induction step assuming equation holds true for time step t

$$\begin{aligned}
\theta^{(t+1)} &= \theta^{(t)} + v^{(t)} && (23)\\
&= \theta^{(t)} - \eta\, g(\theta^{(t)}) + \alpha\, v(\theta^{(t-1)}) && (24)\\
&= \theta^{(t)} - \eta\, g(\theta^{(t)}) + (\alpha - \eta) \sum_{s=0}^{t-1} \alpha^{(t-1-s)} g(\theta^{(s)}) && (25)\\
&= \theta^{(t)} - \eta\, g(\theta^{(t)}) - \eta \sum_{s=0}^{t-1} \alpha^{(t-s)} g(\theta^{(s)}) && (26)\\
&= \theta^{(t)} - \eta \sum_{s=0}^{t} \alpha^{t-s} g(\theta^{(s)}) && (27)
\end{aligned}$$

□

Theorem A.3. Consider a neural network evolving according to SGD with learning rate η and momentum μ as

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t)}; \qquad v^{(t)} = \mu g(\theta^{(t)}; z^{(t)}) + \mu v^{(t-1)}; \tag{28}$$

where g is the gradient of the loss w.r.t. parameters $\theta^{(t)}$ at time point t. Then, the tangent map $Y^{(t+1)}$ of the dynamical system is given by

$$Y^{(t+1)} = Y^{(t)} \left(\sum_{s=1}^{t} \mu^{t-s} H^{(s)} Y^{(s)} \right). \tag{29}$$

Proof. At time step t, the update rule for the gradient depends on all previous time steps according to Theorem A.2. Thus, the tangent map can be obtained through application of the chain rule:

$$Y^{(t+1)} = \frac{d\theta^{(t+1)}}{d\theta^{(0)}} = \sum_{s=0}^{t} \frac{\partial \theta^{(t+1)}}{\partial \theta^{(s)}} \frac{d\theta^{(s)}}{d\theta^{(0)}} = \sum_{s=0}^{t} \frac{\partial \theta^{(t+1)}}{\partial \theta^{(s)}} Y^{(s)}. \tag{30}$$

Using Theorem A.2 again, the partial derivatives can be resolved as

$$\frac{\partial \theta^{(t+1)}}{\partial \theta^{(s)}} = \begin{cases} I - \eta \nabla_{\theta^{(t)}} g(\theta^{(t)}) = I - \eta H^{(t)}; & s = t \\ \mu^{t-s} \nabla_{\theta^{(s)}} g(\theta^{(s)}) = \mu^{t-s} H^{(s)}; & s < t \end{cases} \tag{31}$$

And reinserting into equation (30), we obtain the desired result

$$Y^{(t+1)} = (I - \eta H^{(t)}) Y^{(t)} \left(\sum_{s=0}^{t-1} \mu^{t-s} H^{(s)} Y^{(s)} \right) = Y^{(t)} \left(\sum_{s=0}^{t} \mu^{t-s} H^{(s)} Y^{(s)} \right) \tag{32}$$

□

Hessian eigenvalues and Local Chaos

Theorem A.4. Let $A \in \mathbb{R}^{N \times N}$ be a positive definite matrix, and the natural logarithm of a matrix be defined as the infinite series

$$\ln A = \sum_{k=1}^{\infty} \frac{(-1)^{k}}{k} (A - I)^{k} \tag{33}$$

Then, if $(\lambda_i; v_i)_{i=1}^{N}$ is the set of all eigenpairs of A, $(\ln \lambda_i; v_i)_{i=1}^{N}$ is the set of all eigenpairs of $\ln A$.

Proof. Since A is positive definite, its eigenvalues λ_i are all positive, so $\ln \lambda_i$ is defined for all eigenvalues of A. Consider an arbitrary eigenvalue of A satisfying $Av_i = \lambda_i v_i$. Clearly, $(A - I)v_i = Av_i - Iv_i = \lambda_i v_i - v_i = (\lambda_i - 1)v_i$. It follows immediately that $(A - I)^k v_i = (\lambda_i - 1)^k v_i$ and thus:

$$\ln A v_i = \sum_{k=1}^{\infty} \frac{(-1)^{k}}{k} (A - I)^{k} v_i = \sum_{k=1}^{\infty} \frac{(-1)^{k}}{k} (\lambda_i - 1)^{k} v_i = \ln \lambda_i v_i \tag{34}$$

Since $\ln A \in \mathbb{R}^{N \times N}$, the matrix can have at most N eigenvalues, so we know that these are all eigenvalues of $\ln A$ and we are done. □

Theorem A.5. Given a neural network with N parameters, let H be the Hessian of the loss w.r.t. network parameters, and let $(\lambda_i; v_i)_{i=1}^{N}$ be the Hessian's eigenpairs. Furthermore, let us assume that the network is trained using SGD with learning rate η. Consider the eigenvalues for which $\lambda_i \leq \frac{1}{\eta}$. Then, the network's LLEs indicate locally chaotic training behaviour in the direction v_i if

$$\lambda_i > 0 \wedge \lambda_i > \frac{2}{\eta} \tag{35}$$

Proof. If $(\ _i; v_i)$ is an eigenpair of H, $(1\ \ _i; v_i)$ is an eigenpair of $J_f = (I\ \ H)$, since $(I\ \ H)v = (1\ \ _i)v$. Due to the identity matrix and the Hessian being symmetric, $(I\ \ H)^T(I\ \ H) = (I\ \ H)^2$. Furthermore, if A is a positive de nite matrix and $(\ _i; v_i)$ is an eigenpair of A, then $(\ln\ _i; v_i)$ is an eigenpair of $\ln A$ (see Appendix A.4). If all eigenvalues of $(I\ \ H)^2$ are positive and nonzero due to $\ _i \ \ 6\ \ \frac{1}{}$, the matrix is positive de nite and it follows that $\frac{1}{2}\ln(1\ \ _i)^2$ is an eigenvalue of

$$\ _x^{(1)} = \frac{1}{2}\ln(I\ \ H)^T(I\ \ H)$$

for the same eigenvector v_i and thus an LLE. To indicate locally chaotic training behaviour, the LLE must satisfy

$$\ln(1\ \ _i)^2 \quad 0 \quad , \quad (1\ \ _i)^2 \quad 1 \tag{36}$$

Solving for $\ _i$ gives the desired result and we are done. □

Now, consider the edge case where some eigenpair $(\ _i, v_i)$ of the Hessian H satis es $\ _i = \frac{1}{}$ s.t. for an in nitesimally small displacement $v\ \ k\ v_i$

$$J_f\ v = (I\ \ H)\ v = (1\ \ 1 =)\ v = 0: \tag{37}$$

That is, the dynamical system exhibits a maximal contraction along the axis of v_i, meaning that the system does not evolve along this axis. Therefore, these eigenpairs can be excluded from our considerations on chaotic behaviour. The respective eigenvalues can still be accounted for in the de nition of the Local Lyapunov spectrum by either accepting eigenvalues of 1, or by adapting the de nition of the LLEs to be the singular values of

$$\tilde{U}\ \tilde{V} = U\ \ V = (I\ \ H)^2; \tag{38}$$

where the rst decomposition is the zero-truncated SVD. In this case, the singular pairs $(\ _i, u_i)$, where u_i is the singular vector corresponding to singular value $\ _i$, de ne Local Lyapunov Exponent (LLE) and corresponding Lyapunov vector, respectively. Note that since $(I\ \ H)^2$ is always positive-semide nite, the singular value decomposition is identical to the eigenvalue decomposition $(I\ \ H) = U\ D\ U^{\ 1}$ provided it exists. Therefore the Lyapunov spectrum can be constrained to exclude Hessian eigenpairs where $\ = \frac{1}{}$ by considering the zero-truncated SVD. This de nition is also interesting because it allows us to loosely interpret the top Lyapunov vectors as the directions of maximum trajectory variance if we performed a PCA on trajectory distances in the limit for an in nite number of perturbed trajectories sampled, with the perturbation magnitude and evolution time of the trajectories going to 0.

Theorem A.6. Given a neural network with N parameters, let $H^{(t)}$ be the Hessian of the loss w.r.t. network parameters at time step t. Furthermore, let $\ _1^{(s)}$ be the maximum LLE of the training dynamics at time step s for $0 < s \quad t$. Then, the mLCE of the system evolution can be calculated as

$$\ _1 = \lim_{t!1} \frac{1}{t} \sum_{s=1}^{t} \ _1^{(s)} \tag{39}$$

Proof. As previously established, the mLCE $\ _1$ is de ned as the maximum eigenvalue of $\ _1^{(1)}$, which is precisely the limit of the time-normalized spectral norm of the tangent map:

$$\ _1 = \ _{max} \lim_{t!1} \frac{1}{2t} \ln Y^{(t)T} Y^{(t)} = \lim_{t!1} \frac{1}{t}\ _{max}\ \overline{Y^{(t)T} Y^{(t)}} = \lim_{t!1} \frac{1}{t} k Y^{(t)} k \tag{40}$$

For SGD without momentum, this can be further rewritten using the chain rule and the rules for the logarithm:

$$\ _1 = \lim_{t!1} \frac{1}{t} \ln \sum_{s=1}^{t} \frac{@^{(s)}}{@^{(s\ \ 1)}} = \lim_{t!1} \frac{1}{t} \sum_{s=1}^{t} \ln \frac{@^{(s)}}{@^{(s\ \ 1)}} \tag{41}$$

Using the de nition of the spectral norm again, we realize the eigenvalues in the sum are the maximum LLEs and we end up with

$$_1 = \lim_{t!1} \frac{1}{t} \sum_{s=1}^{t} \ln \max \sqrt{J_f(\ ^{(s+1)})^T J_f(\ ^{(s+1)})} \tag{42}$$

$$= \lim_{t!1} \frac{1}{t} \sum_{s=1}^{t} \max \ln \sqrt{J_f(\ ^{(s+1)})^T J_f(\ ^{(s+1)})} = \lim_{t!1} \frac{1}{t} \sum_{s=0}^{t} \ ^{(s)}_1 : \tag{43}$$

□

Theorem A.7. Given a neural network with N parameters, let $H^{(t)}$ be the Hessian of the loss w.r.t. network parameters at time step t. If for all time steps t the Hessian $H^{(t)}$ has at least one negative eigenvalue, then the mLCE of the training satis es $_1 > 0$ and thus the global training dynamics are chaotic.

Proof. If at any time step s, the Hessian has a negative eigenvalue $^{(s)} < 0$, then the Oseledet matrix at that time step has an eigenvector $(1 \quad ^{(s)})^2 > 0$ (see Theorem A.5). Thus, the maximum LLE at time step s is lower bounded by $0 < \ ^{(s)} \quad ^{(s)}_1$. Using the result of Theorem A.6, we conclude

$$_1 = \lim_{t!1} \frac{1}{t} \sum_{s=1}^{t} \ ^{(s)}_1 \quad \lim_{t!1} \frac{1}{t} \sum_{s=1}^{t} 0 = 0; \tag{44}$$

□

Theorem A.8. Given a neural network with Hessian $H^{(t)}$, eigenpairs $(\ ^{(t)}_i; v^{(t)}_i)_{i=1}^{n}$ of $\ ^{t;t+1}$ and gradient updates $\ ^{(t)}$, let there be $k \quad n$ chaotic eigenpairs, i.e. $_1 \quad ::: \quad _k > 0$. Suppose all chaotic components are pruned through projection, leading to a modi ed update vector

$$\tilde{}^{(t)} = v^{(t)}(v^{(t)})^T \quad ^{(t)}; \qquad v^{(t)} = \sum_{l>k} v^{(t)}_l v^{(t)T}_l : \tag{45}$$

Then, for SGD training with learning rate $\ $ of $\ $ and without momentum, the updated training dynamics are given by

$$^{(t+1)} = \ ^{(t)} \quad v^{(t)} v^{(t)T} g(\ ^{(t)}); \qquad Y^{(t+1)} = (I \quad v^{(t)} v^{(t)T}) Y^{(t)} \tag{46}$$

and for suf ciently small variations of the initial parameters, the system is guaranteed not to have exponentially diverging orbits.

Proof. The updated training dynamics are given by

$$^{(t+1)} = \ ^{(t)} \quad \tilde{}^{(t)} = \ ^{(t)} \quad v^{(t)} v^{(t)T} \ ^{(t)} = \ ^{(t)} \quad v^{(t)} v^{(t)T} g(\ ^{(t)}; x^{(t)}; Y^{(t)}): \tag{47}$$

Thus, we can derivate w.r.t. $^{(t)}$ to obtain the dynamics Jacobian and get

$$\frac{@^{(t+1)}_i}{@^{(t)}_j} = \frac{@}{@^{(t)}_j} \left(\ ^{(t)}_i \quad \sum_k \left[v^{(t)} v^{(t)T} \right]_{ik} g^{(t)}_k(\ ^{(t)}; x^{(t)}; Y^{(t)}) \right) \tag{48}$$

$$= \frac{@^{(t)}_i}{@^{(t)}_j} \quad \sum_k \left[v^{(t)} v^{(t)T} \right]_{ik} \frac{@g^{(t)}_k(\ ^{(t)}; x^{(t)}; Y^{(t)})}{@^{(t)}_j} \tag{49}$$

$$= \ _{ij} \quad \sum_k \left[v^{(t)} v^{(t)T} \right]_{ik} H_{kj}; \tag{50}$$

or more succinctly in matrix notation

$$\frac{@^{(t+1)}}{@^{(t)}} = I \quad v^{(t)} v^{(t)(T)} H^{(t)}: \tag{51}$$

Now, we still need to show that the system is guaranteed not to have exponentially diverging orbits. If $v_i^{(t)}$ is a pruned Lyapunov vector (i.e. $i > k$), its eigenvalue for the modified dynamics Jacobian becomes 1.

$$(I \quad v^{(t)} v^{(t)T} H^{(t)}) v_i^{(t)} = v_i^{(t)} \quad v^{(t)} v^{(t)T} \; {}_i v_i^{(t)} \tag{52}$$

$$= v_i^{(t)} \quad {}_i \underbrace{v^{(t)} v^{(t)T} v_i}_{=0} \tag{53}$$

$$= v_i^{(t)} \tag{54}$$

It follows immediately that the eigenvalue of the local Oseledet matrix (the LLE) is $E_i(1) = 0$. Otherwise, if $v_i^{(t)}$ was a preserved Lyapunov vector, then it satisfies

$$v^{(t)} v^{(t)T} v_i^{(t)} = v_i^{(t)} \tag{55}$$

by design and thus its LLE is preserved. Summing up, the eigenspectrum of the modified dynamics Jacobian satisfies the eigenvalue equation

$$\frac{@^{(t+1)}}{@^{(t)}} v_i = \begin{cases} 1 v_i; & k \quad i \\ (1 \quad {}_i) v_i; & 0 \quad i < k \end{cases} \tag{56}$$

such that all eigenvalues translate to non-chaotic or edge-chaotic LLEs smaller than (or equal to) 0. Applying Theorem A.6 and using the fact that the LLEs, in particular the maximum LLE at any time step, are all smaller than 0 yields

$${}_1 = \lim_{t!1} \frac{1}{t} \sum_{s=0}^{t} {}_1^{(s)} \quad \lim_{t!1} \frac{1}{t} \sum_{s=0}^{t} 0 = 0: \tag{57}$$

Hence, the trajectories diverge at most polynomially and we are done. □

Distance Statements

Theorem A.9. Two standard initialized 784-20-10 MLPs in Pytorch have a expected L2-distance of 6:38.

Proof. We calculate the distance between single layers first and differentiate between weight and bias. Pytorch initializes weights w and biases b uniformly with

$$w_i; b_i \quad U\left(\quad \sqrt{\frac{1}{k}}; \sqrt{\frac{1}{k}}\right); \quad k := \#\text{in\_features} \quad \text{and } m := \#\text{out\_features}$$

1. case: weight

So let $w_i^{(1)}; w_i^{(2)} \quad U\left(\quad \sqrt{\frac{1}{k}}; \sqrt{\frac{1}{k}}\right)$ for all i.

$$\mathop{E}_{w_i^{(1)}; w_i^{(2)} \quad U\left(\quad \sqrt{\frac{1}{k}}; \sqrt{\frac{1}{k}}\right)} [k w^{(1)} \quad w^{(2)} k_2^2] = \mathop{E}_{w_i \quad U\left(\quad \sqrt{\frac{2}{k}}; \sqrt{\frac{2}{k}}\right)} [k w k_2^2]$$

$$= \sum_{i=1}^{km} \mathop{E}_{w_i \quad U\left(\quad \sqrt{\frac{2}{k}}; \sqrt{\frac{2}{k}}\right)} [w_i^2] = \sum_{i=1}^{km} \mathrm{Var}(w_i) = km \frac{4^2}{12k} = \frac{4m}{3}$$

Thus $E[k w^{(1)} \quad w^{(2)} k_2] = \sqrt{\frac{4m}{3}}$.

2. case: bias

Analogously for the bias where the dimension of the bias is not k but m, we end up with $E[k b^{(1)} \quad b^{(2)} k_2] = \sqrt{\frac{4m}{3k}}$.

We now calculate the expected distance over all layers of the two 784-20-10 MLPs $^{(1)}$, $^{(2)}$:

$$E[k \; {}^{(1)} \quad {}^{(2)} k_2] = \sqrt{\frac{4(m_1 + m_2)}{3} + \frac{4m_1}{k_1} + \frac{4m_2}{k_2}} = \sqrt{\frac{4(30)}{3} + \frac{4(20)}{3(784)} + \frac{4(10)}{3(20)}} \quad 6:38$$

□

Theorem A.10. Two random Gaussian walks $x_1(t); x_2(t) = x_i(t-1) + \epsilon_i(t)$ with $x_1(0); x_2(0) = 0$ and $\epsilon_1(t); \epsilon_2(t) \sim N(0; \sigma^2 I)$ have an expected distance of $\sqrt{2TD}$ until time step T.

Proof.

$$
\begin{aligned}
E[\|x_1(t) - x_2(t)\|_2] &= \mathop{E}_{\epsilon_1(t);\,\epsilon_2(t)\sim N(0;\,\sigma^2 I)} \left\| \sum_{t=1}^{T} \epsilon_1(t) - \epsilon_2(t) \right\|_2 \\
&\overset{(1)}{=} \mathop{E}_{\epsilon'(t)\sim N(0;\,\sigma^2 I)} \left\| \sum_{t=1}^{T} \epsilon'(t) \right\|_2 \overset{(2)}{=} \mathop{E}_{\epsilon'(t)\sim N(0;\,2T\sigma^2 I)} [\|\tilde{\epsilon}(t)\|_2] \\
&\overset{(3)}{=} \sqrt{2TD}.
\end{aligned}
\tag{58}
$$

Equation (1) is valid, because $N(0; \sigma^2 I) = \sigma N(0; \sigma^2 I)$ and (2), because variances of the sum two independent variables add up. For (3), observe that

$$
\mathop{E}_{x\sim N(0;\,\sigma^2 I)} [\|x\|_2^2] = \sum_{i=1}^{D} \mathop{E}_{x_i\sim N(0;\,\sigma^2)} [x_i^2] = \mathop{E}_{x'\sim N(0;\,D\sigma^2)} [x'^2] = D\sigma^2 \tag{59}
$$

□

Theorem A.11. Let the gradients of two models $\epsilon_1; \epsilon_2 \in R^D$ be defined as

$$
g_i(t) = w_i(t) + \epsilon_i
$$

where $\epsilon_1; \epsilon_2 \sim N(0; \sigma^2 I)$ are independent and identically distributed, isotropic noise terms and $\|\sum_{s=1}^{t} (w_1(s) - w_2(s))\|_2 = \alpha t$ diverges linearly over time. Assume both models are initialized with the same set of parameters $\theta_1^{(0)} = \theta_2^{(0)}$. Then distance of the two models evolves as

$$
\mathop{E}_{\epsilon_1;\,\epsilon_2\sim N(0;\,\sigma^2 I)} [\|\epsilon_1 - \epsilon_2\|_2] = \sqrt{\sigma^2 t^2 + \alpha t},
$$

where $\sigma = 2D\sigma^2$.

Proof.

$$
\mathop{E}_{\epsilon_1;\,\epsilon_2\sim N(0;\,\sigma^2 I)} \left\| \sum_{s=0}^{t} (w_1(s) - w_2(s)) + \sum_{s=0}^{t} (\epsilon_1(s) - \epsilon_2(s)) \right\|_2 \tag{60}
$$

We want to calculate the term (60) where $\sum_{s=0}^{t} (w_1(s) - w_2(s))$ defines the linear divergence term and $\sum_{s=0}^{t} (\epsilon_1(s) - \epsilon_2(s))$ is the random Gaussian walk as in equation (15). For simplification, assume that we take the expected value over $\epsilon_1(s); \epsilon_2(s) \sim N(0; \sigma^2 I)$ in all of the following expression

$$
\begin{aligned}
&E\left[\left\| \sum_{s=0}^{t} (w_1(s) - w_2(s)) + \sum_{s=0}^{t} (\epsilon_1(s) - \epsilon_2(s)) \right\|_2^2\right] \\
&= E\left[\left\| \sum_{s=0}^{t} (w_1(s) - w_2(s)) \right\|_2^2 + \left\| \sum_{s=0}^{t} (\epsilon_1(s) - \epsilon_2(s)) \right\|_2^2 \right. \\
&\qquad \left. + \sum_{s,s'=0}^{t} \langle (w_1(s') - w_2(s')), \epsilon_1(s) - \epsilon_2(s) \rangle \right] \\
&= \sigma^2 t^2 + \alpha t + E\left[\sum_{s,s'=0}^{t} \langle (w_1(s') - w_2(s')), \epsilon_1(s) - \epsilon_2(s) \rangle\right]
\end{aligned}
$$

where $\sigma := 2D\sigma^2$, which follows from Theorem A.10. Now this leaves to show that the last summand is zero. In fact, observe that for all $v \in R^D$

$$
\mathop{E}_{\epsilon' \sim N(0;\,2\sigma^2 I)} [\langle v, \epsilon' \rangle] = \sum_{i=1}^{D} v_i \mathop{E}_{\epsilon' \sim N(0;\,2\sigma^2 I)} [\epsilon'_i] = 0;
$$

which can be applied to $v = w_1(s) - w_2(s)$ and $\epsilon' = \epsilon_1(s) - \epsilon_2(s)$ for all s. □

Theorem A.12. The modified Subspace Similarity defined as

$$\text{overlap}(V;W) = \sqrt{\frac{\text{Tr}(P_V;P_W)}{\text{Tr}(P_V)\text{Tr}(P_W)}} \sqrt{\frac{d_{max}}{d_{min}}} = \frac{1}{d_{min}}\|\text{atten}(V^TW)\|_2^2$$

where P_V, P_W are the orthogonal profectors onto the subspaces V, W fulfills the following properties:

1. If $V \perp W$, then $\text{overlap}(V;W) = 0$
2. If $V \subseteq W$ or $W \subseteq V$, then $\text{overlap}(V;W) = 1$
3. For arbitrary subspaces $V;W$, $\text{overlap}(V;W) \in [0;1]$

Proof. We first show the equality of the definition. The orthogonal projector is defined as $P_V := VV^T$ using V as a matrix of orthonormal basis vectors (in the columns) and as the subspace simplify the following equations. Now the numerator equals

$$\text{Tr}(P_V;P_W) = \text{Tr}(VV^TWW^T) = \text{Tr}((W^TV)(V^TW)) = \|\text{atten}(V^TW)\|_2^2$$

where for * we used the cyclic property of the trace. We utilize this property again to calculate

$$\text{Tr}(P_V) = \text{Tr}(VV^T) = \text{Tr}(V^TV) = \text{Tr}(\text{Id}_V) = \dim(V)$$

The rest of the equation follows by applying the two results.

Now statement 1. follows directly from the fact that if $V \perp W$ then $V^TW = 0$. For statement 2. observe that if $W \subseteq V$ then the orthogonal projection of W onto V is the identity: $P_V \downarrow_W = \text{Id}_W$. So the similarity is

$$\text{overlap}(V;W) = \frac{1}{\dim(W)}\text{Tr}(VV^TWW^T) = \frac{\text{Tr}(WW^T)}{\dim(W)} = \frac{\text{Tr}(W^TW)}{\dim(W)} = 1$$

Finally for statement 3., we see that $\|\text{atten}(V^TW)\|_2^2 \geq 0$ is clearly positive and what is left to prove is that it is bounded by 1. Let $n := \dim(V)$, $m := \dim(W)$ and $v_1;\dots;v_n$ and $w_1;\dots;w_m$ be an orthonormal basis of V, W, respectively. Assume with out loss of generality $m \leq n$

$$\|\text{atten}(V^TW)\|_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{m} |\langle v_i;w_j\rangle|^2$$

Now extend the orthonormal basis V of to a orthonormal basis $B := v_1;\dots;v_n;\dots;v_k$ of the whole space $V \supseteq V;W$. And write the orthonormal basis of W in terms of B: $w_j = \sum_{i=1}^{k} \alpha_{ji} v_i$. Then

$$\sum_{i=1}^{n}\sum_{j=1}^{m} |\langle v_i;w_j\rangle|^2 = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_{ji}^2 |\langle v_i;v_i\rangle|^2 = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_{ji}^2$$

Now $\sum_{i=1}^{n} \alpha_{ji}^2 \leq 1$, since

$$1 = \|w_j\|_2^2 = \left\langle \sum_{i=1}^{k} \alpha_{ji} v_i; \sum_{i=1}^{k} \alpha_{ji} v_i \right\rangle = \sum_{i=1}^{k} \alpha_{ji}^2 \geq \sum_{i=1}^{n} \alpha_{ji}^2$$

And we apply this inequality to what we previously ended up with

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_{ji}^2 \leq \sum_{j=1}^{m} 1 = m$$

to arive at what we wanted to show: $\text{overlap}(V;W) = \frac{1}{m}\|\text{atten}(V^TW)\|_2^2 \leq \frac{m}{m} = 1$. □

A.1 Generalization and long-term chaos

In this subsection, we provide a mathematical argument for why we can assume chaotic dynamics (in the sense of sensitivity to initial conditions) to generally have an impact on generalization properties of ANNs trained with SGD. Consider an ANN trained with some procedure M to yield a parametrized function f^M by the end of the training, where $:= {}^{(T)}$ are the parameters at the end of the training. Let S be the set of all data points that could potentially be used to train a network for its respective task, and let S^{train} S be the entirety of data points used to actually train the network. Furthermore, let $L(f^M; x)$ be the loss from calculating the network outputs, given a sample $x \in S$. Let us assume the input data to be structured s.t. $8x \in S \; 9 > 0$ s.t. $8y \in B(s) : y \in S$. A training procedure that leads to good generalization should be one that minimizes the variance of the loss of the learned function with respect to data points $x \in S$. Thus, the generalization capacity can be metricized by

$$\mathrm{Var}[L(f^M)] = E_{x \in S}\left[L(f^M; x) - E_{x \in S}\left[L(f^M; x)\right]\right]^2 = E_{x \in S}\left[L(f^M; x) - L\right]^2 \tag{61}$$

In particular, the generalization properties should be good within any finite region S. Assuming $L(f^M; x)$ is a piecewise-smooth function with respect to x and assuming x is not a border point at a derivative discontinuity, the loss function may be expressed using a Taylor polynomial:

$$L(f^M; x_s + \Delta x) = L(f^M; x_s) + \Delta x^T \nabla_L f^M + \frac{1}{2} \Delta x^T H_L(f^M; x_s)\, \Delta x + \cdots \tag{62}$$

Let us consider the ability of the function f to generalize on points in a vicinity S of a point x_s which was part of the training data, and where all other points $x \in S \cap \{x_s\}$ were not seen during the training procedure M. For the sake of simplicity, let us assume that x_s was seen exactly once during the training, namely at time step s. Then from the chain rule, it follows that

$$L^0(f^M; x_s) = \frac{dL}{dx}\Big|_{x = x_s} = \frac{dL}{d\,{}^{(T)}} \frac{d\,{}^{(T)}}{d\,{}^{(s+1)}} \frac{d\,{}^{(s+1)}}{dx_s} = \frac{dL}{d\,{}^{(T)}} Y^{(s+1;T)} \frac{d\,{}^{(s+1)}}{dx_s} \tag{63}$$

When the training procedure M is SGD (without momentum), the update rule gives us that

$$\frac{d\,{}^{(s+1)}}{dx_s} = \frac{d}{d\,{}^{(s)}}\left({}^{(s)} - \Delta g({}^{(s)}; x_s)\right) = I - \frac{\partial^2 L({}^{(s)}; x_s)}{\partial x_s \partial\,{}^{(s)}} = I - H_x^{(s)}; \tag{64}$$

where the tangent map $Y^{(s+1;T)}$ gives us the sensitivity of the parameters at the end of the training procedure to the parameters at a previous time point ${}^{(s+1)}$. Due to the product rule, the higher-order terms of the Taylor expansion will also contain some dependency on $Y^{(s+1;T)}$. However, for more clarity, we restrict ourselves to first-order terms to end up with the expression

$$\mathrm{Var}_S[L(f^{SGD})] = E_{x \in S}\left[L(f^{SGD}; x_s) + \frac{dL}{d\,{}^{(T)}} Y^{(s+1;T)} (I - H_x^{(s)})\, \Delta x - L\right]^2 \tag{65}$$

What this argument tells us is that, in order for the generalization capacity of a trained function f^{SGD} to be good at least for unseen samples within some vicinity of previously seen samples x_s, the training dynamics should not be too sensitive on initial conditions. The argument can be extended to vicinities of arbitrary sets of points seen during training by considering the expected values over respective regions.