Interpretation and Optimization of Recurrent Neural Network Performance Through Lyapunov Exponents Methodology

Ryan Vogt

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Eli Shlizerman, Chair

Eric Shea-Brown

Panagiotis Stinis

Program Authorized to Offer Degree:

Applied Mathematics

University of Washington

Abstract

Interpretation and Optimization of Recurrent Neural Network Performance Through Lyapunov Exponents Methodology

Ryan Vogt

Chair of the Supervisory Committee:
Eli Shlizerman
Department of Applied Mathematics, Department of Electrical and Computer Engineering

Common deep learning models for learning multivariate time series data are Recurrent Neural Networks (RNN). These models are ubiquitous computing systems which have been studied for decades. The propagation of gradients over long time-sequences can make training RNNs particularly challenging and difficult to interpret. The hidden states of RNNs can be viewed as non-autonomous dynamical systems which can be analyzed using dynamical systems tools. In this work, we leverage Lyapunov Exponents, a dynamical systems tool which measures the rate at which nearby trajectories expand or contract over time to analyze the propagation of information in RNNs and relate these properties to RNN training and performance. We show that several statistics of the Lyapunov spectrum have moderate correlation with network loss on both classification and regression tasks, and emerge early in training. We also train an autoencoder to learn the relation between the full Lyapunov spectrum and an RNN's loss on given tasks. The latent representation of the autoencoder distinguishes between high- and low-accuracy networks across a variety of network hyperparameters, including initialization parameter, network size, and network architecture more effectively than direct statistics of the Lyapunov spectrum. From a theoretical perspective to further analyze Lyapunov Exponents of RNNs, we derive a direct expression for gradient in terms of the components of RNNs' Lyapunov Exponents which measure directions (vectors Q) and factors (scalars R) of expansion and contraction over a sequence. We find that the Q vectors associated with the greatest degree of expansion become increasingly aligned with the dominant directions of the gradient extracted by singular value decomposition. Furthermore, we show that the predictions generated by RNN are maximally affected by input perturbations at moments which the R values are maximal. These results showcase correlation between dynamical systems stability theory for RNNs, network performance, and loss gradients. This may open the way to design hyperparameter optimization algorithms and adaptive training methods that account for state-space dynamics as measured by Lyapunov Exponents to improve computations. It may also provide a unifying dynamical systems framework to study RNN performance across network architectures and tasks.

Contents

Chapter 1

Introduction: Dynamical Systems and RNNs

Artificial Neural Networks (ANNs) revolutionized machine learning and provided novel and robust solutions to problems in classification and prediction. Such advancements contributed to scalable and efficient solutions to challenging applications, such as computer vision [1, 2], signal processing [3, 4], and medicine [5, 6]. However, these networks are not easy to explain and the objectives being optimized by them many times are not necessarily relying on robust semantic understanding but rather complex pattern matching. As a result, ANN systems are typically treated as black box models with little understanding of the underlying dynamics which determine their performance. The objective of this work is thereby to draw from methods in dynamical systems theory applied to complex systems such as fluid dynamics and turbulence and apply them to analysis, interpretation, and performance enhancement of ANNs. My proposed research is divided into three major aims: 1) Computation Lyapunov Exponents for Recurrent Neural Networks (RNNs) [7], 2) study of Lyapunov Spectrum characteristics for training and performance of RNNs [7, 8], and 3) analysis of Lyapunov Exponent components' direct relation to training gradients and input sensitivity. . Accomplishing these goals could help to lay a better theoretical foundation for RNN in particular and machine learning in general. It may also provide strategies for designing network architectures suitable for particular tasks and more robust optimization approaches.

Challenges in Training Recurrent Neural Networks

Recurrent neural networks (RNNs) specialize in classifying sequential data which evolves over different time scales. Each layer of the network can be viewed as a composing a function $\boldsymbol{f}$ at each time step of the input sequence. For RNNs to be effective in handling the different time scales, it is important to employ the concept of parameter sharing across inputs of different length. This allows the network to generalize across inputs which have similar features.

For a standard RNN (i.e., vanilla RNN), the function $\boldsymbol{f}$ defines the value of the hidden state h_t as a function of the previous hidden state, h_{t-1} and the input x_t:

$$\boldsymbol{h}_t = \sigma_h(\mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{W}\boldsymbol{x}_t + \mathbf{b}), \tag{1.1}$$

Since input data may exhibit these features in different orders or duration, the relationship between elements in a sequence must be learned by the network. Learning the dependence on the previous state in tandem with the initial input sequence allows RNNs to develop memory over extended periods of time. For long sequences, the memory required to learn meaningful relations across time steps can be quite long. Such requirements, however, impose several fundamental bottlenecks that hinder performance of RNNs. The first bottleneck is due to long-term propagation of gradients during training which typically results in diminishing or exploding gradients [9]. In addition, even if gradient propagation is stabilized, another bottleneck stems from recurrent networks involving composing the function $\boldsymbol{f}$ at each time step in a sequence. The long-term repetition of the composition may amplify small inaccuracies at individual time step to large contributions.

In order to have a network which can store memories in a robust way, it was shown that the eigenvalues of $\boldsymbol{f}$ must all decay close to zero [10]. These limitations on $\boldsymbol{f}$ can make general RNNs impractical and ineffective. Therefore, variations on the general RNN structure have been proposed to mitigate the extreme effects of these long-term dependencies.

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks remedy these problems by incorporating self-loops (gates) which allow the network to learn the amount that each component should preserve and the amount that it should forget. By implementing this structure, LSTMs are able to learn long-term dependencies. While such specific architectures are robust and intuitive, the details of what makes these models robust is poorly understood, and they represent only singular points in the space of RNNs. Developing novel tools inspired by and enhancing the gates employed by these models requires a more generic and rigorous interpretation of RNN properties [11]. Such methods are becoming more warranted in typical practical systems in which neural networks contain multiple interconnected hidden layers handling high-dimensional data. For such networks, mathematical and statistical interpretation is both essential and challenging. Machine learning research asserts that meaningful interfaces for interpretability of neural networks will not only enable meaningful human oversight, but prove useful in building new aligned AI systems [12].

To better understand the propagation of information in Recurrent neural networks and the robustness of training, we make use of the mathematical framework of *dynamical systems*. We treat Equation (1.1) as a *nonlinear*, *nonautonomous iterative map* with h_t as the state variable. The parameters of this dynamical system, $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{b}$ are updated through the training process, meaning the dynamics of this system evolve over training time.

Dynamical Systems

When systems only depend on first-order terms of the state variables, they are said to be *linear* systems. These systems can be written in the form

$$\dot{x} = Ax$$

and their solutions take the form of $x(x_0, t) = e^{tA}x_0$ [13]. Linear systems have been studied extensively, as they can easily be decomposed into parts which can be analyzed separately using various methods such as normal modes, Fourier analysis, superposition, and more.

Nonlinear systems – those for which there are non-linear factors of the state variables determining their evolution– cannot easily be solved analytically, and the above analysis tools are not applicable when nonlinear interactions are present. Therefore, dependence on finding exact analytical solutions to understand the behavior of a system is limiting for a large number of problems which have complex, non-linear behaviors. The study of *dynamical systems* is concerned with the geometry of the solutions of such systems. This geometric reasoning is used to understand the trajectories of the systems without solving for them explicitly [14].

Dynamical systems can be divided into two distinct categories: differential equations and iterative maps. Differential equations describe systems in continuous time with ordinary or partial derivatives. Iterative maps, on the other hand, describe systems in discrete time with quantities (such as state variables) at defined time-steps.

Dynamical systems can be further divided into two additional categories: autonomous and nonautonomous. Autonomous systems are ones for which the evolution of the state variables can be completely described by the values of those variables throughout the trajectory. For such systems, the difference or differential equations do not have any dependence on time, or any other external input variable. One such example of an autonomous system is the harmonic oscillator pendulum shown in Equation (1.2).

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -k \sin x_1 \end{aligned} \tag{1.2}$$

where x_1 represents the position of the oscillator, and x_2 represents the velocity. Since the system is determined by only the state variables x_1 and x_2 along with constant k, this system is autonomous.

Nonautonomous systems, on the other hand, do have a dependence on external variables. For example, a nonautonomous system could have an external input, like the driving force of the forced harmonic oscillator in Equation (1.3). The forced harmonic oscillator introduces a periodic driving force of magnitude F and period $\frac{\tau}{2\pi}$. Additionally, it has a damping force which is proportional to the velocity, with damping constant b.

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \frac{1}{m}\left(-kx_1 - bx_2 + F\cos\left(\frac{t}{\tau}\right)\right) \end{aligned} \tag{1.3}$$

.

The study of dynamical systems led to the observation of chaos. A dynamical system is chaotic if it exhibits deterministic aperiodic behavior with sensitive dependence on the initial conditions. This led to the definition of an *ergodic* system [15], a system for which the time averages along trajectories originating from almost all initial points are equal. It was found that systems from many different application areas transition to chaos in similar ways, showing that this lens for studying dynamics can provide insights across various types of systems.

Lyapunov Exponents of Maps [16]

Lyapunov exponents (LEs) have been developed in the theory of turbulence as effective measures of the degree of instability (chaos) in autonomous systems [13, 14, 16]. Lyapunov exponents describe the rate of divergence of nearby trajectories.
Consider a map F of the form

$$X_{n+1} = F(X_n), \tag{1.4}$$

where X is the state vector. To determine the effect of small changes in the state vector about a point X_n, we start by finding the Jacobian matrix,

$$J_{ij}(X_n) = \left.\frac{\partial F_i}{\partial X^{(j)}}\right|_{X=X_n} \tag{1.5}$$

If the perturbation of X_n is ϵ_n, then we have,

$$\epsilon_{n+1} = \mathbf{J}(X_n)\epsilon_n, \tag{1.6}$$

which is a linear approximation of the evolution of the perturbations of the system about the point X_n. If we consider the difference in X resulting from a perturbation to the starting point X_0 after n time steps, this becomes

$$\frac{\partial X_n^{(i)}}{\partial X_0^{(j)}} = M_{ij}^n = \left[\mathbf{J}(X_{n-1})\mathbf{J}(X_{n-2})\dots\mathbf{J}(X_0)\right]_{ij}. \tag{1.7}$$

According to Oseledec's multiplicative ergodic theorem, for large values of n, the eigenvalues of $\mathbf{M}$ approach $e^{\lambda_i n}$ [17]. For almost any point X_0, there exists an orthonormal set of vectors $\boldsymbol{v}_i^0$, $1 \le i \le d$, where d is the dimension of the state space, such that:

$$\lambda_i = \lim_{n\to\infty} \frac{1}{n} \log \|\mathbf{M}^n \boldsymbol{v}_i^0\| \tag{1.8}$$

For ergodic systems, the exponents $\{\lambda_i\}$ do not depend on the initial point X_0.

Classical Interpretation of Lyapunov Exponents

There are multiple features of the Lyapunov spectrum which provide insight into the dynamical system.

The maximum LE indicates the linear stability. The linear distance, d, between nearby trajectories tends to grow as

$$d = \mathrm{e}^{\lambda_1 t}. \tag{1.9}$$

A system having Lyapunov exponents greater than zero represents chaotic dynamics. As the first exponent decreases in magnitude and approaches zero, the degree of chaos decreases. When all exponents are negative, the dynamics converge towards a fixed point attractor. Exponents of zero represent limit cycles or quasiperiodic orbits.

The Kaplan-Yorke conjecture [18] estimates the dimension of an attractor of a dynamical system. If j is the index such that

$$\sum_{i=1}^{j} \lambda_i \geq 0 \text{ and } \sum_{i=1}^{j+1} < 0 \tag{1.10}$$

Then, the Kaplan-Yorke dimension is defined as:

$$D = j + \frac{\sum_{i=1}^{j} \lambda_i}{|\lambda_{j+1}|} \tag{1.11}$$

The mean exponent describes the mean rate of expansion or contraction of full-volume (n-dimensional) elements [19], R_V.

$$R_V = \mathrm{e}^{\frac{1}{n}(\lambda_1+\lambda_2+\ldots+\lambda_n)t} \tag{1.12}$$

The LE variance measures heterogeneity in stability across different directions and can reflect the conditioning of the product of many Jacobians. In the study of turbulence, the Lyapunov exponents close to and below zero in a chaotic system have been showed to align with the inertial subrange of the turbulent system [20].

Example: Lorenz System

We use the Lorenz system to demonstrate the application of Lyapunov exponents to dynamical systems. The Lorenz system is defined by the following equations:

$$\begin{aligned} \frac{dx}{dt} &= -\sigma(x-y) \\ \frac{dy}{dt} &= \rho x - y - xz \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \tag{1.13}$$

If we introduce a perturbation $\epsilon_n = (\delta x, \delta y, \delta z)$ to this system, then it would evolve according to the linearized form of (1.13). This gives

$$\frac{d\epsilon}{dt} = \begin{bmatrix} -\sigma & \sigma & 0 \\ \rho - z & -1 & -x \\ y & x & -\beta \end{bmatrix} \epsilon \tag{1.14}$$

Before calculating the Lyapunov exponents, we evolve the system (1.13) from an initial condition for a number of time steps T_{init} so that it can approach the attractor. Then, we evolve the system for a much larger number of time steps to understand the long-term behavior of the system. To calculate the Lyapunov exponents, we start with three orthogonal vectors and evolve them according to (1.14). For simplicity, we choose this orthogonal basis to be the unit vectors $\epsilon^{(1)} = (1,0,0)$, $\epsilon^{(2)} = (0,1,0)$, $\epsilon^{(3)} = (0,0,1)$. For all values of $t > T_{init}$, we use the values of x, y, z to define (1.14), which is used to evolve the perturbation vectors. After each time step, the vectors are once again orthogonalized. When this orthogonalization occurs, the factor by which perturbation vector is stretched is calculated. The final Lyapunov exponents are found

by taking the sum of the logarithms of these stretching factors divided by the total amount of time for which these vectors were evolved. The Lyapunov exponents are defined by the limit as $t \to \infty$, but this value can be well approximated for sufficiently large simulation times.

The standard values for the Lorenz system are $\rho = 28$, $\sigma = 10$, $\beta = 8/3$. In addition to these values, we consider different values of ρ to see the effect it has on the dynamics (Fig. 1.1) and the Lyapunov exponents. The values of σ and β are not altered. We take a time step of $\Delta = 0.001$ and take $T_{init} = 10$. The system and perturbation vectors are then evolved for another $40,000$ time steps, corresponding to a simulation time of $t = 4$.

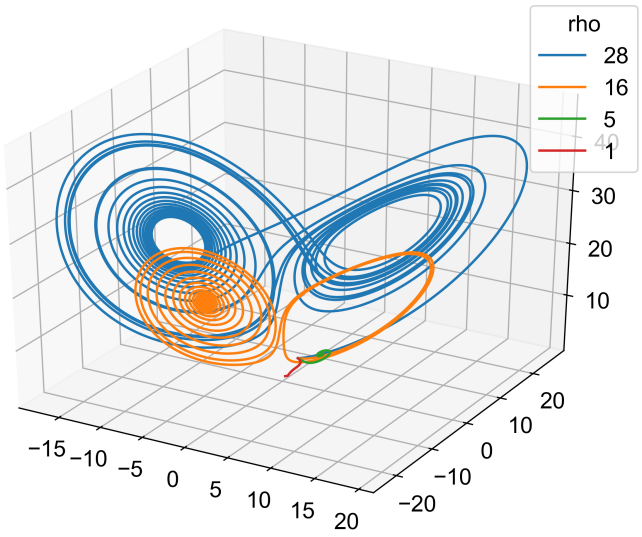

Figure 1.1: *Lorenz system trajectories for various values of ρ.* The standard value of $\rho = 28$ (blue) is well-known to be a chaotic system. The systems defined by $rho = 16$ (orange) and $rho = 5$ (green) are similar to the standard form in that they orbit around an attractor, although the size of the orbits is much smaller. When $\rho = 1$ (red), the trajectory instead rapidly collapses onto a single fixed point attractor.

For the Lorenz system, it can be shown that the sum of the Lyapunov exponents must add to the trace of the matrix in (1.14), or $-(1 + \sigma + \beta)$. With our choice of parameters, this is -13.667 for all systems simulated. For all four values of ρ, the sum of the exponents was within 0.16 of this value. As referenced above, the first Lyapunov exponent indicates the linear stability of the system, so larger values (especially greater than 0) indicate greater degrees of instability. If all exponents are less than 0, this indicates the existence of a fixed point attractor. The values of $\rho = 28, 16, 5$ all correspond to unstable/chaotic systems, as they have a positive first Lyapunov exponent (See Figure 1.2), but the smaller values of ρ, which have smaller orbits as well, also have smaller first exponent, indicating a lesser degree of chaoticity. When $\rho = 1$, the first exponent is negative, which is consistent with the trajectory approaching a fixed point attractor.

Related Work

Spectral Analysis and Model Quality

Since vanishing and exploding gradients arise from long products of Jacobians of the hidden states dynamics whose norm could exponentially grow or decay, much effort has been made to mathematically describe the link between model parameters and the eigen- and singular-value spectra of long products [21, 22, 23, 24]. For architectures used in practice, these approaches appear to have a limited scope [25, 26]. This is likely due to spectra having non-trivial properties reflecting intricate long-time dependencies within the trajectory, and due to the need to take into account the dynamic nature of RNN systems.

Other techniques for inferring model quality include performing spectral analysis of deep neural network weights and fitting their distribution to truncated power laws to correlate with performance [27]. Another

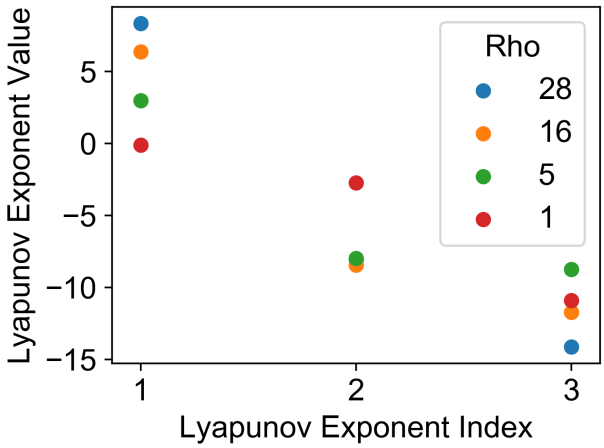

Figure 1.2: *Lyapunov exponents of Lorenz system for various values of ρ.* The positive sign of the first exponent when $\rho = 28$ (blue) is consistent with the chaotic nature of the standard Lorenz system. The systems for which the first exponent is still greater than zero (orange, green), but smaller in magnitude indicate that the trajectories still diverge over time, but to a lesser degree. This is consistent with the trajectories shown above, as they clearly do not converge to a fixed point but instead an orbit. When $\rho = 1$ (red), the first exponent being negative is consistent with its convergence to a single fixed point attractor.

approach using Koopman operators to linearize RNNs demonstrates that linear representations of RNNs capture the dominant modes of the networks and are able to achieve comparable performance to their non-linear counterparts [28].

Dynamical Systems Approaches

The identification of RNN as dynamical systems and as such developing appropriate analyses appears as a prospective direction. Recently, dynamical systems methodology has been applied to introduce constraints to RNN architecture to achieve better robustness, such as orthogonal (unitary) RNN [29, 30, 31, 32, 33, 34, 35] and additional architectures such as coupled oscillatory RNN [36] and Lipschitz RNN [37]. These approaches set network weights to form dynamical systems which have the desired Jacobians for long-term information propagation. In addition, analyses such as stochastic treatment of the training procedure have been shown to stabilize various RNN [38]. Furthermore, a universality analysis of fixed points of different RNN, proposed in [39], hints that RNN architectures could be organized in similarity classes such that despite having different architectural properties would exhibit similar dynamics when optimally trained. In light of this analysis, it remains unclear to which similarity classes in the space of RNN the constrained architectures belong and what is the distribution of the architectures within each class. These unknowns warrant development of dynamical systems tools that *characterize and classify RNN variants.*

Chapter 2

Lyapunov Exponents of Recurrent Neural Networks

RNN dynamics are non-autonomous because the hidden state dynamics $\mathbf{h}_t$ are driven by inputs $\mathbf{x}_t$. The theory of *random dynamical systems* generalizes stability analyses to non-autonomous dynamics driven by an input sequence sampled from a stationary distribution [40]. Analytical results typically employ uncorrelated Gaussian inputs, but the framework is expected to apply to a wider range of well-behaved input statistics. This includes those with finite, low-order moments and finite correlation times like character streams from written language and sensor data from motion capture systems. The time course of the driven dynamics depends on the specific input realization but, critically, the theory guarantees that the stationary dynamics for all input realizations share the same stability properties, which will in general depend on the input distribution (e.g., its variance). This chapter is adapted from [7].

Calculation of Lyapunov Exponents for RNNs

In this section, we apply the terminology and dynamical systems analysis tools outlined in the previous chapter to RNNs and describe the calculation of Lyapunov exponents in this setting.

Recurrent Neural Nets as Maps

Let us consider the standard vanilla RNN cell.

$$\boldsymbol{h}_t = \sigma_h(\mathbf{V}\boldsymbol{h}_{t-1} + \mathbf{U}\boldsymbol{x}_t + \mathbf{b}), \tag{2.1}$$

where σ_h is either *tanh* or *ReLU*. The output of this RNN cell can be viewed as a map on the state space of h_t. For this mapping, the Jacobian is given by:

$$\mathbf{J} = \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \sigma_h'(\mathbf{V}\boldsymbol{h}_{t-1} + \mathbf{U}\boldsymbol{x}_t + \mathbf{b}) \circ \mathbf{V}, \tag{2.2}$$

where $\circ$ is used to indicate elementwise multiplication (if both vectors) or row-wise multiplication (if vector and matrix).
Alternatively, if $\boldsymbol{v}, \boldsymbol{u} \in \mathcal{R}^n, \mathbf{U} \in \mathcal{R}^{n\times n}$, then

$$\boldsymbol{u} \circ \boldsymbol{v} = \begin{bmatrix} u_1 \cdot v_1 \\ \vdots \\ u_k \cdot v_k \\ \vdots \\ u_n \cdot v_n \end{bmatrix}, \quad \boldsymbol{u} \circ \mathbf{U} = \begin{bmatrix} u_1 \cdot W_{11} & \dots & u_1 \cdot W_{1j} & \dots & u_1 \cdot W_{1n} \\ \vdots & & \vdots & & \vdots \\ u_k \cdot W_{k1} & \dots & u_k \cdot W_{kj} & \dots & u_k \cdot W_{kn} \\ \vdots & & \vdots & & \vdots \\ u_n \cdot W_{n1} & \dots & u_n \cdot W_{nj} & \dots & u_n \cdot W_{nn} \end{bmatrix}$$

If σ_h is $tanh$,

$$\sigma_h'(t) = \operatorname{sech}^2(t) \tag{2.3}$$

If σ_h is $ReLU$,

$$\sigma_h'(t) = \begin{cases} 1, & t \geq 0, \\ 0, & t < 0 \end{cases} \tag{2.4}$$

Note that when input is present, this dynamical system becomes non-autonomous, as the map is a function of both the previous hidden state h_{t-1} and the input x_t.

Jacobians of Common RNN Architectures

Besides the vanilla RNN, there are two gated architectures that are commonly used in practice. These gating mechanisms are intended to improve the long-term memory of the network by learning the time-scales over which information should be retained. These commonly-used architectures are Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU).

LSTM

The LSTM is defined by the equations

$$\begin{aligned} f_t &= \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right) \\ i_t &= \sigma_g\left(W_i x_t + U_i h_{t-1} + b_i\right) \\ o_t &= \sigma_g\left(W_o x_t + U_o h_{t-1} + b_o\right) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_h\left(W_c x_t + U_c h_{t-1} + b_c\right), \end{aligned}$$

where σ_g is the sigmoid function and σ_h is $tanh$ (both applied element-wise). We use the following notation when calculating these derivatives:

$$y_* = W_* x_t + U_* h_{t-1} + b_*,$$

where $_*$ is determined by the gate/state. For LSTM, there are 4 different gates/states, (f, i, o, c), each with a corresponding y_*, W_*, U_*, b_*.

The derivatives of each of these gates/states with respect to the hidden states and the input is shown below. The final line shows what the derivative of the hidden state is relative to the hidden state at the previous time step or input at this time step.

$$\begin{aligned} \frac{\partial f_t}{\partial h_{t-1}} &= \left[\sigma(y_f) \circ (1 - \sigma(y_f))\right]^T \circ U_f \\ \frac{\partial i_t}{\partial h_{t-1}} &= \left[\sigma(y_i) \circ (1 - \sigma(y_i))\right]^T \circ U_i \\ \frac{\partial o_t}{\partial h_{t-1}} &= \left[\sigma(y_o) \circ (1 - \sigma(y_o))\right]^T \circ U_o \\ \frac{\partial c_t}{\partial h_{t-1}} &= \frac{\partial f_t}{\partial h_{t-1}} \circ c_{t-1} + \frac{\partial i_t}{\partial h_{t-1}} \circ \tanh(y_c) + i_t \circ \operatorname{sech}^2(y_c) \circ U_c \\ \frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial o_t}{\partial h_{t-1}} \circ \tanh(c_t) + o_t \circ \operatorname{sech}^2(c_t) \circ \frac{\partial c_t}{\partial h_{t-1}} \end{aligned}$$

$$
\begin{aligned}
\frac{\partial f_t}{\partial x_t} &= \left[\sigma(y_f) \circ (1 - \sigma(y_f))\right]^T \circ W_f \\
\frac{\partial i_t}{\partial x_t} &= \left[\sigma(y_i) \circ (1 - \sigma(y_i))\right]^T \circ W_i \\
\frac{\partial o_t}{\partial x_t} &= \left[\sigma(y_o) \circ (1 - \sigma(y_o))\right]^T \circ W_o \\
\frac{\partial c_t}{\partial x_t} &= \frac{\partial f_t}{\partial x_t} \circ c_{t-1} + \frac{\partial i_t}{\partial x_t} \circ \tanh(y_c) + i_t \circ \operatorname{sech}^2(y_c) \circ W_c \\
\frac{\partial h_t}{\partial x_t} &= \frac{\partial o_t}{\partial x_t} \circ \tanh(c_t) + o_t \circ \operatorname{sech}^2(c_t) \circ \frac{\partial c_t}{\partial x_t}
\end{aligned}
$$

GRU

The equations for the Gated Recurrent Unit (GRU) are given below.

$$
\begin{aligned}
r_t &= \sigma_g \left(W_r x_t + U_r h_{t-1} + b_r\right) \\
z_t &= \sigma_g \left(W_z x_t + U_z h_{t-1} + b_z\right) \\
n_t &= \sigma_h \left(W_n \circ x_t + b_{xn} + r_t \circ (U_n h_{t-1} + b_{hn})\right) \\
h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ n_t,
\end{aligned}
$$

Where σ_g is the sigmoid function and σ_h is tanh. Similar to the approach in the above section, we use the notation below to represent the arguments inside the activation functions for each gate.

$$
\begin{aligned}
y_{1*} &= W_* x + b_{x*} \\
y_{2*} &= U_* h_{t-1} + b_{h*}
\end{aligned}
$$

The corresponding derivatives for the GRU are shown below.

$$
\begin{aligned}
\frac{\partial r_t}{\partial h_{t-1}} &= \left[\sigma(y_{1r} + y_{2r}) \circ (1 - \sigma(y_{1r} + y_{2r}))\right]^T \circ U_r \\
\frac{\partial z_t}{\partial h_{t-1}} &= \left[\sigma(y_{1z} + y_{2z}) \circ (1 - \sigma(y_{1z} + y_{2z}))\right]^T \circ U_z \\
\frac{\partial n_t}{\partial h_{t-1}} &= \operatorname{sech}^2(y_{1n} + r_t \circ y_{2n}) \circ \left(\frac{\partial r_t}{\partial h_{t-1}} \circ y_{2n} + r_t \circ U_n\right) \\
\frac{\partial h_t}{\partial h_{t-1}} &= -\frac{\partial z_t}{\partial h_{t-1}} \circ n_t + (1 - z_t) \circ \frac{\partial n_t}{\partial h_{t-1}} + h_{t-1} \circ \frac{\partial z_t}{h_{t-1}} + z_t \circ I
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial r_t}{\partial x_t} &= \left[\sigma(y_{1r} + y_{2r}) \circ (1 - \sigma(y_{1r} + y_{2r}))\right]^T \circ W_r \\
\frac{\partial z_t}{\partial x_t} &= \left[\sigma(y_{1z} + y_{2z}) \circ (1 - \sigma(y_{1z} + y_{2z}))\right]^T \circ W_z \\
\frac{\partial n_t}{\partial x_t} &= \operatorname{sech}^2(y_{1n} + r_t \circ y_{2n}) \circ \left(\frac{\partial r_t}{\partial x_t} \circ y_{2n} + W_n\right) \\
\frac{\partial h_t}{\partial x_t} &= -\frac{\partial z_t}{\partial x_t} \circ n_t + (1 - z_t) \circ \frac{\partial n_t}{\partial x_t} + h_{t-1} \circ \frac{\partial z_t}{x_t}
\end{aligned}
$$

Multi-layered RNNs

When the network has multiple recurrent layers, we can partition the hidden state h into the hidden states at layer k. Then, if a given recurrent network has l layers, the hidden states of that network are

$$\boldsymbol{h}_t = \begin{bmatrix} \boldsymbol{h}_t^1 \\ \boldsymbol{h}_t^2 \\ \vdots \\ \boldsymbol{h}_t^k \\ \vdots \\ \boldsymbol{h}_t^l \end{bmatrix} \tag{2.5}$$

We denote the input of layer k as $\boldsymbol{x}_t^k$. We can easily use the equations for a given recurrent network type to calculate the derivative of the hidden state of a layer k, $\boldsymbol{h}_t^k$ with respect to the hidden states of that same layer at the previous time step $\boldsymbol{h}_{t-1}^k$, as well as with respect to the external input into that layer for this time step $\boldsymbol{x}_t^k$. Since layer k takes the output of layer $k-1$ as input, $x_t^k = h_t^{k-1}$ for $k \geq 2$. Then, in order to find the derivative of the k^{th} layer with respect to a previous layer,

$$\frac{\partial \boldsymbol{h}_t^k}{\partial \boldsymbol{h}_{t-1}^j}, \quad j \leq k,$$

we compose the derivative with respect to hidden state the previous layer with the derivative with respect to the input for each preceding layer until layer j is reached, at which point we calculate the derivative with respect to the hidden state at the previous time step, $\frac{\partial h_t^j}{\partial h_{t-1}^j}$. This gives the composition:

$$\frac{\partial \boldsymbol{h}_t^k}{\partial \boldsymbol{h}_{t-1}^j} = \frac{\partial \boldsymbol{h}_t^k}{\partial \boldsymbol{h}_t^{k-1}} \cdot \frac{\partial \boldsymbol{h}_t^{k-1}}{\partial \boldsymbol{h}_t^{k-2}} \cdot \ldots \cdot \frac{\partial \boldsymbol{h}_t^{j+1}}{\partial \boldsymbol{h}_t^j} \cdot \frac{\partial \boldsymbol{h}_t^j}{\partial \boldsymbol{h}_{t-1}^j} \tag{2.6}$$

For $j < i \leq k$,, the hidden state at the previous layer (h_t^{i-1}) is treated as the input signal to layer i, analogous to x_t. Thus, $\frac{\partial \boldsymbol{h}_t^i}{\partial \boldsymbol{h}_t^{i-1}}$ is treated as $\frac{\partial \boldsymbol{h}_t^i}{\partial \boldsymbol{x}_t^i}$.

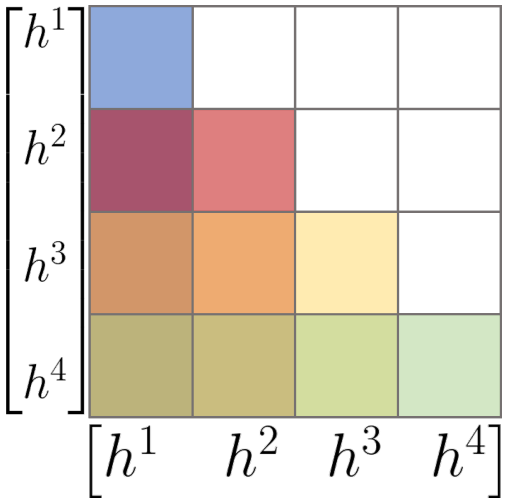

Figure 2.1: Block structure of multilayer Jacobian for one-directional RNN with four layers. The contribution of each layer's input at time step $t-1$ (columns) on the derivative of each layer at time step t (rows) is represented by a different color. Thus, the square corresponding to $\frac{\partial h^4}{\partial h^1}$ has all four colors because it involves contribution from the first through fourth layer.

When implementing this in matrix form, we start by calculating the derivatives for the first layer, then each layer below it. The composition described above can be taken by multiplying by the block directly above it. This reduces redundant computation and requires only a loop as long as the number of layers in the RNN. Given the structure of one-directional recurrent layers, this matrix would be block lower-triangular, with the size of each block corresponding to the number of hidden units in the layer, as is demonstrated in Figure 2.1.

Algorithm for RNN Lyapunov Spectrum Calculation

We adopt the well-established standard algorithm for computing the Lyapunov spectrum [41, 42]. We choose the driving signal to be sampled from fixed-length sequences of the test set for a particular task. For each input sequence in a batch, a matrix $\mathbf{Q}$ is initialized as the identity, and the hidden states, $\mathbf{h}$, are initialized as zero. To track the expansion and contraction of $\mathbf{Q}$ at each step, the Jacobian of the hidden states is calculated and then applied to a set of vectors of $\mathbf{Q}$. The expansion of each vector is calculated and the matrix $\mathbf{Q}$ is updated using the QR decomposition at each step. If r_i^t is the expansion of the i^{th} vector at time step t – corresponding to the i^{th} diagonal element of $\mathbf{R}$ in the QR decomposition– then the i^{th} Lyapunov exponent λ_i resulting from an input signal of length T is given by:

$$\lambda_i = \frac{1}{T}\sum_{t=1}^{T}\log(r_i^t) \tag{2.7}$$

The Lyapunov exponents resulting from each input x^j in the batch of input sequences are calculated in parallel and then averaged. For our experiments, the Lyapunov exponents were calculated over 100 time steps with 10 different input sequences. The mean of the 10 resulting Lyapunov spectra is reported as the spectrum.
The algorithm for computing the Lyapunov exponents is described in Algorithm 1.

Algorithm 1: Lyapunov Exponents Calculation

```
for x^j in Batch do
    initialize h, Q
    for t = 1 → T do
        h ← f(h, x_t^j)
        J ← df/dh
        Q ← J·Q
        Q, R ← qr(Q)
        γ_i += log(R_ii)
    end
    λ_i^j = γ_i^j / T
end
λ_i = mean_j(λ_i^j)
```

It can be seen from Figure 2.2 that the mean calculated in the final step of Algorithm 1 for an example character prediction task (described later in this chapter) rapidly converges for a small number of iterations (j), as shown by the flattening of each of the curves within the first 100 iterations. Moreover, the standard deviation of the resulting Lyapunov spectrum is greater for the exponents with larger index, which corresponds to more negative values. Importantly, this means that the largest exponents have the smallest empirical errors.

Example: Lyapunov Exponents of Vanilla RNN

A demonstration of the various stability regimes for RNNs is given in Figure 2.3. An untrained vanilla RNN exhibits stable, marginally stable, and chaotic dynamics for different initializations of the recurrent weights, $\mathbf{V}$ sampled from Gaussian distributions with variance $\sigma^2_{\mathbf{V}} = 1/500, 100, 500$, respectively.

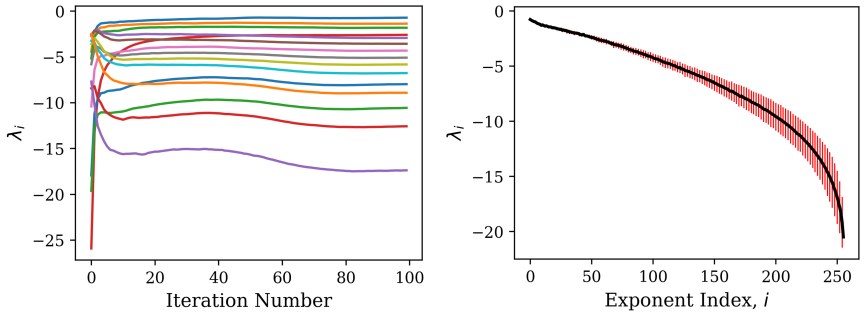

Figure 2.2: *Convergence of Lyapunov spectrum estimator for network trained on character prediction task.* (a) Lyapunov exponent estimation as a function of time, as shown in Equation (2.7), where T is the number of iterations. Each line represents a different exponent in the spectrum. By 100 iterations, it is clear that each exponent is changing very little as a function of iteration number. (b) The mean Lyapunov spectrum over the set of input sequences. Standard deviation is shown by the red bars.

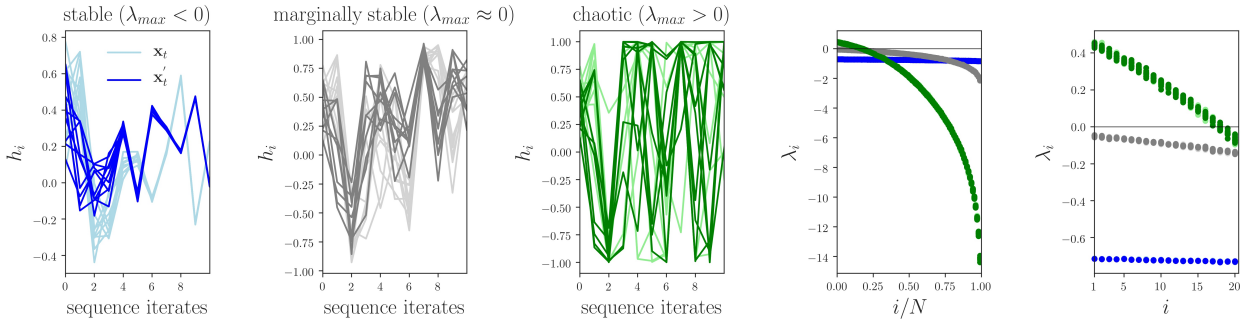

Figure 2.3: *Distinguishing stability regimes of forward dynamics using Lyapunov spectra.* (a-c) Example unit activations from simulations computed from 10 initial conditions and 2 input sequences ($\mathbf{x}_t$ (light) and $\mathbf{x}'_t$ (dark)) for variance of weight matrix $\mathbf{V}$ set to gives 3 examples spanning the 3 qualitatively distinct stability regimes based on the sign of λ_{max}: stability, $\lambda_{max} < 0$ (a; blue, $\sigma^2_{\mathbf{V}} = 1/500$); marginal stability, $\lambda_{max} \approx 0$ (b; gray, $\sigma^2_{\mathbf{V}} = 100$); and chaos $\lambda_{max} > 0$ (c; green, $\sigma^2_{\mathbf{V}} = 500$). (d) Lyapunov spectra computed from (a-c) (same colors). (e) First 20 exponents from (d). (vanilla RNN with $\phi = \tanh$; $\mathbf{V}$ orthogonal; $\mathbf{U} = \mathbb{I}$; 2 realizations (light/dark) of $\mathbf{x}$ Gaussian, $\mathbf{\Lambda}_{\mathbf{x}} = \sigma^2_{\mathbf{x}}\mathbb{I}$ with $\sigma^2_{\mathbf{x}} = 0.6$; 10 Gaussian initial conditions, $\mathbf{\Lambda}_{\mathbf{h}_0} = \sigma^2_{\mathbf{h}_0}\mathbb{I}$ with $\sigma^2_{\mathbf{h}_0} = 1$).

Implementation Details and Computational Improvements

While the computational graphs in machine learning libraries like PyTorch and Tensorflow can be used to calculate the Jacobian, these methods turn out to be inefficient. Therefore we have transcribed the analytic expressions for the Jacobian of common RNNs for any number of layers and published it as an opensource software library[1]. This library can be expanded upon to include Jacobians of additional state-of-the-art architectures so their dynamics can be analyzed using the framework. Preliminary results show several orders of speed up (200 on a small example network) over the built-in Pytorch autograd methods for calculating Jacobians and Lyapunov exponents. Further performance improvements over built-in methods are expected as the size and complexity of the networks grows. With this approach, calculating Lyapunov exponents will be attainable even for very large multi-layered networks.

The algorithm for calculation of LE can be altered by introducing two alterations. (*c.f.* Algorithm 2). For the first alteration, we can 'warm-up' the hidden states $\mathbf{h}$ and the vectors of $\mathbf{Q}$ by evolving them via the network dynamics for a set number of steps before calculating the Lyapunov Exponents. This allows the hidden states and vectors of $\mathbf{Q}$ to relax onto the attractor of the dynamical system defined by the

[1] `https://github.com/lyapunov-hyperopt/lyapunov_hyperopt`

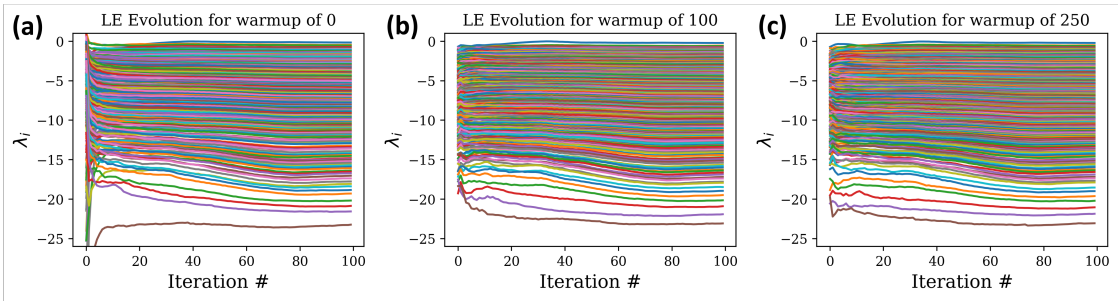

Figure 2.4: *The effect of warmup on the calculation Lyapunov spectrum.* Shown are the Lyapunov exponents as a function of iteration with (a) no warmup and warmups of (b) 100 time steps and (c) 250 time steps. It is clear that the case with no warmup has large changes early in the evolution of the spectrum, whereas those with warmup periods have much smaller changes early in the evolution.

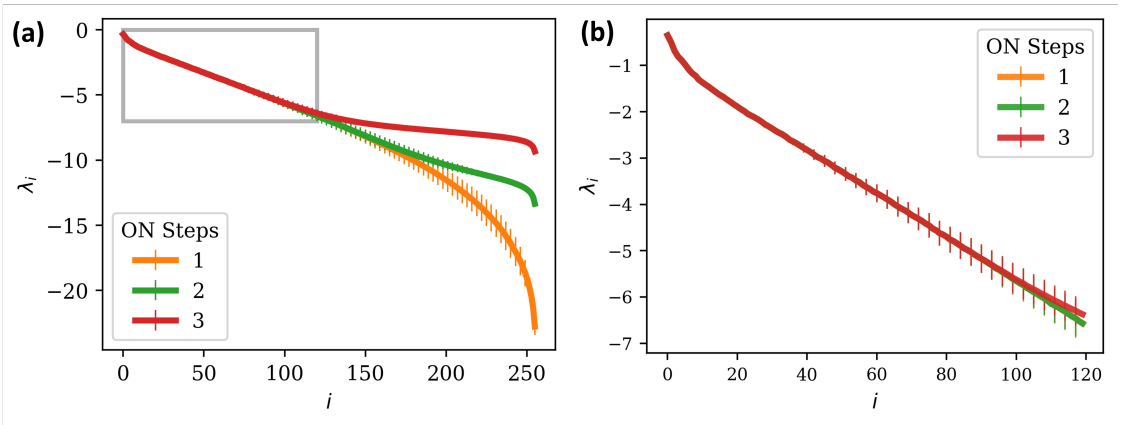

Figure 2.5: *The effect of selection of T_{ON} on the resulting spectrum.* Shown are (a) the full Lyapunov spectrum and (b) the first half of the spectrum for a trained network with different choices of T_{ON} (part of full spectrum plot indicated by gray box). Whereas the selection of T_{ON} clearly impacts the second half of the spectrum, the first half remains nearly identical for all these selections of T_{ON}.

network. This means that all samples used are from the stationary distribution on the attractor. Without the warmup, the initial samples are from a transient phase with a distribution over initial conditions that lies off the attractor and so has a bias effect on the average for finite sample averaging.
Further improvement in a second alteration can be made by not taking the QR decomposition every time step during the calculation of the exponents, but instead every T_{ON} steps. Since the QR step is the most expensive computation in the algorithm, the increase in speed over orthonormalizing every step is approximately T_{ON}. However, since increasing this interval leads to greater expansion and contraction of the vectors of $\mathbf{Q}$ before orthonormalization steps, this can lead to a spurious plateau at higher indices due to the accumulation of rounding errors [43]. However, if one cares only about the first few exponents, this effect is negligible for reasonable selections of T_{ON}. In Fig. 2.5, we show how increasing T_{ON} flattens out the spectrum beyond a certain index, but that the beginning of the spectrum remains unaffected by the selection of T_{ON}.

Algorithm 2: Modified Lyapunov Exponents Calculation

for x^j *in Ensemble* **do**
  initialize **h**, **Q**
  Warm-up **h**, **Q**
  for $t = 1 \rightarrow T$ **do**
    $\mathbf{h} \leftarrow \mathrm{f}(\mathbf{h}, x_t^j)$
    $\mathrm{J} \leftarrow \frac{d\mathbf{f}}{d\mathbf{h}}$
    $\mathrm{Q} \leftarrow \mathrm{J} \cdot \mathrm{Q}$
    if $t \ \% \ T_{ON} = 0$ **then**
      $\mathrm{Q}, \mathrm{R} \leftarrow qr(\mathrm{Q})$
      $\gamma_i \mathrel{+}= \log(\mathrm{R}_{ii})$
    end
  end
  $\lambda_i = \gamma_i / T$
end
$\lambda_i = \mathrm{mean}_j(\lambda_i^j)$

In order to leverage the Lyapunov spectrum for gradient-based optimization, it will be necessary to adapt the code to support back-propagation. The QR decomposition is a key step in this calculation, but its numerical stability could pose issues during back-propagation. While methods for backpropagation-friendly eigendecomposition do exist [44], the adpation of these methods in replacement of the QR decomposition in the calculation of the exponents without significantly hindering computational speed is unclear and requires further exploration.

Lyapunov Spectrum Characteristics for Training and Performance of RNNs

With these computational tools, we aim to determine the "fingerprints" of Lyapunov spectrum of robust RNNs which successfully learn particular tasks. We accomplish this by observing the evolution of the Lyapunov spectrum over the course of training. Furthermore, surveying these properties across different network setups and examples, we will examine the spectral properties to identify those that correspond to optimally trained networks.

Evolution of Lyapunov Exponents over Training

Training changes the weights in RNN, which leads to changes in how h evolve over time. While the choice of initialization can lead to different shapes before training, observing the common features that emerge among all networks throughout training reveals that, for a given task, a general shape emerges throughout training. The evolution of the Lyapunov spectrum of a network from untrained to optimally trained gives a general picture of the dynamics that correspond to improved performance. For an LSTM network with uniform initialization on the symmetric interval $[-p, p]$, different values of p clearly yield different initial Lyapunov spectra.

We find that the general shape of the spectrum is reached early in training. As the network parameters change during training, the resulting dynamics of the system defined by these parameters similar converge, leading to significantly different spectra between untrained and trained networks. In Fig. 2.6, we use the CharRNN task as an example to show that the spectrum rapidly changes in the first few epochs of training, with the dynamics of the system becoming more concentrated in the first few dimensions, resulting in a small number of exponents near zero and increasingly negative exponents for all other directions. This general spectrum shape quickly converges after a small number of training epochs to a spectrum near that of the fully trained network. The reduced mean-squared difference shows a $\mathcal{O}(t^{-1})$ convergence with learning epoch t, while the mean difference shows this convergence is from below. This held across the range of initialization parameters tested.

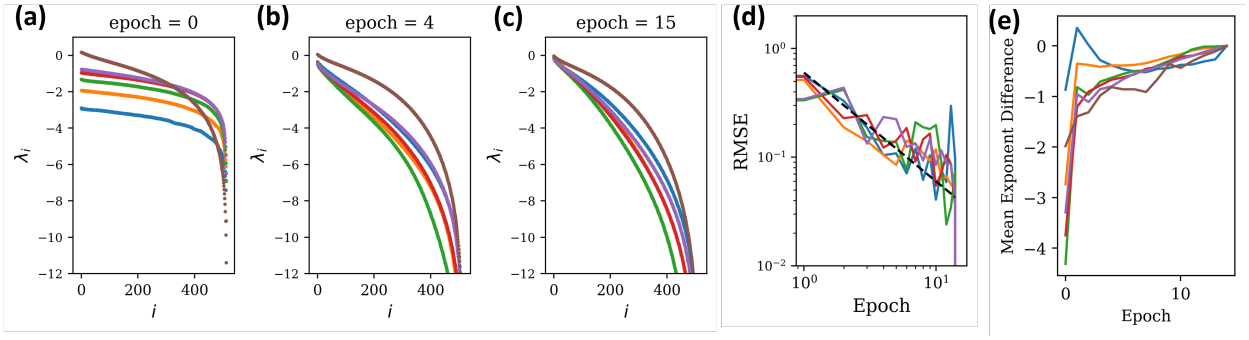

Figure 2.6: *Evolution of the Lyapunov spectra over training.* For values of initialization parameter given in legend in (a) ($p = 0.04$ (Blue), $p = 0.08$ (Orange), $p = 0.12$ (Green), $p = 0.16$ (Red), $p = 0.2$ (Purple), $p = 0.5$ (Brown)). The spectra before training (a) and after four (b) and fifteen (c) epochs. The reduced mean-squared error between successive epochs (d) shows that the each spectrum converges quickly during training, with t^{-1} plotted as a dashed black line for reference. Similarly, the difference between the mean of the Lyapunov exponents at the fifteenth epoch and the mean at previous epochs (e) rapidly approaches zero throughout training.

Lyapunov Spectrum as a Robust Readout of Training Stability

In general, the dependence of Lyapunov spectrum on hyperparameters is tangled. To direct our study of this dependence and how it relates to performance, we used the task constraints to guide our interpretation of spectra behavior of trained networks from randomly sampled hyperparameters. We perform this experiment on two tasks: a character prediction classification task (CharRNN) and a signal reconstruction regression task (CMU Mocap). We choose the training and validation split for the data corresponding to each task as 80/20.

CharRNN: Character Prediction Task

For our CharRNN experiment, the text used for training the networks was Leo Tolstoy's *War and Peace*, which contains about 3.2 million characters of mostly English text, with a total of 82 distinct characters in the vocabulary. We split the data into train, validation, and test sets with 80/10/10 ratio, respectively. We study the performance of single-layer LSTM networks (with no bias vectors) with hidden size of 512 units trained on this data set. We train the network using stochastic gradient descent with a batch size of 100 and the Adam optimizer. The network is unrolled for 100 time steps. After 10 epochs of training, the learning rate is decreased by a multiplying by a factor a .95 after each additional epoch. The batch size used for training was 100. For each combination of dropout and learning rate, the network was trained for 40 epochs, and the training epoch at which the network had the lowest validation loss was selected.
For this task, we hypothesized from existing worke [45] that to satisfy the constraint of propagating a scalar signal over many iterates, it would be sufficient to have a single exponent approaching 0 with the other exponents more negative. Focusing here on hyperparameters, we uniformly sampled a fixed range of log-dropout and learning rate while keeping the initialization parameter fixed. In Fig. 2.7(a,b), we observe the spectra and indeed find a correlation between maximum Lyapunov exponent and validation loss. Here, we opt to report performance with loss value, as opposed to task accuracy. This is because loss is the exact quantity underlying gradients, and as such highlights the direct link between RNN dynamics and learning. Nevertheless, we verify that the phenomena reported here persist when measuring task accuracy instead, and refer the reader to Figure B2 in the appendix for further details.
Networks with a maximum LE closer to zero indeed performed better. Of these 60 spectra, we selected a smaller subset of 6 that spanned the range of the gross shapes. Interestingly, they also spanned the range of losses, suggesting there is a consistent signal about the loss encoded in the complex, yet systematic variation of the maximum Lyapunov exponent with hyperparameters. The gross shape of these spectra also correlated with the loss, supporting our hypothesis. However, the metrics to best characterize this gross shape are not immediately apparent. A straightforward interpretation of the signal we have observed is

that the slower the contraction of the effect of an input on the activity in the activity mode associated with this maximum LE, the longer gradient information can propagate backward in time. This would give the dynamics more predictive power and thus better performance, but it remains to be verified.

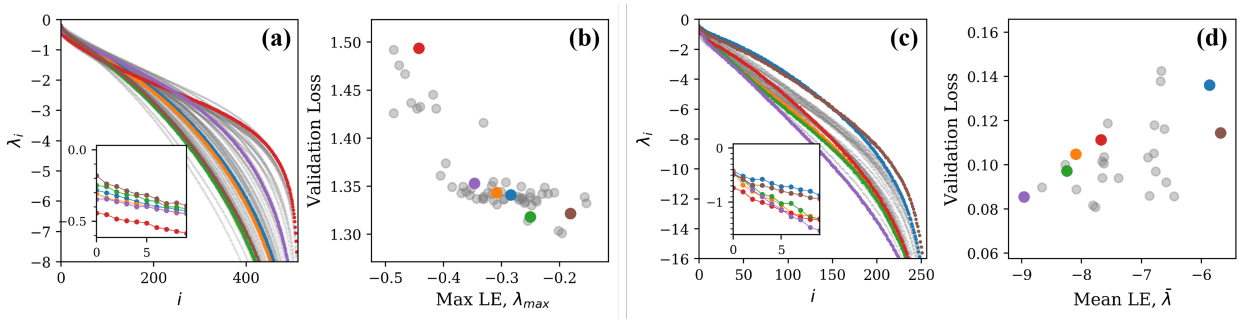

Figure 2.7: *Correlation between Lyapunov spectra and performance.* Shown are spectra (a,c) and loss dependence on spectra (b,d) for networks trained on CharRNN (a,b) and CMU MoCap (c,d). Those in (a,b) are for 60 random combinations of learning rate and log-dropout rate, while those in (c,d) are for 30 values of the initialization parameter, p. The colors in (a,b) and (c,d) are 6 samples selected by eye to roughly span the range of gross spectra shapes seen in (a) and (c), respectively. All other spectra are shown in gray.

CMU Mocap: Signal Prediction Task

The CMU Motion Capture (CMU Mocap) dataset is a high-dimensional data set capturing the location of various sensors on a subject over time as they conduct a variety of tasks. In particular, we narrow our focus onto the Sports and Activities category of the data set. The basketball, soccer, walking, and jumping subcategories were selected to compile a variety of motions which require consistent full-body movement (unlike, for example, the "directing traffic" and "car washing" sub-categories). For this task, we use a standard 256-unit LSTM. For each dimension of the data, the mean and the standard deviation across all times points is calculated, and those dimensions for which the standard deviation is less than 10^{-4} are ignored. All other dimensions are normalized by their standard deviation. For our example, we consider the first 20 features of the remaining data. The network is trained to predict the 20-dimensional output at the time step t given the preceding input sequence. The loss function is defined as mean-squared error across the 20 dimensional-output. We chose the batch size of 250 for training.

For this task, the higher dimensional data means that the changes that training induces in the shape of the Lyapunov spectra will likely be more complicated. We hypothesized that nevertheless there would be more information in the gross shape of the spectra of the dynamics of trained networks and less in the maximum LE. Here, we vary the initialization parameter from 0.05 to 0.5 while keeping the dropout and learning rates fixed. Indeed, we find (Fig. 2.7c,d) consistent variation of the gross shape with loss. We have quantified the variation of loss with gross shape using the mean Lyapunov exponent, though the spectra vary with higher order features as well.

The plots of Fig. 2.7 illustrate that distinct properties of the LE spectrum correlate with validation performance, depending upon task structure. For the CharRNN task, the maximum exponent being larger (closer to 0), corresponds to better performance. For the Mocap learning task, the larger mean Lyapunov exponent correspond to better performance.

The evolution of these statistics over the course of training of the CharRNN task is demonstrated in Figure 2.8. In particular, we observe that both the maximum and mean Lyapunov exponents change systematically with respect to validation loss early in training. While further training leads to further changes in these spectrum statistics, increasing the maximum LE beyond the threshold of -0.5 or decreasing the mean LE beyond the threshold of -4 has little impact on performance as these changes occur later in training, after the loss has converged.

Given our observation that the spectrum changes most rapidly early in training, we believe the most strongly-correlated metrics for a task could allow an alternative way to assess performance of a network early in training.

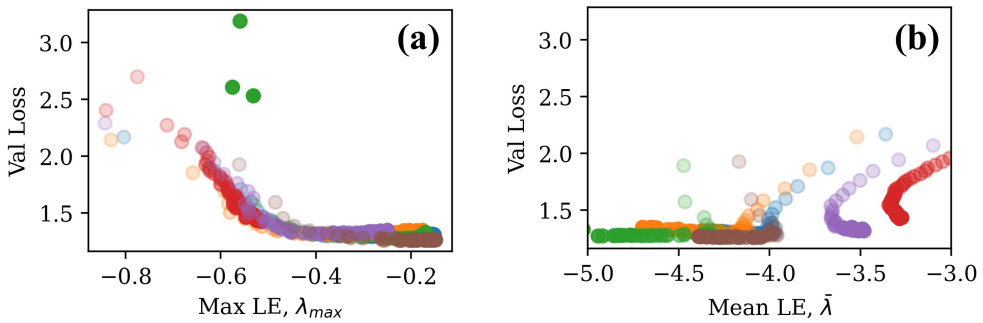

Figure 2.8: *Evolution of relation between performance and (a) Max LE and (b) Mean LE for CharRNN task over the course of training.* The evolution of the spectra highlighted in Figure 2.7(a-b) are shown in the corresponding color. For each color, the opacity of the points increases with epoch, so the most transparent points are from early in training. For this example, we see that, in general, (a) the maximum Lyapunov exponent increases over the course of training, corresponding to a decrease in the validation loss, but the decrease in loss is limited once the maximum increases beyond about -0.5, whereas (b)the mean LE tends to decrease over training, but has limited impact when decreasing beyond -4.

Conclusion and Discussion

In this chapter, we presented an exposition and example application of Lyapunov exponents for understanding training stability in RNNs. We motivated them as a suitable quantity related to stability of dynamics and useful in a complementary approach to existing mathematical approaches for understanding training stability focused on the singular value spectrum. We adapted the standard algorithm for their computation to RNNs, and showed how it could be made more efficient in standard machine learning development environments. We demonstrated that even the basic implementation runs almost 2 orders of magnitude faster than typical training time (see [43] for further precision and efficiency considerations) and that it converges relatively quickly with learning time. The latter implies it could be useful as a readout of performance early in training. To test this application, we studied it in two tasks with distinct constraints. In both we find interpretable variation with performance reflecting these distinct constraints.

There are a few points to be made about LEs in RNNs. First, one should consider carefully the degrees of freedom in the architecture under consideration. We have considered the hidden state dynamics of an LSTM such that there are as many Lyapunov exponents as there are units. LSTMs have gating variables that can also carry their own dynamics and could be considered as degrees of freedom, adding another N exponents to the spectrum. While we were able to obtain evidence from the spectra of hidden states alone, recent work [46, 43] suggests that studying the stability in the subspace of gating variables can also be informative. A second point is that one should also consider the role of the input weights $\mathbf{V}$ (c.f. Eq. 2). The strength of inputs (e.g. input signal variance) that drive high-dimensional dynamics has long been known to have a stabilizing effect [47, 48, 49]. Thus, the input statistics of the task's data can play a role that is modulated by $\mathbf{V}$, making the latter a target for gradient-based algorithms aiming to decrease loss. Due to limited space, we have not analyzed these weights here, but believe it is important to do so in applications. Finally, we raise perhaps the most glaring adaptation: that of approximating this asymptotic quantity to settings of finite-length sequences. We proposed to use averaging over inputs, in addition to averaging over initial conditions, as a way to take advantage of batch tensor mechanics to achieve faster LE estimates. This is in contrast to long-time averages used in the LE definition, although our method is justified under assumptions of ergodicity and short transient dynamics. Analyzing the influence of input batch size on this precision is a topic of future research.

The task analysis we have performed suggests a use case for the Lyapunov spectrum as having features that serve as an early readout of performance useful to speed up hyperparameter search. We looked at the maximum and mean Lyapunov exponent as two features highlighted by hypotheses we made based on our knowledge of the task constraints. Presumably, there are others relevant here and in other tasks. The search for generic features that do not require knowledge of task constraints (e.g. dynamical isometry) is important, as is demonstrating how useful these features are in practise with complex sequence data. One

limitation is how quickly they converge with learning. For example, as a composite quantity, the mean LE converges rather quickly, while the maximum LE is significantly slower to converge. In [43], it is shown that loss function/learning rule combinations can also sculpt the spectra and alter its statistics, demonstrating that these are more sources of variation to understand.

Interestingly, Full Force [50], an alternative to Back-propagation-through-time, has a qualitatively similar spectra (same max, min, and mean LE) but with much lower variance, presumably a desirable feature for generalization. This method also demands more information about performance than just a gradient, suggesting the interesting hypothesis that the precision with which the features of the Lyapunov spectra can be sculpted can scale with the amount of information provided in training. Finally, recent work has given theoretical grounding to why LSTMs avoid vanishing gradients [46]. Of course this was the reason for the design of LSTM so it only recapitulates the original design intuition. Looking forward however as more complex architectures are developed with less interpretable design, this approach based on LEs can still provide novel insight into why they work.

We close with a short discussion of open problems as this is a prominent area in understanding network dynamics at the intersection of machine learning and computational neuroscience. Having supporting analytical results is essential for robust control of complex problems. To this end, understanding how and where the assumption of untied weights breaks down and how forward propagation differs from backward propagation will be important in extending analytical results on spectral constraints. The Lyapunov spectrum can guide this work. Lastly, there could be insightful parallels in theoretical neuroscience work. For example, Lyapunov spectrum have been derived for additional degrees of freedom in the unit dynamics [51], different neuron types (such as hybrid neurons) [52], and more. Also, a discrepancy between the loss of linear stability and the onset of chaotic dynamics in driven systems must be understood [49]. Making these connections explicit will serve both fields.

Appendix

A Trained Network Hyperparameters

The following are the network parameters for the networks highlighted for each task in Figure 2.6.

Color	Dropout	Learning Rate	Initialization Parameter, p
Blue	0.034	0.00301	0.08
Orange	0.000	0.00319	0.08
Green	0.039	0.00574	0.08
Red	0.028	0.00055	0.08
Purple	0.014	0.00153	0.08
Brown	0.106	0.00485	0.08

Table A1: Network Hyperparameters for CharRNN

Color	Dropout	Learning Rate	Initialization Parameter, p
Blue	0.0	0.05	0.476
Orange	0.0	0.05	0.305
Green	0.0	0.05	0.196
Red	0.0	0.05	0.309
Purple	0.0	0.05	0.059
Brown	0.0	0.05	0.439

Table A2: Network Hyperparameters for CMU Mocap

B Spectrum metrics' correlation with loss

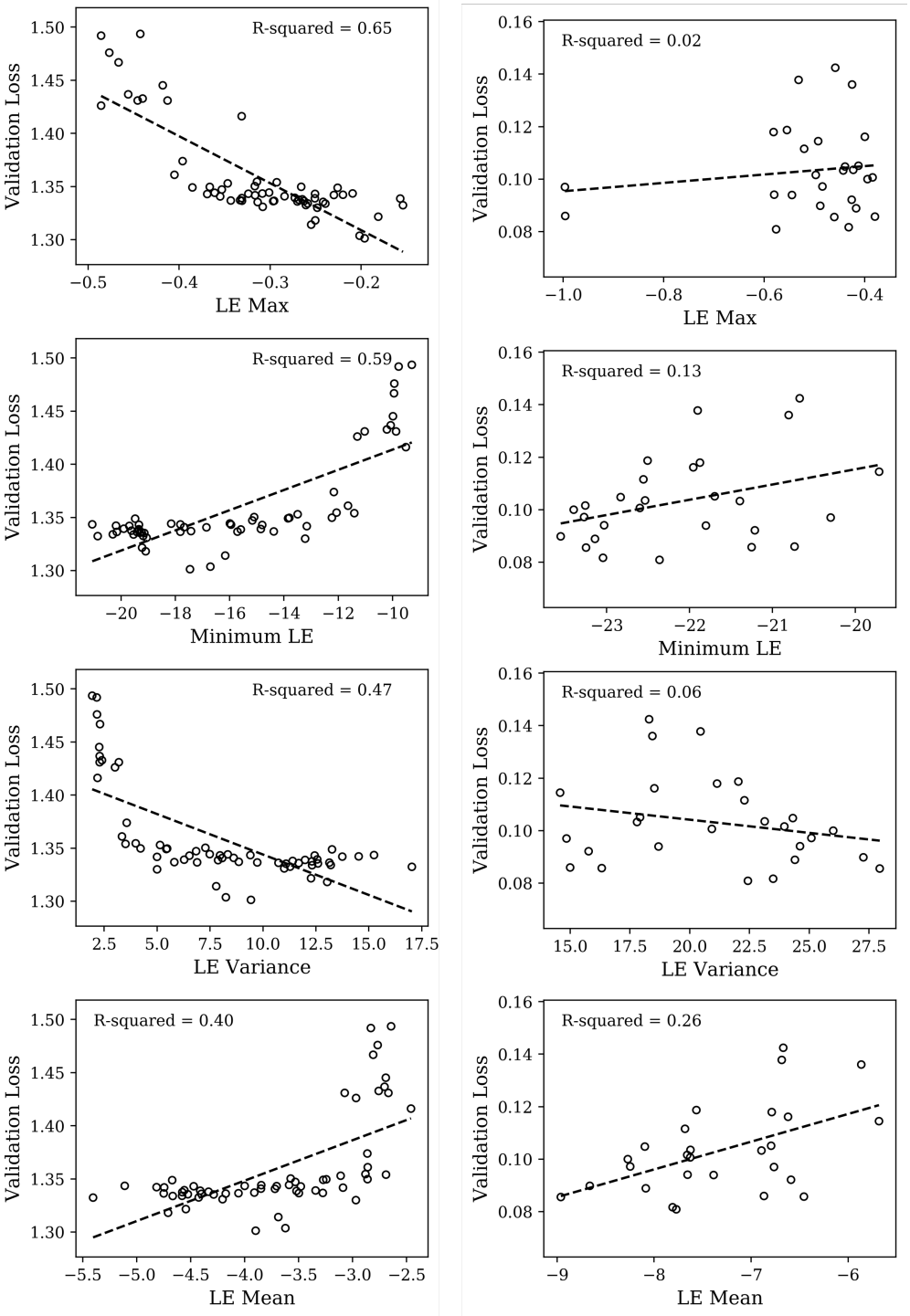

Figure B1: *Lyapunov spectrum statistics against validation loss for both tasks.* The results for the CharRNN task are shown on the left and CMU Mocap are shown on the right. Each row represents a different statistic of the spectrum.

Effect of replacing validation loss with accuracy

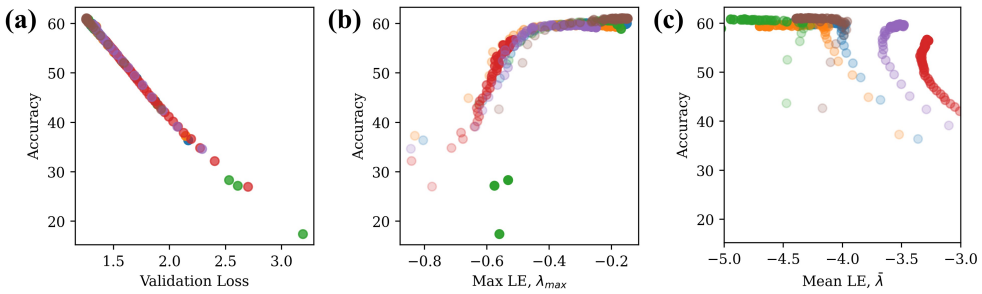

Figure B2: *Validation Loss vs Accuracy for CharRNN task.* For the CharRNN task, (a) we find that there is a strong correlation between the top-1 accuracy and the validation loss for the six networks highlighted in Figs. 2.7 and 2.8. The range of accuracy is from about 20% to 60%. As with Figure 2.8, the evolution of the spectra highlighted in Figure 2.7(a-b) are shown in the corresponding color in panel (b) and (c). For each color, the opacity of the points increases with epoch, so the most transparent points are from early in training. We see the evolution of the (b) maximum Lyapunov exponent and (c) mean Lyapunov exponent is simply inverted when plotted as a function of accuracy as opposed to validation loss (Figure 2.8), since accuracy generally increasing over training instead of decreasing.

Chapter 3

Lyapunov-Guided Representation of Recurrent Neural Network Performance

In the previous chapter, we presented data showing that there could exist a correlation between some features of the Lyapunov Spectrum and the performance of networks on certain tasks. However, since the statistics that best correlate with performance vary across tasks, it can be challenging to know *a priori* which features are most indicative of high accuracy. Therefore, we propose to learn these features and their relation to performance using a neural network. By using just the Lyapunov spectra as input and the validation loss as the target, we train a neural network to predict RNN accuracy without any direct information about the weights.

This chapter is adapted from [8].

Inferring Network Performance from Lyapunov Spectra with Fully-Connected PredNet

Initial results show that a simple fully connected neural network with just 50 neurons can be trained on the Lyapunov spectra of LSTM networks of varying sizes and predict the validation loss of the networks later in training on a character prediction task with moderate accuracy. We call such a predictive network PredNet. Furthermore, predictions of accuracy by this simple neural network accurately capture the dependence of the validation loss on initialization hyperparameters, allowing it to serve as a diagnostic tool for learning early in training (see Figure 3.1).

More specifically, we see that, when provided only with the Lyapunov spectra of the underlying networks, the network is able to determine that in the small-initialization-parameter regime (< 0.2), larger networks achieve better accuracy than smaller ones. However, it also finds that, upon increasing the initialization parameter beyond 0.24, the largest network tested (512 units) has deteriorating accuracy. Furthermore, the next largest network size of 256 experiences a deterioration of accuracy at a larger initialization parameter value (around 0.3). These results are consistent with the ground truth.

However, these observations also indicate that the precision and the resolution of the network's loss predictions are limited. Specifically, PredNet struggles to determine the capture the extent to which the larger networks become worse than the smaller networks for larger initialization parameters. Meanwhile, the the exact loss value for many of the networks is not correctly predicted, but is roughly approximated, leading to significant variation in predicted loss for very similar networks. This could indicate an overfitting of the data, suggesting that alternative deep-learning approaches which allow for a more robust representation are necessary to learn the relation between LE and accuracy.

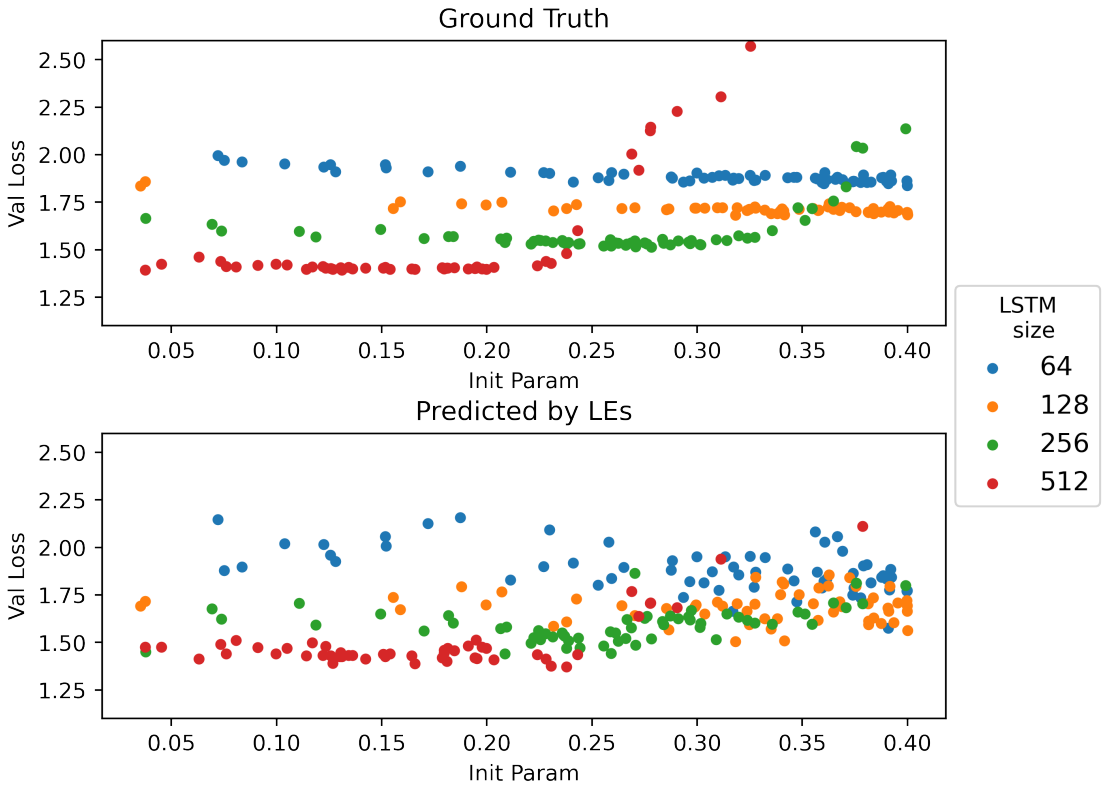

Figure 3.1: The validation loss of LSTMs learning the character prediction task described in [7] as a function of initialization parameter (top) shows that larger networks perform better for the right parameter selection, but are more sensitive to changes in initialization parameter above a certain threshold. A simple fully-connected network trained on the Lyapunov spectrum of these networks (PredNet) after just 40% of training predicts the future validation loss after training (bottom) and effectively captures this dependence

Proposed Network for Stable RNN: ASRNN

One network design which was developed specifically to preserve information over long periods of time is the Anti-symmetric RNN (ASRNN) [34]. This network constrains the hidden-to-hidden weight matrices to be antisymmetric so that the Jacobian's eigenvalues are all imaginary and multiplication corresponds to a rotation of the hidden states. We use this network architecture to analyze the dynamical properties as captured by the eigenvalues versus the Lyapunov exponents and their relation to accuracy. The Jacobian of this system is given by:

$$\boldsymbol{J}(t) = \text{diag}\left[\text{sech}^2\left((\mathbf{U} - \mathbf{U}^T)\boldsymbol{h}_{t-1} + \mathbf{W}\boldsymbol{x}_t + \boldsymbol{b}\right)\right](\mathbf{U} - \mathbf{U}^T) \tag{3.1}$$

This leads to the equation for the ASRNN

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \epsilon\tanh\left((\mathbf{U} - \mathbf{U}^T)\boldsymbol{h}_{t-1} + \mathbf{W}\boldsymbol{x}_t + \boldsymbol{b}\right), \tag{3.2}$$

where ϵ is a hyperparameter which represents the step size.

In order to ensure the stability of the system when solved using the forward Euler method, a diffusion hyperparameter γ is introduced to ensure the eigenvalues of the Jacobian (3.1) have slightly negative real parts. With this modification, the model becomes

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \epsilon\tanh\left((\mathbf{U} - \mathbf{U}^T - \gamma\mathbf{I})\boldsymbol{h}_{t-1} + \mathbf{W}\boldsymbol{x}_t + \boldsymbol{b}\right) \tag{3.3}$$

We train this network on sequential MNIST data using the initialization and hyperparameters outlined in [34]. The input to hidden matrices $\mathbf{W}$ are initialized to $N(0, 1)$. The hidden to hidden matrices are initialized to $N(0, \sigma^2/n)$, where $n = 128$ is the hidden size and $\sigma \in \{0, 1, 2, 4, 8, 16\}$. The step size $\epsilon \in \{0.01, 0.1, 1\}$ and diffusion $\gamma \in \{0.001, 0.01, 0.1\}$. The optimizer used was Adagrad, with batch size of 128 and learning rate chosen from $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1\}$. All models are trained for 50,000 iterations. The MNIST dataset was split into a train and validation set using a 75/25 split. For each network, the validation loss after 50,000 iterations is reported.

We analyze the state-to-state derivative, $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}$ of (3.3), which is

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \mathbf{I} + \epsilon \cdot \text{diag}\left[\tanh'\left((\mathbf{U} - \mathbf{U}^T - \gamma\mathbf{I})\boldsymbol{h}_{t-1} + \mathbf{W}\boldsymbol{x}_t + \boldsymbol{b}\right)\right](\mathbf{U} - \mathbf{U}^T - \gamma\mathbf{I}) \tag{3.4}$$

Once each of these networks is trained, we find the eigenvalues of this derivative when given a random input and hidden state and plot the real and imaginary parts. Each eigenvalue is associated with the validation loss of that network after training by the color of the dots. Purple dots correspond to high accuracy whereas yellow corresponds to worse accuracy on the validation set. A sample of network eigenvalues for one choice of γ is shown in Figure 3.2. We found that many networks had similar eigenvalue spectra and distinguishing between networks with high and low accuracy is difficult in this regime.

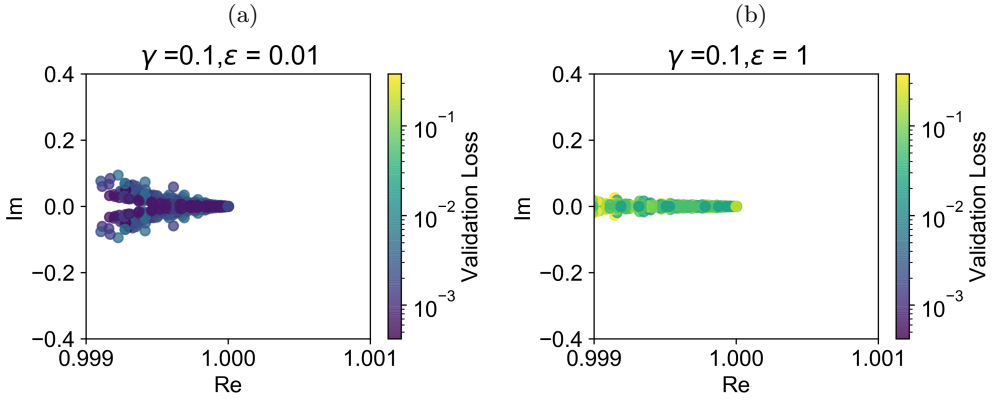

Figure 3.2: *Jacobian Eigenvalues for ASRNN with fixed diffusion parameter* $\gamma = 0.1$. Each plot shows the real and imaginary parts of the 128 eigenvalues of each network with $\gamma = 0.1$ and (a) $\epsilon = 0.01$, (b) and $\epsilon = 1$. This corresponds to 35 networks each. The color of each dot indicates the validation loss for the corresponding network, ranging the best (purple) to worst (yellow). These spectra indicate that while some structure may correlate with accuracy, there are many overlapping eigenvalues which correspond to a large variety of network accuracy.

However, if we instead consider the Lyapunov spectra of the ASRNN networks, we are able to get a more clear pattern distinguishing between the performance of the networks. The Lyapunov exponents are calculated using the validation set, and are warmed up for 500 time steps before evolving for another 500 time steps. The batch size used for the Lyapunov exponent calculation was 20. The corresponding Lyapunov exponent plots for the parameter combinations shown in Figure 3.2 are shown in Figure 3.3. The Lyapunov spectra of these networks shows that the better-performing networks tend to have smaller first Lyapunov exponents, and there is a gradient between good and bad performance as the first exponent increases. While this pattern tends to hold, it is not easily quantifiable and does not necessarily translate perfectly across all hyperparameter combinations. Nonetheless, these results suggest that there is important information in the Lyapunov spectra which correlates with performance in a way that the eigenvalues do not.

In order to explore this correlation, we train two very simple 10-neuron fully-connected PredNets on the Lyapunov spectra and the eigenvalues plotted above, where the targets are the validation loss. One network is trained on the exponents, while the other is trained on the eigenvalues. The data on which the PredNets are trained comes from a total of 315 ASRNN networks. We split these data into a training set and validation set, with an 80/20 split, corresponding to 252 and 63 networks, respectively. This split was randomly selected

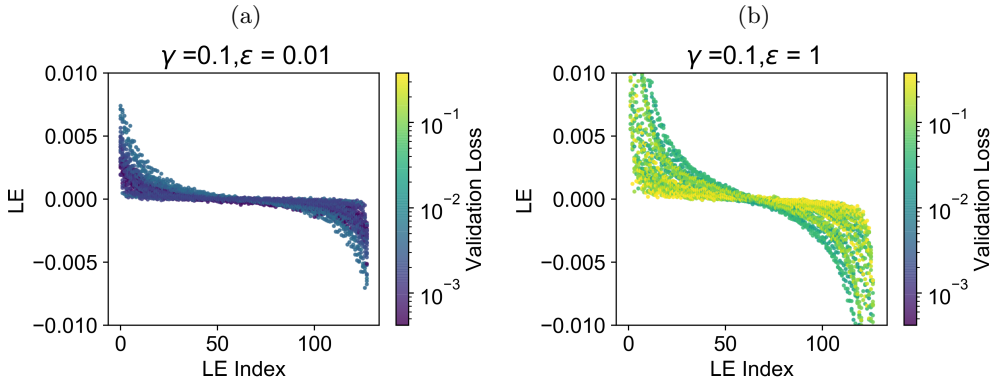

Figure 3.3: *Lyapunov Spectra for ASRNN with fixed diffusion parameter* $\gamma = 0.1$. As above, each plot shows the Lyapunov exponents of each network with $\gamma = 0.1$, (a) $\epsilon = 0.01$ and (b) $\epsilon = 1$. The pattern is not always consistent across all values of ϵ, but there tends to be a gradient in performance when we consider the first part of the spectrum. In general, good-performing networks tend to have smaller first Lyapunov exponents.

across all hyperparameter combinations. Each Prednet was trained using the Adam optimizer with learning rate of 0.0001 for 15000 epochs.

The validation loss of each PredNet over the course of training is shown in Figure 3.4. The fact that the validation loss for the Lyapunov exponents is lower than that of the Jacobian eigenvalues throughout training confirms the intuition that the Lyapunov exponents contain more useful correlations with performance. A deeper analysis of the features which correlate with performance will be necessary, but this prediction network will be a valuable tool to discover these correlations.

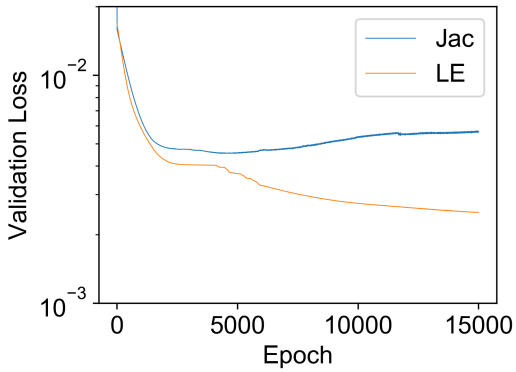

Figure 3.4: *Validation loss for PredNets trained on Jacobian eigenvalues (blue) and Lyapunov spectra (orange) of trained Anti-Symmetric RNNS.* Each PredNet has ten neurons and is trained independently on either the Jacobian eigenvalues or Lyapunov exponents dataset. The validation loss is shown at each epoch of training. After the first 5000 epochs of training, it is clear that the Lyapunov exponents data performs better than the Jacobian eigenvalues, indicating that this information is more informative regarding training performance.

AeLLE: Latent Lyapunov Exponent Embedding

In order to learn a robust relation between Lyapunov exponents and performance, we propose the use of AeLLE, an autoencoder which uses latent Lyapunov exponents to predict the corresponding network's accuracy. The proposed AeLLE methodology consists of three steps: 1) Computation of LE spectrum, 2) Autoencoder for LE spectrum, and 3) Embedding of Autoencoder Latent representation. The process for calculating the LE spectrum of RNNs is described in the previous chapter.

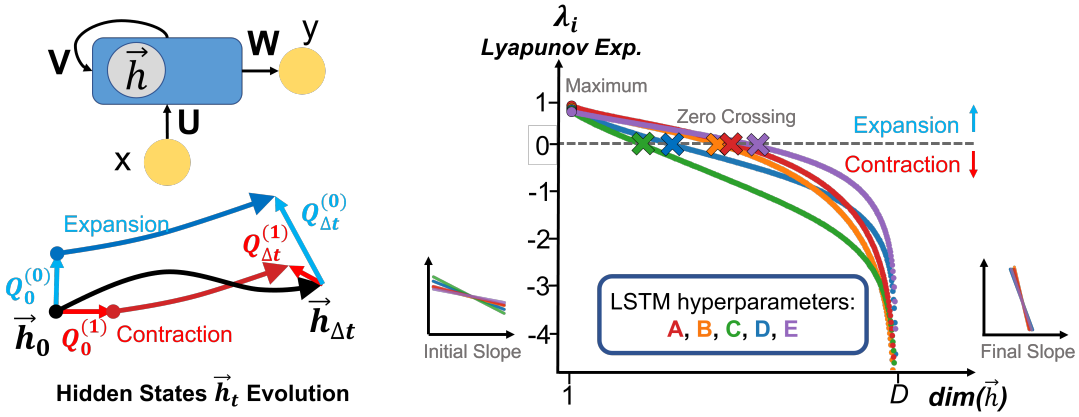

Figure 3.5: **LE Spectrum of RNN variants.** RNN (top left) hidden states h evolution is tracked to calculate the LE spectrum of the network. The exponents are found by calculating the expansion and the contraction of nearby trajectories over time (bottom left). LE spectra correspond to points on LE spectrum curves (right). Variation of hyperparameters will correspond to distinct LE spectra curves, even upon initialization before training on any task has occurred (right). Basic known LE features such as Maximum, Initial Slope, Final Slope, Number of exponents greater than zero marked on the plot, do not directly correlate to task accuracy.

Autoencoder for LE spectrum

An autoencoder consists of two components: an *encoder* network ϕ which transforms the input into a representation in the latent layer, and a *decoder* network ψ which transforms the latent representation into a reconstruction of the original input. Over the course of training, the Latent layer becomes representative of the variance in the input data and extracts key features that might not immediately be apparent in the input.

In addition to the reconstruction task, it is possible to include constraints on the optimization by formulating of a loss function for the Latent layer values (Latent space), e.g., a classification or prediction criterion. This can constrain the organization of values in the Latent space [9, 53]. We propose an adapted Autoencoder methodology for correlating LE spectra and RNN task accuracy. In this setup, we consider the LE spectrum as the input $\boldsymbol{Z}$. Our autoencoder consists of a fully-connected encoder network ϕ, a fully-connected decoder network ψ, and an additional linear prediction network ξ defined by

$$\begin{aligned} \hat{Z} &= (\psi \circ \phi)Z, \\ \hat{T} &= (\xi \circ \phi)Z, \end{aligned} \tag{3.5}$$

where $\hat{\boldsymbol{Z}}$, and $\hat{\boldsymbol{T}}$, correspond to the output from the decoder and predicted accuracy, respectively, with loss of $L = \|Z - \hat{Z}\|^2 + \alpha \cdot \|T - \hat{T}\|_l$. Ae performs the reconstruction task, optimization of the first term of Eq. 3.6, mean-squared reconstruction error of LE spectrum, as well as prediction of the associated RNN accuracy $\boldsymbol{T}$ (best validation loss), the second term of Eq. 3.6.

$$\phi, \psi, \xi = \arg\min_{\phi,\psi,\xi}(\|Z - \hat{Z}\|^2 + \alpha \cdot \|T - \hat{T}\|_l). \tag{3.6}$$

The parameter l can defined based on the desired behavior. The most common choices are $l = 1$, indicating the 1-norm, and $l = 2$, indicating the 2-norm.

During training of Ae, the weight α in the prediction loss is gradually being increased so that Ae emphasizes RNN error prediction once the reconstruction error has converged. We found that this approach allows to capture features of both RNN dynamics and accuracy. A choice of α being too small leads to dominance of the reconstruction loss such that the correlation between LE spectrum and RNN accuracy is not captured. Conversely, when α is initially set to a large value, the reconstruction along with the prediction diverge. The convergence of Ae for different RNN variants, as we demonstrate in Results section,

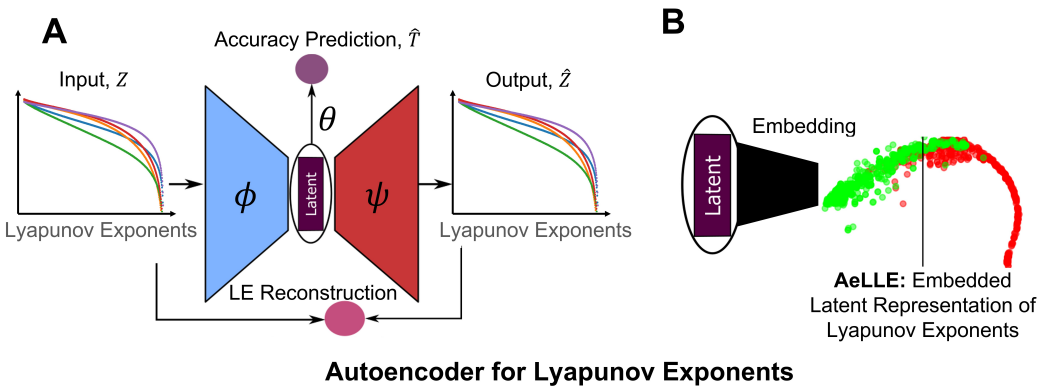

Figure 3.6: **AeLLE: LE spectrum Autoencoder and Latent Representation Embedding.** A) The autoencoder takes Lyapunov Exponents as input. This input is then embedded into a latent space (purple) by the encoder (blue). From this latent space, the autoencoder predicts the accuracy of the corresponding network, which is compared to the true accuracy of the network to get the prediction loss. Simultaneously, the Lyapunov spectrum is reconstructed from this latent space by the decoder(red). These reconstructed Lyapunov Exponents are then compared to the input Lyapunov exponents to get the reconstruction loss. These sums are added together with normalization factor α to get the total loss. B) The Latent space of the LE spectrum Autoencoder correlates LE spectrum and accuracy. Embedding of Latent space representation provides a low-dimensional clustering and classification space, leading to separation between high-accuracy (green) and low-accuracy (red) networks. We find that Latent space clusters the LE spectra well such that simple embedding (PCA) and simple linear classifiers (hyperplane, hyperellipse or threshold) classify RNN variants according to accuracy. Shown is an example linear classifier along the first PC dimension for networks trained on the CharRNN task (see Results section for more details).

shows that correlative features between LE spectrum and RNN accuracy can be inferred. The dependency of Ae convergence on a delicate balance of the two losses reconfirms that these features are tangled and thus the need for Ae embedding. We describe the settings of α and additional Ae implementation details in Supplementary Materials.

Embedding of Autoencoder Latent Representation

When the loss function of Ae converges, it indicates that the Latent space captures the correlation between LE spectrum and RNN accuracy. However, an additional step is typically required to achieve an organization of the Latent representation based on RNN variants accuracy. For this purpose, a low dimensional embedding, denoted as AeLLE, of the Latent representation needs to be implemented. An effective embedding would indicate the number of dominant features needed for the organization, provide a classification space for the LE spectrum features, and connect them with RNN parameters. We propose to apply the Principal Component Analysis (PCA) embedding to the Latent representation [54, 55]. The embedding consists of performing Principal Component Analysis and projecting the representation on the first few Principal Component directions (e.g. 2 or 3). While other, nonlinear, embeddings are possible, e.g., tSNE or UMAP [56, 57], the simple linear projection onto the first two principal components of the latent space results in an effective organization. This indicates that the Latent representation has successfully captured the characterizing features of performance. We show in the Results section that, for all examples of RNN architectures and tasks that we considered, the PCA embedding is sufficient to provide an effective space. In particular, in this space, most accurate RNN variants (green) can be separated from other variants (red) through a simple clustering procedure.

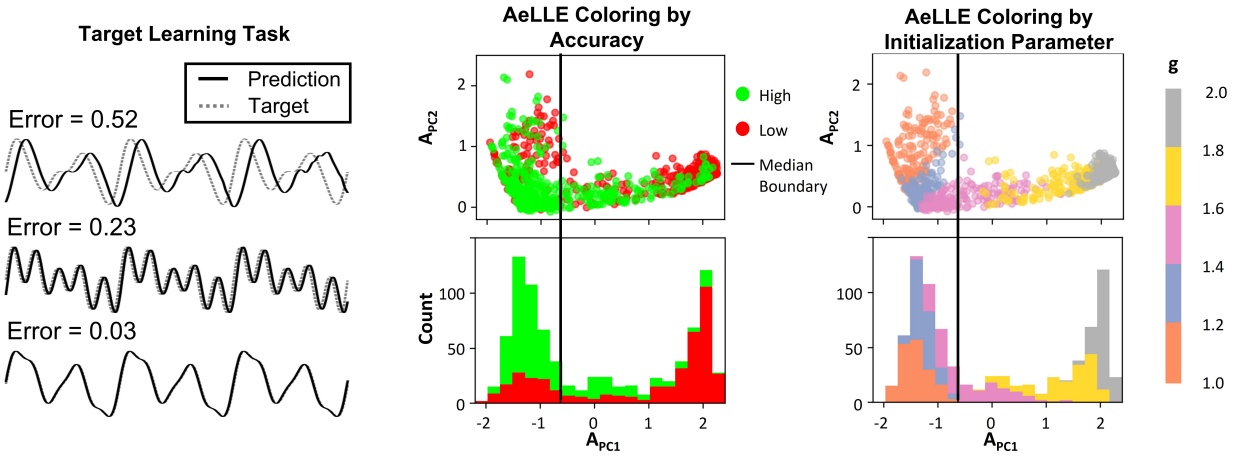

Figure 3.7: **Clustering by performance across initialization parameter: AeLLE for networks trained on Signal Reconstruction Task.** *Left:* The signal reconstruction task involves recreating a target signal. Higher losses indicate a greater pointwise difference between the target and predicted signals. *Center:* In the AeLLE representation, most of the high-accuracy networks (green) cluster to the left of most of the low-accuracy networks (red). The black vertical line indicates the location of the median classifier along the first principal component of the AeLLE space. Moreover, the greatest concentration of the high-accuracy networks is in the bottom-left of the space shown, which is consistent with the stacked histogram indicating that there are relatively few high-accuracy networks to the right of the median compared to low-accuracy, and many more to the left. *Right:* In comparison, the cluster to the bottom-left of the space shown contains a mixture of networks with initialization parameters ranging from 1.0 to 1.6. In general, networks with larger initialization parameter g (particularly once $g > 1.6$), tend to cluster further to the right in this space. This is consistent with the fact that networks with $g > 1.6$ tend to have lower accuracy on this task, but networks with $1 < g < 1.6$ tend to have similarly high accuracy (see Supplemental Materials).

Results

To investigate the applications and generality of our proposed method, we consider tasks with various inputs and outputs and various RNN architectures that have been demonstrated as effective models for these tasks. In particular, we choose three tasks: Signal Reconstruction, Character Prediction and Sequential MNIST. All three tasks involve learning temporal relations in the data with different forms of the input and objectives of the task. Specifically, the inputs range from low-dimensional signals to categorical character streams to pixel greyscale values. Nonetheless, across this wide variety of inputs and tasks, AeLLE space and clustering is consistently able to separate variants of hyperparameters according to accuracy in a way that is more informative than network hyperparameters alone.

More specifically, we consider *(i) Signal Reconstruction* task, also known as target learning. In this task, a random RNN is being tracked to generate a target output signal from a random input [58, 26]. This task involves intricate time-dependent signals and a generic RNN for which dynamics in the absence of training are chaotic. With this example, we demonstrate that our method is able to distinguish between networks of high and low accuracy across *initialization parameters*.

(ii) Character Prediction is a common task which takes as an input a sequence of textual characters and outputs the next character of the sequence. This task is a rather simple task and is used to benchmark various RNN variants. With this task, we demonstrate that our method is able to distinguish across *network sizes*, in addition to initialization parameters.

(iii) Sequential MNIST is a more extensive benchmark for RNN classification accuracy. The input in the task is an image of a handwritten digit unrolled into a sequence of numerical values (pixels' greyscale values) and the output is a corresponding label of the digit. We investigate accuracy of various RNN variants on row-wise SMNIST, demonstrating that our method distinguishes according to performance across *network architectures*. We describe the outcomes of AeLLE application and resulting insights per each task below.

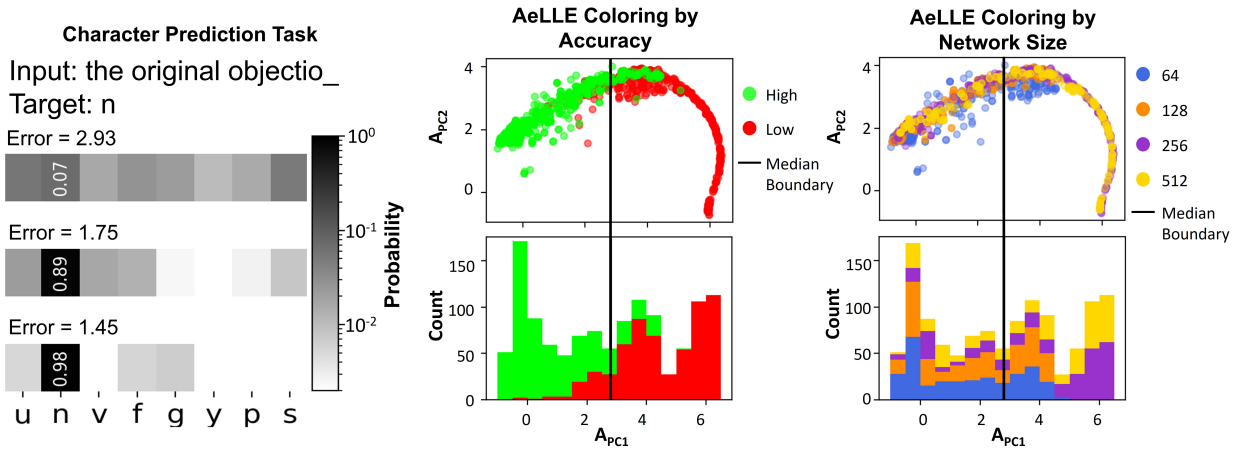

Figure 3.8: **Clustering by performance across network size: AeLLE for LSTM RNNs trained on CharRNN task.** *Left:* The CharRNN task involves predicting the next character in a sequence. Larger losses correspond to less confidence in predicting the next character in a sequence. *Center:* In the AeLLE representation the higher-accuracy networks, regardless of size, tend to cluster by performance, with the low-accuracy networks to the left in this representation. *Right:* By contrast, LSTM of different sizes are often mixed together in this representation, with larger networks covering a wider range within the AeLLE space, but still overlapping with smaller networks throughout the space.

Signal Reconstruction via Target Learning with Random RNN

To examine how AeLLE interprets generic RNN with time evolving signals as output and input, we test Rank-1 RNN. Such a model corresponds to training a single rank of the connectivity matrix, the output weights W, on the task of target learning. We set the target signal (output) to be a four-sine wave, a benchmark used in [58]. A key parameter in Rank-1 RNN is the amplification factor of the connectivity g which controls the output signal in the absence of training. For $g \leq 1$, the output signal is zero, while for $g \geq 1.8$ the output signal is strongly chaotic. In the interval $1 < g < 1.8$ the output signal is weakly chaotic. Previous work has shown that the network can generate the target signal when it is in the weakly chaotic regime, i.e., $1 < g < 1.8$ and trained with FORCE optimization algorithm [58, 50].

However, not all samples of the random connectivity correspond to accurate target generation. Even for g values in the weakly chaotic interval, there would be Rank-1 RNN variants that fail to follow the target. Thereby, the target learning task, Rank-1 RNN architecture, and FORCE optimization are ideal candidates to test whether AeLLE can organize the variants of Rank-1 RNN models according to accuracy.

The candidate hyperparameters for variation would be of 1) samples of fixed connectivity weights (from normal distribution) and 2) the parameter g within the weakly chaotic regime. We structure the benchmark set to include 1200 hyperparameter variants and compute LE spectrum for each of them. After training is complete, LE in the validation set are projected onto the Autoencoder's AeLLE space, depicted in Fig. 3.7, where each sample is a dot in the $A_{PC_1} - A_{PC_2}$ plane.

Our results show that AeLLE organizes the variants in a 2D space of $A_{PC_1} - A_{PC_2}$ according to accuracy. The variants with smaller error values (high accuracy) (< 0.57) are colored in *green* and variants with larger error values (low accuracy) (> 0.57) are colored in *red*. We demonstrate the disparity in the signals that different error values correspond to in Fig. 3.7-left. AeLLE space succeeds to correlate LE spectrum with accuracy such that most low-error networks are clustered in the bottom-left of the two-dimensional projection (see Fig. 3.7-center), whereas large-error networks are concentrated in primarily to the right and top of the region shown. This clustering of high-accuracy networks allows for the identification of multiple candidates as top performing variants in this space. Comparison of AeLLE clustering with a direct clustering according to values of g, Fig. 3.7-right, shows that while most networks with $g < 1.7$ include variants with low-error, there are also variants with high-error for each value of g. This is not the case for all variants in the low-error hyperellipse of the AeLLE space. These variants have different g and connectivity values and sampling from

the hyperellipse provides a higher probability Rank-1 RNN variant to be accurate.

For comparison, we calculated the F1 score of the classifier which uses the median first principal component value as the decision boundary to classifiers which use simple statistics of the Lyapunov spectra. When we use the median value of the Lyapunov spectrum means and maximum Lyapunov exponent as the decision boundaries, the resulting F1 scores are 0.705 and 0.504, respectively, meaning that the mean LE is much more indicative of performance than the maximum LE for this task. Additionally, we define another classifier by projecting the raw Lyapunov exponents onto their first two principal components and using the median of the first principal component as the decision boundary to get a LE PC classifier. For this task, we find the LE PC classifier achieves similar performance to the LE mean, with an F1 score of 0.703. Meanwhile, the AeLLE classifier achieves an F1 score of 0.724, indicating a significant improvement over the max LE classifier and a modest improvement over the mean LE and LE PC classifiers (see Supplementary Materials for more details).

Character Prediction with LSTM RNNs of Different Size

Multiple RNN tasks are concerned with non-time dependent signals, such as sequences of characters in a written text. Therefore, we test AeLLE on LSTM networks that perform the character prediction task (CharRNN) in which for a given sequence of characters (input) the network predicts the character (output) that follows. In particular, we train LSTM networks on English translation of Leo Tolstoy's *War and Peace*, similar to the setup described in [11]. In this setup, each unique character is assigned an index (number of unique characters in this text is 82), and the text is split into disjoint sequences of a fixed length $l = 101$, where the first $l - 1 = 100$ characters represent the input, and the final character represents the output. The loss is computed as the cross entropy loss between the expected character index and the output one.

The hyperparameters of *network size* (number of hidden states) and initialization of weight parameters appear to have most impact on the accuracy. We create 1200 variants of these parameters, varying the number of hidden units from 64 to 512 (by factors of 2) and sample initial weights from a symmetric uniform distribution with the parameter p denoting the half-width of the uniform distribution from which the initial weights are sampled in the range of $[0.04, 0.4]$. We split the variants into Autoencoder training set (80%) and validation set (20%).

Similar to the target learning task, we project the LE of the variant networks onto the first two PC dimensions of the latent space of the trained autoencoder and mark them according to accuracy. LSTM networks with loss below the median among these networks (loss <1.75) are considered as high-accuracy (*green*), while those with loss above the median are considered low-accuracy (*red*).

We find that AeLLE in 2D space separates the spectra of the variants according to accuracy across network sizes. Performing principal component analysis of the AeLLE illustrates that the low- and high-accuracy networks are separated along the PC1 dimension, with higher-accuracy networks being further to the left in these space and lower-accuracy ones clustering to the right (Fig. 3.8). For comparision, we show the median value of the first principal component across all networks (black line), showing that the vast majority of high-accuracy networks are to the left of this line. In contrast, the distribution of the network sizes (Fig. 3.8-right) is more evenly distributed in this space. This demonstrates that this method is able to learn properties from the LE spectrum which correlate with performance across network sizes which are more informative than network size alone.

Comparing the separation of in the AeLLE space to a classifier based on simple LE statistics, we find that using the median value of the mean Lyapunov Exponent or max Lyapunov Exponent as the decision boundary gives classifiers with F1 scores of .834 and .859, respectively, suggesting that both statistics are strongly indicative of performance on this task. The LE PC classifier with decision boundary defined by the median value of the first principal components of the raw Lyapunov exponents has a similar F1 score of 0.860. Meanwhile, the AeLLE classifier achieves an F1 score of .877, indicating a modest improvement on both of the direct statistics (see Supplementary Materials for more details). This shows that, while all metrics used are indicative of performance, the AeLLE method is able to achieve slightly greater discrimination of network performance.

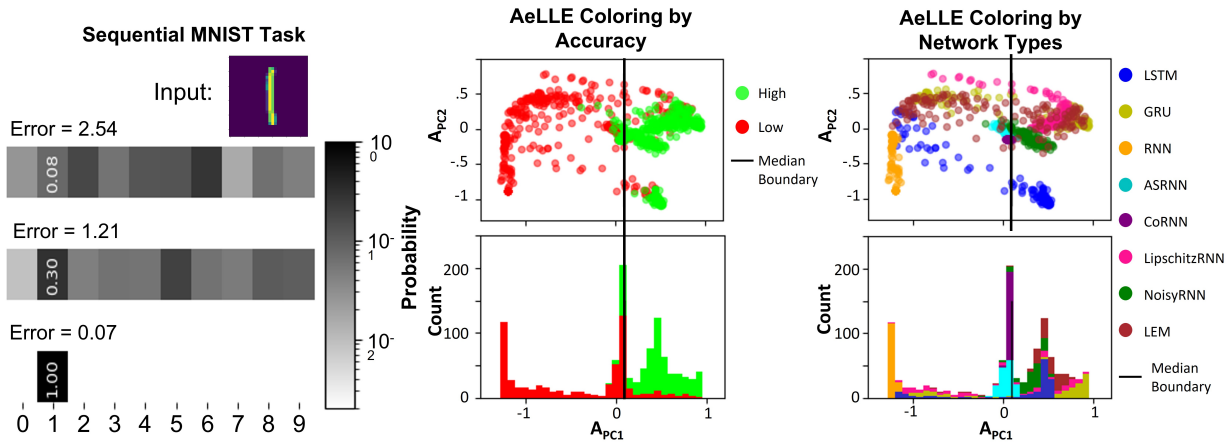

Figure 3.9: **Clustering across network architecture: AeLLE for networks trained on SMNIST Task.** *Left:* In this task, the network predicts the digit shown in a given image. Higher losses correspond to lower confidence in the correct digit label. *Center:* AeLLE representation shows that networks of the similar error are clustered together. When classifying these networks according to high accuracy (green) and low accuracy (red), the high-accuracy networks, regardless of network architecture, are consistently located further to the right (more positive PC1) than low-accuracy networks. *Right:* Networks of the same type often clustered together in AeLLE representation, with some overlap between similar architectures (such as coRNN and ASRNN). For each network architecture (except vanilla RNN), there are variants with high and low accuracy separated across the first PC dimension.

Sequential MNIST Classification with Different Network Types

A common benchmark for sequential models is the sequential MNIST task (SMNIST) task [59]. In this task, the input is a sequence of pixel greyscale values unrolled from an image of handwritten digits from $0-9$. The output is a prediction of the corresponding label (digit) written in the image. We follow the SMNIST task setup in [60], where each image is treated as sequential data and each row is the input at one time, and the number of time steps is equal to the number of columns in the image. The loss corresponds to the cross entropy between the predicted and the expected one-hot encoding of the digit.

We train a larger number of RNN variants on this task to demonstrate how the AeLLE properties translate across *network architectures*. The architectures trained on this task were: LSTM, Gated Recurrent Unit (GRU), (vanilla) RNN, Anti-Symmetric RNN (ASRNN) [34], Coupled Oscillatory RNN (coRNN) [36], Lipschitz RNN [37], Noisy RNN [38], and Long-Expressive Memory Network (LEM) [61]. For each network type, we train 200 variants of hidden size 64. Every network was trained for 10 epochs and LE of post-trained networks are collected. This constitutes a set of 1600 variants, where we use 70% for Autoencoder training, 10% for validation, and 20% for testing. For more details on the training of this Sequential MNIST task, see the Appendix A.3.

Similarly to previously described tests, we color code the variants according to accuracy. Networks with loss $< 2.2 \times 10^{-3}$ are considered as high-accuracy (*green*) which includes 50% of networks, while the rest of the networks with higher loss are considered as low-accuracy (*red*).

As in previous tests, AeLLE analysis is able to unravel variants and their accuracy according to LE spectrum. With a median value of PC1 in AeLLE plane (black line in Fig. 3.9), the AeLLE plane could be divided into two clusters where low-accuracy are in the left part of the plane and high-accuracy are in the right part. The distribution of network architectures in this representation (see Fig. 3.9-right), we find clusters of mixed architectures with similar accuracy. Moreover, we find that GRU, Lipschitz RNN, LEM, and to a lesser extent Noisy RNN, all occupy a similar part of this space, with their best-performing variants generally being located to the top-right of the space shown, and then moving to the left as the perfomance of the variants deteriorates. While no vanilla RNN variants achieve high accuracy, even networks that have variants with high accuracy, such LSTM and LEM, have low-accuracy variants that are projected onto the same space as the cluster of vanilla RNNs. Meanwhile, ASRNN and coRNN, which are both constrained to

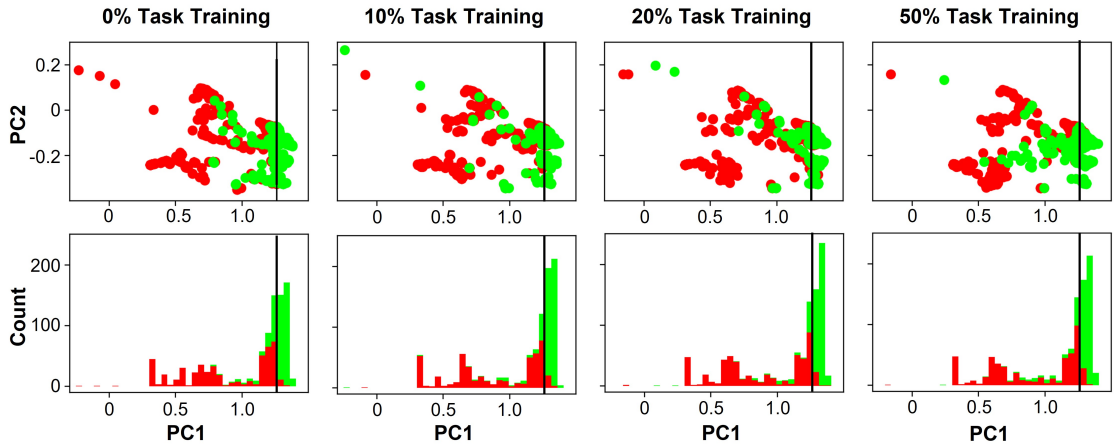

Figure 3.10: **Pre-Trained AeLLE during RNN training on SMNIST task.** Prediction of task accuracy of inputs in a test set by pre-trained AeLLE threshold classifier ($PC1 < -0.03$) of SMNIST variants is shown at $(0\%, 10\%, 20\%, 50\%)$ of training of networks on SMNIST. Throughout training, the high-accuracy networks (green) tend to cluster to the left of the Pre-trained AeLLE median classifier (black line). Furthermore, the distribution of networks in this space changes little over training, indicating that the dynamical properties which facilitate networks' successful learning of a task emerge early in training.

have dynamics that preserve information, are projected very close to each other in this space into relative small clusters near the median boundary.

The LE mean and LE max classifier on this dataset achieve F1 scores of .609 and .566, respectively, suggesting that both statistics are poor predictors of performance on their own. The LE PC classifier has an F1 score of 0.608, representing only a minor improvement in accuracy. However, the AeLLE classifier using the median value of the first principal component achieves an F1 score of 0.859 (see Supplementary Materials for more details). This score is a significant improvement on the simple LE statistics and demonstrates that AeLLE is particularly advantageous for this task, suggesting that characteristics of the dynamics shared across architectures which determine network performance are non-trivial.

Pre-Trained AeLLE for Accuracy Prediction Across Training Epoch

In the three tests described above, we find that the same general approach of AeLLE allows the selection of variants of hyperparameters of RNN associated with accuracy. LE spectrum is computed for fully trained models to set apart the sole role of hyperparameter variation. Namely, all variants in these benchmarks have been trained prior to computing LE spectrum. Over the course of training, connectivity weight parameters vary and as a result LE spectrum undergoes deformations. However, it appears that the general properties of LE spectrum such as the overall shape emerge early in training.

From these findings and the success of AeLLE, a natural question arises: how early in training can AeLLE identify networks that will perform well upon completion of training? To investigate this question we use a pre-trained AeLLE classifier, i.e., trained on a subset of variants that were fully trained for the task. We then propose to test how such pre-trained fixed AeLLE represents variants that are only partially trained, e.g., underwent $0\% - 50\%$ of training. This test is expected to show how robust are the inferred features within AeLLE correlating hyperparameters and LE spectrum subject to optimization of connectivity weights. Also it would provide insight into how long it is necessary to train the network to predict the accuracy of a hyperparameter variant.

We select the SMNIST task with LSTM, GRU, RNN, ASRNN, CoRNN, Lipschitz RNN, Noisy RNN, and LEM models with 64 hidden states size, the initialization parameter p is the same as test three (200 variants for each model).

We then compute LE spectrum for the first five epochs of the training (out of all 10 epochs) for all variants. Note, LE spectrum before training are also computed. Therefore, 6000 LE spectrum are considered. We

Training	**AeLLE** vs. [Loss]			
%	Recall		Precision	F1
0%	93.8%	[–]	77.0% [–]	0.85 [–]
10%	**91.5%**	[16.9%]	**86.6%** [95.1%]	**0.89** [0.35]
20%	**91.8%**	[45.8%]	**86.5%** [97.7%]	**0.90** [0.66]
50%	**88.6%**	[77.7%]	**83.7%** [97.3%]	**0.86** [0.86]

Table 3.1: Precision, Recall, and F1 Score of pre-trained AeLLE classifier for RNN final accuracy evaluated at different stages of training. indicated by **bold numbers**. This classification is compared with a classification based on loss value at the corresponding epoch, indicated by [·].

then select 20% variants into a training set, and they span over all epochs. We define such AeLLE as a Pre-Trained AeLLE and investigate its performance.

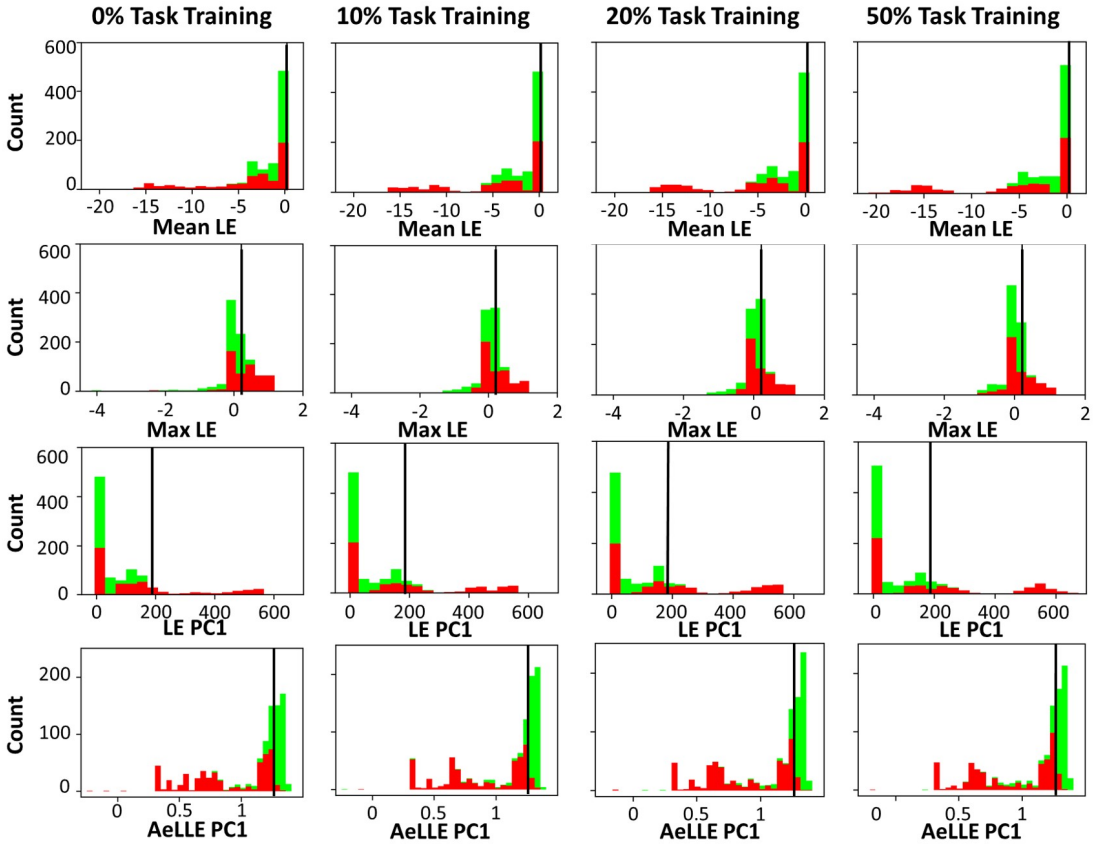

Figure 3.11: **Comparison of LE mean, LE max, and AeLLE classifiers and distributions throughout network training.** The distribution of the Mean LE (first row), Max LE (second row), and first principal component of the raw Lyapunov exponents (third row), and first principal component of the AeLLE (bottom) are shown from 0% to 50% training on the SMNIST task. The decision boundary value of each metric is shown as a vertical black bar, which is used as a classifier in Table 3.2.

We select another 20% data as the validation set (1800 variants), and apply the same loss threshold of 2.2×10^{-3} as above. We use this validation set to define a simple threshold that classifies low- and high-accuracy variants according to AeLLE (Fig. 3.10 green shaded region of $PC1 > -0.03$).

We then apply the same Pre-Trained AeLLE and accuracy threshold to variants in the test set (3600 variants) at different epochs (formulated as % of Training) illustrated in Fig. 3.10 with training progressing

from left to right and Table 3.1. We observe that projection of variants LE spectrum onto AeLLE (points in AeLLE 2D embedding) does not change significantly over the course of training. Specifically, before training, we find that the Recall, i.e., the number of the low-error networks which fall within the low-error threshold region, is 93.8%. The Precision, i.e., how many networks in the region are low-error, is 77.0%. Then after a single epoch, 10% of training, the Recall becomes 91.5% and the Precision improves to 83.6%, indicating more of the networks in the region are correctly identified as high-accuracy. The Recall and Precision does not change too much over training. This indicates that the LE spectrum captures properties of network dynamics which emerge with network initialization and remain throughout training.

In summary, we find that Pre-Trained AeLLE is an effective classifier that predicts the accuracy of the given RNN when fully trained, even before it has undergone 50% training. To further quantify the effectiveness of AeLLE classifier prediction, we compare it with a direct feature of training, the loss value at each stage of training, see Table 3.1. We find that variants with low loss early in training do correspond to variants that will be classified as low-error, indicated by almost perfect Precision rate of $99\% - 100\%$. However, it appears that many variants of high accuracy do not converge quickly. Indeed, the Recall rate for a classifier based on loss values is $17\% - 77\%$ for $10\% - 50\%$ of training, while in contrast, AeLLE Recall rate is consistently above 88% across all training epochs.

Training	Classifier	Recall	Precision	F1
0%	LE Mean	99.1%	55.6%	0.71
	LE Max	65.3%	**90.3%**	0.72
	LE PCA	**100%**	55.9%	0.76
	AeLLE	93.8%	77.0%	**0.85**
10%	LE Mean	**97.7%**	58.3%	0.73
	LE Max	60.7%	**92.6%**	0.74
	LE PCA	96.8%	60.0%	0.74
	AeLLE	91.5%	86.6%	**0.89**
20%	LE Mean	**97.6%**	58.5%	0.73
	LE Max	59.5%	**94.3%**	0.73
	LE PCA	96.6%	60.3%	0.74
	AeLLE	91.8%	86.5%	**0.90**
50%	LE Mean	**96.4%**	58.9%	0.73
	LE Max	58.8%	**94.3%**	0.72
	LE PCA	94.9%	60.4%	0.74
	AeLLE	88.6%	83.7%	**0.86**

Table 3.2: Precision, Recall, and F1 Score of LE mean, LE Max, and AeLLE classifiers throughout training. The best of each score (Precision, Recall, F1) is indicated by **bold numbers**.

For further comparison, we construct classifiers at each epoch using the LE mean and LE max, and the first principal component of the raw LEs. For each classifier, we select the value of the statistic that yields the best F1 score on the training set as the decision boundary, and then test the accuracy of such a classifier in determining whether a network's performance at the end of training will be high- or low-accuracy on the test set . The distribution of LE statistics and the first PC of the AeLLE classifier over training is shown in Figure 3.11. We see that the distribution of the Mean LE is heavily concentrated near its maximum value near 0 and does not change significantly over training. Meanwhile, the Max LE distribution changes more, but there is a similar number of high- and low-accuracy networks to either side of the boundary throughout training. The first principal component of the AeLLE has the greatest concentration of high-accuracy networks to one side of the decision boundary, suggesting this is a more useful classifier. In Table 3.2, we present the recall, precision, and F1 score of each of these classifiers. Here, we can see that the AeLLE classifier has significantly higher F1 scores than the other classifiers across all epochs.

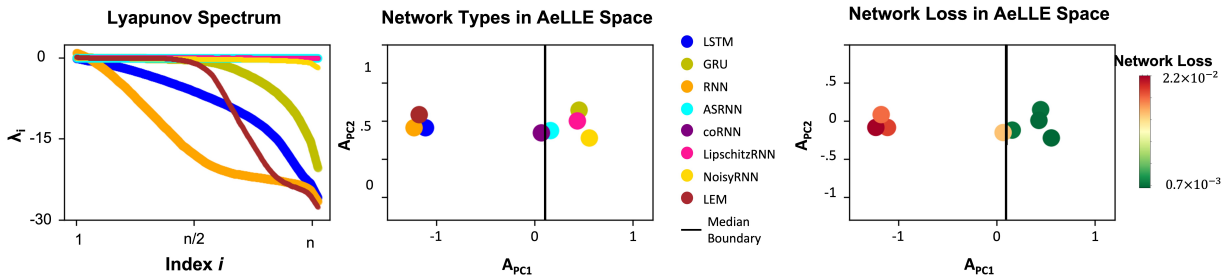

Figure 3.12: **Comparison of Lyapunov Spectrum curves, AeLLE, and loss for different network architectures on SMNIST task.** Lyapunov Spectrum curves (left) of five different architectures show that the ASRNN and coRNN have very similar spectra with many exponents close to zero. Meanwhile, the spectra of LSTM, GRU, and RNN dip well below zero, but at different rates. These networks in AeLLE space (center), are thus grouped such that ASRNN and coRNN are closer to each other than to other network types, but still are separable. LSTM, GRU, and RNN networks on the other hand are similarly close to each other in AeLLE space while having seemingly significant differences in their spectra. Such mapping appears to be correlated with accuracy (right; colors indicate loss from low-green to high-red). Despite the similar spectra of coRNN and ASRNN, the loss of coRNN is larger than that of ASRNN by a factor of 3, and indeed they appear to be separated in this space. While the LE curves of LSTM, RNN and GRU are distinct, their loss values are high (red) and they are clustered in the same vicinity farther from more optimal networks, ASRNN and coRNN. Indeed, while LSTM and RNN are very different networks, in AeLLE representation these are mapped close to each other since in this task and configuration LSTM accuracy was lower than RNN.

	#PCs			
Training %	1	2	4	10
0%	0.81	0.81	0.84	0.89
50%	0.84	0.84	0.87	0.91

Table 3.3: F1 Score of pretrained AeLLE classifier on networks at various levels of training using median from increasing numbers of principal components (PCs). The total dimension of the AeLLE space is 32.

Higher Dimensions of AeLLE

AeLLE space is not restricted to two dimensions. In general, the more dimensions are used, the representation is expected to become more accurate. To test this hypothesis we extend the classification to higher dimensions. Specifically, we use the first d PCs where the median of each PC divides the space into 2^d subspaces. In each subspace, we check the number of optimal and non-optimal network samples. Then, we take the union of all optimal subspaces (those which contain more optimal networks than sub-optimal networks) to be the overall optimal region for the classifier. We then test the classifier on the test dataset for each epoch. Our results are reported in Table 3.3, where we show the F1 score for 0% and 50% training with d being the number of PCs is set to $d = 1, 2, 4, 10$.

The results show that as additional PCs are included, F1 score increases at initial phase before training and during training, hence the full AeLLE space includes in higher dimensions additional features of LE and the corresponding networks. The results also show that the first-order PC classifier is able to capture a major portion of high- and low-accuracy networks.

LE Features Visualization in AeLLE Space

Lyapunov Spectrum for different architectures can have highly variable maximum values, slopes, variances, means, and more. Whereas certain properties of these spectra would intuitively correlate with performance, the relation between each individual feature and accuracy is unclear. This is evident from comparison of Lyapunov Spectra of LSTM, RNN, GRU, and LEM to coRNN, ASRNN, Lipschitz RNN, and Noisy RNN

trained on SMNIST in Figure 3.12(left). Whereas coRNN, ASRNN, Lipschitz RNN, and Noisy RNN all have exponents close to zero for all indices, the spectra of LSTM, GRU, RNN, and LEM all dip well below zero, with RNN decreasing much faster than the others. Meanwhile, GRU maintains a far greater number of exponents close to zero, similar to coRNN, for about half of the indices. It would then be natural to assume the performance of all networks with all LEs close to zero would be most similar and that LSTM, GRU, and GRU would all be most similar to RNN in performance.

The representation in AeLLE space (Figure 3.12 center) shows two clusters, with one tight cluster representing RNN, LSTM, and LEM and another looser cluster with all other sample networks. Within this looser cluster, we find both the coRNN network, which has a very similar spectrum to ASRNN and Lipschitz RNN, and the GRU, which is visually much closer to LSTM and LEM. However, we find that, within this cluster, GRU is located closest to Lipschitz RNN and Noisy RNN, all of which are found to the right of the median boundary along with ASRNN. Meanwhile, coRNN is located just to the left of the boundary, in the direction of the RNN, LSTM, and LEM. While visualy inspection of the spectra does not immediately indicate the reason for this ordering, it becomes more clear when we observe the loss of the networks (Figure 3.12 right). Since ASRNN, Lipschitz RNN, Noisy RNN, and GRU obtain the optimal accuracy (indicated by green color), they are mapped to the right of coRNN and the other networks point in AeLLE plane. Furthermore, while LSTM, RNN, and LEM are different networks in architectures and dynamics, these appear to be mapped to the same cluster in AeLLE. The reason for such non-intuitive mapping could be explained by accuracy again, since on this task, RNN LSTM, and LEM all exhibit low accuracy (indicated by red color).

Such experiments indicate that easily observable LE features such as number of exponents near zero or overall spectrum shape do not vary uniformly with performance. Instead, more complex, non-linear combination of LE features extracted by AeLLE would be required to determine this relation.

Discussion

LE methodology is an effective toolset to study nonlinear dynamical systems since LE measure the divergence of nearby trajectories, and thus indicate the degree of stability and chaos in that system. Indeed, LE has been applied to various dynamical systems and applications, and there exist theoretical underpinning for characterization of these systems by LE spectrum.

Our results demonstrate that the information that LE contain regarding the dynamics of a network can be related to network accuracy on various tasks. In particular, we show that AeLLE representation encodes information about the dynamics of recurrent networks (represented by their Lyapunov Exponents) along with the performance. We demonstrated that this relation to performance can be learned across choice of weight initialization, network size, network architecture, and even training epoch. Effectively, AeLLE representation discovers the implicit parameters of the network.

Such a representation is expected to be invaluable in uniting research which looks to assess and predict model quality on particular tasks and those which emphasize and constrain model dynamics to encourage particular solutions. Our approach allows mapping of dynamics of a network to accuracy through the latent space representation of LE autoencoder. This mapping appears to capture multiple characteristics of the networks, with some are direct such as network type, accuracy, number of units, and some are implicit. All these appear to be be contained in AeLLE representation and effectively provides salient features/parameters for the networks that are being considered.

Specifically, our results show that AeLLE representation is able to distinguish networks according to accuracy across choice of network architectures with greater accuracy than using simple spectrum statistics alone. Such findings suggest that the features of the dynamics which correspond to optimal performance on given tasks are consistent across network architectures, whether they are RNN, gated architectures, or dynamics-constrained architectures, even if they are not immediately apparent upon visual inspection or through first-order statistics of the spectrum. AeLLE is able to capture these. While the accuracy of such a classifier is enhanced over those using simple statistics, this comes at the cost of training time of the autoencoder and interpretabilty of extracted features. While it was not explored the scope of this work, analysis of the individual components of this autoencoder methodology (Equation (3.5)) could provide more insight into the interpretability of these features. Namely, the representations of the Lyapunov exponents

which AeLLE produces ($\phi(Z)$) could be directly analyzed statistically or otherwise. The contribution of each dimension of the latent space of the autoencoder to the predicted loss value could be extracted from the linear prediction layer (ξ). Finally, the corresponding LE spectrum features for these latent dimensions could be analyzed using the decoder (ψ). Such future explorations would give invaluable insight to the interpretation of the extracted LE features we demonstrated in this paper.

Furthermore, while the application of AeLLE for search of optimal networks given a task is outside of the scope of this work, exploration into AeLLE representation to find predicted optimal dynamics for a task would be a natural extension of the results that are reported here. Furthermore, extension of AeLLE methodology can be used to search and unravel novel architectures with desired dynamics defined by a particular LE spectrum. AeLLE approach can be also adopted to analyze other complex dynamical systems. For example, long-term forecasting of temporal signals from dynamical systems is a challenging problem that has been addressed with a similar data-driven approach using autoencoders and spectral methods along with linearization or physics-informed constraints [62, 63, 64, 65, 66]. Application of AeLLE could unify such approaches for dynamical systems representing various physical systems. The key building blocks in AeLLE that would need to be established for each of these extensions is efficient computation of Lyapunov exponents, and sufficient sampling of data to train the Autoencoder to form an informative Latent representation, and stable back-propagation of gradients across many iterations of QR decomposition in the LE calculation.

Perturbations of Latent Space Embedding

The construction of AeLLE makes predictions the networks' accuracy from the latent representation by using a simple linear layer. By starting with initial Lyapunov spectra and their corresponding network loss, we can observe how their representations in the latent space change when we perturb the predicted loss upwards or downwards. The change in the predicted loss is represented by ΔT. We use L to denote the latent representation of the Lyapunov spectrum ($\phi \circ Z$ from (3.5)).

$$\begin{aligned} \Delta T &= T' - \hat{T} \\ \Delta L &= L' - \hat{L} \\ \Rightarrow \Delta L &= \theta^{-1}(\Delta T) \end{aligned} \tag{3.7}$$

By introducing a small perturbation ΔT to predicted loss of the system, we can observe the change in the latent representation ΔL (3.7). Introducing a negative perturbation to the predicted loss corresponds to a

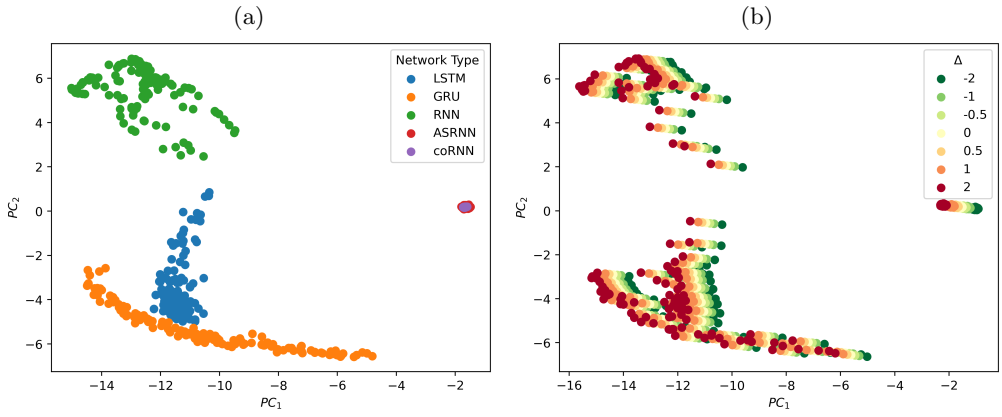

Figure 3.13: *AeLLE of networks trained on SMNIST and the results of perturbations to loss predictions.* (a) The AeLLE of five different network architectures trained on SMNIST. ASRNN and coRNN achieved the best performance on this task and clustered together to the right of this space. (b) When introducing a perturbation to the predicted loss ΔT, the latent representations move uniformly in this space along a single axis, with improving performance moving to the right.

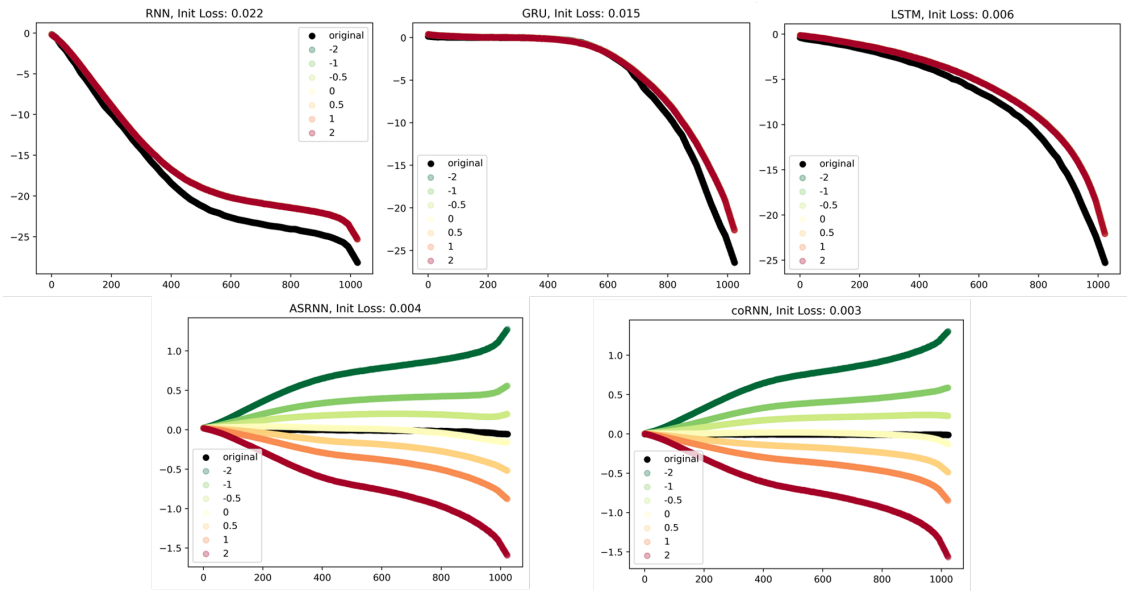

Figure 3.14: Perturbations of output Lyapunov spectra for different network types. We can see that the decoder is much more sensitive to changes for the ASRNN and coRNN networks (bottom row) and less sensitive for the other architectures (top row), for which all reconstructed spectra are overlapping.

lower error, suggesting that the resulting AeLLE would correspond to better accuracy. Positive perturbations would have the opposite effect. The original latent space for networks trained on SMNIST and the result of perturbations on these representations is shown in Figure 3.13. Note that, since θ is a simple linear function, it can only give a first-order local approximation of the direction in AeLLE space corresponding with accuracy improvements. This is particularly evident when considering the most negative values of ΔT shown in Figure 3.13(b), since the network loss was between 0 and 1 for almost all networks shown, and negative cross entropy loss is impossible. Nonetheless, we find that the direction in the AeLLE space is mostly horizontal, consistent with the clustering of the better-performing ASRNN and coRRN away from the other networks along the horizontal axis.

Moreover, we can map the perturbed latent representations back into the original LE space using the decoder of the trained autoencoder. This mapping can assess the capability of the decoder and observe the changes to the underlying spectrum introduced by these perturbations (see Figure 3.14). We can see that perturbations have a greater impact on the reconstructed spectra of ASRNN and coRNN than they do on the other network types. This may be because the Lyapunov spectra of ASRNN and coRNN are all close to zero, whereas LSTM, GRU, and RNN all have spectra which eventually dip well below zero.

We can attempt to isolate the space within the AeLLE representation for which the network predicts zero loss. To do this, we take the null space of the matrix θ. The autoencoder used for Figures 3.13 and 3.14 had a latent space of 32, and since the loss prediction output has dimension 1, the resulting dimension of the null space is 31. The projection of this space onto the AeLLE space of the network is shown alongside the AeLLE of the trained network architectures in Figure 3.15. We can see that these points are to the right of all other networks, consistent with our perturbation studies. Notably, the cluster of ASRNN and coRNN networks in this space are located very close to the points associated with zero loss, which is consistent with their high accuracy on this task. Further studies like this could reveal the predicted level sets of the loss in the AeLLE space. Using the decoder to reconstruct the spectra of these AeLLE corresponding to zero loss, a wide variety of shapes of the output spectra emerge, indicating that the decoder may struggle to reconstruct these networks with high accuracy, especially those with very low loss.

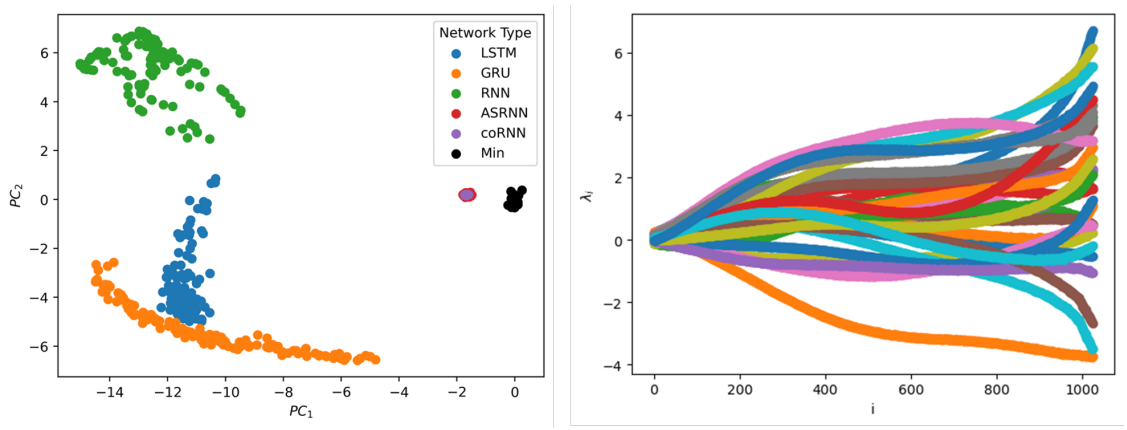

Figure 3.15: (Left) AeLLE of network architectures and the minimum loss value as predicted by the linear layer θ (black). (Right) The reconstructed LE spectra of the null space of ϕ.

Appendix

A Task Implementation and LE Computation Details

Signal Reconstruction via Target Learning

The random four-sine is a combination of four sine waves whose frequencies and amplitudes are integers randomly generated from the range $[1, 6]$ and $[1, 11]$, respectively. We train each network for 15 epochs where each epoch contains a 120s signal with time interval 0.1s, i.e., the signal length is 1200. After training the last epoch, the final recorded validation loss is a mean absolute error (MAE) between the reconstructed and target signal.

In terms of hyperparameter selection, we vary the $\boldsymbol{g}$ value in the network which determines the degree of chaos of the network as larger $\boldsymbol{g}$ means a more chaotic network. All $\boldsymbol{g}$s are selected from range $[1.1, 2.0]$ with an interval of 0.1 and we train 120 networks for each $\boldsymbol{g}$, i.e., 1200 networks in total.

We leave other hyperparameters, such as network size and learning rate constant over all experiments. We use 10 sequences of 200 time steps to calculate the LEs during testing, i.e., 10 random starting points and their following 200 time steps are used for LEs calculation. The final LEs is the average of those 10 results.

Character Prediction: CharRNN

The character sequences are randomly split into train and validation sets with an 80%/20% split. In order to ensure the independence of the trained networks, each trained network then takes a random sample of 30% of the sequences in the full training set and validation set to form its own training and validation sets.

The network is then trained for 15 epochs on the random training data and the reported validation loss is calculated from the random validation data after all training. The selection of most hyperparameters for the LSTM, including the dropout, and learning rate, and optimizer, is held constant for each instance. However, the distribution used to initialize the weight parameters of each network instance is a uniform distribution on the interval $[-p, p]$, with p sampled uniformly from the interval $[0.04, 0.40]$. Furthermore, the size of the hidden layer of the LSTM is selected from [64, 128, 256, 512], with 300 networks of each size trained on this task.

We use 10 sequences of length 100 to calculate the LEs. To treat the spectrum for each network size with the same model, we use linear interpolation to increase the dimension of each spectrum to 1024. This gives a total of 1200 spectra of LSTMs on which to train a model.

Sequential MNIST

Our model is trained with the training dataset for 10 epochs and after all training, the cross entropy loss on testing dataset is recorded as the final validation loss. We train eight different architectures, namely LSTM, GRU, RNN, ASRNN, coRNN, Lipschitz RNN, Noisy RNN and LEM. Each architecture has 200 networks with hidden sizes of 64, i.e., 1600 networks. The initial weights of the model are uniformly initialized between $-p$ and p. For LSTM, GRU, and RNN, their p is randomly generated from $[0.1, 3.0]$. For ASRNN and coRNN, their p is randomly sampled from $[0.1, 50]$ and $[0.1, 30]$, For Lipschitz RNN, Noisy RNN and LEM, their initialization are $[0.1, 20]$, $[0.1, 5.0]$, $[0.1, 10.0]$, respectively. This is because we want to have a diverse performance in each architecture. More specifically, since ASRNN and CoRNN are very robust to initialization, we can a wider range of initialization and for LSTM, RNN, the range is narrower. Other hyperparameters, such as the learning rate, and the number of layers are the same for all experiments. Lyapunov exponents are calculated on one mini-batch of the test dataset which includes 128 different input sequences. The final LEs is the average over those 128 sequences.

Task	**Network Type**	**Hidden Size(s)**	**Init Param Range**	**Total Networks**	Loss **Type**
Target Learning	Random RNN	512	$1.0 - 2.0$	1200	MAE
CharRNN	LSTM	$64, 128, 256, 512$	$0.04 - 0.40$	1200	Cross Entropy
SMNIST	RNN	64	$0.1 - 3.0$	200	Cross Entropy
	LSTM	64	$0.1 - 3.0$	200	Cross Entropy
	GRU	64	$0.1 - 3.0$	200	Cross Entropy
	coRNN	64	$0.1 - 30$	200	Cross Entropy
	ASRNN	64	$0.1 - 30$	200	Cross Entropy
	Lipschitz RNN	64	$0.1 - 20$	200	Cross Entropy
	Noisy RNN	64	$0.1 - 5$	200	Cross Entropy
	LEM	64	$0.1 - 10$	200	Cross Entropy

Table A1: Network training details for RNN tasks

B LE Statistics vs. AeLLE Classifiers

We compare the AeLLE clustering shown in the Results section to clustering and classifiers based on two first-order LE statistics: the spectrum mean and the maximum LE. In comparison to the first two principal components of the AeLLE space shown in the Results section, we plot these two statistics simultaneously with each network labeled by color according to whether the network has high (green) or low (red) accuracy.

In addition, we compare the classifier obtained by taking median value of the first principal component of the raw Lyapunov spectra. We compare the Precision, Recall, and F1 scores of all classifiers to show that AeLLE achieves the optimal F1 score among these classifiers.

Signal Reconstruction via Target Learning

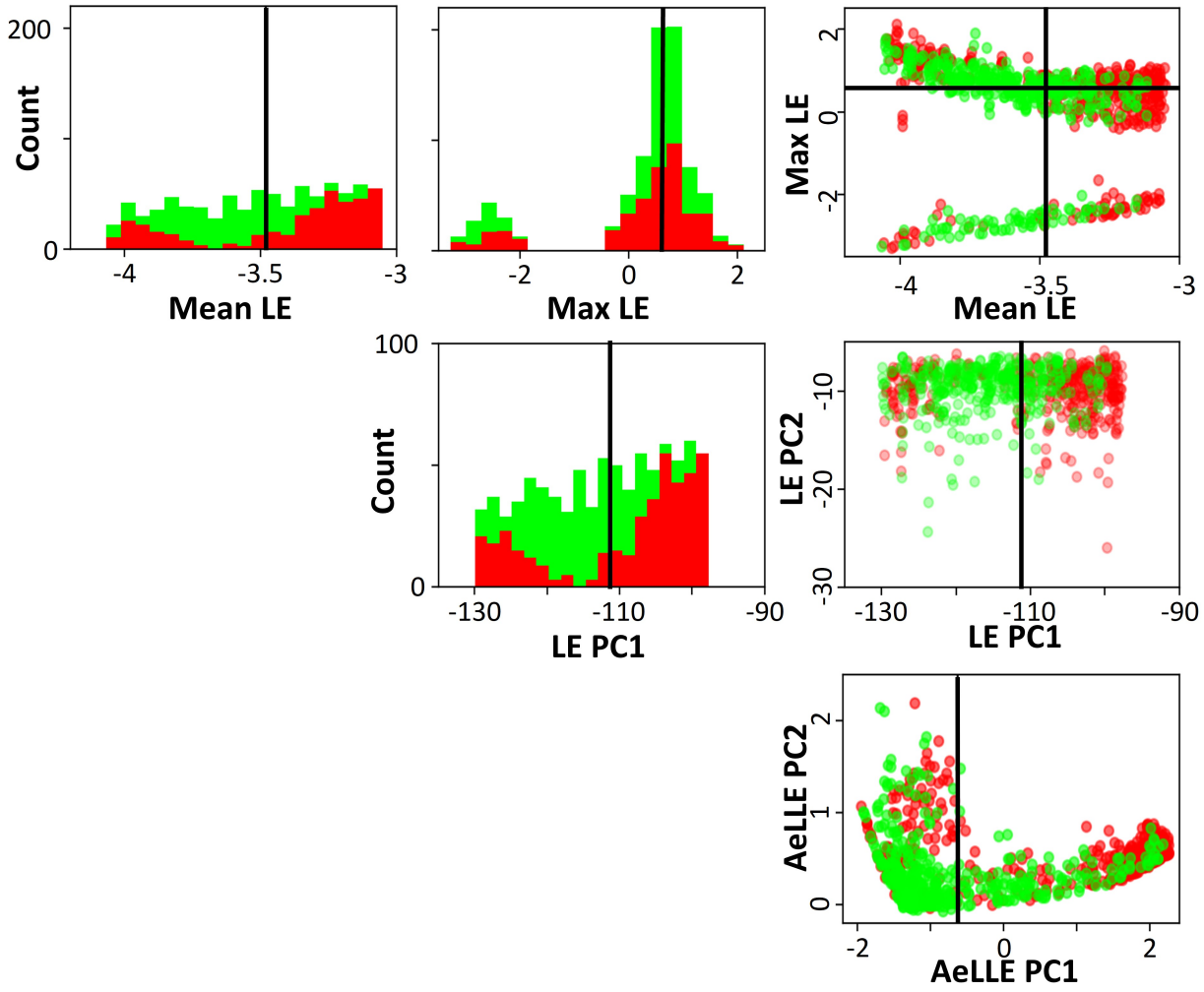

Figure B1: Distribution of mean and max LE for Signal Reconstruction task using Target Learning (top), first two principal components of the raw LE (middle), and the first two principal components of AeLLE (bottom) with colors indicating high- (green) and low-accuracy (red) networks, and black bars indicating median value of the indicated LE statistic.

Plotting the distribution of the mean LE shows that both high- and low-accuracy networks have mean LE which span the entire range of this statistic. Meanwhile, the max LE distribution is bimodal, with each mode containing a relatively even mix of both high- and low-accuracy networks. Consequently, when both

statistics are plotted together, there are two resulting clouds, but there is no clear separation between high- and low-accuracy networks (see Figure B1).

Similar to the LE mean, the first principal component of the raw LE spectra has high- and low-accuracy networks across the full range that it spans, with greater concentration of high-accuracy networks in the center. However, there is still mixing throughout. We compare this to the AeLLE projection presented in the results, which has a concentration of high-accuracy variants in the bottom left of the space shown, but still has mixing of high- and low-accuracy variants.

	Recall	Precision	F1
Mean LE	70.2%	70.9%	0.705
Max LE	50.1%	50.6%	0.504
LE PCA	70.6%	70.0%	0.703
AeLLE	**72.1%**	**72.8%**	**0.724**

Table B2: Precision, Recall, and F1 Score of AeLLE vs. LE stats classifiers for Signal Reconstruction task using Target Learning.

The precision, recall, and F1 score of the classifiers which result from using the median value of each of these statistics is shown in table B2. The maximum LE classifier achieves rather poor performance with an F1 score of 0.504, while the mean LE and LE PC classifiers achieve almost identical performance, with F1 scores of .705 and .703, respectively. Meanwhile, the AeLLE classifier F1 score represents a slight improvement on both of these classifiers, achieving an F1 score of 0.724.

Character Prediction: CharRNN

Plotting the distribution of the mean LE shows that most high-accuracy networks have a greater mean LE than low-accuracy networks for this task. Meanwhile, the max LE is generally smaller for high-accuracy networks than for low-accuracy networks. Consequently, when both statistics are plotted together, there is a greater concentration of high-accuracy networks in the quadrant corresponding to larger mean LE and smaller max LE (see Figure B2).

Similarly, the projection of the raw LEs onto their first two principal components shows a greater concentration of high-accuracy networks to the left of the space shown, with low-accuracy networks concentrated further to the left. This is very similar to the AeLLE projection presented in the Results section.

	Recall	Precision	F1
Mean LE	83.6%	83.3%	0.834
Max LE	86.0%	85.8%	0.859
LE PCA	86.0%	86.0%	0.860
AeLLE	**87.6%**	**87.7%**	**0.877**

Table B3: Precision, Recall, and F1 Score of AeLLE vs. LE stats classifiers for Character Prediction task using LSTM.

The precision, recall, and F1 score of the classifiers which result from using the median value of each of these statistics is shown in table B3. For this task, all classifiers tested achieve high accuracy. The mean LE and max LE classifiers achieve F1 scores of 0.834 and 0.859, respectively, while the LE PC classifier has a nearly identical F1 score of 0.860. Meanwhile, the AeLLE classifier achieves a slightly better accuracy, with an F1 score of 0.877.

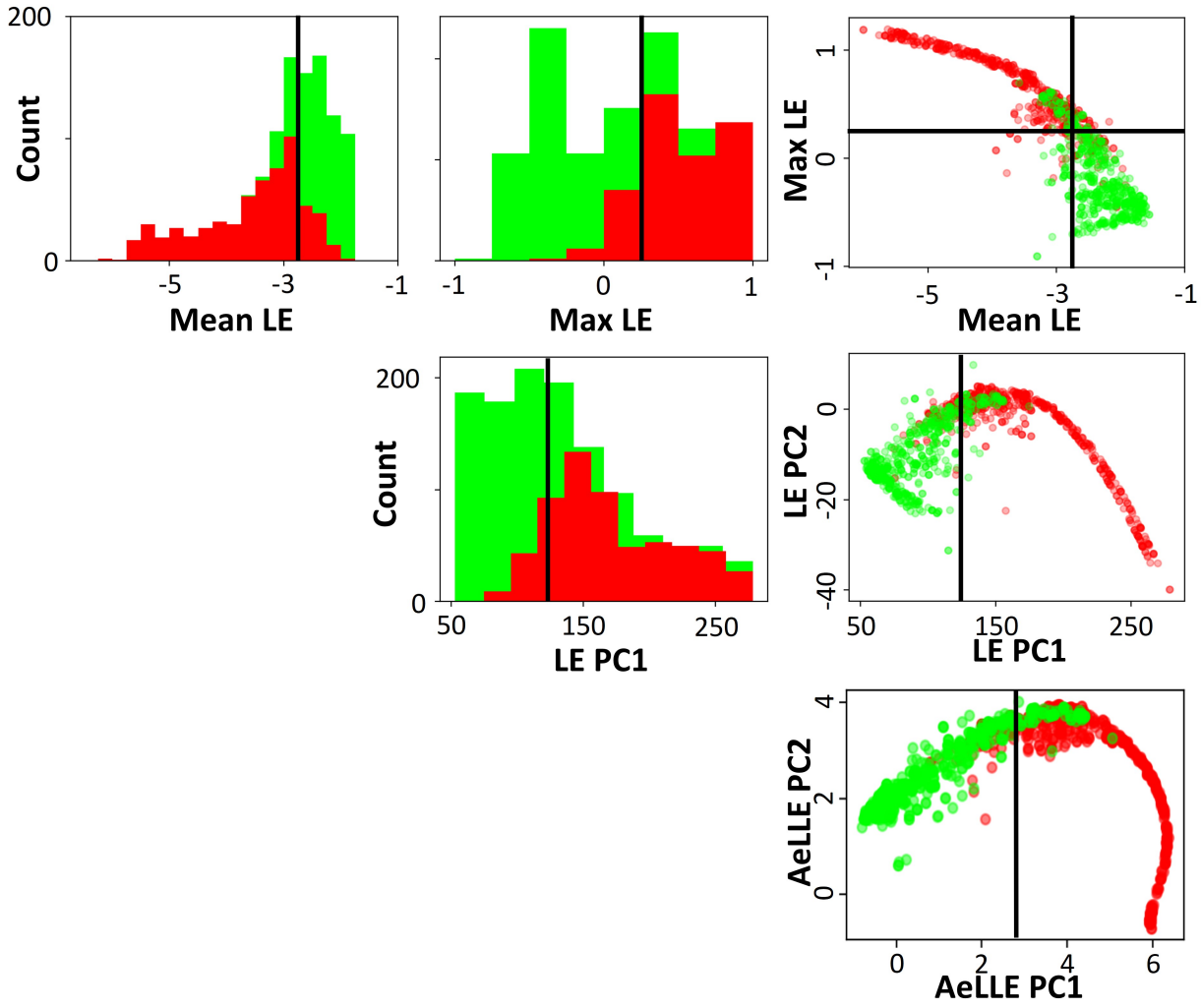

Figure B2: Distribution of mean and max LE for Character Prediction task using LSTM (top), first two principal components of the raw LE (middle), and the first two principal components of AeLLE (bottom) with colors indicating high- (green) and low-accuracy (red) networks, and black bars indicating median value of the indicated LE statistic.

Sequential MNIST

Plotting the distribution of the mean LE shows that most of the networks we trained for this task have a mean LE that is close to zero. Whereas only low-accuracy networks have mean LE values that are very low, there are both high- and low-accuracy networks with LE mean anywhere between -5 and 1. Conversely, networks with particularly a large max LE are all low-accuracy, but again the distribution is more mixed between high- and low-accuracy networks for all but the most extreme values of max LE. Consequently, when both statistics are plotted together, The low-accuracy networks span a greater range of locations, but both low- and high-accuracy networks are mixed in similar locations, particularly close to the median values of LE mean and max LE (see Figure B3).

Meanwhile, the projection of the raw LEs onto their first two principal components shows that there is a large concentration of high-accuracy networks near the median value of PC1, but that there are mixtures of similar numbers of high- and low-accuracy networks to either side of the median, even though there very large values of PC1 contain only low-accuracy networks. In contrast, the AeLLE projection shown in the Results sections displays a more significant separation between the concentrated areas of high-accuracy and

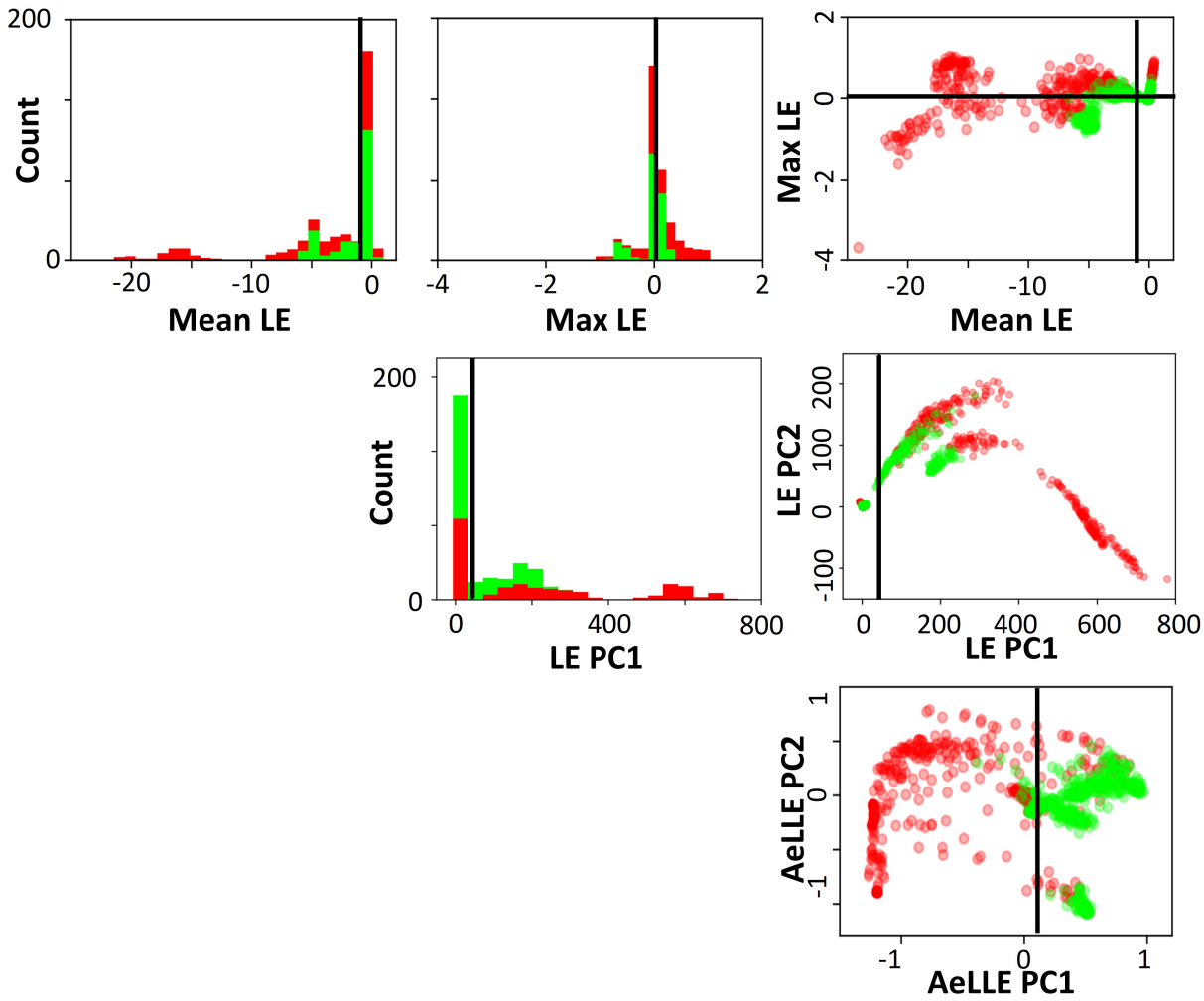

Figure B3: Distribution of mean and max LE for Sequential MNIST Classification task with Different Network Types (top), first two principal components of the raw LE (middle), and the first two principal components of AeLLE (bottom) with colors indicating high- (green) and low-accuracy (red) networks, and black bars indicating median value of the indicated LE statistic.

low-accuracy networks.

	Recall	Precision	F1
Mean LE	61.0%	60.9%	0.609
Max LE	56.6%	56.7%	0.566
LE PCA	60.8%	60.8%	0.608
AeLLE	**85.9%**	**85.9%**	**0.859**

Table B4: Precision, Recall, and F1 Score of AeLLE vs. LE stats classifiers for Sequential MNIST Classification task with Different Network Types.

The precision, recall, and F1 score of the classifiers which result from using the median value of each of these statistics is shown in table. B4. For this task, the mean LE and max LE classifiers achieve similarly

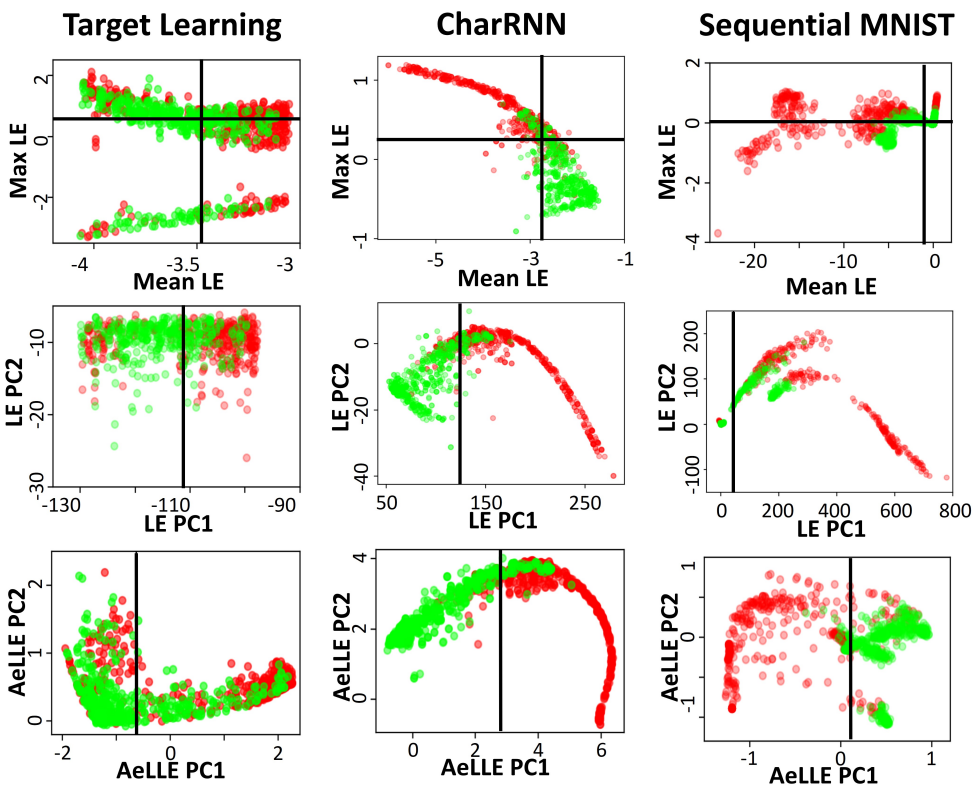

Figure B4: Projections of Lyapunov Exponents onto 2D plane using LE max and LE mean (top), raw LE PCA (middle), and AeLLE (bottom). The results are shown for Target Learning (left), CharRNN (center), and Sequential MNIST (right).

poor F1 scores of 0.609 and 0.566, respectively. The LE PC classifier achieved a similarly bad F1 score of 0.608. Meanwhile, the AeLLE classifier achieves a significantly better accuracy, with an F1 score of 0.859. This shows that the AeLLE is a much more effective performance classification method than simple LE statistics or linear projections across network architectures, suggesting that the dynamic properties which inform performance across architectures are non-trivial.

C Jacobians of RNN Architectures

In the calculation of the Lyapunov spectrum, we use the Jacobian $\frac{\partial h_t}{\partial h_{t-1}}$ (or equivalent) to perform the evolution of the tangent space at each time step. For each network, we calculate the expression for the Jacobian so we can code it directly, significantly improving the speed of calculation over numerical derivative estimation. The expressions for all non-vanilla RNN networks are shown (Note: ASRNN has the same Jacobian as RNN, but with constraints on the weights, so it is not shown either).

LSTM

$$y_* = W_* x_t + U_* h_{t-1} + b_*,$$

where $_*$ is determined by the gate/state. For LSTM, there are 4 different gates/states, (f, i, o, c), each with a corresponding y_*, W_*, U_*, b_*.
The derivatives of each of these gates/states with respect to the hidden states and the input is shown below. The final line shows what the derivative of the hidden state is relative to the hidden state at the previous time step.

$$\begin{aligned}
\frac{\partial f_t}{\partial h_{t-1}} &= \left[\sigma(y_f) \circ (1 - \sigma(y_f))\right]^T \circ U_f \\
\frac{\partial i_t}{\partial h_{t-1}} &= \left[\sigma(y_i) \circ (1 - \sigma(y_i))\right]^T \circ U_i \\
\frac{\partial o_t}{\partial h_{t-1}} &= \left[\sigma(y_o) \circ (1 - \sigma(y_o))\right]^T \circ U_o \\
\frac{\partial c_t}{\partial h_{t-1}} &= \frac{\partial f_t}{\partial h_{t-1}} \circ c_{t-1} + \frac{\partial i_t}{\partial h_{t-1}} \circ \tanh(y_c) + i_t \circ \operatorname{sech}^2(y_c) \circ U_c \\
\frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial o_t}{\partial h_{t-1}} \circ \tanh(c_t) + o_t \circ \operatorname{sech}^2(c_t) \circ \frac{\partial c_t}{\partial h_{t-1}}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial f_t}{\partial x_t} &= \left[\sigma(y_f) \circ (1 - \sigma(y_f))\right]^T \circ W_f \\
\frac{\partial i_t}{\partial x_t} &= \left[\sigma(y_i) \circ (1 - \sigma(y_i))\right]^T \circ W_i \\
\frac{\partial o_t}{\partial x_t} &= \left[\sigma(y_o) \circ (1 - \sigma(y_o))\right]^T \circ W_o \\
\frac{\partial c_t}{\partial x_t} &= \frac{\partial f_t}{\partial x_t} \circ c_{t-1} + \frac{\partial i_t}{\partial x_t} \circ \tanh(y_c) + i_t \circ \operatorname{sech}^2(y_c) \circ W_c \\
\frac{\partial h_t}{\partial x_t} &= \frac{\partial o_t}{\partial x_t} \circ \tanh(c_t) + o_t \circ \operatorname{sech}^2(c_t) \circ \frac{\partial c_t}{\partial x_t}
\end{aligned}$$

GRU

$$\begin{aligned}
y_{1*} &= W_* x + b_{i*} \\
y_{2*} &= U_* h_{t-1} + b_{h*}
\end{aligned}$$

The corresponding derivatives for the GRU are shown below.

$$\frac{\partial r_t}{\partial h_{t-1}} = [\sigma(y_{1r} + y_{2r}) \circ (1 - \sigma(y_{1r} + y2r))]^T \circ U_r$$
$$\frac{\partial z_t}{\partial h_{t-1}} = [\sigma(y_{1z} + y_{2z}) \circ (1 - \sigma(y_{1z} + y2z))]^T \circ U_z$$
$$\frac{\partial n_t}{\partial h_{t-1}} = \text{sech}^2(y_{1n} + r_t \circ y_{2n}) * \frac{\partial r_t}{\partial h_{t-1}} \circ y_{2n} + r_t \circ U_n$$
$$\frac{\partial h_t}{\partial h_{t-1}} = -\frac{\partial z_t}{\partial h_{t-1}} \circ n_t + (1 - z_t) \circ \frac{\partial n_t}{\partial h_{t-1}} + \frac{\partial z_t}{h_{t-1}} \circ h_{t-1} + z_t \circ I$$

CoRNN

$$y_t = y_{t-1} + z_t \Delta t$$
$$z_t = z_{t-1} + (tanh(W_y y_{t-1} + W_z z_{t-1} + V u_t + b) - \gamma y_{t-1} - \epsilon z_{t-1})\Delta t$$

we need to calculate the derivative of y_t w.r.t y_{t-1}.

$$\begin{aligned}
\frac{\mathrm{d}y_t}{\mathrm{d}y_{t-1}} &= \frac{\mathrm{d}(y_{t-1} + z_t \Delta t)}{\mathrm{d}y_{t-1}} \\
&= \frac{\mathrm{d}(y_{t-1} + (z_{t-1} + (tanh(W_y y_{t-1} + W_z z_{t-1} + V u_t + b) - \gamma y_{t-1} - \epsilon z_{t-1})\Delta t)\Delta t)}{\mathrm{d}y_{t-1}} \\
&= \frac{\mathrm{d}y_{t-1}}{\mathrm{d}y_{t-1}} + \Delta t \frac{\mathrm{d}z_{t-1}}{\mathrm{d}y_{t-1}} + \Delta t^2 \frac{\mathrm{d}(tanh(W_y y_{t-1} + W_z z_{t-1} + V u_t + b))}{\mathrm{d}y_{t-1}} \\
&- \Delta t^2 \frac{\mathrm{d}(\gamma y_{t-1} + \epsilon z_{t-1})}{\mathrm{d}y_{t-1}} \\
&= I + 0 + \Delta t^2 W_y^T (1 - tanh^2(W_y y_{t-1} + W_z z_{t-1} + V u_t + b)) - \Delta t^2 \gamma I \\
&= (1 - \Delta t^2 \gamma) I + \Delta t^2 W_y^T (1 - tanh^2(W_y y_{t-1} + W_z z_{t-1} + V u_t + b))
\end{aligned}$$

Lipschitz RNN

$$\begin{cases} \dot{h} = A_{\beta_A, \gamma_A} h + tanh(W_{\beta_W, \gamma_W} h + Ux + b) \\ y = Dh \end{cases}$$

where $A_{\beta,\gamma} \in \mathbb{R}^{N \times N}, W_{\beta,\gamma} \in \mathbb{R}^{N \times N}$,

$$\begin{cases} A_{\beta_A, \gamma_A} = (1 - \beta_A)(M_A + M_A^T) + \beta_A (M_A - M_A^T) - \gamma_A I \\ W_{\beta_W, \gamma_W} = (1 - \beta_W)(M_W + M_W^T) + \beta_W (M_W - M_W^T) - \gamma_W I, \end{cases}$$

where $\beta_A, \beta_W \in [0, 1], \gamma_A, \gamma_W > 0$ are tunable parameters, and $M_A, M_W \in \mathbb{R}^{N \times N}$ are trainable matrices. $\dot{h}$ is the derivative with respect to time, i.e.,$\dot{h} = \frac{\partial h(t)}{\partial t}$. Therefore the Jacobian matrix of h_t is:

$$\begin{aligned}
\frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial (h_t + \triangle t \cdot \alpha A h_t + \triangle t \cdot \tanh(W h_t + U x_t + b)}{\partial h_t} \\
&= I + \triangle t \alpha A + \triangle t W \text{sech}^2(W h_t + U x_t + b)
\end{aligned}$$

, where α is default to 1 and $\triangle t$ is the time interval.

Noisy RNN

$$\begin{aligned}\eta_{mulpi} &= k * \mathcal{N} \sim (0,1) + (1-k)\\ h_{update} &= \alpha h A + \tanh\left(h_t W + U x_t + b\right)\\ h &= h + d \cdot \eta_{mulpi} \cdot h_{update} + noise\end{aligned}$$

where, k, α are scale constant, and $\mathcal{N} \sim (0,1)$ is the normal distribution

$$\begin{cases} A = \beta(B - B^T) + (1-\beta)(B + B^T) - \gamma_A I \\ W = \beta(C - C^T) + (1-\beta)(C + C^T) - \gamma_W I \end{cases}$$

$$\begin{aligned}\frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial(h_t + d \cdot \eta_{mulpi} \cdot (\alpha h_t A + \tanh\left(h_t W + U x_t + b\right)) + noise)}{\partial h_t}\\ &= I + d \cdot \eta_{mulpi} \cdot (\alpha A + W \cdot \operatorname{sech}^2(h_t W + U x_t + b))\end{aligned}$$

LEM

$$z_n = (1 - \boldsymbol{\Delta} t_n) \odot z_{n-1} + \boldsymbol{\Delta} t_n \odot \sigma(W_z y_{n-1} + V_z u_n + b_z) \tag{3.8}$$

$$y_n = (1 - \bar{\boldsymbol{\Delta}} t_n) \odot y_{n-1} + \bar{\boldsymbol{\Delta}} t_n \odot \sigma(W_y z_n + V_y u_n + b_y) \tag{3.9}$$

, where

$$\begin{aligned}\boldsymbol{\Delta} t_n &= \triangle t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1)\\ \bar{\boldsymbol{\Delta}} t_n &= \triangle t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)\end{aligned}$$

We substitute $\bar{\boldsymbol{\Delta}} t_n$ into Eq.3.9 and get the following:

$$\begin{aligned}y_n =& (1 - \underbrace{\triangle t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)}_{\bar{\boldsymbol{\Delta}} t_n}) \odot y_{n-1} +\\ & \underbrace{\triangle t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)}_{\bar{\boldsymbol{\Delta}} t_n} \odot (W_y z_n + V_y u_n + b_y)\end{aligned}$$

Therefore, the Jacobian is:

$$\begin{aligned}\frac{\partial y_n}{\partial y_{n-1}} &= -\triangle t W_2 \hat{\sigma}'(W_2 y_{n-1} + V_2 u_n + b_2) \odot y_{n-1}\\ &+ (1 - \triangle t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)\\ &+ \triangle t W_2 \hat{\sigma}'(W_2 y_{n-1} + V_2 u_n + b_2) \odot (W_y Z_n + V_y u_n + b_y)\\ &+ \triangle t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2) \odot (W_y \frac{\partial Z_n}{\partial y_{n-1}})\end{aligned}$$

where

$$\begin{aligned}\frac{\partial Z_n}{\partial y_{n-1}} &= \frac{\partial(1 - \triangle t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1) \odot Z_{n-1})}{\partial y_{n-1}}\\ &+ \frac{\partial(\triangle t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1) \odot \sigma(W_Z y_{n-1} + V_Z u_n + b_Z)}{\partial y_{n-1}}\\ &= -\triangle t W_1 \hat{\sigma}'(W_1 y_{n-1} + V_1 u_n + b_1) \odot Z_{n-1}\\ &+ \triangle t W_1 \hat{\sigma}'(W_1 y_{n-1} + V_1 u_n + b_1) \odot \sigma(W_Z y_{n-1} + V_Z u_n + b_Z)\\ &+ \triangle t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1) \odot W_Z \sigma'(W_Z y_{n-1} + V_Z u_n + b_Z)\end{aligned}$$

D Alternative AeLLE Classifiers

We show the distribution of each network type in the left region of Fig.5 in Table. D5 column 1. We find that RNN variants are not in $R - A_{PC1}$ and this is consistent with known instabilities of vanilla RNN causing them to underperform on SMNIST. Other network types include variants of both high and low accuracies. Lipschitz and Noisy RNN have the most number of variants in R-APC1 (23.9% and 20.7%, respectively). This distribution might change depending on the classifier we use. For example, in Table. D5 row 2, as we use mean as the classifier, many more ASRNN and coRNN variants are included in this region. More sophisticated classifiers are of course possible (e.g. sloped line, polynomial curve, ellipse) and would improve the accuracy based distinction.

Table D5: Percentages of LSTM, GRU, RNN, ASRNN, coRNN, Liptschitz RNN, Noisy RNN, and LEM on the Right side of region separated by A_{PC1} median (column 2) and mean (column 3).

Distribution (%)	A_{PC1} Median	A_{PC1} Mean
LSTM	19.8	14.5
GRU	18.6	13.5
RNN	0	0
ASRNN	5.5	12.6
coRNN	0.4	17.9
Lipschitz RNN	23.9	17.2
Noisy RNN	20.7	16.1
LEM	11.1	8.2

Chapter 4

Linking Finite-Time Lyapunov Exponents to RNN Gradient Subspaces and Input Sensitivity

In the previous chapters, we have studied how Lyapunov Exponents could be related to a metric of training performance, such as the loss or accuracy. In this chapter, we explore a more direct connection between Lyapunov Exponents and the RNN training process. We expand our Lyapunov Exponents calculation to consider the Finite-Time Lyapunov Exponents and their associated vectors at each time step in a sequence. By casting the gradient of the recurrent weights of an RNN as a tensor instead of a vector, we investigate the dominant modes of the RNN gradient and compare their geometry to the FTLE vectors. Since we can order each of these sets, either by degree of expansion or contraction for FTLEs or variance captured by each direction with matrix decompositions of the gradient, we are able to compare the relation between the order of these vectors and their geometric similarity. Furthermore, we show that the task outcome of an RNN is maximally affected by input perturbations at moments where high state space expansion is taking place (as measured by FTLEs). Our results showcase deep links between computations, loss gradients, and dynamical systems stability theory for RNNs.

Introduction

Sequential inputs comes in a wide variety of forms, from natural speech or text, to electrical signals in the brain, financial markers, audio input and music, and more broadly, multivariate time series data [67, 68, 69, 70, 71]. Recurrent Neural Networks (RNN) specialize in processing such data by iteratively updating their hidden states h_{t+1} based on previous states h_t modulated by recurrent connectivity weights, and input x_t via input weights. The compounding effect of signal amplification and dampening across many RNN iterations can lead to high sensitivity in some h_t directions and very little in others, making training RNN over long sequential inputs challenging [10]. RNN form non-autonomous dynamical systems and thus, the evolution of the hidden states and memory of RNNs can be understood through the lens of dynamical systems theory and analysis. While RNN have been extensively studied, the relationship between state space dynamics — especially localized sensitivity to perturbations — and task performance remains relatively misunderstood. In this work, we explore this link and reveal new properties of RNN state space flows that inform, and can help guide, computations.

An important method for characterization of dynamical systems is *Lyapunov Exponents (LE)* [72, 73] which measure the average separation/contraction rates of infinitesimally close trajectories. Recent work identifying RNN as dynamical systems has extended LE calculation and analysis to these systems [43, 7]. Specifically of interest is the relation between Lyapunov exponent spectra and RNN performance as measured by network post-training accuracy, either by calculating the correlation between direct LE statistics and loss [7], or by training networks to capture a latent representation which separates networks according to accuracy [8]. A limitation of this approach takes root in the fact that LEs are defined as asymptotic quantities, averaged

as time approaches infinity and therefore, only capture averaged effects of space expansion and contraction. As a result, some more localized phenomena that may influence specific computations on particular input features are missed.

Here, we build on these previous works and derive connections between the gradient of the recurrent weight matrix and the finite-time intermediate values involved in LE computations over the course of a sequence. To do so, we leverage Finite Time Lyapunov Exponents (FTLE), quantities that have been initially derived to characterize fluid flows and the formation of Lagrangian Coherent Structures [74, 75, 76], time-dependent regions of state space that show high sensitivity to perturbations and act as dynamic separatrices. These methods are especially relevant in light of a recent resurgence in RNN-like methods to model long-term temporal structure called State Space Models, which have been demonstrated to exceed the capability of transformers while performing faster generation [77, 78, 79, 80]. Ongoing work translates these advances to new RNN structures which exhibit similarly impressive performance for very long sequences [81]. Moreover, RNNs are often used to model neural circuits in neuroscience systems [82], for which measuring temporal and spatial patterns of activity simultaneously is necessary to fully understand modes of behavior [83]. The lessons drawn from our work contribute to computational neuroscience by linking localized dynamic stability to task outcome.

Our contribution can be outlined as follows. We show that loss gradients in RNNs can be expressed explicitly in terms of the components extracted in FTLE calculations, and we explore the evolution of both the FTLEs and their associated vectors to analyze their correlation and influence on performance and confidence on classification tasks. We demonstrate that the vectors associated with the maximal FTLEs of an RNN become further aligned over the course of an input sequence with the dominant modes of the gradient matrix as defined by the singular vectors. As a consequence of this, we show that the step-wise expansion and contraction of FTLE subspaces during Jacobian QR decomposition steps involved in LE computations can serve as indicators of input sensitivity. We show that selection of moments of high state space expansion are indicative of instabilities such that subtle input perturbations impact network output in significant adversarial ways.

Background and Motivation

Computation of Lyapunov Exponents

Lyapunov Exponents can be computed by adopting the well-established algorithm [41, 42] and following the implementation of [43, 7], which principally relies on the QR matrix decomposition method. By definition, Lyapunov Exponents are an asymptotic quantity, but they can be estimated over very long iterates. A batch of input sequences $\mathbf{x}$ is sampled from a set of fixed-length sequences of the same distribution. For each input sequence in a batch, a matrix of Q vectors, $\mathbf{Q}$, is initialized as the identity to represent an orthogonal set of nearby initial states. The hidden states h_t are initialized as zeros.

The partial derivatives of the RNN hidden states at time t, h_t, with respect to the hidden states at time at $t-1$, h_{t-1} form the Jacobian at time step t, $\mathbf{J}_t$.

$$[\mathbf{J}_t]_{ij} = \frac{\partial \mathbf{h}_t^j}{\partial \mathbf{h}_{t-1}^i}. \tag{4.1}$$

$\mathbf{J}_t$ is calculated and then multiplied by the vectors of $\mathbf{Q}_t$ to track the expansion and the contraction of the Q vectors.

The QR decomposition of the Jacobian-Q matrix product is then used to retrieve an updated $\mathbf{Q}_{t+1}$ and expansion factors $\mathbf{R}_{t+1}$ at each time step:

$$\mathbf{Q}_{t+1}, \mathbf{R}_{t+1} = QR(\mathbf{J}_t \mathbf{Q}_t). \tag{4.2}$$

r_t^k represents the expansion factor of the k^{th} Q vector at time step t – corresponding to the k^{th} diagonal element of $\mathbf{R}_t$ in the QR decomposition. Then, the k^{th} Lyapunov Exponent λ_k of the system with an input signal of length T ($T \gg 1$) is given by

$$\lambda_k = \frac{1}{T} \sum_{t=1}^{T} \log(r_t^k). \tag{4.3}$$

Finite-Time Lyapunov Exponents

In order to adapt the above algorithm for FTLE, Eq. (4.3) is changed to:

$$\lambda_{k,t_f}^{FT} = \frac{1}{t_f} \sum_{t=1}^{t_f} \log(r_t^k) \tag{4.4}$$

Note that, in comparison to the Lyapunov Exponents (LE) in (4.3), the FTLE that we find in (4.4) are also a function of the intermediate time, t_f, as opposed to a fixed long sequence length T. This means that FTLEs are a sequence of values over time. Notably, this sequence is determined by the component parts of the QR decomposition used at each step in (4.2), $\mathbf{Q}_{t+1}$ and $\mathbf{R}_{t+1}$. We denote the columns of $\mathbf{Q}_t$ as the Q *vectors*, where the k^{th} column of $\mathbf{Q}_t$ is the vector associated with $\lambda_{k,t}^{FT}$. The FTLE calculated in Eq. (4.4) are ordered in dimension k, meaning that the first Q vector at time t corresponds to the largest FTLE, the second Q vector corresponds to the second largest, and so on. We denote the diagonal elements of $\mathbf{R}_t$ as the R *values*, where $\mathbf{R}_t^{kk}$ represents the expansion or contraction factor in the k^{th} dimension at time t, respectively.

A Covariant Basis for Lyapunov Exponents: the Covariant Lyapunov Vectors (CLVs)

At each point $\boldsymbol{h}$ of the state space, there exists a non-orthogonal basis, $\{\boldsymbol{v}_i(\boldsymbol{h})\}$, that transforms *covariantly*, i.e., component indices map across the dynamics, $\boldsymbol{v}_i(\boldsymbol{J}(\boldsymbol{h})\boldsymbol{h}) = \boldsymbol{J}(\boldsymbol{h})\boldsymbol{v}_i(\boldsymbol{h})$, where $\boldsymbol{J}(\boldsymbol{h})$ is the Jacobian at $\boldsymbol{h}$. The evolution of a perturbation aligned with $\boldsymbol{v}_i(\boldsymbol{h})$ remains in the ith subspace and scales exponentially with a rate given by the corresponding Lyapunov exponent, λ_i. The Gram-Schmidt basis, $\{\boldsymbol{q}_i\}$, computed as the columns of $\boldsymbol{Q}$ (after some burn-in time) within the FTLE algorithm are not aligned with $\{\boldsymbol{v}_i\}$ directly, but rather lie in the set of nested spans of subspaces orthogonal to the $\{\boldsymbol{v}_i\}$. That is, $\boldsymbol{q}_1 = \boldsymbol{v}_1$, $\boldsymbol{q}_2 \in \text{span}(\perp \boldsymbol{v}_1)$, $\boldsymbol{q}_3 \in \text{span}(\perp (\boldsymbol{v}_1, \boldsymbol{v}_2))$, etc. A perturbation in the $\boldsymbol{q}_i$ direction grows at a rate given by λ_i. However, since these vectors are not covariant (i.e., the dynamics (given by $\boldsymbol{J}$) do not map one Gram-Schmidt basis onto the next), there is no single component associated with λ_i along which the perturbation evolves.

The computation of the Covariant Lyapunov Vectors (CLVs) is thereby more involved. It requires a forward phase in which the $\boldsymbol{R}$ matrices are obtained, then used in a backward phase to backward map vectors initialized in the nested subspaces ($\boldsymbol{q}_1$, span($\boldsymbol{q}_1, \boldsymbol{q}_2$), ...) of the final Gram-Schmidt basis. The resulting back-transformed vectors align with the CLVs [84]. For this work, we use the Q-vectors as a finite-time approximation of the CLVs, as is done in other practical applications. Thus, we will not make further reference to CLVs in this work, but instead to their approximation in the form of the Q-vectors.

Jacobian Relation to Gradient

We present the problem of spectral constraints for robust gradient propagation, following the derivation introduced in [7]. For transparent exposition, in this section we will consider the vanilla RNN, while the derivation is applicable to more complex RNN:

$$\mathbf{o}_t = \mathbf{W}\mathbf{h}_t,\ \mathbf{h}_t = \phi(\mathbf{a}_t),\ \mathbf{a}_t = \mathbf{V}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\ , \tag{4.5}$$

where $\mathbf{V}$ is the recurrent weight matrix, $\mathbf{h}_t \in \mathbb{R}^N$ is the hidden state vector, $\mathbf{U}$ is the input weight matrix, $\mathbf{x}_t$ is the input into the network, $\mathbf{b}$ is a constant bias vector, ϕ is the non-linearity, and $\mathbf{W}$ is the output weight. The loss over T iterates is the cumulative loss over each iterate $1 \leq t \leq T$. The loss at time t is given by $L_t = f(\mathbf{y}_t, \hat{\mathbf{y}}_t)$, with f some scalar loss function, $\hat{\mathbf{y}}_t$ the prediction, and $\mathbf{y}_t$ is a target vector. The gradient of the loss in the space of recurrent weights $\mathbf{V}$, is given by

$$\nabla_{\mathbf{V}} L = \sum_{t=1}^{T} \sum_{i=1}^{N} \frac{\partial L}{\partial h_{t,i}} \nabla_{\mathbf{V}} h_{t,i} = \sum_{t=1}^{T} \text{diag}(\phi'(\mathbf{a}_t)) \nabla_{\mathbf{h}_t} L\ \mathbf{h}_{t-1}^{\top}\ , \tag{4.6}$$

Here

$$\nabla_{\mathbf{h}_t} L = \sum_{s=t}^{T} \left(\prod_{r=t+1}^{s} \mathbf{J}_r^{\top} \right) \mathbf{W}^{\top} \nabla_{\mathbf{o}_s} L\ , \tag{4.7}$$

where $\nabla_{\mathbf{o}_s} L$ is an expression depending on the loss type (*e.g.* $\hat{\mathbf{y}} - \mathbf{y}_t$ for cross-entropy loss) and $\mathbf{J}_t = \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ is the Jacobian of the hidden state dynamics,

$$\mathbf{J}_t = \text{diag}\left(\phi'(\mathbf{a}_t)\right)\mathbf{V} \ . \tag{4.8}$$

$\mathbf{J}_t$ varies in time with $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ via $\mathbf{a}_t$ and so is treated as a random matrix with ensemble properties arising from the specified input statistics and the emergent hidden state statistics.

Derivation of Gradient Matrix Representation and Link to FTLEs

In previous sections, we described how FTLEs are computed. We now present how we link FTLEs to RNN loss gradients, and ultimately to the loss.

Loss Gradient Matrix for V, ΔV

Let us consider the gradient of a function L, ∇L. We are interested in comparing the updates to the hidden-state connection weights $\mathbf{V}$. We reshape the (non-reduced) loss gradient vector with respect to the recurrent weight matrix, $\nabla_V L$, as a Tensor with the same shape as $\mathbf{V}$, such that the gradient vector has the shape $N \times H \times H$, where N is the batch size, and H is the hidden size. We will call this recast version $\Delta\mathbf{V}$. Finally, the subscript t denotes the time step at which it was calculated.

After reshaping the gradient of the recurrent weights as $\Delta\mathbf{V}$, both $\Delta\mathbf{V}$ and the Jacobian $\mathbf{J}_t$ have the same shape. Over the course of training using Stochastic Gradient Descent with learning rate η, the i^{th} update to the recurrent weight $\mathbf{V}$ takes the form

$$\mathbf{V}_{i+1} = \mathbf{V}_i - \eta \Delta \mathbf{V}_i. \tag{4.9}$$

With this construction, we observe the multiplication of the recurrent weight matrix at training iteration $i+1$, $\mathbf{V}_{i+1}$ by the hidden states, h_t can be decomposed into component parts using Eq. (4.9). This gives the resulting equation for $\mathbf{a}_t$ in Eq. (4.5) (we ignore the notation for the iteration number for the other parameters as we are most interested in $\mathbf{V}_t$):

$$\begin{aligned} \mathbf{a}_t &= \mathbf{V}_{i+1}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b} \\ &= (\mathbf{V}_i - \eta\Delta\mathbf{V}_i)\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b} \\ &= \mathbf{V}_i\mathbf{h}_{t-1} - \eta\Delta\mathbf{V}_i\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b} \end{aligned} \tag{4.10}$$

Notably, this demonstrates that this multiplication can be split into two contributions: the multiplication by the weight at the previous iteration, $\mathbf{V}_i\mathbf{h}_{t-1}$, and the multiplication by the gradient of the weights (multiplied by the learning rate), $\eta\Delta\mathbf{V}_i$. Thus, we consider each matrix in a batch of $\Delta\mathbf{V}$ to operate in the same space as $\mathbf{V}$, acting on the hidden states, $\mathbf{h}_t$.

FTLE-Gradient Relation

We derive the gradient of the RNN recurrent weight given by Eq. (4.7) in terms of the Q vectors and R values used in the calculation of FTLE. Given the algorithm for calculating FTLEs and Q-vectors requires multiplying the Q vectors by the Jacobian $\mathbf{J}$ and then performing the QR decomposition, we can write the expression for the Q and R at time step t in the following way:

$$\mathbf{Q}_t\mathbf{R}_t = \mathbf{J}_t^\top \mathbf{Q}_{t-1} \tag{4.11}$$

Eq. (4.11) can be used to solve for $\mathbf{J}_t$, allowing Eq. (4.7) to be recast by replacing the product of of Jacobians:

$$\prod_{r=t+1}^{s} \mathbf{J}_r^\top = \prod_{r=t+1}^{s} \mathbf{Q}_r\mathbf{R}_r\mathbf{Q}_{r-1}^{-1} \tag{4.12}$$

$$= \mathbf{Q}_s \left(\prod_{r=t+1}^{s} \mathbf{R}_r \right) \mathbf{Q}_t^\top \ , \tag{4.13}$$

using $Q^{-1} = Q^\top$ on the first $(r = t + 1)$ Q vectors with index $r - 1 = (t + 1) - 1 = t$. When $t = 0$, the logarithm of the diagonals of the product of $\mathbf{R}$'s is equivalent to the FTLEs up to time s before we divide by the time, s.

This gives the following expression for the full gradient with respect to the hidden state h_t:

$$\nabla_{\mathbf{h}_t} L = \sum_{s=t}^{T} \mathbf{Q}_s \left(\prod_{r=t+1}^{s} \mathbf{R}_r \right) \mathbf{Q}_t^\top \mathbf{W}^\top \nabla_{\mathbf{o}_s} L \,, \tag{4.14}$$

Experiments

In the previous section, we derived quantities that show the FTLEs and loss gradients are related in fundamental ways. We now set out to verify and exploit these links to show that state space dynamics have an impact on computations. Namely, we aim to show that the Q vectors indicate the geometry of the gradient, aligning with the dominant modes of $\Delta\mathbf{V}$, and that the R values represent the temporal contributions to the loss gradient, indicating the network's input sensitivity.

For our experiments, we consider a vanilla RNN (Eq. (4.5)) trained on the sequential MNIST task. For such a task, the MNIST dataset of handwritten numbers is fed to the RNN the image as a sequence of one or more pixels at a time, and the RNN must predict the number that was written at the end. In its standard form, the MNIST dataset images are 28×28 pixels, for a total of 784 pixels. We consider two different setups for this task.

1) Analysis of $\Delta\mathbf{V}$ and the geometric alignment with Q vectors: We calculate the cosine similarity (alignment) between singular vectors of $\Delta\mathbf{V}$ and the Q vectors and demonstrate the evolution of this alignment over the course of training and sequence index as the confidence of the network's predictions increases. We consider a smaller network (128 hidden units) trained on shorter sequences, given the significant computational cost of calculating the gradient at each step. For this experiment, we consider the row-wise SMNIST task [85], in which the network receives a full row of the image (28 pixels). Since there are 28 rows in each image, the sequence length for this setup is 28.

2) Analysis of R values and input sensitivity: We identify input locations to perturb based on times at which the greatest degree of expansion in state space occurs. We demonstrate the impact these perturbations have on network output is greater than that of perturbations at randomly-selected input locations. We use a larger RNN (512 units) trained on MNIST dataset with permuted pixel-wise input, giving an input length of 784. This size of network is necessary to achieve good performance on sequences of this length. Furthermore, by reducing the dimension of the input, we are able to isolate the individual input pixels which correspond to the R values at that point in the sequence.

Alignment of Gradient with Q Vectors

The decomposed product of the recurrent weight and hidden states shown in Eq. (4.10) shows two component parts: the original weight matrix at the previous iteration and the gradient of the weight matrix at that step. The gradient indicates the direction in parameter space that will lead to the greatest change in the local loss landscape, which we represent in matrix form as $\Delta\mathbf{V}$.

When computing the singular value decomposition (SVD) of $\Delta\mathbf{V}$, the first singular vectors capture the dominant modes of this gradient. We assume these first vectors correspond to the directions in parameter space which correspond to the greatest loss increase, with later vectors capturing less of this effect. Meanwhile, the Q vectors are also ordered at each time step to reflect the direction of decreasing expansion/increasing contraction as the index increases. In effect, for high-index Q-vectors, the rate of contraction is higher, meaning that information stored along that direction is rapidly forgotten, whereas earlier indices have a greater degree of information preservation, and even extra sensitivity if expansion is pronounced.

To test the degree of alignment between the Q vectors (q_t) and the singular vectors of $\Delta\mathbf{V}$ (e_{V_t}), we take the inner product of all pairs of vectors between each set. As both of these sets of vectors are respectively orthonormal, this inner product yields the cosine similarity, a value between -1 and 1 indicating the degree to which the vectors are parallel/aligned. 1 indicates parallel, 0 indicates orthogonal, and -1 indicates antiparallel. We store these alignment values in a matrix A_t.

$$[A_t]_{ij} = q_{ti}^{\top} \left(e_{V_t}\right)_j \tag{4.15}$$

Since the number of hidden states of the RNN under consideration is 128, the Q vectors and gradient singular vectors are each 128-dimensional. In this high-dimensional space, the probability of two randomly-selected variables being effectively orthogonal is very high, due to concentration of measure on the sphere [86]. Thus, most vectors in this high-dimensional space are expected to have alignment values close to 0.

The Q vectors are calculated for twenty five different random initializations of h_0, along with the loss and gradient associated with this initial hidden state. Eq. (4.15) is then used to find the alignment across all combinations of Q-vectors and gradient singular values.

We begin by analyzing the dominant directions of $\Delta\mathbf{V}$ measured by the first five singular vectors. To study the level of alignment with the Q-vectors, we show histograms of the alignment values between these first five singular vectors and the first ten, the last ten, and ten randomly-selected Q-vectors across all time steps in Figure 4.1.

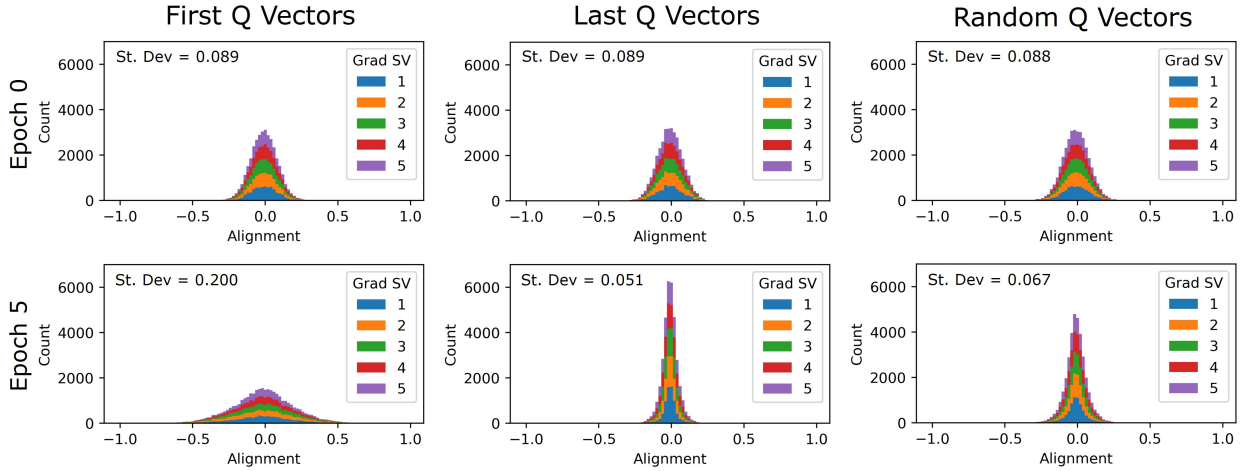

Figure 4.1: Distribution of alignment values at epoch 0 (top row) and epoch 5 (bottom row) across all time steps between the first 5 Singular Values of ΔV and three sets of ten Q-vectors: The first ten (left column), the last ten (middle column), and ten randomly selected indices not including the first or last ten (right column). The standard deviation of each cumulative distribution is indicated in the top-left of each plot.

Whereas the distributions of alignment are similar across these three sets for an untrained network, once the network has been trained for 5 epochs, the distributions of alignment between the dominant directions of $\Delta\mathbf{V}$ and each set of Q vectors differ. The first ten Q vectors, corresponding to the ten largest FTLEs, have a wider distribution, indicating a greater number of vectors aligned with the first five singular vectors of $\Delta\mathbf{V}$. As a result, the standard deviation of this distribution increased from 0.089 when untrained to 0.200 after 5 epochs.

In contrast, the distribution of the alignment between the last ten Q vectors and first five singular vectors of $\Delta\mathbf{V}$ becomes narrower, concentrating more around 0, with its standard deviation decreasing from 0.089 to 0.051. The set of randomly-selected Q vectors has a slightly narrower and taller peak around zero than it did at epoch 0, causing the standard deviation to decrease from 0.088 to 0.067.

To study how the alignment of these vectors changes depending on the Q-vector index and the step in the sequence, we show in Figure 4.2 the standard deviation of the distribution of alignment values for the first five and last five singular vectors of the gradient $\Delta\mathbf{V}$ for each Q vector index over a sequence.

As the network trains, a clear structure emerges in the standard deviations of the alignment values as a function of the Q vector index and sequence time step. As seen in Figure 4.2, before training (Epoch 0), the standard deviation of the alignment with the first and last singular vectors of the gradient at each Q-vector index fluctuates around a mean value consistent with the standard deviation for the randomly-selected Q

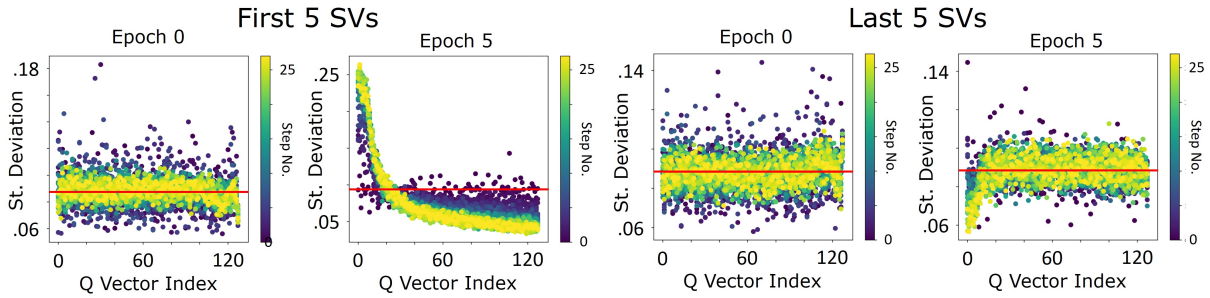

Figure 4.2: Standard deviation of alignment values as a function of Q-vector index and sequence step number before and after training. Alignment is calculated between each Q-vector and the first five singular vectors of $\Delta\mathbf{V}$ (left) as well as the last five singular values of $\Delta\mathbf{V}$ (right) for each time step in the sequence. The average standard deviation of the alignment values for randomly selected vectors is shown as a red horizontal line.

vectors (see red line). This is true both early in the input sequence (purple dots) and towards the end of the sequence (yellow dots).

However, once the network has been trained, the standard deviation of the alignment between the first five singular vectors of $\Delta\mathbf{V}$ and the first several Q-vectors is much greater than the average value for the random indices. Moreover, there is a sharp decrease in the standard deviation of the alignment as the Q-vector index increases. Both of these effects become more pronounced later in the sequence, leading to a gradual decreasing curve over Q vector index at the final step in the sequence. Notably, for Q vectors after index 40, the standard deviation at the end of the sequence decreases to well below that of the random vectors, indicating that these directions are increasingly orthogonal to the dominant directions of the gradient.

Meanwhile, the standard deviation of the alignment with last five singular values of the gradient has a similar but mirrored pattern. Once the network has been trained, the first several Q-vectors are less aligned and therefore more orthogonal to these singular vectors. Then, the alignment gradually increases for approximately the first twenty Q vectors until reaching the same average baseline that the untrained network had.

Through this analysis, we find that the basis of Q vectors reveal the directions in hidden space which are more aligned with and which are more orthogonal to the dominant modes of the gradient update. Additionally, we find that the least informative modes of the gradient have effectively random alignment with all Q vectors except the first few, with which it is more orthogonal. This shows that state-space dynamics and its sensitivity to inputs is shaped throughout training in a manner that aligns with directions that are relevant to the task, as measured by the loss' gradient. We now investigate how state-space expands or contracts along these directions, and how these transformations are related to computations.

Sensitivity Analysis of R values and Input

For this experiment, we consider the pixel-wise SMNIST task, since we want to analyze the sensitivity to individual inputs. The input size is 1 pixel, and the input sequence length is 784, with some fixed permutation of the pixels. We train an RNN with a hidden size of 512 units and initialize the weight matrices with the Xavier normal initialization. The loss used for this task is cross entropy loss.

We study the expansion and contraction factors, R values (r_t^k in Eq. (4.4)), throughout the sequence, and demonstrate that they can indicate the network's sensitivity to input. We perform this analysis by comparing the predictions of the network based on the original input, versus input which has been perturbed based on a threshold which depends on the ordering of R values.

We observe the general structure of R values of the network over a given sequence by plotting $\frac{1}{n}\sum_{i=1}^{i} r_t^i$, the mean of the first n R values as a function of sequence index, t. It can be seen in Figure 4.3 (Left) that $\log(r_t^1)$ generally fluctuates around a mean value over the course of the sequence. This mean value is what ultimately determines the value of the FTLE over that sequence. To determine the variation of R values over the sequence, we order them according to value (see Figure 4.3 (Right)), as opposed to

according to time index. For $n = 1, 10$, and 100, we compute $\frac{1}{n}\sum_i = 1^i r_t^i$, which represents the relative momentary expansion/contraction of n-dimensional volumes. We observe that the slope of the plot for $n = 1$ is consistently the most negative, indicating a greater distinction between successive indices, whereas the slope is much more flat for $n = 10$ and $n = 100$. Thus, we choose to use the first R value, r_t^1, as the metric to predict which input pixels are most sensitive to perturbations. Going forward, we denote $R_1 = r_t^1$.

Using this ordering of input indices, we select a number of pixels we wish to perturb to test the sensitivity of the network. The pixel corresponding to the index of the k largest R_1 values over a sequence are then chosen to be "flipped". If a candidate pixel is chosen to "flip", we do the following:

If the grayscale value of the pixel is non-zero, we set the grayscale value to zero (we call this a "down flip"); If the grayscale value of the pixel is already zero, we instead increase it to the maximum value of 1 (we call this an "up flip").

The full algorithm for flipping the pixels of an input image X_t according to R_1 values is shown in 3.

Algorithm 3: Pixel Flipping

Re-index t according to R_1 values : $t \to t(R_1)$;
for $i \in \{1, ..., k\}$ **do**
  if $X_{t(R_1)}[i] = 0$ **then**
    $X_{t(R_1)}[i] = 1$ (Up flip)
  else
    $X_{t(R_1)}[i] = 0$ (Down flip)
  end
end

For comparison, we also randomly select samples of k pixels from the input to flip. For each choice of k, we select 50 samples of k random pixels on which to perform flips. To determine the relative sensitivity of the network to these two choices of perturbation, we calculate the difference in the network loss between the original input and the perturbed inputs (with k flipped pixels). This is performed over 15 different original input images.

The difference in network loss (defined as the network's cross entropy loss on the perturbed input minus the loss on the original input) as a function of the number of pixels flipped, k, is shown for each sample as an individual dot in Figure 4.4. Additionally, the mean loss difference across all samples for a given k is shown as a line of the same color.

For $20 < k < 300$, the mean loss difference is greater when flipping according to R values as opposed to random pixels. We investigate the significance of the difference between these losses by performing a p-test on the hypothesis for each k that the mean of the R value loss (μ_R) is greater than mean of the

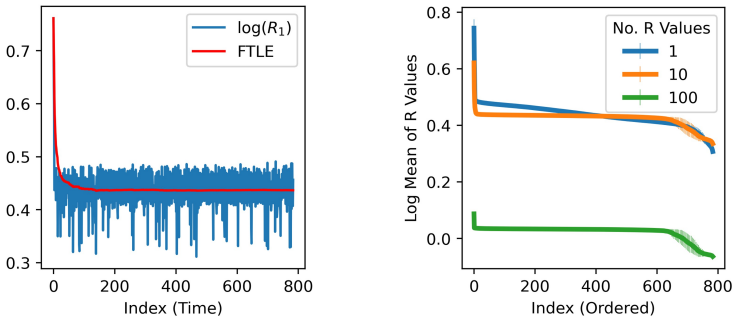

Figure 4.3: (Left) For a single input, the time-average of the logarithm of the first R value determines the first FTLE. The value of this first FTLE rapidly converges in this sequences, despite the fluctuations in the R value. (Right) The mean log of the first n R values can be ordered according to magnitude across the sequence of pixels. With this ordering, the first R shows a clear non-zero slope throughout across the sequence, whereas the mean over larger numbers shows a significant range over which the slope seems to be flat. This indicates that using the first R value provides the clearest ordering of pixels according to R values.

k	$\mu_R - \mu_{\text{rand}}$	Variance	p-value
1	-0.028	0.009	0.998
5	-0.008	0.084	0.540
10	0.025	0.303	0.468
20	0.313	0.341	0.179
30	0.260	0.359	0.234
50	0.794	0.523	0.065
75	0.814	0.475	0.043
100	1.050	0.530	**0.024**
200	1.363	0.824	0.049

Table 4.1: Statistics of loss differences between R value-flipped and randomly-flipped inputs

random-selection loss (μ_{rand}). We show the results of this test ($\mu_R - \mu_{\text{rand}} > 0$) in Table 4.1. The smallest p value is achieved for $k = 100$, at $p = 0.024$, but we find that $k = 75, 100$, and 200 pixels, we get $p < 0.05$.

To illustrate this sensitivity, we show the result of the network predictions as a result of selecting 100 pixels to flip corresponding to the locations where r_t^1 is the largest. We show in Figure 4.5 the input images after these 100 flips are performed according to the largest r_t^1. The "up flips" lead to more bright (yellow) pixels which used to be dark (purple), while the "down flips" lead to more dark pixels which used to be lighter (blue, green, or yellow). Furthermore, we show how the network predictions resulting from the original input (black), the perturbed input according to R values, and the randomly-perturbed inputs. As expected, the flipping according to the R values leads to changed predictions by the network, and to a greater extent than the average across the randomly-perturbed inputs. We demonstrate in two examples that, whether the network originally predicts the correct label and consequently has very low loss, or predicts the incorrect label (with low confidence) and thus has higher loss, flips on these 100 selected pixels can cause the network to predict a new (and incorrect) label with high confidence. Furthermore, we find the resulting confidence in this incorrect label is much greater than the mean for random perturbations, which tend to have relatively low confidence for all logits.

Discussion

In this work, we demonstrated that the stability of state space dynamics in RNNs is functionally linked to task computations and can help identify features in dynamics that are crucial for performance. Indeed, we show that increments in expansion/contraction rates that are used to estimate FTLEs, together with the orthogonal directions associated with the linearization of dynamics, are representative of directions in neural

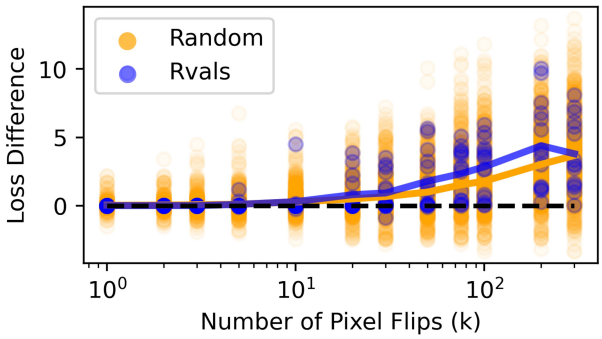

Figure 4.4: Difference in Loss Values between initial input and flipped inputs as a function of the number of "up" (blue) and "down" (orange) flips. When flipping between 20 and 200 pixels (see vertical dashed lines), the increase in loss is greater when the flips are based on R values rather than randomly-selected (see Table 4.1).

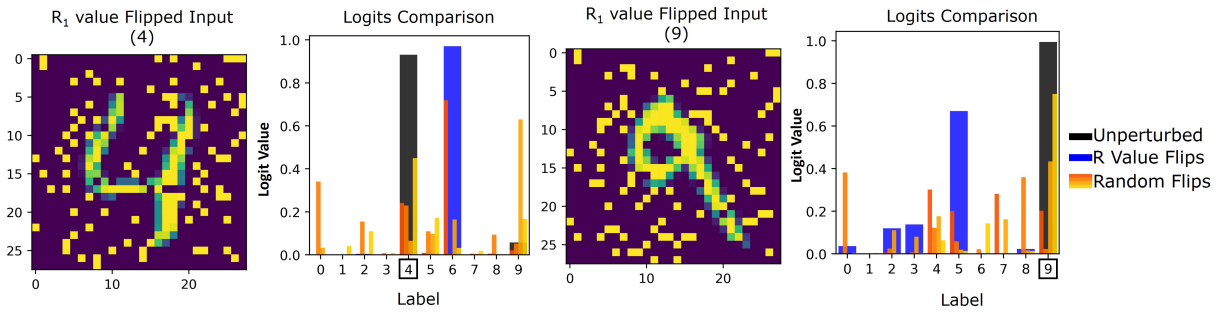

Figure 4.5: Comparison of network output logits between original input and perturbed outputs for two sample images. The perturbed input has had $k = 100$ pixels flipped from the original input image. The bar plots over 10 digit classes show network output logits for each example input for different perturbation scenarios: unperturbed input (black), the$R$ value-flipped input as shown (blue), 4 examples of 100 randomly-flipped pixels (red, red-orange, orange, yellow).

activity space for which loss gradients are highly sensitive. Our arguments rest on the fact that the Q vectors at some time t indicate the directions associated with the expansion or contraction factors given by the R values. In analytic derivations, we show that the gradient of the hidden state can be expressed explicitly using a basis of the Q vectors and R values, as shown in Eq. (4.14).

Through complementary numerical experiments, we validated that the Q vectors can capture hierarchically the geometry of the gradient. Q vectors associated with the greatest degree of expansion aligned with the dominant directions of the gradient, as measured by the singular values of $\mathbf{\Delta V}$. We further validated that rate of expansion and contraction in the state space measured by the R values leads to increased input sensitivity. This manifests in the form of perturbation effects timed at moments of heightened state space expansion. Indeed, when perturbed at moments corresponding peaks in aggregate effects of the 100 largest R values, the impact on the network's loss was significantly more pronounced than for random perturbations. Beyond loss value, such perturbations lead to reliably wrong digit classification in sequential MNIST tasks with very high confidence and measured by logit magnitude, a phenomenon unobserved when randomly perturbing.

In sum, the geometric picture provided by the Q vectors characterizes gradient propagation, and temporally localized R values link state space expansion to task uncertainty. Since the first Q vectors become increasingly aligned with the dominant gradient modes as the network confidence increases (both over training epoch and over an input sequence), it would seem that greater expansion in state space correspond to more definite network outputs. This means that changes that impact these moments have the greatest impact on the loss, showing that sensitivity in dynamics translates to a bigger impact on credit assignment. Thus, these directions can be interpreted as "ridges" along which the network's sensitivity is greatest. Such an interpretation is analogous to the ridges found in the study of Lagrangian Coherent Structures [74], but of the gradient as opposed to the state space.

Further investigations into how FTLE and related quantities can be leveraged to improve and/or analyze RNN training are warranted. The decomposition of the gradient into these components parts seems to be a promising direction for further development, and could lead to novel regularization strategies. While outside the scope of this work, regularization of the gradient to encourage greater alignment with the first few Q vectors, analysis of the per-neuron contributions to the gradient directions through the Q vectors, introducing an attention mechanism that depends on the R values, or generating adversarial learning examples based on the largest R values, could be interesting extensions.

Appendix

A Decomposition of Loss Gradient ΔV_t

In order to analyze the geometry of the loss gradient, we use the framework of the recast ΔV_t to perform matrix decompositions. With each decomposition, we are able to extract information about the orthogonal modes of the gradient, as well as the weight of each mode. While each decomposition provides different insights into the geometry of the tensor, each provides an ordering of modes that indicates the relative contribution of each mode to capturing the information contained in the matrix.

Eigendecomposition One possible decomposition of $\Delta \mathbf{V}_t$ is the the eigendecomposition.

$$\lambda_{V_t}, E_{\lambda V_t} = eig(\Delta \mathbf{V}_t)$$

The eigenvalues from this decomposition would describe the amount of variance that can be described by the associated eigenvector of the gradient. This means that the eigenvectors associated with the largest eigenvalues of ΔV are the directions which contain the greatest variance within the gradient, indicating the directions of greatest change to the recurrent weight matrix $\mathbf{V}$ needed to improve the loss in that update step.

Singular Value Decomposition When we take the singular value decomposition (SVD), we get

$$U_{V_t}, S_{V_t}, \mathbb{V}_{V_t}^T = svd(\Delta \mathbf{V}_t),$$

where

$$\Delta \mathbf{V}_t = U_{V_t} S_{V_t} \mathbb{V}_{V_t}^T$$

The columns of $\mathbf{U}$ give the orthogonal directions of the input which map to the columns of $\mathbb{V}$, which are an orthogonal basis for the output space. The degree of expansion or contraction along each direction is captured by the diagonal elements of $\mathbf{S}$, the singular values. Similar to the eigendecomposition, we associate the directions of $\mathbb{V}$ corresponding to the largest singular values of $\mathbf{S}$ with the directions of greatest change to $\mathbf{V}$ at that step to improve the loss.

For this work, we primarily use the singular value decomposition in our analysis, but other decompositions such as eigen-decomposition would be valid.

B Rank of Local-time Gradient, ΔV

Since this is a classification task, the loss would usually only be assessed at the final step of the sequence, meaning that $\nabla_{o_s} L = 0$ for all $s \neq T$. However, when we assess the gradient at the intermediate steps, we replace the total sequence length T with the current sequence length, t. Thus, we calculate what the gradient would be at each step based on the predictions after a portion is seen (at intermediate time steps). We will call these intermediate gradients ΔV_t.

To find the approximate rank of ΔV_t, we use the participation ratio, defined as the sum of squares of the singular values divided by the square of sums of the singular values:

$$P = \frac{\sum_i s_i^2}{\left(\sum_i s_i\right)^2}$$

Therefore, P ranges from $(0, 1]$, where values closer to 1 correspond to low-rank behavior dominated by a single (or small number of) singular value(s). As the value decreases, this indicates that the variance is distributed over a greater number of singular values/dimensions.

The mean participation ratio for each time step throughout two epochs of training is shown in Figure B1. The standard deviation of the participation ratio is shown as the error bars. As the network training loss converges rapidly over the first two epochs of training, we see that the shape of the participation ratio curve over sequence index also converges. Whereas the gradient of the untrained network has a participation ratio near 1 for all time steps, indicating a very low-rank matrix, the ratio later in training decreases significantly.

By the end of the second epoch of training, the average participation ratio reaches less than 0.4 for all time steps after the third, indicating significant contributions from multiple dimensions, thus implying a higher rank.

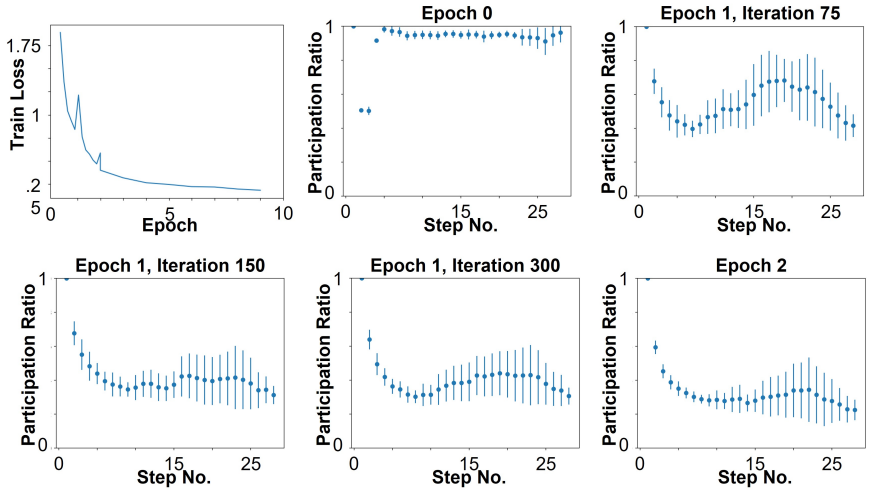

Figure B1: Participation Ratio of ΔV_t. The training loss of the network falls most rapidly in the first epoch of training (top-left). During this time, the participation ratio evolves from being close to 1 for all time steps for an untrained network (top-center) to much more varied. Once the network has been trained for 2 epochs, the general shape of the participation ratio curve over the sequence index does not change significantly.

C Distribution of Alignments for Last 5 Gradient Singular Vectors

To complement the presentation of the distribution of the alignment between the Q vectors and the first five singular vectors, we present the last five singular values of ΔV, corresponding to the least dominant directions of the gradient (See Figure C2). As before, we show the distributions for both an untrained network and a trained network.

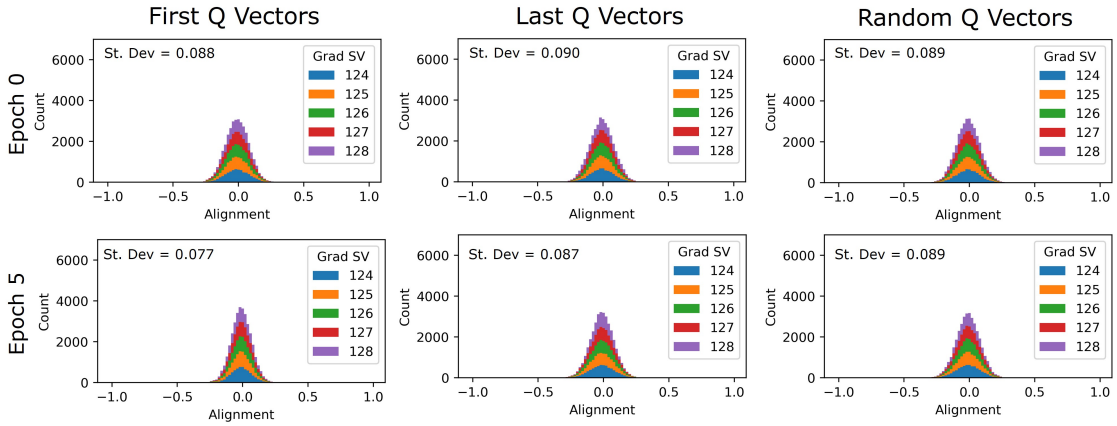

Figure C2: Distribution of alignment values at epoch 0 (top row) and epoch 5 (bottom row) across all time steps between the last 5 Singular Values of ΔV and three sets of ten Q-vectors: The first ten (left column), the last ten (middle column), and ten randomly selected indices not including the first or last ten (right column). The standard deviation of each cumulative distribution is indicated in the top-left of each plot.

As with the first five singular vectors, the distributions of the alignment between the three sets of Q-vectors and the last ten singular vectors is nearly identical at epoch 0, and they are nearly identical to the corresponding distributions for the first five singular vectors. However, after five epochs of training, only the distribution for the first ten Q-vectors changes considerably. In this case, the distribution gets narrower, indicating the vectors become less aligned with and more orthogonal to the least dominant directions of the gradient. Meanwhile, the other two sets of Q-vectors maintain approximately the same level of alignment.

D Additional Samples of Perturbed Input Predictions

We present further examples of input image which have been perturbed at locations corresponding to the largest 100 R values (see Figure D3). The top-left example, for which the correct prediction is 2 and the R-value flipped input predicts 2 with similar confidence to the original input, is a notable exception to the pattern noted across the other examples provided. For every other case, the R value-perturbed input leads to not only a different predicted label, but an incorrect label with greater confidence than any label is predicted by the randomly-flipped input.

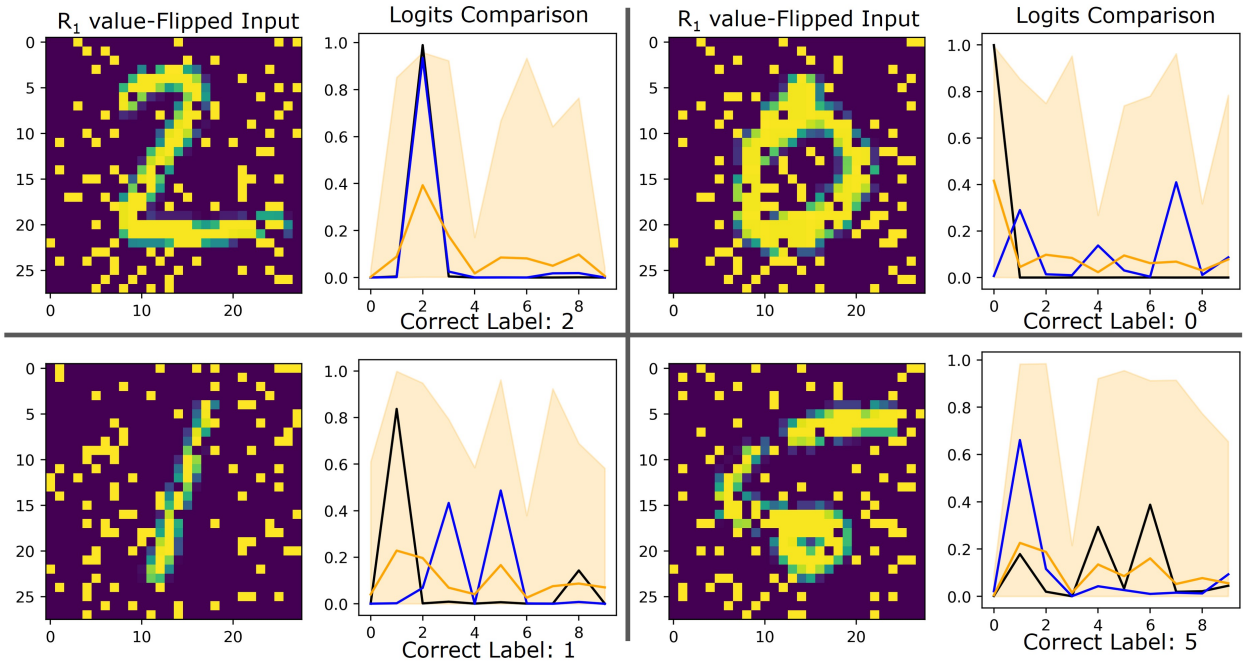

Figure D3: Additional examples of perturbed input images. For each image, we show the resulting network predictions after 100 pixels are flipped according to the largest R values (blue) or based on randomly-selected pixels (orange).

Chapter 5

Discussion and Future Work

In this work, we have investigated the application of Lyapunov exponents as a way to interpret RNN stability, accuracy, and training.

We demonstrated that the hidden states h_t of RNNs can be viewed as *non-autonomous*, discrete-time dynamical systems with parameters which evolve over the course of training. In measuring the rate of convergence or divergence of nearby trajectories, LEs serve as indicators of stability (or chaoticity) of a dynamical system [13, 14, 16]. Originally formulated for autonomous systems, we use random dynamical systems theory to extend this analysis to non-autonomous systems such as RNNs [40].

If we consider the hidden states to operate as the "memory" state of an RNN, then the LEs of RNNs can serve as indicators of the stability of information propagation and memory preservation within a system. In other words, they can indicate the degree to which an RNN preserves, forgets, or amplifies information over the course of a sequence along all directions of its state memory. With this approach, we are able to capture the cumulative dynamics of an RNN over the course of a sequence, and not the momentary impact of single inputs.

Moreover, analysis of the dynamics of the hidden state space of RNNs allows for a more direct comparison across network architectures than direct analysis of weights, activations, or gradients would. This is particularly evident for gated architectures, which have designated functions for different gates and their associated weights by construction. Consequently, comparison of gated architectures such as LSTM [87] and GRU [88] usually takes the form of accuracy or performance comparisons on specific tasks [89]. Some work has demonstrated, however, that analysis of dynamics across network architectures can reveal universality classes which are not present when observing the geometry (such as the weights and activations) of these network architectures [39]. We build on and extend this analysis by connecting network dynamics, as measured by the LEs, to the accuracy of RNNs.

As the parameters of the dynamical system evolve over the course of training, we expect to see the greatest change in the LEs when the changes in the weights are greatest. This occurs when the learning gradient has the greatest magnitude. We demonstrate in 2.2 that this intuition is accurate, as characteristics of the LE spectra such as the maximum, mean, and gross shape of the LE converge rapidly early in training. Later, we show in Table 3.3 that the features extracted by the AeLLE method can accurately predict the final accuracy of a network after only a small number of training epochs. From these results, we conclude that the dominant LE features evolve rapidly with larger-magnitude gradients and fine-tune as the loss converges for a task.

However, the previous observation leads to further questions which were not explored in this work. Namely, how does the choice of optimization procedure effect the resulting LEs of the trained network? For example, would using Adam [90] or RMSProp [91] lead to different LEs than those of Stochastic Gradient Descent, even if they converge to similar final loss? Since these different optimization methods would navigate the loss landscape in different ways, how are the paths that these methods take in the parameter space reflected in their Lyapunov Exponents, and what do those spectra tell us about other properties of the solution that each of these methods finds, such as generalization? These questions regarding the details of the impact of optimization procedure on the dynamics of the RNNs warrant additional investigation.

Investigation of the relation between further properties of training could provide additional insights.

For example, connecting the transition from the "lazy" regime (in which during training the network can be well approximated by a kernelized linear predictor) to "rich" regime (in which training encodes rich representations and induces implicit biases) [92], to the Lyapunov Exponents could help to explore the generalization of LE properties and how they indicate the richness of representations encoded by RNNs.

Regardless of the details of the training, we find that Lyapunov Exponents serve as important indicators of network accuracy. However, identifying the properties of the spectrum which are most indicative of network accuracy poses a challenge. Using natural statistics of the spectrum such as the maximum or mean Lyapunov Exponent can yield a strong correlation with the network accuracy. Comparing the correlation of loss with a large selection of LE statistics can illuminate which of those metrics correlates most strongly for a given task. This allows for a direct connection between the classical interpretation of Lyapunov Exponents and the accuracy of RNNs.

It is noteworthy to mention that this approach is limited by the selection of candidate statistics under consideration and may not capture the optimal features of the LEs for distinguishing network accuracy. Thus, we introduce a data-driven approach (AeLLE) to extract the LE features which best indicate network accuracy. The autoencoder setup we employ generates a rich representation in the latent space which allows the linear prediction layer to distinguish the accuracy of networks according only to their Lyapunov spectra. Notably, these results demonstrate this capability across a wide variety of network hyperparameters, including network initialization parameter, network size, network architecture, and training progress. Moreover, we show that the null space of the linear prediction layer can reveal the locations in the latent space corresponding to a predicted loss of zero. Though it is a compelling direction of inquiry to inverty such LEs to network architectures, it is currently unclear how to implement this. While outside the scope of this work, further investigations into this linear layer and the level sets of its loss prediction in the latent space could provide valuable insight into the structure of this representation and the optimal Lyapunov spectrum for a task.

Furthermore, even if an optimal Lyapunov spectrum could be identified using this method, using this in training would need to be resolved as well. This is challenging due to several reasons. First, because the calculation of the Lyapunov exponents involves products over long sequences, propagating error across all steps of this long sequence could be difficult, as it is for standard gradient backpropagation for RNNs. Additionally, each step of the sequence requires a QR decomposition, which can be unstable in backpropagation. To rectify this instability of QR, one could implement the algorithm outlined in [93]. Assuming the stability issues could be addressed, the computational cost of computing LEs would be prohibitive for use at ever iteration of training. However, the distance from the optimal LE spectrum could serve as a useful regularization at key points in training to encourage optimal dynamics.

There were several limitations to the scope of the AeLLE experiments. Firstly, we did not consider generalization across task. Since the output of the prediction layer of AeLLE is a single loss value for a given Lyapunov spectrum, it is unclear how this particular setup would generalize for tasks with even minor differences in loss. For example, simultaneous prediction of the loss on a regression (use mean-squared error) and classification (using cross entropy loss) tasks seems nonintuitive, as these two losses measure distances in such different spaces. Even regression or classification tasks with different numbers of output dimensions could be difficult to compare directly. Therefore, adapting this approach to generalize across task (using a single autoencoder) could require computing a proxy for network accuracy/loss which can be used across tasks.

Finally, additional investigation of the autoencoder setup is warranted. While our results demonstrate that the AeLLE method can be used to extract LE features indicative of network accuracy, we did not explore the optimality of the network architecture employed. Most notably, the size of the latent space of the autoencoder was held fixed across all experiments. Future work investigating the minimum size of this latent space to adequately distinguish between networks with similarly varied hyperparameters could reveal the dimensionality of the LE features which encode performance. Furthermore, when varying network architectures, such an investigation could reveal the degree to which these LE features (and thus the dynamics that they describe) are universal or specific to network construction.

Our derivation of the gradient of the recurrent weight of an RNN in terms of the component parts of the LE calculation (Q vectors and R values) demonstrate a direct connection between these two quantities. Our experiments investigating the consequences of this connection show that the geometry of the dominant modes of the gradient become more aligned with the directions of greatest expansion in hidden state space. Furthermore, we demonstrate that these directions of greatest expansion can serve as indicators of network

input sensitivity.

These observations were limited to a single classification task, sequential MNIST. To investigate how Q vectors and R values relate to the network gradient and input sensitivity in general, additional experiments on a wider class of learning tasks are necessary. Notably, it is unclear how this relation between R values and network sensitivity would translate to a regression task. Furthermore, as mentioned earlier in this section, the choice of optimizer would have an impact on how the recurrent weight matrix is updated. Comparing the alignment between Q vectors and the weight updates yielded by Adam and RMSProp to SGD could provide additional insight into how the dynamics of the network relate to the relative success of these optimization algorithms.

This work draws explicit connections between the accuracy of RNNs and the dynamics that they exhibit, as measured by the Lyapunov Exponents. With the findings demonstrated in this dissertation, we demonstrate how to unite the study of dynamical systems with that of RNNs in order to investigate network training and accuracy. The results and intuition developed in this work lay the foundation for further investigations and experiments which could benefit these two related fields.

Bibliography

[1] Feng Wang and David MJ Tax. "Survey on the attention based RNN model and its applications in computer vision". In: *arXiv preprint arXiv:1601.06823* (2016).

[2] Julieta Martinez, Michael J Black, and Javier Romero. "On human motion prediction using recurrent neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2891–2900.

[3] Jun Zhang and Kim-Fung Man. "Time series prediction using RNN in multi-dimension embedding phase space". In: *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*. Vol. 2. IEEE. 1998, pp. 1868–1873.

[4] Danilo Mandic and Jonathon Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley, 2001.

[5] Ya-Qi Jing, Qing-Hao Meng, Pei-Feng Qi, Meng-Li Cao, Ming Zeng, and Shu-Gen Ma. "A bioinspired neural network for data processing in an electronic nose". In: *IEEE Transactions on Instrumentation and Measurement* 65.10 (2016), pp. 2369–2380.

[6] Awni Y. Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H. Tison, Codie Bourn, Mintu P. Turakhia, and Andrew Y. Ng. "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network". In: *Nature Medicine* 25.1 (Jan. 2019), pp. 65–69. ISSN: 1546-170X. DOI: 10.1038/s41591-018-0268-3. URL: https://doi.org/10.1038/s41591-018-0268-3.

[7] Ryan Vogt, Maximilian Puelma Touzel, Eli Shlizerman, and Guillaume Lajoie. "On Lyapunov Exponents for RNNs: Understanding Information Propagation Using Dynamical Systems Tools". In: *Frontiers in Applied Mathematics and Statistics* 8 (2022). ISSN: 2297-4687. DOI: 10.3389/fams.2022.818799. URL: https://www.frontiersin.org/article/10.3389/fams.2022.818799.

[8] Ryan Vogt, Yang Zheng, and Eli Shlizerman. *Lyapunov-Guided Embedding for Hyperparameter Selection in Recurrent Neural Networks*. 2022. arXiv: 2204.04876 [cs.LG].

[9] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[10] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[11] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. "Visualizing and Understanding Recurrent Networks". In: *arXiv e-prints* (2015), arXiv–1506.

[12] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. "The Building Blocks of Interpretability". In: *Distill* (2018). https://distill.pub/2018/building-blocks. DOI: 10.23915/distill.00010.

[13] John Guckenheimer and Philip Holmes. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. Vol. 42. Springer Science & Business Media, 2013.

[14] Steven Strogatz. *Nonlinear dynamics and chaos : with applications to physics, biology, chemistry, and engineering*. eng. Second edition. Studies in nonlinearity. Boulder, CO: Westview Press, 2015. ISBN: 0813349117.

[15] G. D. Birkhoff. "What is the Ergodic Theorem?" In: *The American Mathematical Monthly* 49.4 (1942), pp. 222–226. DOI: 10.1080/00029890.1942.11991212. eprint: https://doi.org/10.1080/00029890.1942.11991212. URL: https://doi.org/10.1080/00029890.1942.11991212.

[16] J. R. Dorfman. "Lyapunov exponents, baker's map, and toral automorphisms". In: *An Introduction to Chaos in Nonequilibrium Statistical Mechanics*. Cambridge Lecture Notes in Physics. Cambridge University Press, 1999, 100–117. DOI: 10.1017/CBO9780511628870.009.

[17] J. P. Eckmann and D. Ruelle. "Ergodic theory of chaos and strange attractors". In: *Rev. Mod. Phys.* 57 (3 July 1985), pp. 617–656. DOI: 10.1103/RevModPhys.57.617. URL: https://link.aps.org/doi/10.1103/RevModPhys.57.617.

[18] Paul Frederickson, James Kaplan, Ellen Yorke, and James Yorke. "The Lyapunov dimension of strange attractors". In: *Journal of Differential Equations* 49 (Aug. 1983), pp. 185–207. DOI: 10.1016/0022-0396(83)90011-6.

[19] Nestor Norio Oiwa and Nelson Fiedler-Ferrara. "A fast algorithm for estimating Lyapunov exponents from time series". In: *Physics Letters A* 246.1-2 (1998), pp. 117–121.

[20] Michio Yamada and Koji Ohkitani. "The Inertial Subrange and Non-Positive Lyapunov Exponents in Fully-Developed Turbulence". In: *Progress of Theoretical Physics* 79.6 (June 1988), pp. 1265–1268. ISSN: 0033-068X. DOI: 10.1143/PTP.79.1265. eprint: http://oup.prod.sis.lan/ptp/article-pdf/79/6/1265/5295968/79-6-1265.pdf. URL: https://doi.org/10.1143/PTP.79.1265.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. 2013, pp. 1310–1318.

[22] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in neural information processing systems*. 2016, pp. 3360–3368.

[23] Boyu Wang and Minh Hoai. "Predicting body movement and recognizing actions: an integrated framework for mutual benefits". In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE. 2018, pp. 341–348.

[24] Minmin Chen, Jeffrey Pennington, and Samuel Schoenholz. "Dynamical isometry and a mean field theory of RNNs: Gating enables signal propagation in recurrent neural networks". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 873–882.

[25] Greg Yang. "Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation". In: *arXiv preprint arXiv:1902.04760* (2019).

[26] Yang Zheng and Eli Shlizerman. "R-FORCE: Robust Learning for Random Recurrent Neural Networks". In: *arXiv preprint arXiv:2003.11660* (2020).

[27] Charles Martin, Tongsu Peng, and Michael Mahoney. "Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data". In: *Nature Communications* 12 (July 2021), p. 4122. DOI: 10.1038/s41467-021-24025-8.

[28] Ilan Naiman and Omri Azencot. *A Koopman Approach to Understanding Sequence Neural Models*. 2021. DOI: 10.48550/ARXIV.2102.07824. URL: https://arxiv.org/abs/2102.07824.

[29] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. "Full-capacity unitary recurrent neural networks". In: *Advances in neural information processing systems* 29 (2016), pp. 4880–4888.

[30] Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. "Tunable efficient unitary neural networks (eunn) and their application to rnns". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1733–1741.

[31] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. "Efficient orthogonal parametrisation of recurrent neural networks using householder reflections". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2401–2409.

[32] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. "On orthogonality and learning recurrent networks with long term dependencies". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3570–3578.

[33] Omri Azencot, N Benjamin Erichson, Mirela Ben-Chen, and Michael W Mahoney. "A Differential Geometry Perspective on Orthogonal Recurrent Models". In: *arXiv preprint arXiv:2102.09589* (2021).

[34] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. "AntisymmetricRNN: a dynamical system view on recurrent neural networks". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=ryxepo0cFX.

[35] Giancarlo Kerg, Kyle Goyette, Maximilian Puelma Touzel, Gauthier Gidel, Eugene Vorontsov, Yoshua Bengio, and Guillaume Lajoie. "Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics". In: *Advances in neural information processing systems* 32 (2019).

[36] T. Konstantin Rusch and Siddhartha Mishra. "Coupled Oscillatory Recurrent Neural Network (coRNN): An accurate and (gradient) stable architecture for learning long time dependencies". In: *International Conference on Learning Representations*. 2021.

[37] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. "Lipschitz Recurrent Neural Networks". In: *International Conference on Learning Representations*. 2020.

[38] Soon Hoe Lim, N Benjamin Erichson, Liam Hodgkinson, and Michael W Mahoney. "Noisy recurrent neural networks". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 5124–5137.

[39] Niru Maheswaranathan, Alex Williams, Matthew Golub, Surya Ganguli, and David Sussillo. "Universality and individuality in neural dynamics across large populations of recurrent networks". In: *Advances in neural information processing systems* 32 (2019).

[40] L. Arnold. *Random Dynamical Systems*. Springer Monographs in Mathematics. Springer Berlin Heidelberg, 2013. ISBN: 9783662128787. URL: https://books.google.ca/books?id=KCbyCAAAQBAJ.

[41] Giancarlo Benettin, Luigi Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. "Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems; a method for computing all of them. Part 1: Theory". In: *Meccanica* 15.1 (1980), pp. 9–20.

[42] L Dieci and E S Van Vleck. "Computation of a few Lyapunov exponents for continuous and discrete dynamical systems". In: *Applied Numerical Mathematics* 17.3 (1995), pp. 275–291.

[43] Rainer Engelken, Fred Wolf, and Larry F Abbott. "Lyapunov spectra of chaotic recurrent neural networks". In: *arXiv preprint arXiv:2006.02427* (2020).

[44] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. "Backpropagation-friendly eigendecomposition". In: *Advances in Neural Information Processing Systems*. 2019, pp. 3162–3170.

[45] Mikael Henaff, Arthur Szlam, and Yann LeCun. "Recurrent orthogonal networks and long-memory tasks". In: *arXiv preprint arXiv:1602.06662* (2016).

[46] Tankut Can, Kamesh Krishnamurthy, and David J Schwab. "Gating creates slow modes and controls phase-space complexity in GRUs and LSTMs". In: *arXiv preprint arXiv:2002.00025* (2020).

[47] L. Molgedey, J. Schuchhardt, and H. G. Schuster. "Suppressing chaos in neural networks by noise". In: *Phys. Rev. Lett.* 69.26 (1992), pp. 3717–3719. ISSN: 00319007. DOI: 10.1103/PhysRevLett.69.3717. URL: http://link.aps.org/doi/10.1103/PhysRevLett.69.3717.

[48] Guillaume Lajoie, Kevin K. Lin, and Eric Shea-Brown. "Chaos and reliability in balanced spiking networks with temporal drive". In: *Phys. Rev. E* 87.5 (2013), pp. 1–5. ISSN: 15393755. DOI: 10.1103/PhysRevE.87.052901. URL: http://journals.aps.org/pre/abstract/10.1103/PhysRevE.87.052901.

[49] Jannis Schuecker, Sven Goedeke, and Moritz Helias. "Optimal Sequence Memory in Driven Random Networks". In: *Physical Review X* 8.4 (2018), p. 41029. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.8.041029. URL: https://doi.org/10.1103/PhysRevX.8.041029.

[50] Brian DePasquale, Christopher J Cueva, Kanaka Rajan, G Sean Escola, and LF Abbott. "full-FORCE: A target-based method for training recurrent networks". In: *PloS one* 13.2 (2018), e0191527.

[51] L.H. Miranda Filho, M.A. Amato, Y. Elskens, and T.M. Rocha Filho. "Contribution of individual degrees of freedom to Lyapunov vectors in many-body systems". In: *Communications in Nonlinear Science and Numerical Simulation* 74 (2019), pp. 236–247. ISSN: 1007-5704. DOI: https://doi.org/10.1016/j.cnsns.2019.03.011. URL: https://www.sciencedirect.com/science/article/pii/S1007570419300796.

[52] Federico Bizzarri, Angelo Brambilla, and Giancarlo Storti Gajani. "Lyapunov exponents computation for hybrid neurons". In: *Journal of computational neuroscience* 35 (2013), pp. 201–212.

[53] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[54] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2.11 (1901), pp. 559–572.

[55] Kun Su and Eli Shlizerman. "Clustering and Recognition of Spatiotemporal Features through Interpretable Embedding of Sequence to Sequence Recurrent Neural Networks". In: *Frontiers in artificial intelligence* 3 (2020), p. 70.

[56] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[57] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. "UMAP: Uniform Manifold Approximation and Projection". In: *Journal of Open Source Software* 3.29 (2018), p. 861.

[58] David Sussillo and Larry F Abbott. "Generating coherent patterns of activity from chaotic neural networks". In: *Neuron* 63.4 (2009), pp. 544–557.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[60] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: *arXiv e-prints* (2018), arXiv–1803.

[61] T. Konstantin Rusch, Siddhartha Mishra, N. Benjamin Erichson, and Michael W. Mahoney. *Long Expressive Memory for Sequence Modeling*. 2022. arXiv: 2110.04744 [cs.LG].

[62] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. "Deep learning for universal linear embeddings of nonlinear dynamics". In: *Nature Communications* 9.1 (Nov. 2018), p. 4950. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07210-0. URL: https://doi.org/10.1038/s41467-018-07210-0.

[63] Henning Lange, Steven L Brunton, and J Nathan Kutz. "From Fourier to Koopman: Spectral Methods for Long-term Time Series Prediction." In: *J. Mach. Learn. Res.* 22.41 (2021), pp. 1–38.

[64] Jeremy Morton, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden. "Deep dynamical modeling and control of unsteady fluid flows". In: *Advances in Neural Information Processing Systems* 31 (2018).

[65] N. Benjamin Erichson, Michael Muehlebach, and Michael W. Mahoney. *Physics-informed Autoencoders for Lyapunov-stable Fluid Flow Prediction*. 2019. DOI: 10.48550/ARXIV.1905.10866. URL: https://arxiv.org/abs/1905.10866.

[66] Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael Mahoney. "Forecasting sequential data using consistent Koopman autoencoders". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 475–485.

[67] Bo Pang, Kaiwen Zha, Hanwen Cao, Chen Shi, and Cewu Lu. "Deep RNN Framework for Visual Sequential Applications". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[68] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. "Recurrent neural network based language model". In: *Eleventh annual conference of the international speech communication association*. 2010.

[69] S. Das and O. Olurotimi. "Noisy recurrent neural networks: the continuous-time case". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 913–936. DOI: 10.1109/72.712164.

[70] Peter Tino, Christian Schittenkopf, and Georg Dorffner. "Financial volatility trading using recurrent neural networks". In: *IEEE transactions on neural networks* 12.4 (2001), pp. 865–874.

[71] Kun Su, Xiulong Liu, and Eli Shlizerman. "Predict & cluster: Unsupervised skeleton based action recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9631–9640.

[72] David Ruelle. "Ergodic theory of differentiable dynamical systems". In: *Publications Mathématiques de l'Institut des Hautes Études Scientifiques* 50.1 (1979), pp. 27–58.

[73] V. Oseledets. "Oseledets theorem". In: *Scholarpedia* 3.1 (2008). revision #142085, p. 1846. DOI: 10.4249/scholarpedia.1846.

[74] Shawn C. Shadden, Francois Lekien, and Jerrold E. Marsden. "Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows". In: *Physica D: Nonlinear Phenomena* 212.3 (2005), pp. 271–304. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2005.10.007. URL: https://www.sciencedirect.com/science/article/pii/S0167278905004446.

[75] George Haller. "Lagrangian Coherent Structures". In: *Annual Review of Fluid Mechanics* 47.1 (2015), pp. 137–162. DOI: 10.1146/annurev-fluid-010313-141322. eprint: https://doi.org/10.1146/annurev-fluid-010313-141322. URL: https://doi.org/10.1146/annurev-fluid-010313-141322.

[76] M. A. GREEN, C. W. ROWLEY, and G. HALLER. "Detection of Lagrangian coherent structures in three-dimensional turbulence". In: *Journal of Fluid Mechanics* 572 (2007), 111–120. DOI: 10.1017/S0022112006003648.

[77] Albert Gu, Karan Goel, and Christopher Re. "Efficiently Modeling Long Sequences with Structured State Spaces". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=uYLFoz1vlAC.

[78] Ankit Gupta, Albert Gu, and Jonathan Berant. "Diagonal State Spaces are as Effective as Structured State Spaces". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 22982–22994. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9156b0f6dfa9bbd18c79cc459ef5d61c-Paper-Conference.pdf.

[79] Ankit Gupta, Harsh Mehta, and Jonathan Berant. "Simplifying and Understanding State Space Models with Diagonal Linear RNNs". In: *arXiv preprint arXiv:2212.00768* (2022).

[80] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. "Simplified State Space Layers for Sequence Modeling". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=Ai8Hw3AXqks.

[81] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. "Resurrecting Recurrent Neural Networks for Long Sequences". In: *arXiv preprint arXiv:2303.06349* (2023).

[82] Akanksha Kaushik, Jyotsna Singh, and Shilpa Mahajan. "Recurrent Neural Network: A Flexible Tool of Computational Neuroscience Research". In: *Proceedings of the Third International Conference on Information Management and Machine Intelligence: ICIMMI 2021*. Springer. 2022, pp. 377–384.

[83] Omri Barak. "Recurrent neural networks as versatile tools of neuroscience research". In: *Current Opinion in Neurobiology* 46 (2017). Computational Neuroscience, pp. 1–6. ISSN: 0959-4388. DOI: https://doi.org/10.1016/j.conb.2017.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0959438817300429.

[84] F. Ginelli, P. Poggi, A. Turchi, H. Chaté, R. Livi, and A. Politi. "Characterizing Dynamics with Covariant Lyapunov Vectors". In: *Phys. Rev. Lett.* 99 (13 Sept. 2007), p. 130601. DOI: 10.1103/PhysRevLett.99.130601. URL: https://link.aps.org/doi/10.1103/PhysRevLett.99.130601.

[85] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[86] Michel Talagrand. "A new look at independence". In: *The Annals of Probability* 24.1 (1996), pp. 1 –34. DOI: 10.1214/aop/1042644705. URL: https://doi.org/10.1214/aop/1042644705.

[87] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[88] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[89] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[90] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[91] T Tieleman and G Hinton. "Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude". In: *COURSERA: Neural Networks for Machine Learning* 4 (2012), pp. 26–31.

[92] Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. "Kernel and rich regimes in overparametrized models". In: *Conference on Learning Theory*. PMLR. 2020, pp. 3635–3673.

[93] Denisa AO Roberts and Lucas R Roberts. "QR and LQ Decomposition Matrix Backpropagation Algorithms for Square, Wide, and Deep–Real or Complex–Matrices and Their Software Implementation". In: *arXiv preprint arXiv:2009.10071* (2020).