Beyond exploding and vanishing gradients: analysing RNN training using attractors and smoothness

Antônio H. Ribeiro
Federal University of Minas Gerais

Koen Tiels
Eindhoven University of Technology

Luis A. Aguirre
Federal University of Minas Gerais

Thomas B. Schön
Uppsala University

Abstract

The exploding and vanishing gradient problem has been the major conceptual principle behind most architecture and training improvements in recurrent neural networks (RNNs) during the last decade. In this paper, we argue that this principle, while powerful, might need some refinement to explain recent developments. We refine the concept of exploding gradients by reformulating the problem in terms of the cost function smoothness, which gives insight into higher-order derivatives and the existence of regions with many close local minima. We also clarify the distinction between vanishing gradients and the need for the RNN to learn attractors to fully use its expressive power. Through the lens of these refinements, we shed new light on recent developments in the RNN field, namely stable RNN and unitary (or orthogonal) RNNs.

1 Introduction

Training recurrent neural networks can be challenging. The problem in training these recurrent models is usually stated in terms of the so-called *exploding and vanishing gradient problem* (Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013; Bengio et al., 1994). This problem is easy to understand and has motivated many techniques, including the use of gating mechanisms (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), gradient clipping (Pascanu et al., 2013), non-saturating activation functions (Chandar et al., 2019) and the manipulation of the propagation path of gradients (Kanuparthi et al., 2019).

A recently proposed family of methods based on the same principle are the so-called unitary and orthogonal RNNs (Mhammedi et al., 2017; Vorontsov et al., 2017; Lezcano-Casado and Martínez-Rubio, 2019; Helfrich et al., 2018; Arjovsky et al., 2016; Jing et al., 2017; Maduranga et al., 2019; Wisdom et al., 2016; Lezcano-Casado, 2019). In these architectures, the eigenvalues of the hidden-to-hidden weight matrix are fixed to one to simultaneously avoid exploding gradients (that might appear when the eigenvalues are larger than one) and vanishing gradients (that might appear when they are less than one). Similar, but less strict constraints were proposed by Zhang et al. (2018); Kerg et al. (2019). A different line of study goes in the opposite direction and suggests that using RNNs constrained to be stable can provide as good performance on many tasks as unconstrained RNNs. Miller and Hardt (2019) discuss examples for which projecting all eigenvalues to values less than one does not affect the performance. In this case, RNNs can be truncated with arbitrarily small error and hence could be replaced by feedforward structures. Indeed, feedforward structures, such as transformer-based architectures (Vaswani et al., 2017) and convolutional networks (Bai et al., 2018), have recently matched or outperformed RNNs in many tasks. These feedforward architectures yield impressive results in language and music modeling (Bai et al., 2018; van den Oord et al., 2016; Dauphin et al., 2017; Radford et al., 2018, 2019), text-to-speech conversion (van den Oord et al., 2016), machine translation (Kalchbrenner et al., 2016; Gehring et al., 2017) and other sequential learning tasks for which RNNs have been the default choice until recently (Bai et al., 2018).

Our aim in this paper is to improve the present understanding of recurrent neural networks. We believe the moment is propitious for such analysis, and it might help in explaining this shift of winning paradigm for sequence modeling and the lack of consensus on how to deal with the eigenvalues. On the one hand, we complement the vanishing gradient interpretation with an analysis of the attractors of the RNN for storing long-term information. While the condition for dynamic attractors (other than a single fixed point) to appear is the same as for the gradients to vanish, these two notions are distinct. The numerical examples explore the training mechanism and the attractors of the stable (Miller and Hardt, 2019) and orthogonal RNNs (Lezcano-Casado and Martínez-Rubio, 2019). On the other hand, we study the *smoothness* of the cost function, which builds on and improves the notion of exploding gradients, since it also takes into account higher-order derivatives. As a result of our analysis we have that not only "walls" might be formed in the cost function (as suggested by Pascanu et al. (2013)), but also regions with very intricate behavior, full of local minima. These regions are thus very hard for the optimization algorithms to navigate (cf. Figure 1).

2 Recurrent neural networks

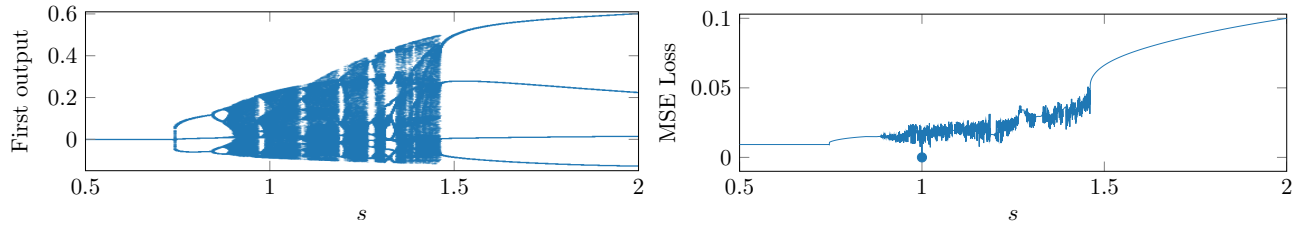
RNNs are nonlinear discrete-time dynamical systems, represented by the expression:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{z}_t; \boldsymbol{\theta}), \tag{1a}$$

$$\hat{\mathbf{y}}_t = \mathbf{g}(\mathbf{x}_t, \mathbf{z}_t; \boldsymbol{\theta}), \tag{1b}$$

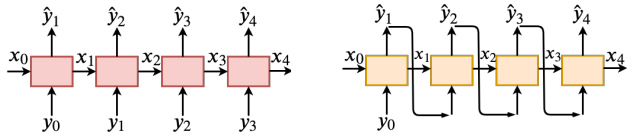
*A.H.R. and L.A.A. are with the Graduate Program in Electrical Engineering the Federal University of Minas Gerais. T.B.S. is with the Department of Information Technology at Uppsala University, which also hosted A.H.R as a visiting researcher and K.T. as a post-doctoral researcher at the moment this work was conceived.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.



(a) (b)

Figure 1: **Chaotic LSTM.** Display: a) Bifurcation diagram; and b) cost function (mean-square error) for LSTM models with parameter vectors $\theta(s) = s\theta_{\text{true}}$.



(a) Training (b) Inference

Figure 2: **Teacher forcing.** Illustration of a recurrent neural network in two different modes. In (a), observed previous values of the output $\mathbf{y}_t$ are used as input to the model. In (b), the neural network outputs are instead fed back as inputs.

where $\mathbf{x}_t \in \mathbb{R}^{N_x}$ is the hidden state and $\hat{\mathbf{y}}_t \in \mathbb{R}^{N_y}$, the predicted output. Training the model amounts to finding a parameter vector $\boldsymbol{\theta}$ that makes the model suitable for a given task. Eq. (1) is sufficiently general to capture vanilla RNNs, LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014), and stacked layers of these units.

When modeling a sequence $\{\mathbf{y}_t\}$, the previously observed outputs are often used as input to the RNN during training i.e., $\mathbf{z}_t = \mathbf{y}_{t-1}$. This technique is called *teacher forcing* and it is commonly used in training language models (Peters et al., 2018).

The mode that is used during training does not have to be used during inference. Furthermore, after training, instead of the observed values, we can successively use the last neural network predicted value to generate many-steps-ahead predictions. The two modes are illustrated in Figure 2. The RNN representation in Eq. (1) is general enough to account for the inference mode by defining an extended hidden state $(\mathbf{x}_t, \hat{\mathbf{y}}_t)$ and new transition and output functions, $\mathbf{f}$ and $\mathbf{g}$, respectively. Hence, teacher forcing yields transition and output functions during inference that are not the same as those used during training.

When the problem being solved has a clearly defined input (c.f. Sections 5.1 and 5.2), the input sequence $\mathbf{u}_t$ should be fed to the neural network. In this case, the neural network input during training can be defined either as $\mathbf{z}_t = \mathbf{u}_t$ or, using teacher forcing, as $\mathbf{z}_t = (\mathbf{y}_{t-1}, \mathbf{u}_t)$.

The model parameters are estimated, i.e., the RNN is trained, by minimizing the cost function:

$$V(\boldsymbol{\theta}) = \frac{1}{N}\sum_{t=1}^{N} l(\mathbf{y}_t, \hat{\mathbf{y}}_t), \tag{2}$$

where l is the loss function and N is the sequence length. For regression and classification problems the squared error and cross entropy loss are common choices. In the case of multiple independent training sequences, the parameters might be estimated by minimizing a weighted average of several Vs defined as above.

3 Information in a recurrent network

It is common to associate the challenges of storing information about a distant past in an RNN with the *vanishing gradients* problem during training (Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013). That is, for long sequences, as the error gradients are back-propagated through the RNN, they might shrink exponentially to zero, making it harder to learn long-term dependencies.

This interpretation has motivated successful strategies for training recurrent neural networks, such as gating mechanisms (Hochreiter and Schmidhuber, 1997). While useful, this notion alone does not give the whole picture, and the presence or absence of dynamic attractors plays an important role in the RNN capability of storing information.

3.1 Entropy of the internal states

We start our analysis by studying the amount of *information* stored by an RNN and how this amount of information changes over time. The discussion applies both to training and inference, and the consequences to each case will be detailed later on.

Assume that at time t the system state $\mathbf{x}_t$ is distributed according to $p_t(\mathbf{x}_t)$. The entropy associated with this probability distribution provides a way of quantifying how much information we would obtain in measuring the state. For the set Ω_x, the entropy H_t at time t can be computed as:

$$H_t = -\int_{\Omega_x} p_t(\mathbf{x}_t) \log p_t(\mathbf{x}_t)\, \mathrm{d}\mathbf{x}_t. \tag{3}$$

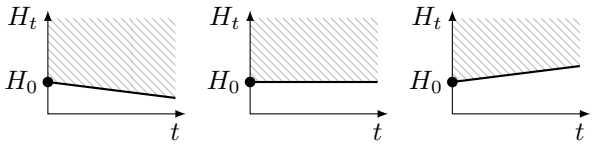
Under mild assumptions, we have (see Section B in the supplementary material) established the inequality:

$$H_t + N_x \log L_f \leq H_{t+1}, \tag{4}$$

where N_x is the dimension of $\mathbf{x}_t$ and L_f is the Lipschitz constant of $\mathbf{f}$ in Eq. (1).

The Lipschitz constant L_f is related to the dynamical behaviour of the RNN, and Eq. (4) establishes that this constant gives a lower bound on the entropy evolution over time. The lower bound is illustrated in Figure 3 and shows that:

1. for a system with $L_f < 1$, the entropy might decay over time towards zero. The larger L_f is, the



(a) $L_f < 1$ (b) $L_f = 1$ (c) $L_f > 1$

Figure 3: **Information in RNNs.** Illustration of the lower bound on the entropy H_t obtained in Eq. (4). Starting from H_0, the entropy H_t can only take values in the shaded region. The entropy may: (a) decay; (b) remain constant; or, (c) increase over time.

slower the decay of information retention can be.

2. for a system with $L_f = 1$, the entropy can stay constant if the bound in Eq. (4) is tight. Hence, such a system might retain the information;
3. for a system with $L_f > 1$, the entropy increase over time. This is the case for chaotic systems, for instance.

By the above discussion, the region of the parameter space for which $L_f < 1$, is the region where the model may not be able to retain information indefinitely. Simultaneously, $L_f < 1$ is a sufficient condition for the gradient to vanish, making it harder for the model to escape from the corresponding regions of the parameter space during training. Hence, information retention and the vanishing gradient problem are related; nevertheless, the two concepts are distinct.

3.2 Contractive *vs* non-contractive systems

A necessary condition for a dynamical system to have $L_f < 1$ is that the system is *contractive*[1].

Definition 1 (**contractive system**). *For a given input sequence $\{\tilde{\mathbf{u}}_t\}$, a dynamical system (1) is said to be contractive in $\Omega_{\mathbf{x}}$ if it satisfies, for some $L_{\tilde{\mathbf{u}}} < 1$ and for all $\mathbf{x}$ and $\mathbf{w}$ in $\Omega_{\mathbf{x}}$:*

$$\|\mathbf{f}(\mathbf{x}, \tilde{\mathbf{u}}_t; \boldsymbol{\theta}) - \mathbf{f}(\mathbf{w}, \tilde{\mathbf{u}}_t; \boldsymbol{\theta})\| < L_{\tilde{\mathbf{u}}} \|\mathbf{x} - \mathbf{w}\|. \quad (5)$$

All contractive systems have a unique fixed point inside the contractive region $\Omega_{\mathbf{x}}$, and all trajectories converge to such a fixed point (Rudin, 1964, Theorem 9.23). Hence, for contractive systems, the distribution $p_t(\mathbf{x}_t)$ will converge to a point mass at the fixed point, and thus zero entropy. Systems with richer nonlinear dynamic behaviors, such as multiple fixed points, limit cycles, and chaotic attractors, are all *non-contractive*.

3.3 Long-term memory

In what follows, the concept of long-term memory will be closely related to the definition of an attractor of a dynamical system.

Definition 2 (**attractor of a dynamical system**). *Let $\mathbf{u}_t$ be equal to a constant $\bar{\mathbf{u}}$ for every t. An **attractor** of the dynamical system $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \bar{\mathbf{u}}; \boldsymbol{\theta})$ is a subset $\mathcal{A} \subset \mathbb{R}^{N_x}$ for which:*

1. *if $\mathbf{x}_{t_0} \in \mathcal{A}$ then $\mathbf{x}_t \in \mathcal{A}$ for all $t \geq t_0$;*
2. *there exists a neighborhood of $\mathcal{A}$, called the **basin of attraction** $\mathcal{B}_{\mathcal{A}}$, such that: if $\mathbf{x}_{t_0} \in \mathcal{B}_{\mathcal{A}}$ then $\mathbf{x}_t \to \mathcal{A}$ as $t \to \infty$;*
3. *there is no proper (non-empty) subset of $\mathcal{A}$ having the first two properties.*

Property 1 is related to the concept of long-term memory. If at a given time instant t_0, $\mathbf{x}_{t_0} \in \mathcal{A}$, the system state will stay in $\mathcal{A}$ for all future time points, that is, it will "remember" this set $\mathcal{A}$. Systems that do not respect this property will just leave (i.e. "forget") set $\mathcal{A}$ at some time $t > t_0$.

Property 2 is related to the *robustness* of the long-term memory. So, for $\mathbf{x}_t \in \mathcal{A}$, we can disturb the system by applying a non-zero finite input sequence $\mathbf{u}_t$ and, if this sequence is sufficiently small so that the state remains in $\mathcal{B}_{\mathcal{A}}$, then the system[2] will converge back to $\mathcal{A}$. In order words, the system will not "forget" $\mathcal{A}$ even in the presence of some (sufficiently small) disturbance.

The idea of defining long-term memory as attractors of a dynamical system is not new. Similar approaches have been pursued in Bengio et al. (1993, 1994). Examples of attractors of after-training RNNs are studied by Sussillo and Barak (2013) and by Maheswaranathan et al. (2019), for toy problems and sentiment analysis, respectively.

In this paper, we will build on this idea and show the RNN may go through bifurcations during training for tasks that require long-term memory and learn attractors to solve the problem. We will also show how stable RNNs and orthogonal RNNs use different mechanisms. For this analysis we will use the bifurcation diagram to visualize the attractors of the RNN. These diagrams show the values visited in steady-state behavior (when a constant input is applied to the system) as a function of some bifurcation parameter s. See Figure 1(a).

As mentioned before, it is possible to have different state transition and output functions for training and inference, respectively. See Figure 2. Hence, the attractors of a recurrent model might be different during training and inference. This is the case for the numerical example in Section 5.3, where we include the analysis of the attractor and bifurcation diagram in both scenarios. Before undertaking this analysis we will study the influence that the internal dynamics has on the smoothness of the cost function.

4 Smoothness of the cost function

We will now establish connections between the *smoothness* of the cost function and the internal dynamics of the RNN *during training*. These connections allow us to extend the concept of exploding gradient with the analysis of the smoothness of the cost function. This analysis is based on the Lipschitz constant of the cost function and of its gradient (sometimes called β-smoothness). Both constants play a crucial role in optimization, see Nesterov (1998), and can be understood as qualitative measurements of how *smooth* the cost function is. Lower values imply that the cost function is less intricate and that optimization algorithms can still converge while taking larger steps. It also provides an upper bound on how different the performance of two close local minima may be.

This smoothness analysis suggests that higher-order derivatives (which contain information about the curvature) might explode in some regions of the parameter

[1] If $\tilde{\mathbf{u}} \in \Omega_{\mathbf{u}}$ it follows that $L_f \geq L_{\tilde{\mathbf{u}}}$. Hence, if the system is not contractive then $L_f \geq L_{\tilde{\mathbf{u}}} \geq 1$.

[2] for $\mathbf{f}$ continuous in $\mathbf{u}$.

space. This indicates that not only walls are formed, but also entire regions of the parameter space exhibit intricate behavior, full of undesirable local minima. This goes beyond what many papers suggest about exploding gradients. For instance, Pascanu et al. (2013) formulate the hypothesis that when gradients explode, they do so in some specific direction, creating *walls* of high curvature. Doya (1993) speculates that bifurcations might cause the jumps in the cost function.

4.1 Example: chaotic LSTM

We consider an LSTM model of dimension two without the bias terms. In Figure 1(a) we show the bifurcation diagram and, in Figure 1(b), the cost function. These are given as a function of s: the weights of the RNN are $\theta(s) = s\theta_{\text{true}}$ and θ_{true} denotes the true data generating weights. See Section C in the supplementary material for additional details.

The bifurcation diagram depicts, for each parameter, the steady-state behavior of the system. It was generated using a simulation of 200 samples for which the first 100 samples were discarded to remove the transient. In this bifurcation diagram we can observe a region with rich nonlinear and chaotic behavior for $0.9 \lesssim s \lesssim 1.45$. This region corresponds to the region where the cost function (Figure 1(b)) is intricate and full of local minima. This connection between internal dynamics and smoothness of the cost function is formalized in the next subsection.

4.2 Theoretical results

In this work, we phrase the exploding gradient problem in terms of the smoothness of the cost function. Here, we quantify smoothness using the Lipschitz constant of the cost function V and of its gradient, the so-called β-smoothness, and establish a connection between these constants, the simulation length N and the state-transition Lipschitz constant L_f. The formulation is quite natural and allows us to include higher-order derivatives in the analysis. We (Ribeiro et al., 2019) have previously studied this result in a different context. Here it is generalized to RNNs.

The Lipschitz constant of V and of its gradient provide upper bounds on $\|\nabla V\|$ and $\|\nabla^2 V\|$, respectively, cf. Khalil (2002, Lemma 3.1). Hence, the first part of the theorem below can indeed be seen as a formalization of the exploding gradient problem. The second part gives information about the explosion of second-order derivatives and curvature and, to the best of our knowledge, is novel. For the case $L_f < 1$, similar results have been presented by Miller and Hardt (2019), but not for the case that interests us the most: $L_f > 1$, for which there might be an explosion in the curvature.

Theorem 1 (Lipschitz constants of V and ∇V). *Let $\mathbf{f}(\mathbf{x}, \mathbf{u}; \boldsymbol{\theta})$ and $\mathbf{g}(\mathbf{x}, \mathbf{u}; \boldsymbol{\theta})$ in Eq. (1) be Lipschitz in $(\mathbf{x}, \boldsymbol{\theta})$ with constants L_f and L_g on a compact and convex set $\Omega = (\Omega_{\mathbf{x}}, \Omega_{\mathbf{u}}, \Omega_{\boldsymbol{\theta}})$. Assume the loss function is either $l(\hat{\mathbf{y}}, \mathbf{y}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2$ or $l(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}^T \log(\sigma(\hat{\mathbf{y}})) - (1-\mathbf{y})^T \log(1-\sigma(\hat{\mathbf{y}}))$. Let $\{\mathbf{u}_t\}_{k=1}^N \subseteq \Omega_{\mathbf{u}}$ and $(\Omega_{\mathbf{x}}, \Omega_{\boldsymbol{\theta}}) \subseteq \mathbb{R}^{N_\theta}$. If there exists at least one choice of $(\mathbf{x}_0, \boldsymbol{\theta})$ for which there is an invariant set contained in Ω, then, for trajectories and parameters confined within Ω:*

1. *The cost function V defined in (2) is Lipschitz with constant:*[3]

$$L_V = \begin{cases} \mathcal{O}(L_f^{2N}) & \text{if } L_f > 1, \\ \mathcal{O}(N) & \text{if } L_f = 1, \\ \mathcal{O}(1) & \text{if } L_f < 1. \end{cases} \tag{6}$$

2. *If the Jacobian matrices of $\mathbf{f}$ and $\mathbf{g}$ are also Lipschitz with respect to $(\mathbf{x}, \boldsymbol{\theta})$ on Ω, then the gradient of the cost function ∇V is Lipschitz with constant:*

$$L'_V = \begin{cases} \mathcal{O}(L_f^{3N}) & \text{if } L_f > 1, \\ \mathcal{O}\left(N^3\right) & \text{if } L_f = 1, \\ \mathcal{O}\left(1\right) & \text{if } L_f < 1. \end{cases} \tag{7}$$

Proof. Section B in the supplementary material. □

Remark (other loss functions). *The above theorem was stated for two different loss functions (squared error and average cross-entropy preceded by the sigmoid function). The theorem still holds for any loss function for which the equivalent of Lemma 2 (in the supplementary material) remains true.*

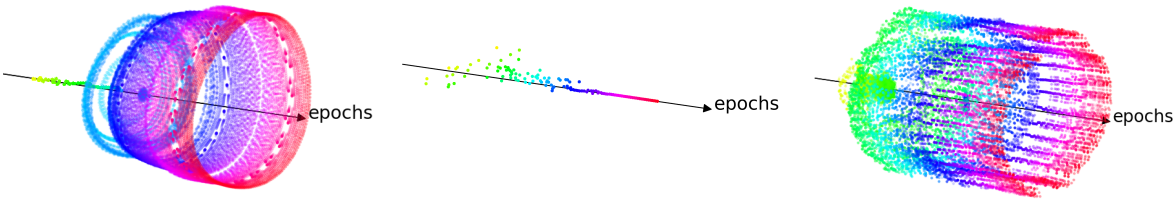
Remark (relation between L_f and eigenvalues). *Let A be the Jacobian matrix of $\mathbf{f}$ with respect to $\mathbf{x}$ for a fixed input $\bar{\mathbf{u}}$. The constant $L_{\bar{\mathbf{u}}}$ in the set Ω is equal to the largest possible eigenvalue of A inside the set. Furthermore, $L_{\bar{\mathbf{u}}}$ is a lower bound on L_f, hence the necessary condition for exploding gradients to appear given in Pascanu et al. (2013) (i.e. largest eigenvalue bigger than 1) also follows from Theorem 1. The necessary condition for the second derivative to explode could also be stated in terms of the largest eigenvalue: it needs to be bigger than 1.*

It is important to highlight that the proof of this more general result follows a different strategy compared to existing proofs of the exploding gradient problem (such as in Pascanu et al. (2013)). Rather than backpropagating the derivatives (which is more natural in the context of neural networks), we use forward propagation (similarly to what is done by Williams and Zipser (1989)) which allows us to arrive at this more general result.

If the system is non-contractive we have $L_f \geq 1$ (cf. Section 3.2). According to Theorem 1, the Lipschitz constants and β-smoothness for all these systems might blow up exponentially (or polynomially for some limit cases) with the maximum simulation length. This might yield very intricate cost functions, see Figure 1(b).

5 Learning attractors during training

Let us now study the mechanism employed by RNNs in learning a task that requires long-term memory. We focus on the LSTM (Hochreiter and Schmidhuber, 1997); on the stable LSTM (sLSTM) proposed by Miller and Hardt (2019), which projects, at each iteration, the parameters into the region of the parameter space that yields contractive models; and, on the orthogonal RNN (oRNN) proposed by Lezcano-Casado and Martínez-Rubio (2019).



(a) LSTM (b) sLSTM (c) oRNN

Figure 4: **Learning oscillations.** Bifurcation diagram during training for the sine-wave generation task showing the steady-state of the output y_t and its first difference $y_t - y_{t-1}$. For each epoch, the plot shows values visited after a burn-in period of 1200 samples, which is used to remove the transient response and to produce a visualization of the system attractors. The plot shows epochs 300 to 1500, the initial 300 epochs are out of scale for the oRNN, and are omitted in all diagrams. The bifurcation diagram is for a constant input of 0.218, and other input values yield similar plots.

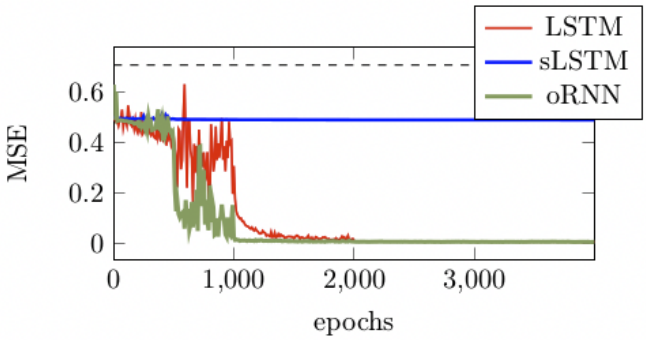


Figure 5: **Sine-wave generator training history.** Mean square error (MSE) per epoch for the sine-wave generation task during training. The baseline performance is indicated by the dashed line. The final performance of the LSTM after 4000 epochs is $4.8 * 10^{-3}$, the final performance of the sLSTM is 0.49 and the final performance of the oRNN is $4.1 * 10^{-3}$.

5.1 Sine-wave generation

We study the use of RNNs for the generation of sine-waves with varying frequencies. The input is a constant, and the output is a sine-wave with unitary amplitude and the frequency specified by the constant input. This artificial task is described in Sussillo and Barak (2013). We have 100 sequences of length 400, each sequence consisting of a sine-wave with the given constant frequency. The 100 sequences are uniformly picked in the interval $\left[\frac{\pi}{16}, \frac{\pi}{8}\right]$. We use those sequences for training.

For all models, we use the same configurations: Adam optimization algorithm (Kingma and Ba, 2014) with initial learning rate 10^{-3} and default parameters. For the stable model, we decrease the learning rate by 10 at epochs $\{500, 1000, 2000\}$. We use a hidden size of 200 and a single layer for the RNN, which is followed by a linear layer for the output. The gradient is clipped when its norm exceeds 0.25.

The training history for the three models is displayed in Figure 5. The stable LSTM fails to perform well in the sine-wave generation task. Figure 4 shows the bifurcation diagram for the three models during training. Here we use a two-dimensional bifurcation diagram containing both the output y_t and its first difference $y_t - y_{t-1}$ to facilitate the visualization of periodic attractors.

For the LSTM, during training, a stable fixed point undergoes a bifurcation at which the fixed point stability switches, and a periodic solution arises. The stable LSTM is constrained to stay in the region of the parameter space for which the system is contractive and, hence has a single stable attractor point (cf. discussion on Section 4). Since the system needs to sustain the oscillations, this is a significant limitation that prevents the model from performing the task well.

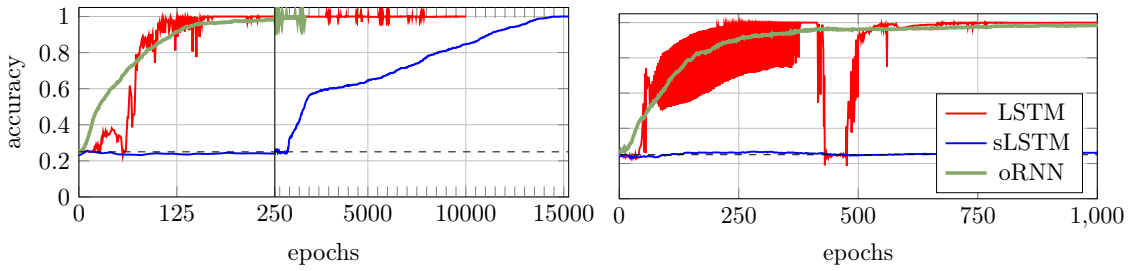
The bifurcation that the fixed point needs to undergo to oscillate is called Hopf (for the continuous case) or Neimark-Sacker bifurcation (for the discrete case). During this bifurcation, a single stable fixed point changes stability and becomes unstable as a pair of complex eigenvalues (of the Jacobian matrix) enters the unstable region. The orthogonal constraint in the orthogonal RNN prevents the occurrence of these bifurcations since all eigenvalues are fixed to one. Nevertheless, orthogonal RNNs still manage to solve the problem and learn the periodic attractor.

Since most real-world sequence tasks have stochastic elements, the deterministic nature of the sine-wave generation is a limitation of this task to be representative of real-world problems. Nevertheless, this example illustrates the challenges of learning steady-state behavior using RNNs. Stable models will not be able to store information in the form of attractors, unless other training mechanisms, such as teacher forcing, are used (see Figure 2). Orthogonal models, do not have this restriction, but, even so, the orthogonal constraints can also prevent some bifurcations, and require different learning mechanisms. For instance, another type of bifurcation that this model will fail to capture is the supercritical pitch-fork bifurcation, where a single stable fixed point loses stability, and two stable fixed points are created.

5.2 Sequence classification based on few relevant symbols

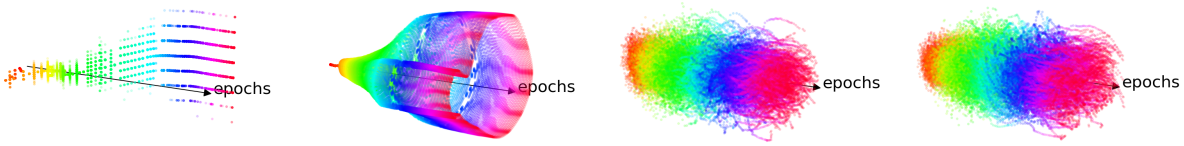
We study the classification of sequences according to a few relevant, widely separated, symbols. This artificial task was originally described by Hochreiter and Schmidhuber (1997), and it requires the neural network to retain information for long periods. For this problem, the sequence contains categorical values $\{p, q, a, b, c, d\}$. The symbols $\{a, b, c, d\}$ act as distractors and are not relevant to the task. Instead, the relevant symbols are

[3]Here $\mathcal{O}$ is the big O notation. It should be read as: $L(N) = \mathcal{O}(g(N))$ if and only if there exist positive integers M and N_0 such that $|L(N)| \leq Mg(N)$ for all $N \geq N_0$.



(a) Sequence length 100 (b) Sequence length 200

Figure 6: **Sequence classification training history.** Accuracy on validation data set for the recurrent models trained to perform the same sequence classification task for two different sequence lengths. The baseline performance (always predicting $\{p, p\}$) is indicated by the dashed line. In (a) two x-axis scales co-exist in the same graph, one scale in [0, 250) and other in [250, 15250], with a relation 1:40 between the two scales.



(a) LSTM, $p \to \{p, p\}$ (b) LSTM, $b \to \{p, p\}$ (c) oRNN, $p \to \{p, p\}$ (d) oRNN, $b \to \{p, p\}$

Figure 7: **Learning to classify sequences.** Bifurcation diagram for the sequence classification task for sequences of length 100. It shows the steady-state of the output y_t and its first difference $y_t - y_{t-1}$. The arrows point towards the evolution of the number of epochs, that varies from 0 to 400. For each epoch, the plot shows values visited after a burn-in period of 400 samples, which is used to remove the transient response and to produce a visualization of the system attractors.

Table 1: **Sequence classification accuracy.** Comparison between the LSTM; stable LSTM (sLSTM); and orthogonal RNN (oRNN); for sequence classification with different sequence lengths (l). As a baseline, always predicting $\{p, p\}$ would yield an accuracy of 0.25.

l	LSTM	sLSTM	oRNN
50	1.00	1.00	1.000
100	1.00	1.00	1.000
200	1.00	0.27	0.999
300	0.25	0.26	0.995
500	0.27	0.26	0.970

picked from $\{p, q\}$ appearing only twice in the sequence, at positions $t_1 \in [10, 18]$ and $t_2 \in [40, 49]$. The task of the neural network is to classify the sequence according to the order in which the relevant symbols occur. The four possible classes are $\{(p, p), (p, q), (q, p), (q, q)\}$. Training and validation datasets each contain 1000 sequences, and all sequences have the same length.

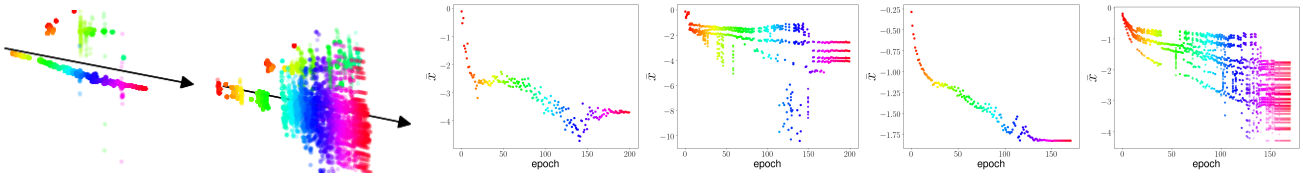
For all models, we use the same configurations: Adam optimization algorithm with an initial learning rate of 10^{-2} and default parameters. We decrease the learning rate by 10 at epochs $\{500, 1000, 2000\}$, and run the optimization for a total of 15000 epochs or until accuracy one is attained. We use a hidden size of 200 and a single layer for the RNN, which is followed by a linear layer with a single output. The gradient is clipped when its norm exceeds 0.25. The batch size is 100.

Figure 6 shows the training history for the different models in the sequence classification task. In Figure 6(a), the sLSTM eventually manage to solve the task, the accuracy, however, increase at a very slow linear rate between epochs $[1000, 10000]$. The linear rate is quite characteristic of first-order optimization methods in a basin of attraction (Nocedal and Wright, 2006), and is very slow due to the vanishing gradient problem. On the other hand, the standard LSTM converges quickly. The convergence, however, is quite dependent on the initial condition. See Supplementary Figure 2 to see how, for a different random seed and identical setup, the model completely fails to converge.

For length 200, the LSTM converges, but very unsteadily and with many bumps. The strong dependency with the initial conditions and hard to navigate landscape is in agreement with the considerations about smoothness and multiple close local minima that might affect this model (cf. Section 4). The orthogonal RNN avoids this hard to navigate landscape by fixing the eigenvalues equal to one, and it does have a much smoother convergence than the LSTM. It is also the model that manages to solve the problem for the longest sequences (see Table 1). The stable LSTM model, on the other hand, fails to solve the task for the length of 200 or longer.

To analyze the attractors during training, we apply a constant unitary input to one of the six input channels (keeping the others at zero) and measure one of the four possible outputs. Bifurcation diagrams for the LSTM and the oRNN for all 6×4 possible combinations of input/output pairs are displayed in Supplementary Figures 3 and 4. Figure 7 shows a subset of the combinations. More specifically, it shows the bifurcation diagram between the input p and the output $\{p, p\}$ and between the input b and the same output, which we believe to be representative of all possible combinations.

The LSTM goes through bifurcations, for instance: in $p \to \{p, p\}$, the number of fixed points increases, and,



(a) LSTM (b) LSTM, feedback (c) oRNN (d) oRNN, feedback (e) sLSTM (f) sLSTM, feedback

Figure 8: **Learning to model language.** Bifurcation diagram for the world-level language model for: the LSTM, in (a)-(b); the oRNN, in (c)-(d); and, the sLSTM, in (e)-(f). In (a) and (b), we plot a 2-D bifurcation diagram showing the steady-state of the average of internal states $\bar{x}_t$ and its first difference $\bar{x}_t - \bar{x}_{t-1}$. In (c)-(f), a 1-D bifurcation of the average $\bar{x}_t$ is used, which facilitates the visualization. In (a), (c) and (e), the bifurcation diagram is obtained from constant inputs. In (b), (d) and (f), the diagram is generated using as input the word predicted with the highest probability at the previous time instant, and using as first input to the sequence the same input as in the constant input case.

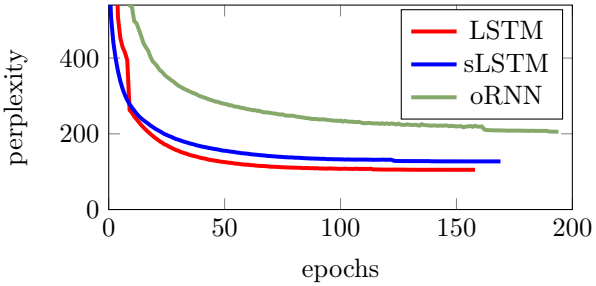


Figure 9: **Word-level language model training history.** Perplexity on validation data per epoch.

in $b \to \{p, p\}$, they transition to periodic behavior. We understand the ability to undergo bifurcations is useful, and having multiple fixed points helps the LSTM to solve the task for longer sequences than it would be able to only within the contractive region. The comparison between LSTM and stable LSTM in Table 1 seems to corroborate such a hypothesis. The periodic behavior in $b \to \{p, p\}$, on the other hand, is spurious, since b and the other distractor symbols do not influence the outcome. The mechanism the orthogonal RNN uses to solve the task is quite distinct: they create a cloud of fixed points and, without suffering any bifurcation, during training, these fixed points change position in the state space. These fixed points do not disappear during training, and we believe the last layer learns to weigh them differently and use the information to solve the task.

5.3 Word-level language model

We train a language model on the openly available dataset Wikitext-2. This dataset contains 600 Wikipedia articles for training (2,088,628 tokens), 60 articles for validation (217,646 tokens), and 60 articles for testing (245,569 tokens) (Merity et al., 2017). The dataset has a vocabulary of 33,278 distinct tokens. The goal is to predict the next token in the article given the previous tokens. The state-of-the-art result for the Wikitext-2 data set when not extending the training set is a perplexity (lower is better) on the test set of 39.14 (Gong et al., 2018).

Since our implementation of the orthogonal RNN is restricted to a single layer, we restrict all the models to a single layer. Despite this our LSTM model achieves a perplexity of 99.2 on the test set, which is close to other results reported in the literature (i.e. 99.3 obtained for a standard LSTM model in Edouard Grave (2017)). The stable LSTM has a perplexity of 118.8 (which is close to Miller and Hardt (2019), where they achieve a perplexity of 113.2) and the orthogonal LSTM has a perplexity of 185.3.

We trained a model that consists of an embedding layer outputting a size 800 vector to a recurrent unit (LSTM, stable LSTM, or orthogonal RNN) with hidden size 800, followed by a softmax decoder layer that outputs probabilities for each token in the vocabulary. The last layer and the embedding have tied layers. A dropout layer is included before and after the LSTM layer with a dropout rate of 0.5. The models are trained and evaluated using a cross-entropy loss. The model is trained using ADAM with betas (0.9, 0.999). The learning rate starts at 10^{-3} and is divided by 10 when the cost function plateaus for more than 7 epochs. The training is stopped when the learning rate drops below 10^{-7}. The gradient norm is clipped when it exceeds 0.25. The batch size is 100, and the gradient is not backpropagated for a sequence length larger than 80.

Figure 8 shows bifurcation diagrams for this task. The problem has a high-dimensional output, and the average of all outputs, i.e., $\bar{x}_t$, is used to condense the information. The projection is extended into two dimensions by including the first difference for the case of the LSTM bifurcation diagram visualization. Other projections, other then the average of internal states, yield similar qualitative results and are displayed in Supplementary Figures 5, 6 and 7. The projection in the direction of the tokens "is", "<unk>", and "Valkyria", which are representative of both frequent and infrequent tokens, are used in this case. The bifurcation diagram for different random initial states and different constant inputs is also displayed in the supplementary material.

When fed with constant input, LSTM presents a wide range of different qualitative behavior: from converging to a single fixed point, as in Figure 8(a), to going through bifurcations that result in chaotic or periodic behavior, as in Supplementary Figure 5(b). When fed with constant input, orthogonal RNNs, on the other hand, converge to a single fixed point. This was verified for several random seeds, and we show a representative example in Figure 8(c). Qualitatively, this behavior is quite similar to the stable LSTM, which also has single fixed-point, by construction.

We also show the attractors for when the input, at instant t, is the word predicted with the highest probability at the previous time instant, $t-1$. This scenario corresponds to using the model for sentence generation. LSTM presents, again, chaotic attractors, which are

quite desirable here, since periodic behavior or a single attractor correspond to repeating a few words or only a single word after sufficient interactions, which would not generate interesting sentences. Both orthogonal RNNs and stable LSTMs present this type of undesirable behavior. Not to say they are the same, in our experiments, sLSTM usually takes more than 600 tokens to reach this steady-state periodic behavior (after <eos> most of the time) while for the oRNN this sometimes happens before 100 tokens. Also, sLSTM converges to a richer periodic behavior, which oscillates between more points (compare Figure 8 (d) and (f)). This example shows how the stable LSTM, during inference, might present richer steady-state behavior due to the feedback connections.

The training history is displayed in Figure 9. In the previous examples, the LSTM training was unsteady and full of jumps (and also highly sensitive to initial conditions, cf. Supplementary Figure 2); here, the curve is very smooth. Chaotic and periodic attractors in the constant input scenario suggest that training the LSTM goes into the region of the parameter space that yields non-contractive models. An experiment presented by Laurent and von Brecht (2017, sec. 2.1) shows that this chaotic behavior, however, has limited influence in the output. This might help explain why the training of the LSTM model progresses so smoothly. It might also help in explaining the good performance of the stable LSTM model.

6 Discussion

We believe the refinements presented above might be relevant in developing new and more capable optimization and regularization methods for RNNs and to understand the limitations and strengths of the many techniques that are already in use.

Constraining the RNN eigenvalues to be smaller than one yields exponentially decreasing transient behavior, which in turn gives rise to vanishing gradients. When the training and inference mode are the same, this constraint also has the effect of not allowing attractors to appear. Much of the current RNN literature, however, does not distinguish between the two effects, which precludes richer analysis and an improved understanding of the training mechanism.

It is also important to draw a clear distinction between the RNN behavior during training and inference. As discussed in Section 3, attractors are needed to store information in a non-volatile way. When the training and inference mode are the same, learning attractors (other than a single fixed point) require the optimization procedure to explore regions of the parameter space where the model is non-contractive. As discussed in Section 4, the smoothness properties of the objective function might be very poor in these regions, which is typically very problematic for traditional optimization procedures.

The symbol classification task studied in this paper is very similar to other artificial benchmarks used to assess the ability of recurrent models to *remember* information seen many time steps before. Other examples include the adding and multiplication problems (Hochreiter and Schmidhuber, 1997) and the copy memory task (Arjovsky et al., 2016). All these problems are useful benchmarks in assessing the model's ability to learn attractors and explore regions of the parameter space where the model is non-contractive. The results in this paper seem to reinforce the effectiveness of oRNN models in solving this class of problems.

Artificial benchmarks are usually formulated in a way that makes it hard or even impossible to use teacher forcing. State-of-the-art models for many relevant tasks, such as word language modeling, however, employ such a mechanism. Teacher forcing introduces the appealing possibility of using different modes for training and inference. In this case, the system output is fed back as an input, allowing new dynamical behavior during inference. Hence, even if the training is limited to regions of the parameter space where the dynamic system is contractive, it might still be possible to have attractors (and hence long-term memory) during inference.

Different training and inference behavior might help explain why the stable model by Miller and Hardt (2019) is successful in many tasks. On the one hand, training the model in the unstable region is challenging, so ruling out this region of the parameter space from the optimization procedure might not be so useful. On the other hand, the model is not necessarily stable during inference since the output is fed back as an input. A similar argument might also justify the good performance of the RNN model without chaotic behavior proposed by Laurent and von Brecht (2017).

7 Related work

Sussillo and Barak (2013) and Maheswaranathan et al. (2019) give examples analyzing attractors in an RNN. However, they focus on the situation after training, while we use the attractors for the analysis of the training procedure.

The understanding of exploding and vanishing gradients attributed to (Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013; Bengio et al., 1994). Pascanu et al. (2013) formulates exploding and vanishing gradients in terms of the eigenvalues of the hidden-to-hidden weight matrix but do not consider the importance of attractors in retaining the information. Bengio et al. (1994) define long-term memory in terms of single fixed points and present a trade-off between vanishing gradients and retaining information robustly in the presence of noise. We extend the analysis by contemplating chaotic and periodic attractors and, also, the possibility of different training and inference modes.

Pascanu et al. (2013) conjectures that the walls in the cost function are due to bifurcations. In our examples, we do not observe this connection between bifurcations and jumps in performance.

The notion of using entropy for a dynamical system is not new. One classical definition is that of the Kolmogorov-Sinai entropy (Sinai, 1959), which is related to how uncertainty (and information) increases with time in a chaotic attractor. However, this definition cannot be applied for regions of the parameter space that do not preserve volume, and hence the definition in Section 3 serves our purpose better.

Code availability: The code for reproducing the numerical examples is available at: github.com/antonior92/attractors-and-smoothness-RNN.

Acknowledgments

We thank Carl Jidling and Manoel Horta Ribeiro for insightful comments and suggestions. This work has been supported by the Brazilian agencies *CAPES - Coordenacão de Aperfeiçoamento de Pessoal de Nível Superior* (Finance Code: 001), *CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico* (contract number: 302079/2011-4, 200931/2018-0 and 142211/2018-4) and *FAPEMIG - Fundação de Amparo à Pesquisa do Estado de Minas Gerais* (contract number: TEC 1217/98), by the Swedish Research Council (VR) via the projects *NewLEADS – New Directions in Learning Dynamical Systems* (contract number: 621-2016-06079) and *Learning flexible models for nonlinear dynamics* (contract number: 2017-03807), and by the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (contract number: RIT15-0012)

References

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML)*, pages 1120–1128.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. page 14.

Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks*, pages 1183–1188.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Chandar, S., Sankar, C., Vorontsov, E., Kahou, S. E., and Bengio, Y. (2019). Towards Non-Saturating Recurrent Units for Modelling Long-Term Dependencies. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3280–3287.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259*.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 933–941.

Doya, K. (1993). Bifurcations of Recurrent Neural Networks in Gradient Descent Learning.

Edouard Grave, Armand Joulin, N. U. (2017). Improving neural language models with a continuous cache. In *Proceedings of the 5th International Conference on Learning Representations (ICML)*.

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. (2017). A Convolutional Encoder Model for Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 123–135.

Gong, C., He, D., Tan, X., Qin, T., Wang, L., and Liu, T.-Y. (2018). Frage: Frequency-agnostic word representation. In *Advances in Neural Information Processing Systems 31*.

Helfrich, K., Willmott, D., and Ye, Q. (2018). Orthogonal recurrent neural networks with scaled Cayley transform. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1969–1978.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljacic, M. (2017). Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs. *Proceedings of the 34th International Conference on Machine Learning (ICML)*.

Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A., and Kavukcuoglu, K. (2016). Neural Machine Translation in Linear Time. *arXiv:1610.10099*.

Kanuparthi, B., Arpit, D., Kerg, G., Ke, N. R., Mitliagkas, I., and Bengio, Y. (2019). H-DETACH: Modifying the LSTM Gradient Towards Better Optimization. *Proceedings of the 7th International Conference for Learning Representations (ICLR)*.

Kerg, G., Goyette, K., Touzel, M. P., Gidel, G., Vorontsov, E., Bengio, Y., and Lajoie, G. (2019). Non-normal Recurrent Neural Network (nnRNN): Learning long time dependencies while improving expressivity with transient dynamics. In *Advances in Neural Information Processing Systems 32*.

Khalil, H. K. (2002). *Nonlinear Systems*. Upper Saddle River, third edition.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations (ICLR)*.

Laurent, T. and von Brecht, J. (2017). A recurrent neural network without chaos. In *Proceedings of the 5th International Conference for Learning Representations (ICLR)*.

Lezcano-Casado, M. (2019). Trivializations for Gradient-Based Optimization on Manifolds. In *Advances in Neural Information Processing Systems 32*.

Lezcano-Casado, M. and Martínez-Rubio, D. (2019). Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3794–3803.

Maduranga, K. D., Helfrich, K. E., and Ye, Q. (2019). Complex unitary recurrent neural networks using scaled cayley transform. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4528–4535.

Maheswaranathan, N., Williams, A., Golub, M. D., Ganguli, S., and Sussillo, D. (2019). Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. In *Advances in Neural Information Processing Systems 32*, pages 15670–15679.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *Proceedings of the 5th International Conference for Learning Representations (ICLR)*.

Mhammedi, Z., Hellicar, A., Rahman, A., and Bailey, J. (2017). Efficient orthogonal parametrisation of recurrent neural networks using Householder reflections. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 2401–2409.

Miller, J. and Hardt, M. (2019). Stable Recurrent Models. In *Proceedings of the 7th International Conference for Learning Representations (ICLR)*.

Nesterov, Y. (1998). *Introductory Lectures On Convex Programming*. Springer Science & Business Media.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 2nd edition.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML)*, pages 1310–1318.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners.

Ribeiro, A. H., Tiels, K., Umenberger, J., Schön, T. B., and Aguirre, L. A. (2019). On the Smoothness of Nonlinear System Identification. *Provisionally accepted at Automatica*.

Rudin, W. (1964). *Principles of Mathematical Analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill.

Sinai, Y. G. (1959). On the notion of entropy of a dynamical system. In *Dokl. Akad. Nauk. SSSR*, volume 124, page 768.

Spivak, M. (1998). *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. Mathematics Monograph Series. Perseus Books, Cambridge, Mass.

Sussillo, D. and Barak, O. (2013). Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. *Neural Computation*, 25(3):626–649.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Vorontsov, E., Trabelsi, C., Kadoury, S., and Pal, C. (2017). On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3570–3578.

Williams, R. J. and Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. (2016). Full-Capacity Unitary Recurrent Neural Networks. *Advances in Neural Information Processing Systems 29*, pages 4880–4888.

Zhang, J., Lei, Q., and Dhillon, I. S. (2018). Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization. *Proceedings of the 35th International Conference on Machine Learning (ICML)*