ARTICLE Communicated by Ilya Sutskever

Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks

David Sussillo
sussillo@stanford.edu
Department of Electrical Engineering, Neurosciences Program, Stanford University, Stanford, CA 94305-9505, U.S.A.

Omri Barak
omri.barak@gmail.com
Department of Neuroscience, Columbia University College of Physicians and Surgeons, New York, NY 10032-2695, U.S.A.

Recurrent neural networks (RNNs) are useful tools for learning nonlinear relationships between time-varying inputs and outputs with complex temporal dependencies. Recently developed algorithms have been successful at training RNNs to perform a wide variety of tasks, but the resulting networks have been treated as black boxes: their mechanism of operation remains unknown. Here we explore the hypothesis that fixed points, both stable and unstable, and the linearized dynamics around them, can reveal crucial aspects of how RNNs implement their computations. Further, we explore the utility of linearization in areas of phase space that are not true fixed points but merely points of very slow movement. We present a simple optimization technique that is applied to trained RNNs to find the fixed and slow points of their dynamics. Linearization around these slow regions can be used to explore, or reverse-engineer, the behavior of the RNN. We describe the technique, illustrate it using simple examples, and finally showcase it on three high-dimensional RNN examples: a 3-bit flip-flop device, an input-dependent sine wave generator, and a two-point moving average. In all cases, the mechanisms of trained networks could be inferred from the sets of fixed and slow points and the linearized dynamics around them.

1 Introduction

A recurrent neural network (RNN) is a type of artificial neural network with feedback connections. Because the network has feedback, it is ideally suited for problems in the temporal domain such as implementing

Both authors contributed equally to this article. O.B. is now at the Rappaport Faculty of Medicine, Israeli Institute of Technology, Haifa, Israel.

temporally complex input-output relationships, input-dependent pattern generation, and autonomous pattern generation. However, training RNNs is widely accepted as a difficult problem (Bengio, Simard, & Frasconi, 1994). Recently, there has been progress in training RNNs to perform desired behaviors (Maass, Natschläger, & Markram, 2002; Jaeger & Haas, 2004; Maass, Joshi, & Sontag, 2007; Sussillo & Abbott, 2009; Martens & Sutskever, 2011). Jaeger and Hass (2004) developed an echostate network, a type of RNN that achieves good performance by allowing a random recurrent pool to receive feedback from a set of trained output units. Sussillo and Abbott (2009) derived a rule to train the weights of such an echostate network operating in the chaotic regime. Further, Martens and Sutskever (2011) employed a second-order optimization technique, Hessian-free optimization, to train all the weights in an RNN using backpropagation through time to compute the gradient (Rumelhart, Hinton, & Williams, 1985). Given these recent developments, it is likely that RNNs will enjoy more popularity in the future than they have to date.

Because such supervised training algorithms specify the function to perform without specifying how to perform it, the exact nature of how these trained RNNs implement their target functions remains an open question. The resulting networks are often viewed as black boxes. This is in contrast to network models that are explicitly constructed to implement a specific known mechanism (see Wang, 2008; Hopfield, 1982). One way to make progress may be to view an RNN as a nonlinear dynamical system (NLDS), and in this light, there is a rich tradition of inquiry to exploit.

A nonlinear dynamical system is, as the name implies, nonlinear. As such, the qualitative behavior of the system varies greatly between different parts of phase space and can be difficult to understand. A common line of attack when analyzing NLDSs is therefore to study different regions in phase space separately. The most common anchors to begin such analyses are fixed points—points in phase space exhibiting zero motion, with the invaluable property that the dynamics near a fixed point are approximately linear and thus easy to analyze. Other, faster points in phase space can also provide insight into the system's mode of operation but are harder to systematically locate and analyze.

In this article, we show that there are other regions in phase space, which we call *slow points*, where linearization is valid. Our hypothesis is that understanding linearized systems around fixed and slow points in the vicinity of typical network trajectories can be insightful regarding computational mechanisms of RNNs. We provide a simple optimization technique for locating such regions. The full analysis would therefore entail finding all candidate points of the RNN, linearizing the dynamics around each one, and finally trying to understand the interaction between different regions (Strogatz, 1994; Ott, 2002). We illustrate the application of the technique using both fixed and slow points. The principles are first illustrated with

a simple two-dimensional example, followed by high-dimensional trained RNNs performing memory and pattern generation tasks.

2 Fixed Points

Fixed points are common anchors to start an analysis of the system. They are either stable or unstable, meaning that the motion of the system, when started in the vicinity of a given fixed point, either converges toward or diverges away from that fixed point, respectively. Stable fixed points, also known as attractors, are important because the system will converge to them or, in the presence of noise, dwell near them. Unstable fixed points come in more varieties, having one or more unstable modes, up to the case of a completely unstable fixed point (repeller). A mode is an independent pattern of activity that arises when the linear system is diagonalized. In efforts to understand the interaction of different attractors, unstable fixed points with a few unstable modes are often very useful. For instance, a saddle point with one unstable mode can funnel a large volume of phase space through its many stable modes and then send them to two different attractors depending on which direction of the unstable mode is taken.

Finding stable fixed points is often as easy as running the system dynamics until it converges (ignoring limit cycles and strange attractors). Finding repellers is similarly done by running the dynamics backward. Neither of these methods, however, will find saddles. The technique we introduce allows these saddle points to be found, along with both attractors and repellers. As we will demonstrate, saddle points that have mostly stable directions, with only a handful of unstable directions, appear to be highly significant when studying how RNNs accomplish their tasks (a related idea is explored in Rabinovich, Huerta, Varona, & Afraimovich, 2008).

3 Linear Approximations

Consider a system of first-order differential equations

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}), \tag{3.1}$$

where $\mathbf{x}$ is an N-dimensional state vector and $\mathbf{F}$ is a vector function that defines the update rules (equations of motion) of the system. We wish to find values $\mathbf{x}^*$ around which the system is approximately linear. Using a Taylor series expansion, we expand $\mathbf{F}(\mathbf{x})$ around a candidate point in phase space, $\mathbf{x}^*$:

$$\mathbf{F}(\mathbf{x}^* + \boldsymbol{\delta}\mathbf{x}) = \mathbf{F}(\mathbf{x}^*) + \mathbf{F}'(\mathbf{x}^*)\boldsymbol{\delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\delta}\mathbf{x}\mathbf{F}''(\mathbf{x}^*)\boldsymbol{\delta}\mathbf{x} + \dots \tag{3.2}$$

Because we are interested in the linear regime, we want the first derivative term of the right-hand side to dominate the other terms. Specifically, we are looking for points $\mathbf{x}^*$ and perturbations around them, $\boldsymbol{\delta x} \equiv \mathbf{x} - \mathbf{x}^*$, such that

$$|\mathbf{F}'(\mathbf{x}^*)\boldsymbol{\delta x}| > |\mathbf{F}(\mathbf{x}^*)|, \tag{3.3}$$

$$|\mathbf{F}'(\mathbf{x}^*)\boldsymbol{\delta x}| > \left|\frac{1}{2}\boldsymbol{\delta x}\mathbf{F}''(\mathbf{x}^*)\boldsymbol{\delta x}\right|. \tag{3.4}$$

From inequality 3.3, it is evident why fixed points ($\mathbf{x}^*$ such that $\mathbf{F}(\mathbf{x}^*) = \mathbf{0}$) are good candidates for linearization: the lower bound for $\boldsymbol{\delta x}$ is zero, and thus there is always some region around $\mathbf{x}^*$ where linearization should be valid. They are not, however, the only candidates. A slow point, having a nonzero but small value of $\mathbf{F}(\mathbf{x}^*)$, could be sufficient for linearization. In this case, examining the linear component of the computations of an RNN may still be reasonable because the inequalities can still be satisfied for some annulus-like region around the slow point.

Allowing $|\mathbf{F}(\mathbf{x}^*)| > 0$ means we are defining an affine system, which can be done anywhere in phase space. However, inequalities 3.3 and 3.4 highly limit the set of candidate points we are interested in. So we end up examining true linear systems at fixed points and affine systems at slow points. The latter are not generic affine systems since the constant is so small as to be negligible at the timescale on which the network operates. It is in these two ways that we use the term *linearization*.

This observation motivated us to look for regions where the norm of the dynamics, $|\mathbf{F}(\mathbf{x})|$, is either zero or small. To this end, we define an auxiliary scalar function,

$$q(\mathbf{x}) = \frac{1}{2}|\mathbf{F}(\mathbf{x})|^2. \tag{3.5}$$

Intuitively, the function q is related to the speed, or kinetic energy, of the system. Speed is the magnitude of the velocity vector, so q is the squared speed divided by two, which is also the expression for the kinetic energy of an object with unit mass.

There are several advantages to defining q in this manner. First, it is a scalar quantity and thus amenable to optimization algorithms. Second, becasue q is a sum of squares, $q(\mathbf{x}^*) = 0$ if and only if $\mathbf{x}^*$ is a fixed point of $\mathbf{F}$. Third, by minimizing $|\mathbf{F}(\mathbf{x})|$, we can find candidate regions for linearization that are not fixed points, assuming inequalities 3.3 and 3.4 are satisfied, thus expanding the scope of linearization techniques.

To understand the conditions when q is at a minimum, consider the gradient and Hessian of q:

$$\frac{\partial q}{\partial x_i} = \sum_k^N \frac{\partial F_k}{\partial x_i} \dot{x}_k, \tag{3.6}$$

$$\frac{\partial^2 q}{\partial x_i \partial x_j} = \sum_k^N \frac{\partial F_k}{\partial x_i} \frac{\partial F_k}{\partial x_j} + \sum_k^N \dot{x}_k \frac{\partial^2 F_k}{\partial x_i \partial x_j}. \tag{3.7}$$

For q to be at a minimum, the gradient (see equation 3.6) has to be equal to the zero vector, and the Hessian (see equation 3.7) has to be positive definite. There are three ways for the gradient to be zero:

1. If $\dot{x}_k = 0$, for all k, then the system is at a fixed point, and this is a global minimum of q.
2. If the entire Jacobian matrix, $\frac{\partial F_i}{\partial x_j}(\mathbf{x})$, is zero at $\mathbf{x}$ and the second term of the Hessian is positive definite, then q may be at a local minimum.
3. Finally, q may be at a minimum if $\dot{\mathbf{x}}$ is a zero eigenvector of $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$. For high-dimensional systems, this is a much less stringent requirement than the previous one, and thus tends to be most common for local minima in RNNs.

In summary, our approach consists of using numerical methods to minimize q to find fixed points and slow points. Because the systems we study may have many different regions of interest, the optimization routine is run many times from different initial conditions (ICs) that can be chosen based on the particular problem at hand. Typically these ICs are points along RNN system trajectories during computations of interest. In the following sections, we demonstrate our approach on cases involving fixed points for both low- and high- dimensional systems. Then we show the advantages of linearizing around regions that are not fixed points.

3.1 Two-Dimensional Example. A simple 2D example helps to demonstrate minimization of the auxiliary function q and evolution of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$. For this section, we define the dynamical system $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ as follows:

$$\dot{x}_1 = (1 - x_1^2)x_2, \tag{3.8}$$

$$\dot{x}_2 = x_1/2 - x_2. \tag{3.9}$$

The system has three fixed points: a saddle at $(0, 0)$ and two attractors at $(1, 1/2)$ and $(-1, -1/2)$. Based on this definition of $\mathbf{F}$, the function q is

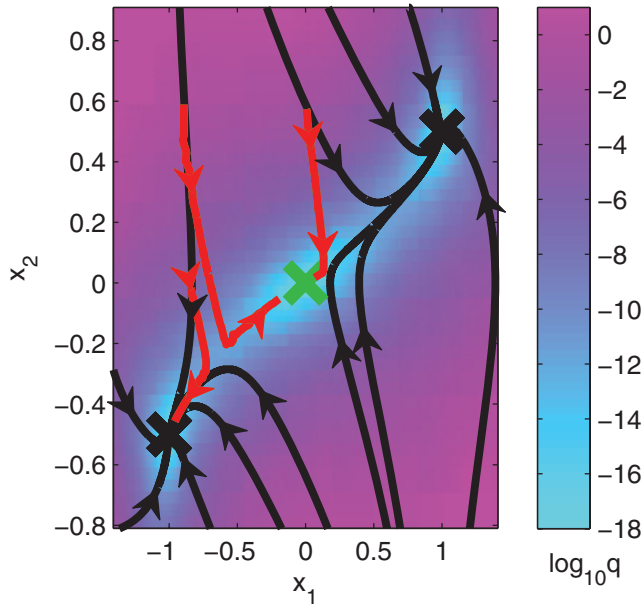

Figure 1: A 2D example of the behavior of minimization of q relative to the defining dynamical system. The trajectories of a simple 2D dynamical system, $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$, defined as $\dot{x}_1 = (1 - x_1^2)x_2$, $\dot{x}_2 = x_1/2 - x_2$ are shown in thick black lines. Shown with a black x are the two attractors of the dynamical system. A saddle point is denoted with a green x. Gradient descent on the auxiliary function q is shown in thick red lines. Minimization of q converges to one of the two attractors or to the saddle point, depending on the IC. The value of q is shown with a color map (bar indicates $\log_{10} q$) and depends on the distance to the fixed points of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$.

defined as $q(\mathbf{x}) = \frac{1}{2}(x_2^2(1 - x_1^2)^2 + (x_1/2 - x_2)^2)$. Note that the fixed points of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ are the zeros of q.

A number of system trajectories defined by evolving the dynamical system $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ are shown in Figure 1 in thick black lines. The three fixed points of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ are shown as large x's. The color map shows the values of $q(\mathbf{x})$, with pink coding large values and cyan coding small values. This color map helps in visualizing how minimization of q proceeds.

Studying the system trajectories alone might lead us to suspect that a saddle point is mediating a choice between two attractors (black x). But the location of this hypothetical saddle is unknown. Using trajectories that are close to the saddle behavior, we can initiate several sample minimizations of q (thick red lines) and see that indeed two of them lead to the saddle point (green x). For this example, two different minimization trajectories originate from the same trajectory of the dynamical system.

3.2 RNNs. Now consider $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ defined as a recurrent neural network,

$$\dot{x}_i = -x_i + \sum_k^N J_{ik} r_k + \sum_k^I B_{ik} u_k, \tag{3.10}$$

$$r_i = h\left(x_i\right), \tag{3.11}$$

where $\mathbf{x}$ is the N-dimensional state called the "activations," and $\mathbf{r} = h(\mathbf{x})$ are the firing rates, defined as the element-wise application of the nonlinear function h to $\mathbf{x}$. The recurrence of the network is defined by the matrix $\mathbf{J}$, and the network receives the I-dimensional input $\mathbf{u}$ through the synaptic weight matrix $\mathbf{B}$. In the cases we study, which are typical of training from random initialization, the matrix elements, J_{ik}, are taken from a normal distribution with zero mean and variance g^2/N, where g is a parameter that scales the network feedback.

After random initialization, the network is trained, typically with a supervised learning method. We assume that such an RNN has already been optimized to perform either an input-output task or a pattern generation task. Our goal is to explore the mechanism by which the RNN performs the task by finding the fixed points and slow points of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ and the behavior of the network dynamics around those points.

To employ the technique on the RNN defined by equations 3.10 and 3.11, we define q as before: $q(\mathbf{x}) = \frac{1}{2} |\mathbf{F}(\mathbf{x})|^2$. The Jacobian of the network, useful for defining the gradient and Hessian of q, is given by

$$\frac{\partial F_i}{\partial x_j} = -\delta_{ij} + J_{ij} r'_j, \tag{3.12}$$

where δ_{ij} is defined to be 1 if $i = j$ and otherwise 0. Also, r'_j is the derivative of the nonlinear function h with respect to its input, x_j.

3.3 3-Bit Flip-Flop Example. We now turn to a high-dimensional RNN implementing an actual task. We first define the task, then describe the training and show the performance of the network. Finally, we use our technique to locate the relevant fixed points of the system and use them to understand the underlying network mechanism.

Consider the three-bit flip-flop task shown in Figure 2. A large recurrent network has three outputs showing the state of three independent memory bits. Transient pulses from three corresponding inputs set the state of these bits. For instance, input 2 (dark green) is mostly zero, but at random times, it emits a transient pulse that is either $+1$ or -1. When this happens, the corresponding output (output 2, light green) should reflect the sign of the last input pulse, taking and sustaining a value of either $+1$ or -1. For

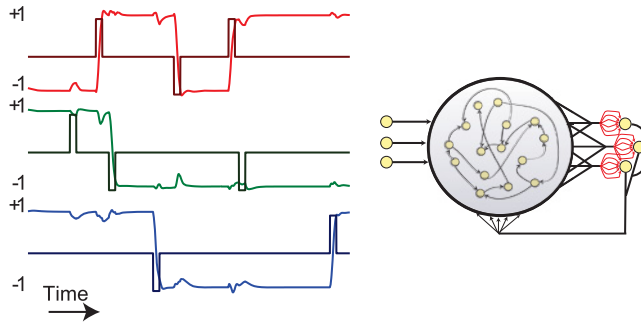

Figure 2: Example inputs and outputs for the 3-bit flip-flop task. (Left) Sample input and output to the 3-bit flip-flop network. The three inputs and outputs are shown in pairs (dark red/red, dark green/green, and dark blue/blue, respectively). Brief input pulses come in with random timing. For a given input-output pair, the output should transition to $+1$ or stay at $+1$ for an input pulse of $+1$. The output should transition to -1 or stay at -1 if the corresponding input pulse is -1. Finally, all three outputs should ignore their nonmatching input pulses. (Right) Network with echostate architecture used to learn the 3-bit flip-flop task. Trained weights are shown in red.

example, the first green pulse does not change the memory state because the green output is already at $+1$. The second green pulse, however, does. Also, the blue and red pulses should be ignored by the green output. An RNN that performs this task for three input-output pairs must then represent $2^3 = 8$ memories and so is a 3-bit flip-flop device, while ignoring the cross talk between differing input-output pairs.

We trained a randomly connected network ($N = 1000$) to perform the 3-bit flip-flop task using the FORCE learning algorithm (Sussillo & Abbott, 2009) (see section 6). We then performed the linearization analysis, using the trajectories of the system during operation as ICs. Specifically, we spanned all possible transitions between the memory states using the inputs to the network and then randomly selected 600 network states out of these trajectories to serve as ICs for the q optimization. The algorithm resulted in 26 distinct fixed points, on which we performed a linear stability analysis. Specifically, we computed the Jacobian matrix, equation 3.12, around each fixed point and performed an eigenvector decomposition on these matrices.

The resulting stable fixed points and saddle points are shown in Figure 3 (left). To display the results of these analyses, the network state $\mathbf{x}(t)$ is plotted in the basis of the first three principal components of the network activations (the transient pulses reside in other dimensions; one is shown in the right panel of Figure 3). In black "x" are the fixed points corresponding to each of the eight memory states. These fixed points are attractors, and the

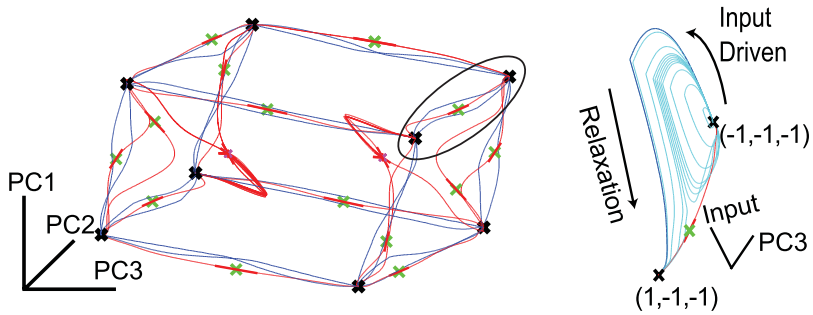

Figure 3: Low-dimensional phase space representation of 3-bit flip-flop task. (Left) The eight memory states are shown as black x. In blue are shown all 24 1-bit transitions between the eight memory states of the 3-bit flip-flop. The saddle fixed points with one unstable dimension are shown with green x. A thick red line denotes the dimension of instability of these saddle points. In thin red are the network trajectories started just off the unstable dimensions of the saddle points. These are heteroclinic orbits between the saddles that decide the boundaries and the attractor fixed points that represent the memories. Finally, the pink x show other fixed points, all with four unstable dimensions. Thick red lines show these dimensions of instability, and thin red lines show network trajectories started just off the fixed points in these unstable dimensions. (Right) Demonstration that a saddle point mediates the transitions between attractors. The input for a transition (circled region in left panel) was varied from 0 to 1, and the network dynamics are shown (blue for normal input pulse and cyan for the rest). As the input pulse increased in value, the network came ever closer to the saddle and then finally transitioned to the new attractor state.

perturbative linear dynamics around each fixed point are stable. Shown in blue are the 24 neural trajectories (activation variable $\mathbf{x}$) showing the effect of an input pulse that flips the 3-bit flip-flop system by one bit. The green crosses are saddles—fixed points with 1 unstable dimension and $N-1$ stable dimensions. Shown in thin red lines are network trajectories started just off the saddle on the dimension of instability. As we will show, the system uses the saddles to mediate the transition from one memory state to another (shown in detail on the right panel). Finally, shown in pink "x" are two fixed points with four unstable dimensions each (thick red lines). Again, sample network trajectories were plotted with the ICs located just off each fixed point on the four dimensions of instability. The unstable trajectories appear to connect three memory states, with no obvious utility.

To verify whether the saddle points are used by the system to switch from one memory to another, we focus on a single transition, specifically from memory $(-1,-1,-1)$ to memory $(1,-1,-1)$. We chose a visualization space that allows both the effects of input perturbation and memory transitions in state space (PC3 and the relevant input vector) to be seen. We varied

the amplitude of the relevant input pulse in small increments from 0 to 1 (see Figure 3, right panel). In blue is shown the normal system trajectory (pulse amplitude 1) during the transition for both the input-driven phase and the relaxation phase. In cyan are shown the results of an input perturbation experiment using intermediate pulse sizes. Intermediate amplitude values result in network trajectories that are ever closer to the saddle point, illustrating its mechanistic role in this particular memory transition.

It is important to note that this network was not designed to have eight attractors and saddle points between them. In fact, as Figure 3 shows, there can be more than one saddle point between two attractors. This is in contrast to networks designed to store eight patterns in memory (Hopfield & Tank, 1986) where no such "redundant" saddle points appear.

3.4 Sine Wave Generator Example. Next, we demonstrate that our analysis method is effective in problems involving pattern generation. Because pattern generation may involve system trajectories that are very far from any particular fixed point, it is not obvious a priori that finding fixed points is helpful. For our pattern generation example, we trained a network ($N = 200$) that received a single input, which specified 1 out of 51 frequency values in radians per unit time. The output of the network was defined to be a sine wave with the specified target frequency and unity amplitude. The target frequency was specified by a static input (see section 6 and Figure 4A). Again, we performed the analysis starting from ICs in the 51 different oscillation trajectories. We took care to include the correct static input value for the optimization of q (see section 6). This means the fixed points found were conditionally dependent on their respective static input value and did not exist otherwise. With this analysis, we found a fixed point for all oscillations centered in the middle of the trajectory.[1]

For all 51 fixed points, a linear stability analysis revealed that the linearized system had only two unstable dimensions and $N - 2$ stable dimensions. A state-space portrait showing the results of the fixed-point and linear stability analyses is shown in Figure 4B. The network state (activation variable $\mathbf{x}$) is plotted on the basis of the first three principal components of the network activations. Shown in blue are the neural trajectories for all 51 oscillations. The green circles show each of the 51 fixed points, with the unstable modes denoted by red lines. As mentioned above, the actual dynamics are far from the fixed points and thus not necessarily influenced by them. To show that the linear analysis can provide a good description of the dynamics, we compared the absolute value of the imaginary components

[1]For the four slowest oscillations, we found exactly two fixed points—one centered in the middle of the oscillation with the other outside the trajectory. All other oscillations had exactly one fixed point. Based on the results of our stability analysis, for these four oscillations with two fixed points, we focused on the fixed point centered in the middle of the oscillation, which was consistent with how the oscillations with one fixed point worked.

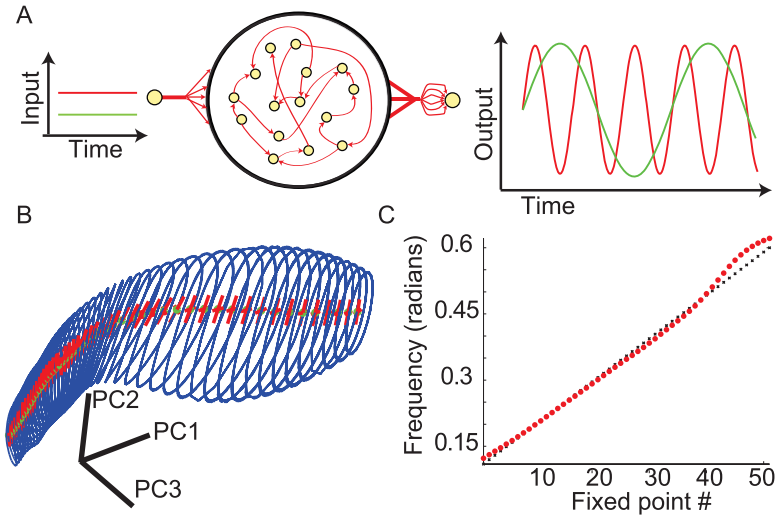

Figure 4: Input-dependent sine generator task. (A) Task definition. The network was given a static input whose amplitude denoted a frequency in radians/sec, and the network was trained to output a sine wave with the specified target frequency with unity amplitude. Shown in red and green are two input-output conditions. The trained weights of the network are shown in red. (B) State-space depiction of trajectories and fixed points (projected to first 3 PCs). Shown in blue are the network trajectories for each of the 51 frequencies during pattern generation. The green circles are 51 fixed points, one for each frequency, that are conditionally defined based on corresponding static input. Shown in red are the two eigenvectors of the unstable plane of the linearized system around each fixed point that are responsible for the oscillatory dynamics. (C) Comparison of the target frequency with that predicted by the fixed points. The abscissa indexes the 51 input dependent fixed points found in the system. The ordinate shows the values of the imaginary part of the eigenvalue of the linearized systems (red) and the desired frequency in radians/sec (black) for all fixed points.

of one of the two unstable dimensions (they come in complex conjugate pairs) to the desired frequency in radians/sec in the target trajectories (see Figure 4C). There is very good agreement between the target frequency and the absolute value of the imaginary part of the unstable mode of the linearized systems (proximity of red circles to the black dots), indicating that the oscillations defined by the linear dynamics around each fixed point explained the behavior of the RNN for all target frequencies. We verified that the validity of the linear approximation stems from inequalities 3 and 4 by computing the norms of the linear and quadratic terms of the Taylor expansion. Indeed, for all 51 trajectories, the norm of the linear term was at least twice that of the quadratic term (not shown).

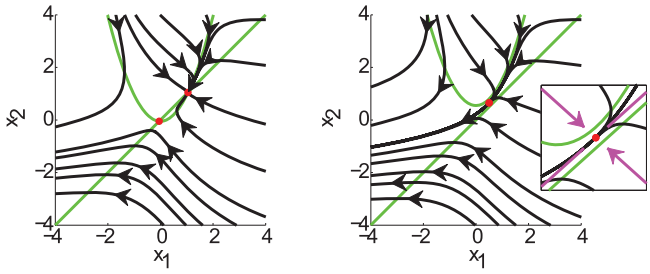

Figure 5: Ghosts and local minima of q. (Left) Dynamics of the system described by equations 4.1 and 4.2 with $a = -0.3$. There are two fixed points, a saddle and a node (red points), both located at the intersection of the two nullclines (green lines), and are global minima of q. (Right) The same system with $a = 0.3$. There are no fixed points, but the local minimum of q (red point) denotes a ghost of a fixed point and clearly influences the dynamics. The linear expansion around this ghost reveals a strongly attracting eigenvector (pink inward arrows), corresponding to the dynamics, and a zero eigenvector (pink lines), along which the dynamics slowly flows because the system is not at a true fixed point.

4 Beyond Fixed Points

While fixed points are obvious candidates for linearization, they are not the only points in phase space where linearization makes sense. Minimizing q can lead to regions with a dominating linear term in their Taylor expansion, even if the zero-order term is nonzero. We first show a 2D example where considering a local minimum of q can provide insight into the system dynamics. Then we analyze a high-dimensional RNN performing a 2-point moving average of randomly timed input pulses and demonstrate that points in phase space with small, nonzero values of q can be instructive in reverse engineering the RNN.

4.1 Local Minima of q. Consider the 2D dynamical system defined by

$$\dot{x}_1 = x_2 - \left(x_1^2 + 1/4 + a\right), \tag{4.1}$$

$$\dot{x}_2 = x_1 - x_2. \tag{4.2}$$

This system has two nullclines that intersect for $a \leq 0$. In this case, q will have two minima coinciding with the fixed points (see Figure 5, left). At $a = 0$, the system undergoes a saddle node bifurcation, and if a is slightly larger than 0, there is no longer any fixed point. The function q, however, still has a local minimum exactly between the two nullclines at $\mathbf{x}^g = (1/2, 1/2 + a/2)$. This can be expected, because q measures the speed of the dynamics, and this "ghost" of a fixed point is a local minimum of speed. The system trajectories

shown in Figure 5 (right) indicate that this point represents a potentially interesting area of the dynamics, as it funnels the trajectories into one common pathway and the dynamics are considerably slower near it (not shown in the figure).

While finding the location of such a ghost can be useful, we will go beyond it and consider the linear expansion of the dynamics around this point:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = F\left(\mathbf{x}^g + \begin{pmatrix} dx_1 \\ dx_2 \end{pmatrix}\right) = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} dx_1 \\ dx_2 \end{pmatrix} - \frac{a}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{4.3}$$

As expected from our discussion of conditions for minima of q, the Jacobian has one zero eigenvalue with eigenvector $(1, 1)$. The second eigenvalue is $-\sqrt{2}$, with an orthogonal eigenvector $(1, -1)$. The full behavior near x_g is therefore attraction along one axis and a slow drift along the other, the latter arising from the fact that x_g is not a true fixed point (pink lines in the figure inset). Though lacking any fixed point, the dynamics of this system can still be understood using linearization around a candidate point, with the additional understanding that the drift will affect the system on a very slow timescale.

In general, when linearizing around a slow point, $\mathbf{x}^s$, the local linear system takes the form

$$\dot{\boldsymbol{\delta}}\mathbf{x} = \mathbf{F}'(\mathbf{x}^s)\, \boldsymbol{\delta}\mathbf{x} + \mathbf{F}(\mathbf{x}^s), \tag{4.4}$$

where the zero-order contribution, $\mathbf{F}(\mathbf{x}^s)$, can be viewed as a constant input to the system. While equation 4.4 makes sense for any point in phase space, it is the slowness of the point as defined by inequalities 3.3 and 3.4 that makes the equation useful. In practice, we have found that the constant term, $\mathbf{F}(\mathbf{x}^s)$, is negligible, but we include it for correctness.

4.2 Approximate Plane Attractor. So far we have shown the utility of considering both fixed points of the dynamics and local minima of q. The inequalities, equations 3.3 and 3.4, suggest, however, that any region with a small value of $q(\mathbf{x})$ is a good candidate for linearization. We demonstrate this point using an RNN trained on a 2-point moving average task. In this task, the network receives analog-valued input pulses at random times. Following each pulse, the network has to output the average of the last two input pulses (see Figure 6).

A perfect implementation of a 2-point moving average requires storage of two values in memory. In a designed network, one might implement such a memory requirement by explicitly constructing a plane attractor: a 2D manifold of attracting fixed points—for example, one dimension to

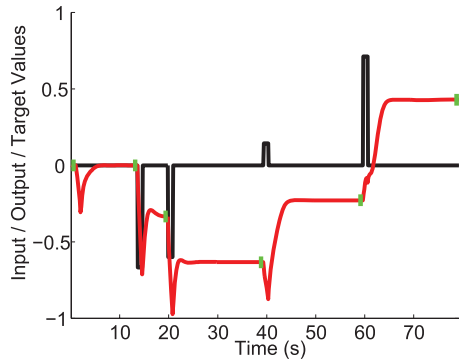

Figure 6: Example inputs, outputs, and targets for the 2-point moving average task. The 2-point moving average task implements the average of the last two input pulses (black). The target moving average is shown in green, and the output of the trained RNN is shown in red.

remember the last input, I1, and another dimension to remember the input before the last, I0.

Continuing with the black box optimization approach, we trained a randomly initialized 100-dimensional RNN to perform this task and then used our algorithm to locate any fixed points, local minima of q, and slow points (see section 6). Our analysis revealed that the network instead created a two-dimensional manifold of slow points, an approximate plane attractor, as shown in Figures 7A and 7B. The black dots in Figure 7A show all points with values of q smaller than 1e-4 in the space of the first three principal components. The linearized dynamics around all slow points had two modes with approximately zero eigenvalues and $N - 2$ fast decaying modes.

In addition to the many slow points, the optimization found two fixed points. These two fixed points organize the extremely slow dynamics on the approximate plane attractor. Shown in Figure 7B is the dynamical structure on the 2D manifold. Each slow point was used as an IC to the network, and the network was run for 1000 s (blue trajectories with arrows). These trajectories demonstrate the precise dynamics at slow timescales that become the approximately fixed dynamics at the timescale of normal network operation. To demonstrate the fast-decaying dynamics to the manifold, the network was run normally with a single trial, which took around 30 s (orange trajectory).

To test the utility of linearizing the RNN around the slow points, we attempted to replace the full nonlinear system with a succession of linear systems defined by the slow points (see Figure 8). Specifically, we defined a linear time-varying dynamical system (LTVDS; see section 6), in which the dynamics at each time point were the linear dynamics induced by the

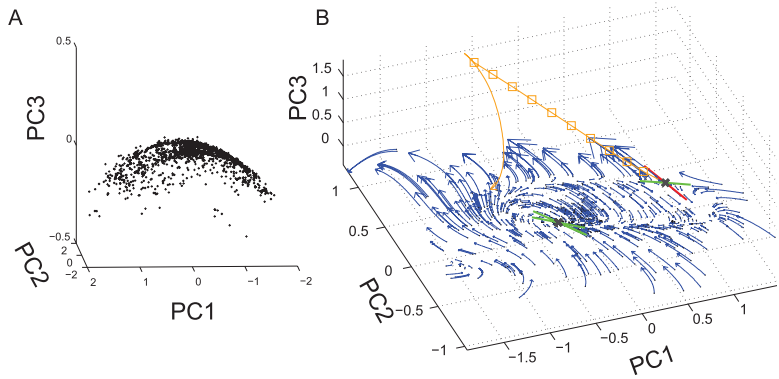

Figure 7: 2D approximate plane attractor in 2-point moving average task. The 2-point moving average task requires two variables to complete the task. The RNN holds those two variables in an approximate plane attractor. (A) The slow points ($q \leq$ 1e-4) form a 2D manifold. The black dots show the slow points of the network. All slow points have two modes with approximately zero eigenvalues. (B) Dynamical picture of 2D manifold of slow points. There are two fixed points (black crosses)—one with two stable directions tangent to the manifold and another with one stable and one unstable dimension tangent to the manifold. The network was simulated for 1000 s from slow points on the manifold ($q \leq$ 1e-7) as the ICs. These slow trajectories are shown with blue lines ending in arrows. To compare with the fast timescales of normal network operation, we show the network computing a moving average that includes a new input. This network trajectory lasted for about 30 s (orange). Each Δt of the 1 s input is shown with orange squares.

closest slow point. We then compared the state transition induced by a single input pulse as implemented by both the RNN and the LTVDS across many random inputs and ICs (Figures 8A and 8B show examples). Note that whenever a new input pulse comes in, the network dynamics must implement two transitions. The I1 value must shift to the I0 value, and the latest input pulse value must shift to the I1 value. Our test of the LTVDS approximation thus measured the mean squared error between the RNN and the LTVDS for the shifted values I1 to I0 and input to I1 (see Figures 8C and 8D, respectively). The errors were 0.074, and 0.0067 for I1→I0 and input→I1.

Finally, we demonstrate that our method is not dependent on a finely tuned RNN. We structurally perturbed the RNN that implemented the 2-point moving average by adding to the recurrent matrix random gaussian noise with zero mean and standard deviation, η (see section 6). The dependency of the output error on the size of the structural perturbation is shown in Figure 9A. An example perturbed RNN with network inputs and output

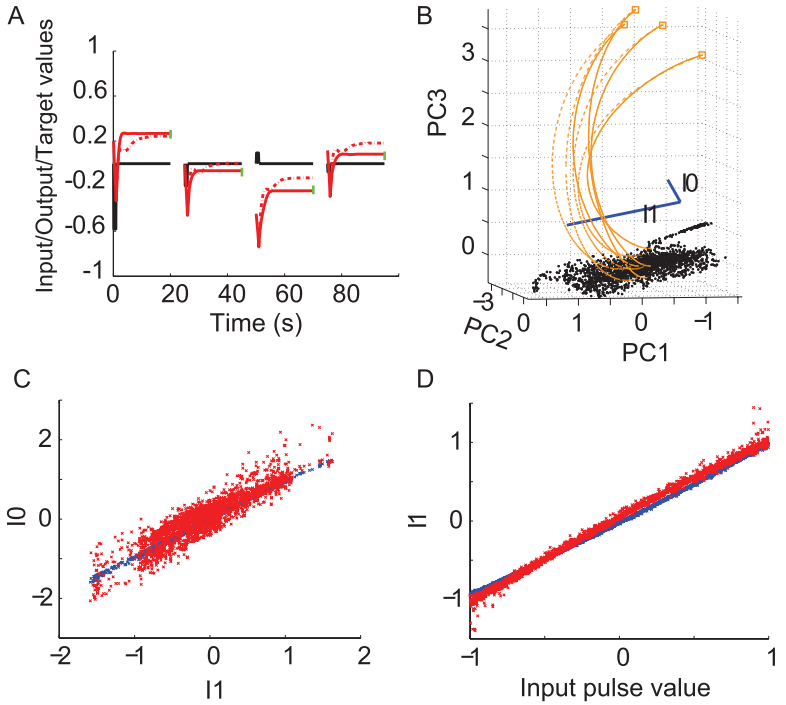

Figure 8: Comparison of the transition dynamics between the RNN and the linear time-varying approximation. (A) Four separate trials of the RNN (red lines) and the LTVDS approximation (dotted red lines). (B) The same trials plotted in the space of the first three PCs of the network activity. The orange squares show the ICs, based on the input pulse, given to both RNN and LTVDS. The RNN trajectories are orange; the LTVDS are dotted orange. Black dots are the 2000 slow points used in simulating the LTVDS. (C, D) In implementing the 2-point moving average, whenever a new input pulse comes in, the RNN must successfully shift the value of the last input (I1) to the value of the input before the last (I0) (panel C) and move the input pulse value to the the value of the last input (I1) (panel D). In blue are the transitions implemented by the RNN. In red are the transitions implemented by the LTVDS using the same ICs. The mean squared error between the LTVDS and the RNN for both transitions was 0.074 and 0.0067, respectively.

is shown in Figure 9B, with $\eta = 0.006$. Unsurprisingly, the main problem is a drift in the output value, indicating that the quality of the approximate plane attractor has been heavily degraded. We applied our technique to the structurally perturbed network to study the effects of the perturbation. Figures 9C and 9D show that the same approximate plane attractor dynamical structure is still in place, though much degraded. The slow point

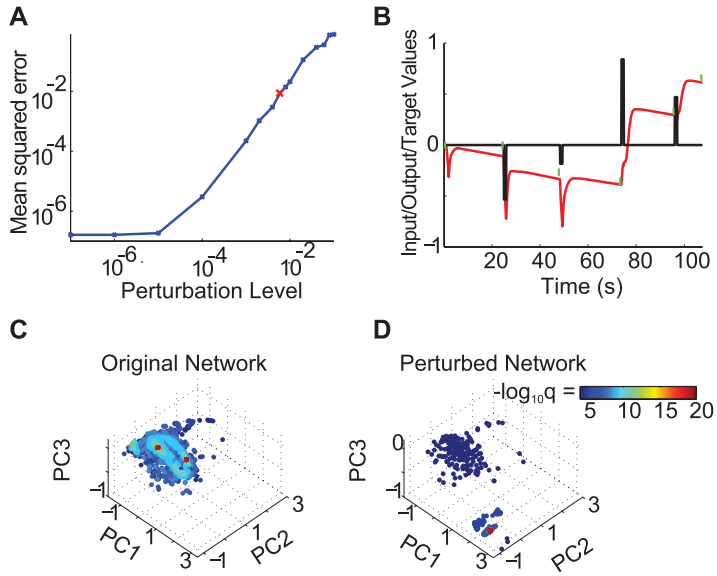

Figure 9: Structural perturbations. (A) To study the ability of our method to find meaningful structure in a perturbed network, gaussian noise was added to the recurrent matrix of the RNN with 0 mean and standard deviation η. We show a plot of mean squared error of the RNN output as a function of the noise level, η. The red x marks the $\eta = 0.006$ noise level used in further studies. (B) Example of perturbed network with $\eta = 0.006$. Same color scheme as Figure 6. (C) A view of the 2D slow point manifold showing the variability in values of q along the manifold for the original network. (D) Similar to panel C for the perturbed network. Note that all q values are higher in this case.

manifold is still in the same approximate place, although the q values of all slow points increased. In addition, there is a fixed point detached from the manifold. Nevertheless, the network is able to perform with mean squared error less than 0.01, on average.

5 Discussion

Recurrent neural networks, when used in conjunction with recently developed training algorithms, are powerful computational tools. The downside of these training algorithms is that the resulting networks are often extremely complex so their underlying mechanism remains unknown. Here we used methods from nonlinear dynamical system theory to pry open the black box. Specifically, we showed that finding fixed points and slow points (points where inequalities 3.3 and 3.4 are satisfied) near relevant network trajectories and studying the linear approximations around those points is

useful. To this end, we outlined an optimization approach to finding slow points, including fixed points, based on the auxiliary function q. Depending on use, minimization of this function finds global and local minima and also regions in phase space with an arbitrarily specified slowness.

We demonstrated how true fixed points were integrally involved in how an RNN solves the 3-bit flip-flop task, creating both the memory states and the saddle points to transition between them. We also analyzed an RNN trained to generate an input-dependent sine wave, showing that input-dependent fixed points were responsible for the relevant network oscillations, despite those oscillations occurring far from the fixed points. In both cases, we found that saddle points with primarily stable dimensions and only a couple of unstable dimensions were responsible for much of the interesting dynamics. In the case of the 3-bit flip-flop, saddles implemented the transition from one memory state to another. For the sine wave generator, the oscillations could be explained by the slightly unstable oscillatory linear dynamics around each input-dependent saddle point.

Finally, we showed how global minima, local minima, and regions of slow speed were used to create an approximate plane attractor, necessary to implement a 2-point moving average of randomly timed input pulses. The approximate plane attractor was implemented using mainly slow points along a 2D manifold.

Studying slow points, and not only fixed points, is important for several reasons. First, a trained network does not have to be perfect: few realistic tasks require infinite memory or precision, so it could be that forming a slow point is a completely adequate solution for the defined timescales of a given input-output relationship. Second, in the presence of noise, the behavior of a fixed point and a very slow point might be indistinguishable. Third, funneling network dynamics can be achieved by a slow point, and not a fixed point (see Figure 5). Fourth, neither trained networks nor biological ones can be expected to be perfect. Studying the slow points of the perturbed plane attractor showed that our technique is sensitive enough to detect dynamical structure in a highly perturbed or highly approximate structure.

The 3-bit flip-flop example was trained using the FORCE learning rule (Sussillo & Abbott, 2009), which affected the network dynamics by modifying the three output vectors. The input-dependent sine wave generator and 2-point moving average were trained using the Hessian-free optimization technique for RNNs (Martens & Sutskever, 2011), and all the synaptic strengths and biases were modified. Despite the profound differences in these two training approaches, our fixed-point method was able to make sense of the network dynamics in both cases.

Although the RNNs we trained were high dimensional, the tasks we trained them on were intentionally simple. It remains to be seen whether our technique can be helpful in dissecting higher-dimensional problems, and almost certainly it depends on the task at hand as the input and task definition explicitly influence the dimensionality of the network state.

Our technique, and the dynamical concepts explored using it, may also provide insight into the observed experimental findings of low-dimensional dynamics in *in vivo* neuroscience experiments (Romo, Brody, Hernández, & Lemus, 1999; Stopfer, Jayaraman, & Laurent, 2003; Jones, Fontanini, Sadacca, Miller, & Katz, 2007; Ahrens et al., 2012). For example, we found that a saddle point funnels large volumes of phase space into two directions and so may be a dynamical feature used in neural circuits that inherently reduces the dimension of the neural activity. Since the networks we studied were trained as opposed to designed, and therefore initially their implementation was unknown, our method and results are compatible with studies attempting to understand biological neural circuits, for example (Huerta & Rabinovich, 2004; Machens, Romo, & Brody, 2005; Rabinovich et al., 2008; Liu & Buonomano, 2009; Ponzi & Wickens, 2010).

There are several possible interesting extensions to this technique. While the system and examples we studied were continuous in nature, the application of our technique to discrete RNNs presents no difficulties. Another extension is to apply the fixed-point and slow-point analysis to systems with time-varying inputs during those times that the input is varying. Because the auxiliary function we used relies only on part of the inequalities, equations 3.3 and 3.4, it is possible that a more elaborate auxiliary function could be more efficient. Finally, an interesting direction would be to apply this technique to spiking models.

6 Methods

6.1 Minimizing $q(\mathbf{x})$. Our proposed technique to find fixed points, ghosts, and slow points of phase space involves minimizing the nonconvex, nonlinear scalar function $q(\mathbf{x}) = \frac{1}{2}|\mathbf{F}(\mathbf{x})|^2$. Our implementation involves using the function *fminunc*, Matlab's unconstrained optimization front end. For effective optimization, many optimization packages, including *fminunc*, require as a function of a particular $\mathbf{x}$ the gradient (see equation 3.6) and the Hessian (see equation 3.7). Because $q(\mathbf{x})$ is a sum of squared function values, we implemented instead the Gauss-Newton approximation to the Hessian. We found it to be effective and simpler to compute (Boyd & Vandenberghe, 2004). It is defined by neglecting the second term on the right-hand side of equation 3.7:

$$G_{ij} = \sum_{k}^{N} \frac{\partial F_k}{\partial x_i} \frac{\partial F_k}{\partial x_j}. \tag{6.1}$$

The Hessian matrix approaches the Gauss-Newton matrix as the optimization approaches a fixed point. In summary, the core of our implementation included a single function that computes three objects as a function of a

specific state vector: $q(\mathbf{x})$, $\frac{\partial q}{\partial \mathbf{x}}(\mathbf{x})$ (see equation 3.6) and $\mathbf{G}(\mathbf{x})$ (see equation 6.1) for the generic RNN network equations defined by equation 6.4.

Since $q(\mathbf{x})$ is not convex, it is important to start the minimization from many different ICs. For all the RNN examples, we first ran the RNN to generate state trajectories during a typical operation of the RNN. Random subsets of these state trajectories were chosen as ICs for the optimization of $q(\mathbf{x})$. After the optimization terminated, depending on whether the value of $q(\mathbf{x})$ was below some threshold, either the optimization was started again from a different IC or the optimization routine ended successfully. To find fixed points and ghosts, the optimization was run until the optimizer returned either success or failure. Assuming a successful optimization run, we called a candidate point a fixed point if the value of q was on the order of magnitude of the tolerances set in the optimization. We called a candidate point a ghost if q was greater than those tolerances, indicative of a local minimum. To find arbitrarily defined slow points, the value of q was monitored at every iteration, and if the value fell below the arbitrary cutoff, the optimization was terminated successfully at that iteration.

For the 3-bit flip-flop and sine wave generator examples, the function tolerance was set to 1e-30.

For the 3-bit flip-flop and 2-point moving average, the input was not used when finding fixed points, as the inputs were simple pulses and we were interested in the long-term behavior of the two systems. However, for the sine wave generator, since the input conditionally defines a new dynamical system for each oscillation frequency, the input was used in finding the fixed points. Concretely, this means $\dot{\mathbf{x}}$ in equation 3.6 included the static input that specified the target frequency and that the fixed points found did not exist in the absence of this input.

6.2 3-Bit Flip-Flop. We extend equations 3.10 and 3.11 to include inputs and outputs of the network,

$$\dot{x}_i = -x_i + \sum_k^N J_{ik} r_k + \sum_k^3 W_{ik}^{FB} z_k + \sum_k^3 B_{ik} u_k, \tag{6.2}$$

$$z_i = \sum_k^N W_{ik} r_k, \tag{6.3}$$

where $\mathbf{u}$ is a 3D input variable coupled to the system through the input weight matrix $\mathbf{B}$. The output $\mathbf{z}$ is the 3D linear readout of the network defined by the output matrix $\mathbf{W}$. In echostate networks, the output is fed back into the system via feedback weights, here denoted $\mathbf{W^{FB}}$. In this example, tanh was used as the nonlinear transfer function, h. Randomly timed input pulses were placed on the input lines and the targets were defined as in

section 3.3. The result of the FORCE training algorithm (Sussillo & Abbott, 2009) was a modification to the output weights **W**, which, due to output feedback, is a rank 3 modification to the effective recurrent connectivity and thus a modification of the network dynamics (see Figure 2, right panel). As the sample input and output traces on the left panel of Figure 2 show, the trained network performed the task well.

6.3 Sine Wave Generator. We used a randomly connected network of the form defined by

$$\dot{x}_i = -x_i + \sum_k^N J_{ik} r_k + \sum_k^I B_{ik} u_k + b_i^x, \tag{6.4}$$

$$z = \sum_k^N w_k r_k + b^z, \tag{6.5}$$

with network dimension $N = 200$ and input dimension $I = 1$. We used tanh as the nonlinearity and added bias terms b_i^x and b^z to the two equations, respectively. To allow the training to go through, before each training batch, the network was initially driven with the signal $\sin(\omega_j t) + j/51 + 0.25$ to set the initial state of the network, where j indexes the 51 different frequency values for the sine waves. During training, the input signal was static at $j/51 + 0.25$. The target frequencies, ω_j, were equally spaced between 0.1 and 0.6 radians/sec. After training, the 1D network readout, z, successfully generated $\sin(\omega_j t)$ with only the static input $j/51 + 0.25$ driving the network (no initial driving sinusoidal input was necessary). We trained the network to perform this task using the Hessian-free optimization technique for RNNs (Martens & Sutskever, 2011). The result of this training algorithm was a modification to matrices **B** and **J**, vector **w**, and biases $\mathbf{b}^x$ and b^z.

6.4 2-Point Moving Average. The network trained to implement the 2-point moving average was of the same form as equations 6.4 and 6.5. The size of the network was 100 dimensions, with two inputs and one output. The first input contained randomly timed pulses lasting 1 second, with each pulse taking a random value between -1 and 1 uniformly. The second input was an indicator variable, giving a pulse with value 1 whenever the first input was valid and was otherwise 0. The minimum time between pulses was 5 s, and the maximum was 30 s. The target for the output was the average of the last two input pulses. To minimally affect the dynamics, the target was specified for only a brief period before the next input pulse and was otherwise undefined (see the green traces in Figure 6). The target was also defined to be zero at the beginning of the simulation. This network was also trained using the Hessian-free optimization technique for RNNs

(Martens & Sutskever, 2011). The network mean squared error was on the order of 1e-7.

For Figure 7, we selected the best local minima we could find by setting the optimizer tolerance to 1e-27 with a maximum number of iterations set to 10,000. This resulted in two ghosts or fixed points whose values were on the order of 1e-21. Given the potential for pathological curvature (Martens, 2010) in q in an example with a line or plane attractor, it is possible that the two fixed points found are actually ghosts. However, the value of q at these two points is so small as to make no practical difference between the two, so we refer to them as fixed points in the main text, since the dynamics are clearly structured by them.

We also selected slow points by varying the slow point acceptance (q value) between 1e-19 and 1e-4, with 200 attempts for a slow point per tolerance level. Slow points with a q value greater than the tolerance were discarded. To visualize the approximate plane attractor, we performed principal component analysis on the slow points of the unperturbed system with $q \leq$ 1e-4. The subspace shown is the first three principal components.

In the comparison of the LTVDS against the full RNN, we compared the transitions of the current input to the last input (I1) and to the input before last (I0). We decoded both I1 and I0 using the optimal linear projection found by regressing the respective I0 or I1 values against the network activations across all trials.

The linear time-varying dynamical system was defined as follows:

$$
\begin{aligned}
&\text{for } t = 1 \text{ to } T, \text{ steps of } \Delta t \\
&\qquad \mathbf{x}^s = \text{closest slow point to } \mathbf{x}(t) \\
&\qquad \boldsymbol{\delta}\mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}^s \\
&\qquad \dot{\boldsymbol{\delta}}\mathbf{x}(t) = \mathbf{F}'(\mathbf{x}^s)\ \boldsymbol{\delta}\mathbf{x}(t) + \mathbf{F}(\mathbf{x}^s) \\
&\boldsymbol{\delta}\mathbf{x}(t + \Delta t) = \boldsymbol{\delta}\mathbf{x}(t) + \Delta t\ \dot{\boldsymbol{\delta}}\mathbf{x}(t) \\
&\mathbf{x}(t + \Delta t) = \mathbf{x}^s + \boldsymbol{\delta}\mathbf{x}(t + \Delta t) \\
&\text{end},
\end{aligned}
$$

where $\mathbf{x}^s$ is the closest slow point to the current state and $\mathbf{F}'(\mathbf{x}^s)$ is the Jacobian around that slow point. The set of potential $\mathbf{x}^s$ was 2000 slow points from the 2D manifold, found by setting the optimizer tolerances such that the q optimization found slow points with values just below 1e-10. This value was chosen as a compromise between the quality of the linearized system at the slow point, which requires a stringent q tolerance, and full coverage of the approximate plane attractor, which requires a larger q tolerance. The transition simulations were initialized by choosing a random input pulse value for every slow point. Then the full RNN was simulated for 1 s to

correctly simulate the strong input. The network state at the end of the 1 s was used as the IC for the comparison simulations of both the RNN and the LTVDS. The simulations were run for 20 s, allowing for equilibration to the 2D manifold for both the full RNN and the LTVDS approximation.

For the perturbation experiments, we chose η values between 1e-7 and 0.1. For each value of η, we generated 50 perturbed networks where the perturbation was zero-mean gaussian noise with standard deviation η added to the recurrent weight matrix. For each of the 50 networks, 1000 random examples were evaluated. The error for each η value was averaged across networks and examples. The approximate plane attractor shown in Figure 9D is with $\eta = 0.006$, a large η value, but where the network still functioned with some drift.

Acknowledgments

We thank Misha Tsodyks, Merav Stern, Stefano Fusi, and Larry Abbott for helpful conversations. D.S. was funded by NIH grant MH093338. OB was funded by the Gatsby and Swartz Foundations and also by NIH grant MH093338.

References

Ahrens, M. B., Li, J. M., Orger, M. B., Robson, D. N., Schier, A. F., et al. (2012). Brain-wide neuronal dynamics during motor adaptation in zebrafish. *Nature*, *485*(7399), 471–477.

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

Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge: Cambridge University Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *PNAS*, *79*(8), 2554–2558.

Hopfield, J. J., & Tank, D. (1986). Computing with neural circuits: A model. *Science*, *233*(4764), 625–633.

Huerta, R., & Rabinovich, M. (2004). Reproducible sequence generation in random neural ensembles. *Physical Review Letters*, *93*(23), 238104.

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, *304*(5667), 78–80.

Jones, L. M., Fontanini, A., Sadacca, B. F., Miller, P., & Katz, D. B. (2007). Natural stimuli evoke dynamic sequences of states in sensory cortical ensembles. *PNAS*, *104*(47), 18772–18777.

Liu, J. K., & Buonomano, D. V. (2009). Embedding multiple trajectories in simulated recurrent neural networks in a self-organizing manner. *J. Neuroscience*, *29*(42), 13172–13181.

Maass, W., Joshi, P., & Sontag, E. D. (2007). Computational aspects of feedback in neural circuits. *PLoS Comput. Biol.*, *3*(1), e165.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560.

Machens, C. K., Romo, R., & Brody, C. D. (2005). Flexible control of mutual inhibition: A neural model of two-interval discrimination. *Science*, *307*(5712), 1121–1124.

Martens, J. (2010). Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*. Madison, WI: Omnipress.

Martens, J., & Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*. Madison, WI: Omnipress.

Ott, E. (2002). *Chaos in dynamical systems*. Cambridge: Cambridge University Press.

Ponzi, A., & Wickens, J. (2010). Sequentially switching cell assemblies in random inhibitory networks of spiking neurons in the striatum. *Journal of Neuroscience*, *30*(17), 5894–5911.

Rabinovich, M. I., Huerta, R., Varona, P., & Afraimovich, V. S. (2008). Transient cognitive dynamics, metastability, and decision making. *PLoS Comput. Biol.*, *4*(5), e1000072.

Romo, R., Brody, C. D., Hernández, A., & Lemus, L. (1999). Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature*, *399*(6735), 470–473.

Rumelhart, D., Hinton, G., & Williams, R. (1985). Learning internal representations by error propagation. *Parallel Distributed Processing*, *1*, 319–362.

Stopfer, M., Jayaraman, V., & Laurent, G. (2003). Intensity versus identity coding in an olfactory system. *Neuron*, *39*(6), 991–1004.

Strogatz, S. H. (1994). *Nonlinear dynamics and chaos, with applications to physics, biology, chemistry, and engineering*. Boulder, CO: Westview Press.

Sussillo, D., & Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, *63*(4), 544–557.

Wang, X. J. (2008). Decision making in recurrent neuronal circuits. *Neuron*, *60*(2), 215–234.

Received July 8, 2012; accepted September 17, 2012.