Engineering recurrent neural networks from task-relevant manifolds and dynamics

Eli Pollock[1], Mehrdad Jazayeri[1*]

[1]Department of Brain & Cognitive Sciences, McGovern Institute for Brain Research,

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA

***Corresponding author:** mjaz@mit.edu (MJ)

Abstract

Many cognitive processes involve transformations of distributed representations in neural populations, creating a need for population-level models. Recurrent neural network models fulfill this need, but there are many open questions about how their connectivity gives rise to dynamics that solve a task. Here, we present a method for finding the connectivity of networks for which the dynamics are specified to solve a task in an interpretable way. We apply our method to a working memory task by synthesizing a network that implements a drift-diffusion process over a ring-shaped manifold. We also use our method to demonstrate how inputs can be used to control network dynamics for cognitive flexibility and explore the relationship between representation geometry and network capacity. Our work fits within the broader context of understanding neural computations as dynamics over relatively low-dimensional manifolds formed by correlated patterns of neurons.

Author Summary

Neurons in the brain form intricate networks that can produce a vast array of activity patterns. To support goal-directed behavior, the brain has to adjust the connections between neurons so that network dynamics can perform desirable computations on behaviorally relevant variables. A fundamental goal in computational neuroscience is to provide an understanding of how network connectivity aligns the dynamics in the brain to the dynamics needed to track those variables. Here, we develop a mathematical framework for creating recurrent neural network models that can address this problem. Specifically, we derive a set of linear equations that constrain the connectivity to afford a direct mapping of task-relevant dynamics onto network activity. We demonstrate the utility of this technique by creating and analyzing a set of network models that can perform a simple working memory task. We then extend the approach to show how additional constraints can furnish networks whose dynamics are controlled flexibly by external inputs. Finally, we exploit the flexibility of this technique to explore the robustness and capacity limitations of recurrent networks. This network synthesis method provides a powerful means for generating and validating hypotheses about how task-relevant computations can emerge from network dynamics.

Introduction

As it becomes possible for neuroscientists to simultaneously record ever-larger numbers of neurons [1], there is a need for theoretical frameworks and models to make sense of the resulting data and explain how behavior arises from the cooperation of many neurons. Rising to this challenge, the field of computational neuroscience has established that cortical neurons can and do perform distributed computations through population-level dynamics [2]. This finding necessitates further development of data analysis techniques and models that make population-level explanations and predictions.

In studying behavior, an important aim is to develop models that incorporate a set of latent variables that can parsimoniously explain observable behavioral states. For instance, in decision-making tasks, drift-diffusion models have explained choice behavior in terms of a one-dimensional latent decision variable [3,4]. A more general framework that can accommodate behaviors with multiple latent variables is to consider a “latent task space” whose dimensions represent those variables (Fig. 1). Within this task space, one can specify (1) the subregion the latent variables occupy during the task (“latent task manifold”), and (2) the dynamics with which those variables evolve (“latent task dynamics”).

To understand how patterns of neural activity give rise to behavioral variables, we need an analogous state space description of neural signals. Neural state space can be straightforwardly defined as a coordinate system in which the instantaneous firing rate of each neuron represents a dimension. Within this framework, the key to understanding the connection between neurons and behavior is to characterize the mapping between the behavioral and neural state spaces. Since the number of behavioral variables is typically much smaller than the number of underlying neurons, numerous studies have sought to link behavior to a set of “latent neural dimensions,” that create a lower-dimensional “latent neural space” during task performance [5,6]. While there is no guarantee of a one-to-one match between the latent neural dimensions and latent task dimensions, mounting evidence suggests the task manifold might be nonlinearly embedded in latent neural space as a “latent neural manifold” (Fig. 1) [7,8].

So far, this provides a descriptive account of how the low-dimensional geometry of neural activity can relate to task-relevant computations. However, manifolds are only abstract mathematical constructs that result from data analysis. Can we provide a generative account for how a population of neurons can support particular manifold geometries? Specifically, can we create a model whose dynamics give rise to a neural manifold that is an arbitrarily embedded task manifold?

To answer this, we turn to recurrent neural network (RNN) models. RNNs capture the recurrent, distributed nature of neural computation and are theoretically able to approximate any dynamical system [9]. Recent advances in machine learning and computing capability have made their use practical for a variety of applications in computational neuroscience. In some studies, RNNs are optimized to reproduce neural data [10,11]. Other studies take a task-oriented approach, training a network to perform a task and then attempting to find similarities between the RNN's population dynamics and those of biological neurons recorded from an animal performing a similar task [12,13]. It is also possible to take a hybrid approach, training a network on a task with constraints that yield more brain-like solutions [14,15].

All of these training approaches can be used to generate hypotheses about how networks solve tasks, as they are generally agnostic to any specific solutions [16]. Valuable insight can come from exploring the geometry of the neural manifolds created by such networks [17,18]. Additionally, one can apply analysis techniques from dynamical systems theory to characterize fixed points and other dynamical features that determine how network states evolve through time [19]. Overall, studying trained RNNs answers questions about the kinds of dynamics that can be used to solve tasks. We are concerned with exploring the inverse question: what kinds of networks are capable of implementing a given low-dimensional dynamical system (Fig. 1)? This requires a synthesis-based approach, which has been explored in some studies but is far from a settled question [20,21].

Here, we propose a novel method for creating RNNs that can map latent task manifolds to arbitrary neural manifolds. This allows us to create RNNs that can explore a range of dynamical solutions to tasks. We use our method to consider how inputs can be used to perform flexible computations, and explore how different embeddings of the task manifold in the neural space can affect network performance and connectivity.

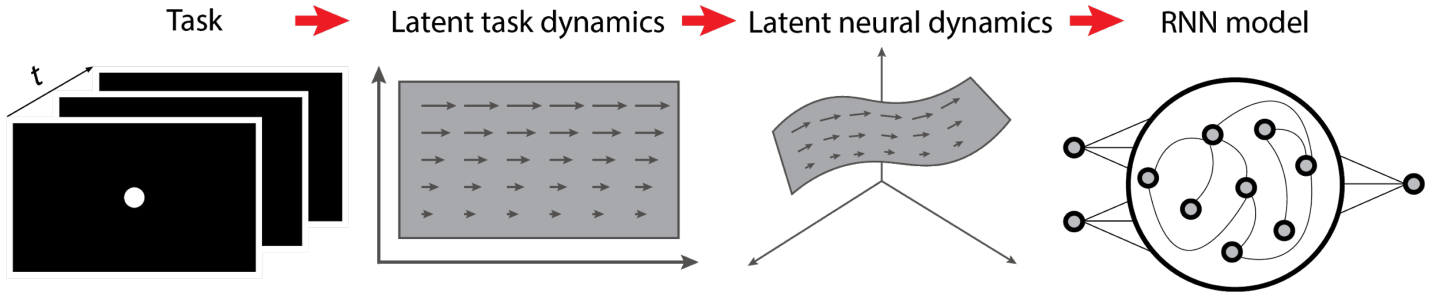


Figure 1 Theoretical framework Left: A hypothetical task involving latent variables. MIddle left: The evolution of these variables can be represented in a latent task space (gray rectangle). In this illustration, adapted from [22], the task might be a time interval production task, where the horizontal axis represents the relative elapsed time. The vertical axis represents the interval duration by a latent variable that specifies the speed of evolution in the horizontal direction, with faster speed (higher on the vertical axis; larger arrows) corresponding to shorter intervals. Middle right: A nonlinear embedding of the task manifold in a neural state space. Right: An RNN models that establishes that nonlinear embedding.

Results

Creating networks that embody task-relevant latent dynamics

Our overarching objective is to examine the computational properties of RNNs whose state dynamics capture the evolution of latent variables in a task, which might be inferred from behavioral models. Since different tasks demand different latent-variable dynamics, as a first step we need a technique for creating RNNs whose state dynamics can be engineered. Here, we describe an approach that achieves this goal rapidly and flexibly.

We start by considering a case in which the objective dynamics are known and we want to synthesize an RNN that can emulate those dynamics. We consider the class of RNNs in which the dynamics of the units are characterized by a differential equation as follows:

$$(1) \quad F(x) = \frac{dx}{dt} = \frac{1}{\tau}(-x + W^T\phi(x) + I)$$

In this equation, *x* is an *N*-dimensional vector specifying the activity of all *N* units in the network, τ is the time constant of the units, *W* is an *N*-by-*N* matrix specifying the synaptic weights between units, and *I* is a vector of inputs into each unit. The superscript T signifies transpose operation. The function $\phi(x)$ is a monotonic, differentiable nonlinearity that transforms the activity into a "firing rate." Here, we use $\phi(x) = \tanh(x)$.

We sought to create the objective dynamics in the RNN by matching the local partial derivatives of the network to that of the objective dynamics. In vector calculus, the matrix of local partial derivatives is known as the Jacobian. As such, we have to adjust *W* so that the Jacobian of the network, denoted J_{RNN}, would match the Jacobian of the objective dynamics, denoted J_{obj}. For the network, J_{RNN} can be written as follows:

$$(2) \quad J_{RNN} = \frac{\delta F_i}{\delta x_j} = \frac{1}{\tau}\left(-\mathbf{1}_N + W^T\Phi\right), \text{ where } \Phi = \text{diag}\left(\phi'(x)\right)$$

with *diag* indicating a matrix with a specified vector along its diagonal and zeros everywhere else. The matrix $\mathbf{1}_N$ is the identity matrix of size *N*.

To adjust RNN dynamics along different dimensions, we used eigendecomposition to factorize J_{RNN} to a set of eigenmodes. Each eigenmode is characterized by an eigenvector, which specifies a single dimension within the state space, and a corresponding eigenvalues that quantifies the rate and direction of movement along that dimension [23]. If we collect the *N* eigenvectors within a matrix U and the eigenvalues within a diagonal matrix Σ, J_{RNN}can be factorized as $U\Sigma U^T$. After substituting J_{RNN} with this eigendecomposition and some linear algebra, we can rewrite (2) as follows:

$$(3) \quad U^T \Phi W = \tau \left(\Sigma + \frac{1}{\tau} \mathbf{1}_N \right) U^T$$

In principle, we can find *W* by replacing U and Σ by their corresponding values based on J_{obj}, and solve for *W*. For example, if we are defining a point attractor, we would specify all eigenvalues to be negative, meaning that perturbations away from the point will decay. However, we have to address one problem beforehand: usually, the dimension of J_{RNN} and J_{obj} do not match: the dimensionality of J_{RNN} is specified by the number of units (*N*), whereas the dimensionality of J_{obj}, denoted *d*, is determined by the number of latent variables needed to perform a given task, and *d* has to be smaller than *N*. From a geometrical perspective, this means that *W* should be adjusted such that the network is low-rank; i.e., activity must reside within a *d*-dimensional subspace associated with the latent dynamics.

Since the number of task-relevant latent variables are usually far smaller than the size of the network [8], J_{obj} can only be used to constrain the first *d* eigenmodes in the state space. To handle the other dimensions, we employ a simple trick: we set the eigenvalues of the remaining *N*-*d* dimensions to a negative value ($-1/\tau$). This serves two purposes. First, it ensures that activity remains within the desired *d*-dimensional subspace (i.e, activity along other dimensions would decay), and second, it allows us to ignore these dimensions and rewrite (3) for the d eigenmodes associated with J_{obj}:

$$(4) \quad U_{obj}^T \Phi W = \tau \left(\Sigma_{obj} + \frac{1}{\tau} \mathbf{1}_d \right) U_{obj}^T$$

where U_{obj}^T contains the *d* eigenvectors of J_{obj} embedded in an *N*-dimensional space (*N*-by-*d*), Σ_{obj} contains the corresponding eigenvalues, and the identity matrix is now of size *d*. This equation can be further simplified to the following form:

$$(5) \quad A_{x_0} W = B_{x_0}$$

Since J_{obj} is *d*-dimensional, equation 5, which is written for a single point in the state space (subscript x_0), provides *d* linear constraints on the connectivity matrix. However, we can rewrite (5) for some number *m* points in the state space for which J_{obj} is defined, and create a system of linear equations to solve for the N^2 unknowns in *W*:

$$(6) \quad \begin{bmatrix} A_{x_0} \\ \dots \\ A_{x_m} \end{bmatrix} W = \begin{bmatrix} B_{x_0} \\ \dots \\ B_{x_m} \end{bmatrix}$$

Using this method, which we refer to as Embedding Manifolds with Population-level Jacobians (EMPJ), we can create an RNN whose activity is confined to a desired manifold and whose slow dynamics over that manifold are fully specified by some objective dynamics (see Methods for full details).

A ring attractor with discrete fixed points

To examine the utility of EMPJ, we attempted to construct a ring attractor that contains a set of discrete fixed points. This choice was motivated by the fact that (1) ring attractors have long served as a canonical example of constrained dynamics [24], and (2) discrete fixed points can be used introduce error-correcting dynamics over the ring. Such semi-discrete ring-attractor dynamics have been implicated in the study of human visual working memory of color [25]. When humans report of a previously seen color after a delay over a color wheel (Fig. 2a, left), their responses exhibit biases that can be captured by fixed-point dynamics over a ring attractor; i.e, with longer delays, the reported color drifts slowly over the color wheel toward a stable set of colors. This behavior can be captured by a one-dimensional drift-diffusion model over the ring that specifies the relaxation dynamics of a single latent variable associated with the internal memory of the color. This behavior of the model depends on two key parameters: a drift function that specifies the average movement direction and speed as a function of position on the ring (Figure 2a, middle), and the noise that causes the internal state to diffuse (see Methods).

Accordingly, we need to construct an RNN whose activity resides on an embedded ring manifold, and whose activity dynamics matches that of a desired drift-diffusion model. For this example, we used a sinusoidal drift function with a period of 60 degrees so that the ring contained six equidistant and alternating stable and unstable fixed points (Figure 2b). The number of fixed points can be changed by changing the frequency of the drift function. Next, we need to create a matching ring manifold in the RNN. We can achieve this in five steps. First, we define an arbitrary 2D subspace (plane) within the state space that would contain the desired ring manifold. Second, we choose a set of points along the ring to construct the equations in (6). We will refer to these as setpoints. Third, for every setpoint, we set the eigenvalue associated with radial eigenvector to a negative constant (see Methods for complete details). This ensures that the embedded ring is stable. Fourth, we must specify the relaxation dynamics over the ring. To do so, for every setpoint, we set the eigenvalues associated with the tangential eigenvector to the derivative of the drift function. This ensures that J_{RNN} over the embedded ring is locally matched to J_{obj} derived from the drift function. Finally, we solve the linear equations in (6) to derive the *W*, for the RNN that satisfies these constraints. For now, we ignore inputs and consider the RNN an autonomous dynamical system.

To test the solution, we initialized the network at various states close to the ring in the state space and allowed the state to evolve according to the imposed relaxation dynamics. As expected, the network state quickly moved onto the ring and evolved towards the nearest stable fixed points (Fig. 2b, left). Moreover, the state dynamics over the ring indicated that the speed of the drift in the state space closely matched the speed predicted from the drift function (Fig. 2b, right).

We also tested drift functions with different number of fixed points, non-periodic drift functions, and drift functions with a non-zero mean. In all cases, EMPJ was able to construct an RNN that would accurately capture the desired dynamics. The case for a drift function with a non-zero mean requires a non-trivial adjustment to what we discussed previously. In general, because eigenvalues are set according to the derivative of the drift function, EMPJ's default solution is a network that corresponds to a drift function with a mean of zero. However, a baseline can be added straightforwardly by an additional constraints to equation (6) that define where fixed points should be located; i.e., where the drift function crosses zero (see Methods) (Fig. 1c). These examples highlight the possibility of using EMPJ as a simple and rapid method for constructing RNN that can express a variety of low-dimensional latent dynamics.

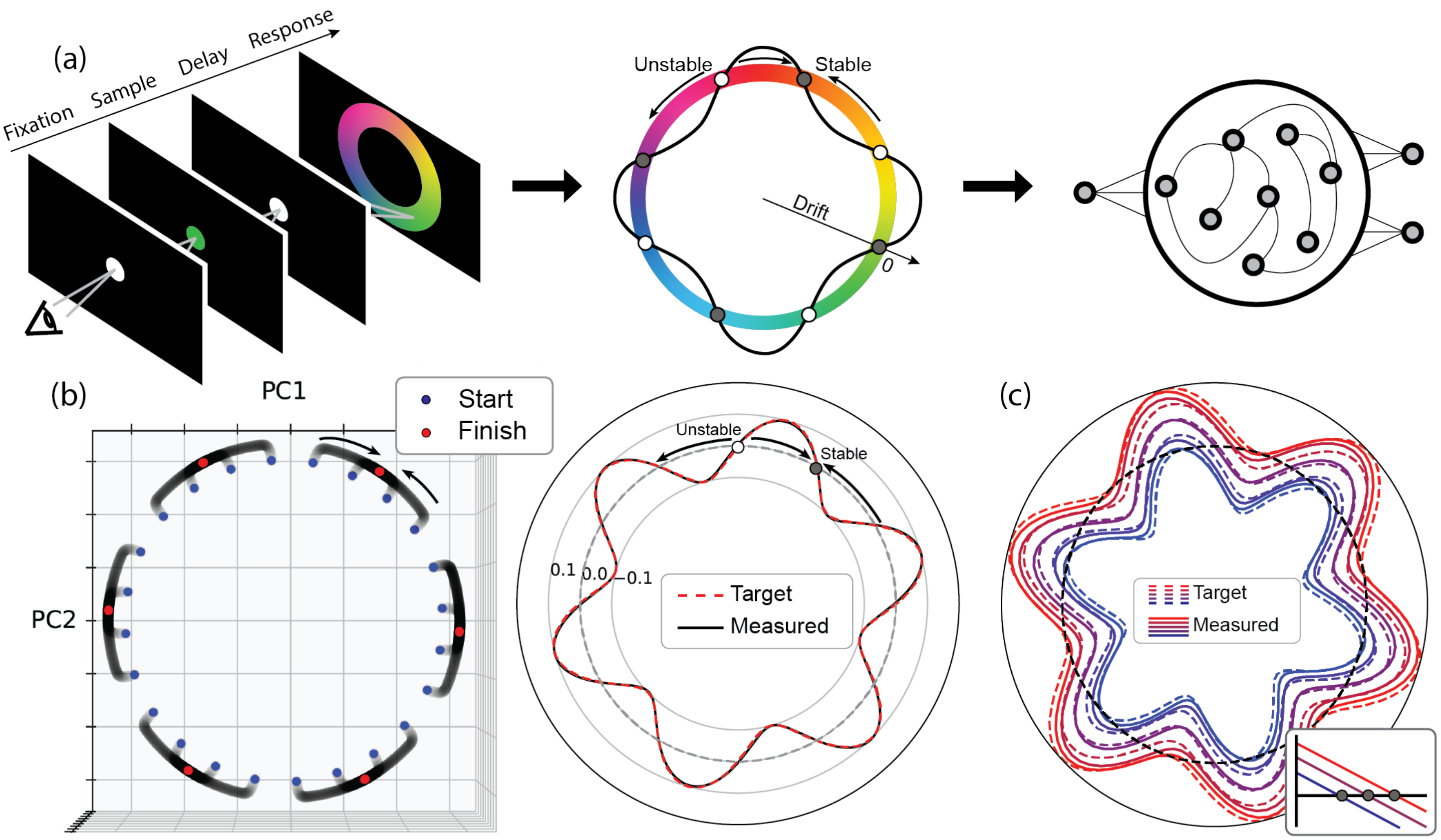


Figure 2 Semi-discrete representations for working memory a) (left) We imagine a task where the goal is to remember a continuous, periodic, one-dimensional variable (such as color from a wheel). (middle) A possible dynamical solution to the task. We have superposed a plot of the drift function in polar coordinates on top of the ring-shaped manifold corresponding to the subject's mental representation of color. A positive drift means clockwise movement on the ring, while negative means counterclockwise. Wherever the drift function crosses zero, there is a fixed point, which is either stable or unstable depending on the slope of the drift function at that point. (right) We wish to create a network that implements this dynamical solution. b) (left) First two principal components of RNN activity, initialized from points close to the ring attractor (blue). Without noise, neural trajectories go towards stable fixed points (red). (right) The drift of the RNN working memory representation compared with the target drift function used to create the RNN. c) Five different networks have drift functions that are shifted from mostly clockwise (red) motion to counter-clockwise (blue) motion. (inset) This is accomplished by constraining the location of fixed points.

Comparison of RNN with drift-diffusion model

Our implementation of a slow drift over a ring-shaped manifold is based on the assumption that a robust circuit for working memory requires corrective dynamics to counter the effect of noise. However, we have so far only analyzed networks under noiseless conditions. We now ask how the system responds to noise by comparing its behavior to a one-dimensional drift-diffusion model (DDM) and investigating whether the semi-discrete representation we have created improves the working memory of the system.

For our analysis, we will add noise to the network through input vectors that are aligned with the plane in which the ring sits. We will refer to this kind of noise as "external noise." The idea that diffusion might be driven by noise introduced through input channels is supported by physiological evidence [26]. An alternative would be introducing noise independently to every unit in the network, but the projection of the variance of a high-dimensional noise vector onto the tangent vector of the ring is inversely proportional to the size of the network, so this "internal noise" vector would need to be quite large to cause the same amount of diffusion as external noise. In simulations, we found that adding such large vectors caused unpredictable network behavior, presumably because the perturbations due to noise brought the network's state so far away from the ring. By calculating the relationship between external noise in the RNN and noise in the DDM (see Methods), we were able to directly compare the behavior of the two models. We found that the distributions of estimates of the initial position on the ring were identical. This was true whether there were two fixed points on the ring or infinitely many (Fig. 3a).

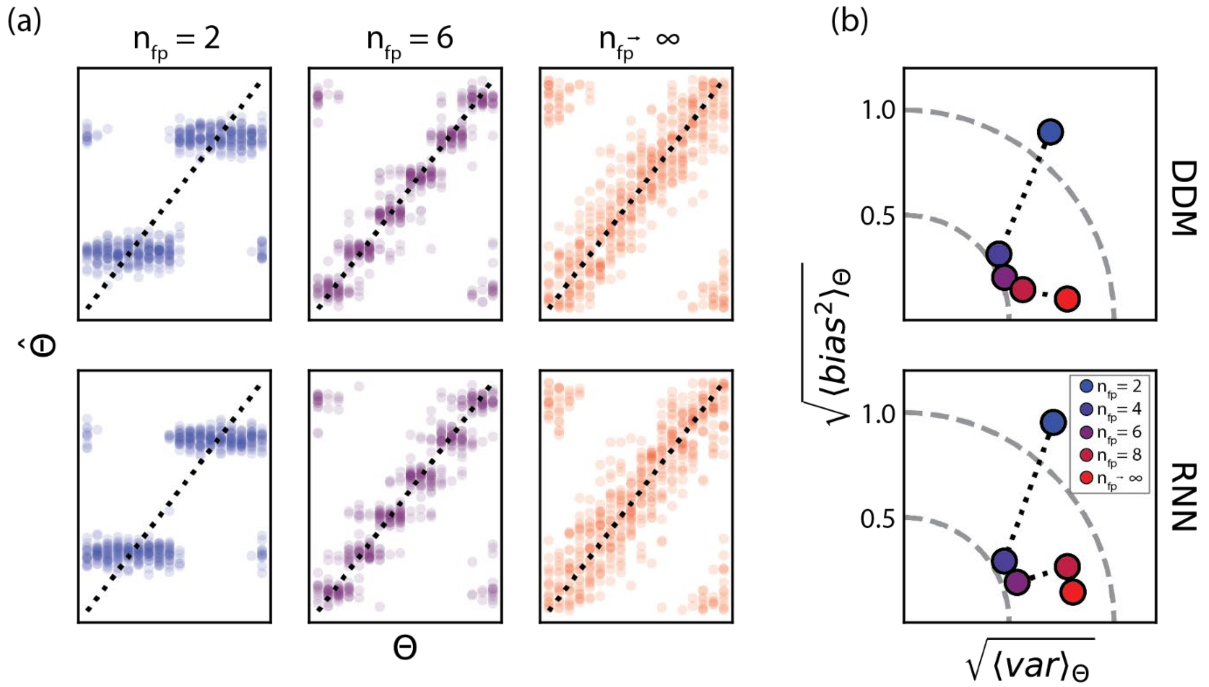


Figure 3 Comparison of bias-variance trade-off a) (top) Simulations of a drift-diffusion model (DDM), given a specified number of stable fixed points in a sinusoidal drift function. Initial values are on the x-axis, while the estimated values after 15 seconds of simulation time are on the y-axis. Each initial condition was simulated 100 times. (bottom) The same simulations in RNNs built to replicate the DDM. The RNNs were given correlated noise confined to the same plane as the ring-shaped manifold. The strength of noise was made to match that in the DDM. b) Average bias and variance for the DDM (top) and RNN (bottom) for various numbers of fixed points.

We further compared the results by computing the average bias and variance of the distributions. Since the overall mean squared error of an estimator is the sum of its variance and bias squared, this was also a way of verifying that our assumptions about the optimality of semi-discrete representations. We found that the rings with only two fixed points were very biased after 15 seconds of simulation time, since estimates were clustered around those two points. However, increasing the number of fixed points decreased both the bias and variance, leading to an overall reduction in error. The lowest total average error occurred with six fixed points. After that, it became easier for noise to push the state in between basins of attraction, and even though bias continued to decrease there were increases in variance that caused overall error to increase. The curves in the bias-variance plots are almost identical for the DDM and RNN simulations, indicating that the RNNs are accurately implementing the DDMs for which they were engineered.

Input control of network dynamics

So far, we have demonstrated the ability of our method to create an RNN that implements an autonomous dynamical system that performs a computation. In this case, that computation is maintaining a semi-discrete representation of a variable. But what if we wish to add some flexibility to the network's dynamics? For example, it could be useful to adjust the strength of the drift function in response to different levels of noise being added to the network. If there is a high level of noise being added to the network, it would make sense to increase the amplitude of the drift function. With very low noise, it would make more sense to have a slower drift.

One possible solution to the problem of creating more flexible computations is the addition of inputs to the RNN. The most common way to do this is to simply project the inputs into the population using linear weights. In this case, inputs are often classified as either "sensory," providing transient information directly relevant to completing a task, or "contextual," providing a cue about what kind of task needs to be done. Contextual inputs are typically modeled as tonic inputs that take a certain value for the duration of the task [12,27]. In a recent study, the geometry of neural trajectories and RNN modeling suggested that tonic inputs might be used by cortical circuits to flexibly switch between different behavioral regimes [18]. How can we understand the computational function of these inputs from a dynamical systems perspective, and can we use that understanding to create networks that flexibly switch between task contexts?

In the case of adjusting the strength of the drift function in the example working memory task, we consider the role that inputs appear to play in modulating the speed of neural trajectories [13,18]. We continue with the same working memory task as before, but assume now that we wish to use tonic inputs to modulate the strength of the corrective drift. In other words, we want the amplitude of the sinusoidal drift function to increase with a tonic input, which will be introduced according to equation (1) by projecting the input value onto the neural population (Fig. 4a).

We begin with our method as described so far: for a set of points on a manifold, we define the first few eigenvalues and eigenvectors of the Jacobian to give the desired recurrent dynamics along a ring-shaped manifold. We will refer to the space spanned by these eigenvectors as the “recurrent subspace,” illustrated by the colored planes in Figure 4b. We then add another dimension to the Jacobian eigendecomposition, such that the eigenvectors are the same as the vector of weights used to project the input onto the population. This dimension can be referred to as the “input subspace.” We also specify the associated eigenvalues to be a negative constant. This means that the projection of the system’s state along the input subspace will exponentially decay (see Methods for further details). Therefore, a tonic input will push the system up to some point where it is canceled out by the exponential decay of activity along the input subspace (Fig. 4b). For constant inputs, the system will reach an equilibrium point where it is stable. Importantly, we can then alter the dynamics in the recurrent subspace so they are parametrized by the position in the input subspace. In this example, we scale the eigenvalues that control the drift function by their position along the input subspace, so that the drift function has zero amplitude when there is no input and has a high amplitude with a high input.

After incorporating the input into the EMPJ framework, we performed a similar simulation as before, initializing the resulting RNN at various points around the ring manifold and at levels in the input subspace corresponding to certain inputs. We were able to control the speed of the drift function as desired: there was very slow drift without any input, and fast drift when there were high inputs (Fig. 4c).

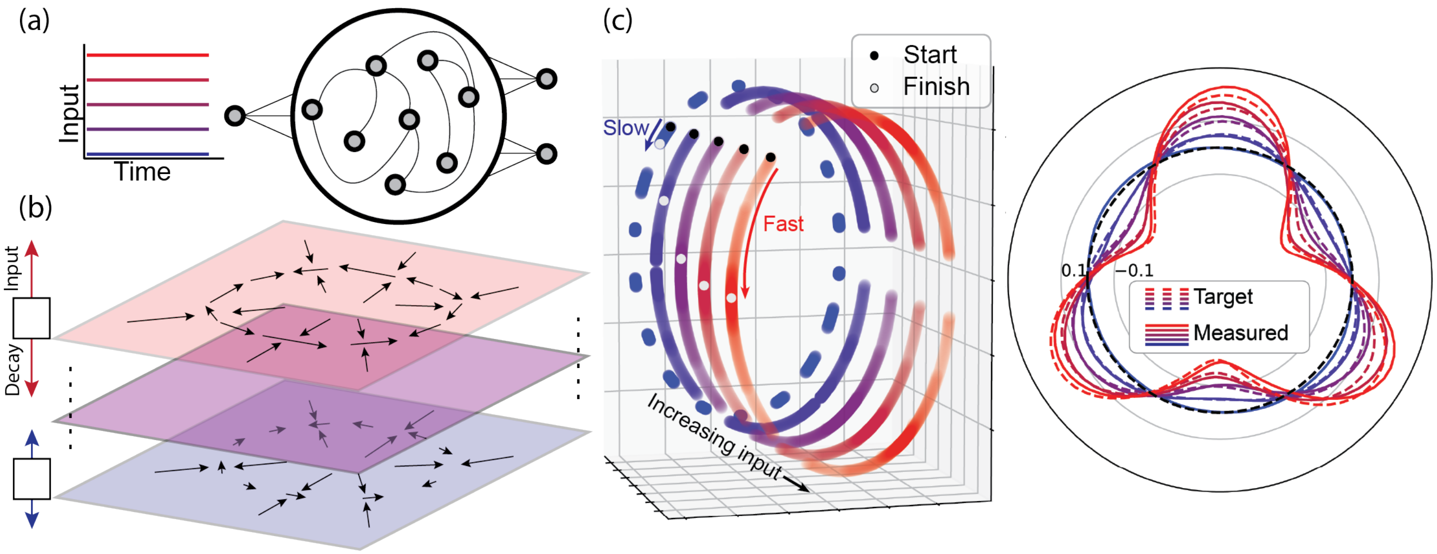


Figure 4 Input control of speed a) A one-dimensional input, which will be used to control the speed of dynamics, is projected by an "input vector" into the neural population. In subsequent panels, blue colors indicate low input, while red indicate higher input. b) We define the input vector to be orthogonal to the plane containing the ring. Using our method, the network's dynamics are set such that a constant input increases the RNN's position along that axis until it is canceled out by decay in the same direction. At that point, the system is stable in a new plane, and the dynamics can be specified in that subspace (in this case, to go faster around the ring). c) (left) PCA of network activity, initialized at various points around the ring and for different input conditions. Black and white dots illustrate start and stop of one initialization, while colors indicate neural trajectories given a particular input level. As expected, tonic inputs confine the dynamics to different rings. (right) Measured drift functions for various inputs closely align with the target drift functions.

Rings embedded in high-dimensional space

Next, we explore the ability of EMPJ to embed rings in more than two dimensions, which will enable us to explore representations between two extremes. At one limit of dimensionality, units have independent tuning curves that fully determine their responses to a stimulus (Fig. 5a, left). In this case, the dimensionality of the system cannot be reduced: we have a ring embedded in the same number of dimensions as there are units in the network. On the other hand, we have rings in only two dimensions, where the tuning curves of units will consist of weighted sums of a sine and cosine. Here, we might say that there are two "latent tuning curves" that project into the population. We can explore rings of intermediate dimensionality by adding other latent tuning curves aligned with other population modes (Fig. 5a, right). This can be thought of as "bending" the ring out of its original plane (Fig. 5b). In our simulations, these bends consist of von Mises functions, which are evenly spaced around the ring and have widths controlled by the parameter κ (see Methods for full details). To keep the overall population activity constant, we normalize these latent tuning curves so that the ring lies on a hypersphere. The total number of latent tuning curves provides the embedding dimension of the ring. The tuning curves of single units are then made of linear combinations of these latent tuning curves (Fig. 5a, right), and will demonstrate the mixed selectivity that is a hallmark of cortical representations [28].

We find that the embedding dimension fully determines the rank of the connectivity matrix for the RNN. No matter what kinds of dynamics occur over the ring, the connectivity matrix only ever has the same number of non-zero eigenvalues as there are embedding dimensions (Fig. 5c). This can be explained by the fact that the linear constraints we used to build our networks occupy the same subspace. The eigendecomposition of the weight matrix reveals its true function: one set of eigenvectors projects the network state into a low-dimensional subspace, the eigenvalues scale it along the relevant dimensions, and the inverse eigenvectors project it back into the full space. Since Equation 1 includes a “membrane leak” term, activity in all other dimensions decays exponentially.

Another finding, unrelated to the RNNs but relevant to questions about optimal representations, is that both the width and number of latent tuning curves affect the total length of the ring manifold (Fig. 5d). Total ring length is a relevant metric to consider, since it means that the distance along the ring between states is greater, making it easier to discriminate between them and reducing the effects of noise. For broad tuning, corresponding to low values of κ, increasing the embedding dimension results in a shorter ring. Intuition for this result can come from the three-dimensional case illustrated in Fig. 5b. An infinitely broad von Mises function consists of a constant value, which would turn the “bend” in the x_1 dimension into an offset from the sphere’s equator. Now the ring would simply lie at a higher “latitude” on the sphere, and would be shorter. Conversely, increasing the embedding dimension of the ring when the tuning curves are relatively narrow will monotonically lengthen the ring. This is consistent with the theoretical result [29] that narrow tuning curves densely tiling the stimulus space optimizes the Fisher information of a population of neurons. Our result here, combined with the previous theory, suggests that there may be pressure on a neural population to have many narrow latent tuning curves, though we have not yet addressed the dynamic stability of these ring shapes.

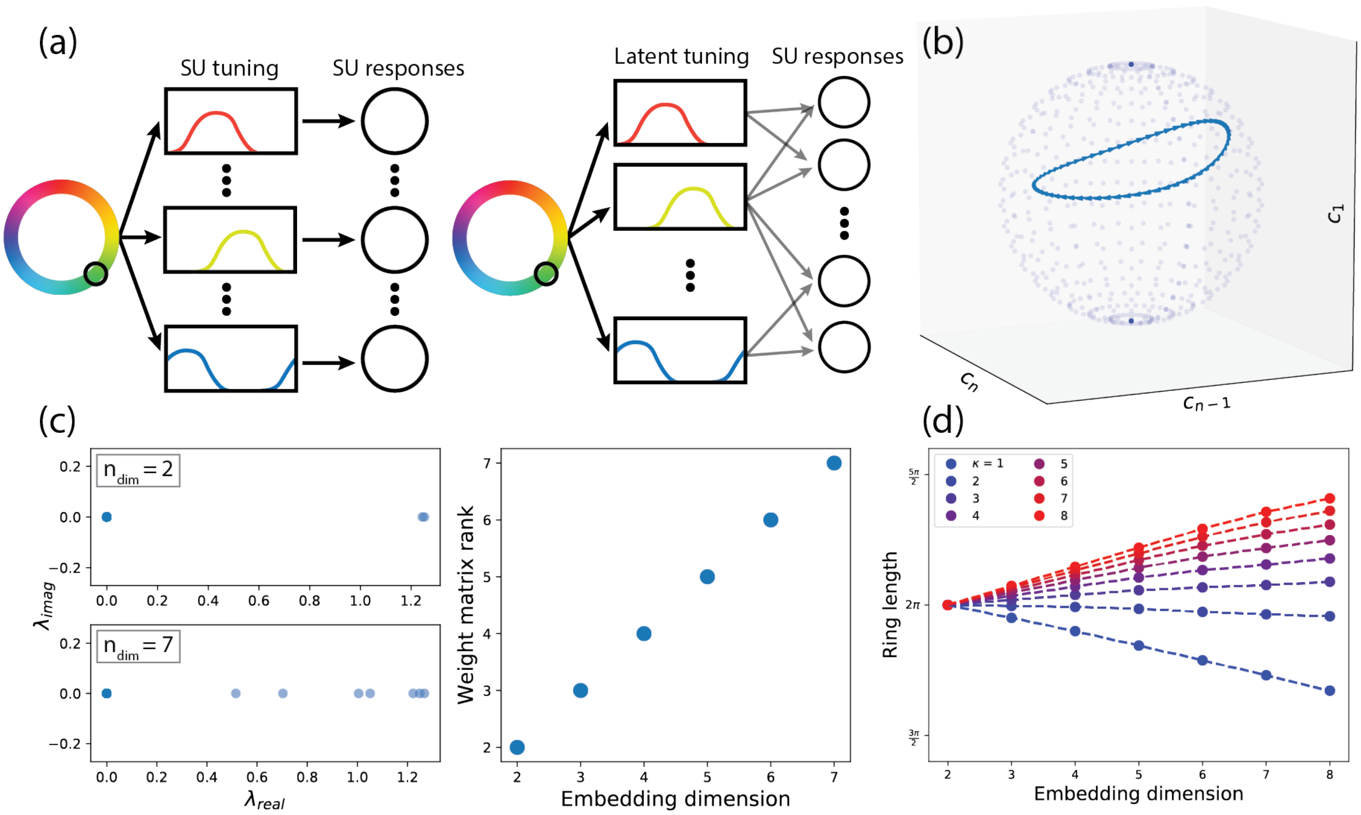


Figure 5 Embedding rings in higher dimensions a) In a framework where tuning curves are independent (left), single-unit (SU) responses depend solely on how much the stimulus overlaps with SN tuning curves. We consider a case (right) where SN responses are the result of random projects of a lower-dimensional set of latent tuning curves. b) View of a ring over a 3D slice of a hypersphere, showing the ring bending out of the plane created by x_{n-1} and $x_n$into the dimension denoted by x_1. The ring's excursions into other dimensions are not visible. c) The eigenvalues of the RNN weight matrix, for a ring lying in a 2D plane (left, top) and for a ring with excursions into five additional dimensions (left, bottom). The rank of the RNN weight matrix, determined by the number of non-zero eigenvalues, matches the ring's embedding dimension. d) The total ring length as a function of the embedding dimension, for different widths of the latent tuning curves (denoted κ).

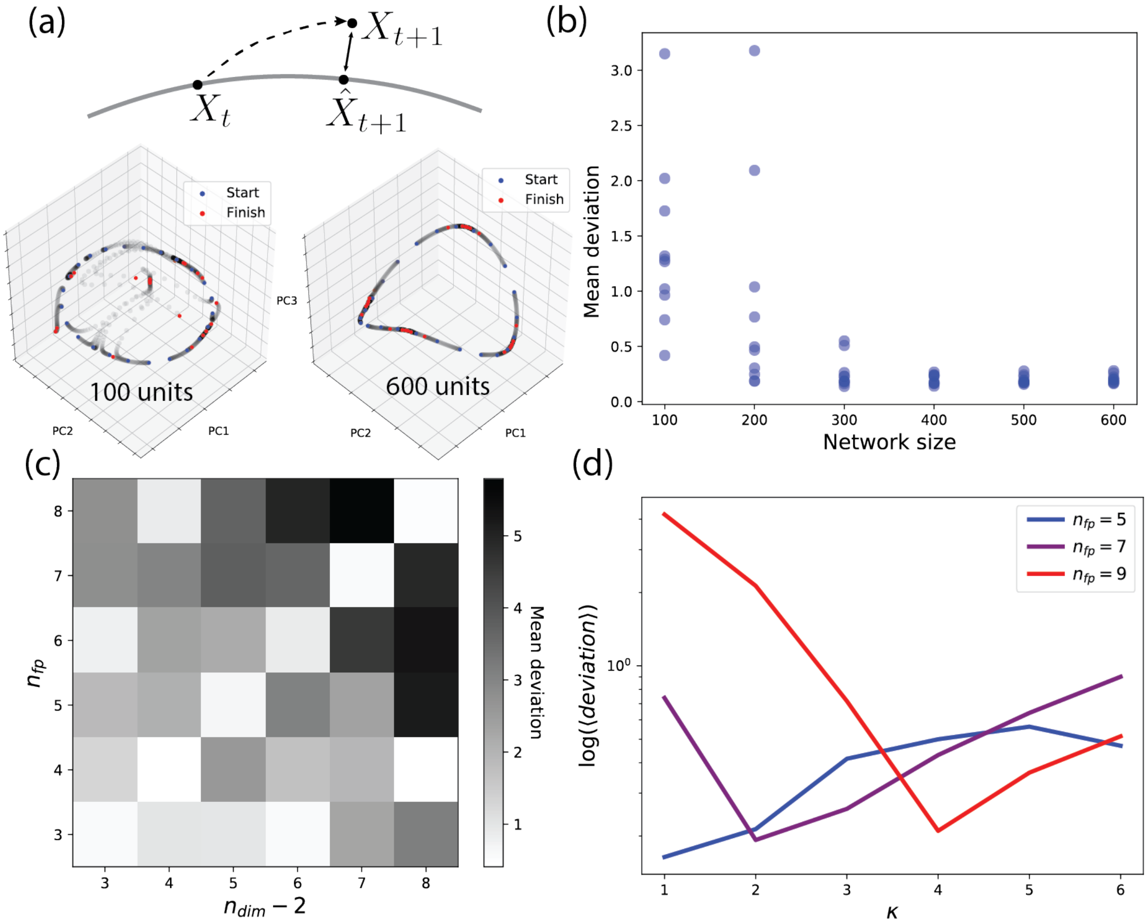
Limitations of RNN dynamic capacity

To address questions about the dynamic stability of rings with higher embedding dimensions, we must first define an appropriate error metric. Measuring the drift of the network state at various initial conditions on the ring, as done in previous figures to see whether the ring implemented the correct drift function, will not suffice, since trajectories might fly off the ring after some time. We therefore introduce a metric referred to "deviation," illustrated in Fig. 6a. At various points in time for an RNN trajectory, we decode the current value of θ on the ring, and then calculate what the RNN state should be given that value. The Euclidean distance between the RNN's actual state and where it should be on the ring provides the deviation at that point in time. We can sample the deviation over time and from many initial conditions to get an average measure of how well the RNN approximates the desired dynamics over the ring. One result of this analysis is the finding that the network must be sufficiently large (Fig. 6b). Our method thus allows us to find the smallest network size capable of creating dynamics over a particular ring.

We next examine the limits introduced by the geometry of the ring and the demands of the drift function. First, we ask whether there is a connection between the embedding dimension and the number of fixed points. Are there symmetries in ring structure than can be exploited to make certain drift functions easier? We find that placing the fixed points of the drift function at the peaks of the latent tuning curves makes the RNN activity more stable on the ring (Fig. 6c). Specifically, this means matching the number of fixed points with the number of "bends" in the ring, which is two less than the embedding dimension.

Taking this finding into consideration, we examine the effect of tuning curve width given matched fixed points and embedding dimension. We find that there are optimal values of κ that depend on the other parameters (Fig. 6d). As the embedding dimension increases, the optimal κ also increases, meaning that higher-dimensional rings require narrower tuning curves for stability. However, this is only true up to a point: making the latent curves too narrow makes the networks less stable.

With these findings, we can make some normative statements. From an information theory perspective, we might assume that higher-dimensional rings with narrower tuning curves are better for encoding a stimulus value. However, this configuration might make it difficult to create stable dynamics. We have found that the most dynamically stable rings have symmetry between the number of fixed points and the embedding dimension, and the latent tuning curves forming those rings have an optimal width.



360

361 **Figure 6 Constraints on network performance** a) Illustration of the deviation metric used to quantify network performance (top). Two examples of a
362 network with high deviation (bottom left) and a network with low deviation (bottom right). b) Influence of number of units in the network on the deviation
363 from the ring. For all networks simulated here, embedding dimension was 6 and the number of fixed points was 4. c) Deviation as a function of number
364 of fixed points and embedding dimension. Darker shading indicates higher deviation d) Logarithmic plot of deviation as a function of latent tuning curve
365 width, assuming embedding dimension and number of fixed points are matched.

Discussion

We have developed a method, EMPJ, for synthesizing RNNs that perform computations by implementing specific task-relevant dynamics. EMPJ works by specifying local constraints on the dynamics, resulting in the desired global behavior. The key innovation in EMPJ is that it derives the network connectivity directly from a set of linear equations given by those constraints. We demonstrated the utility of this technique in the context of a simple working memory task in which the network dynamics were specified by a drift diffusion process over a ring-shaped manifold. The flexibility of EMPJ enabled us to implement a variety of drift functions over the ring accurately. For example, we were able to create networks whose dynamics established drift functions with error-correcting properties in the presence of noise.

Moreover, we used EMPJ to generate networks whose dynamics can be flexibly adjusted by an input. This opens the possibility of creating models of neural systems that perform context-dependent sensorimotor and cognitive computations. We used this approach to model how thalamo-cortical inputs might adjust the speed with which cortical dynamics evolve, as has been suggested by recent findings [13], [30]. However, unlike end-to-end training methods [13], EMPJ enabled us to straightforwardly synthesize RNNs in which an input drove the system to different regions of state space with different drift functions. Although we focused on simple control via tonic inputs, future work should be able to extend EMPJ to incorporate richer time-varying inputs, such as pulses or oscillations, to accommodate more sophisticated control mechanisms.

One question that deserves further consideration is how to choose appropriate target dynamics for the network. In our case, we were able to engineer the target dynamics based on the computational demands of the task we considered. In general, it might be difficult to engineer such simple solutions for complex tasks whose computations involve higher-dimensional manifolds. This problem may be solved by integrating our method with other techniques that furnish the target dynamics. One option would be using Jacobians estimated from neural spiking data recorded from an animal trained to solve the task [31]. Another option is to take Jacobians from an auxiliary artificial neural network that contains task-relevant dynamics [32]. These methods would generate target dynamics from a system able to solve the task, which could then be used with EMPJ to directly engineer an RNN with those dynamics.

As described, EMPJ provides the means for embedding a task parameter manifold directly into an RNN. The approach is similar to that described by the Neural Engineering Framework (NEF), which also matches latent task dimensions to latent neural dimensions and creates a recurrent weight matrix that produces the desired transformations of neural representations [20]. One point of contrast is that EMPJ only requires knowing local linear approximations of dynamics, while the NEF involves specifying the global dynamics equations. This could be advantageous for if the global equations are unknown, but might be disadvantageous if the dynamics are fast enough that linear approximations no longer work. Additionally, population manifolds created through EMPJ are inherently designed to be stable, since we specify that off-manifold activity rapidly decays. The NEF does not use Jacobian matrices, so the local stability is not as well-defined over the manifold. Our approach also makes it easier to create networks for which the latent task manifold is embedded nonlinearly in the neural manifold. Given these differences, we present EMPJ as a complementary technique to the NEF, as it shares the same underlying principles.

EMPJ can also be contrasted with other RNN synthesis methods. For example, one might test the degree to which the connectivity matrix resulting from EMPJ matches predictions from other approaches that relate connectivity to low-dimensional dynamics. Two recent examples of such work are based on mean field theory [21] and distributions of network motifs [33]. Generally, the connectivity matrices found through EMPJ may be different from those found through mean-field methods. This is possibly because mean-field methods rely on the properties of the distribution from which the connectivity matrix weights are drawn, while the weights found through EMPJ are less constrained. As a result, we have been able to use EMPJ to create RNNs with low-dimensional dynamics that are difficult to achieve using mean-field methods (not shown). Further study could elucidate the principles by which connectivity constrains dynamics.

A larger goal of analyzing and synthesizing RNNs is to gain a deeper understanding of the relationship between manifold geometry, complexity of dynamics, and network characteristics. EMPJ makes it easy to generate and test hypotheses about those properties. For example, we used EMPJ to assess how the dimensionality of the manifold and the organization of fixed points impact the ease of implementing different drift functions. Future work could extend this work to further investigate general properties of network models such as capacity [34] and manifold smoothness [35].

Methods

Additional method details

The first step in EMPJ is to define some number of setpoints on a manifold. The exact number does not matter, but the sampling should be sufficiently dense that it is possible to interpolate the drift function between points. The next step is to define both the direction and magnitude of the target vector field over the manifold. This is referred to as the “drift function” previously. The gradient of this vector field is used to define the Jacobian at every point.

The next step is to project the points on the manifold and the vector field gradient into a high-dimensional space. In our method, we accomplish this by performing the Gram-Schmidt process on a set of Gaussian vectors to obtain our “projection vectors.” These vectors can be scaled by some amount to take advantage of the full dynamic range of the network units. For example, we find that scaling these projection vectors so that only a few of the single units ever get close to saturation works well.

Once the Jacobian J_{obj} is determined at each setpoint, we stack the constraints given by equation (4) to produce equation (6), creating a linear equation of the following form:

$$\underline{(7) \quad (A+\xi)W = B}$$

Note that ξ denotes a matrix of white noise ($\sigma = 10^{-6}$ in all cases unless noted otherwise) the same size as *A*, which helps to prevent overfitting and creates a more robust solution. Thus, by placing local constraints on the connectivity matrix, we find a connectivity matrix for a network that has the desired global behavior.

For solving the linear equation, we used the least-squares solver from the NumPy linear algebra library.

Ring attractor example

For the semi-discrete ring attractor, we first created a ring by taking the cosine and sine of 64 evenly spaced values of a parameter θ between 0 and 2π. This yields a list of coordinates on a unit circle. We then projected those points into a 400-dimensional space using two projection vectors, as described in the previous section. The projection vectors were each scaled to have a magnitude of 10.

Next, we needed to define the Jacobian at each setpoint. Since the ring is a locally 1-dimensional object, we only need to worry about defining one eigenvector and corresponding eigenvalue at each point. We obtained the eigenvectors by computing the tangent vector to the ring, using the fact that the tangent vector for a ring at a point specified by the coordinates (*cos*θ, sinθ) has the direction (-*sin*θ, *cos*θ). The eigenvalues were determined by taking the derivative of a drift function of the form f(θ) = -cos($\omega\theta$), where the frequency ω is equal to the number of stable fixed points around the ring. Thus, the eigenvalues were determined by the equation $\lambda(\theta)$= ωsin=($\omega\theta$).

To measure how well the network matched the desired drift function, we initialized the network at points around the ring and measured how the decoded values of θ changed during the first time step of simulation. To decode the value, we used least squares linear regression to decode the cosine and sine of θ, which we used to reconstruct θ. In other words, we solved for the matrix *D* in the equation 8 for known values of θ.

$$(8)\quad [\sin\hat{\theta} \quad \cos\hat{\theta}] = D\tanh(x)$$

$$(9)\quad \hat{\theta} = \tan^{-1}(\frac{\sin\hat{\theta}}{\cos\hat{\theta}})$$

Additional constraints

Since the Jacobian eigenvalues specify the derivative of the drift function, there is an integration constant that is not accounted for when obtaining the actual drift function. We can impose constraints on this value by constraining where the drift function crosses zero. Since zero-crossings of the drift function are by definition fixed points, we can do this by setting equation (1) equal to zero, which gives the following:

$$(10)\quad x_f = W^T\phi(x_f))$$

where x_f refers to the fixed point. This provides another linear constraint, which can be added to the list of other constraints in equation (6):

$$(11)\quad \begin{bmatrix} A_{x_0} \\ \cdots \\ A_{x_m} \\ \phi(x_f) \end{bmatrix} W = \begin{bmatrix} B_{x_0} \\ \cdots \\ B_{x_m} \\ x_f \end{bmatrix}$$

This procedure allows us to achieve the results in Fig. 2c.

Bias/variance comparison

To verify that our RNN model behaved like the drift-diffusion model (DDM) it was designed to implement, we simulated the target DDM over the one-dimensional parameter θ. The change in θ is determined by the following stochastic ordinary differential equation:

$$(12) \quad d\theta = G(\theta)dt + \sigma dW$$

where *G(θ)* is the deterministic drift function and *dW* represents a Wiener process that introduces Gaussian noise at every timestep, scaled by the standard deviation σ.

To compare the models, we used a sinusoidal drift function with a maximum value of 0.2 rad/s and a noise standard deviation σ of 0.2. The frequency determined the number of fixed points of the drift function, and we tested values of 0, 2, 4, 6, and 8. Note that setting the frequency to 0 results in a completely flat drift function, effectively creating infinite fixed points. We simulated the two models 30 times each for 18 different initial conditions. The timestep was set to 50 ms for the DDM, and each trial was simulated for 15 seconds.

As we were interested in exploring the usefulness of semi-discrete representations, we compared the average bias and variance of the model estimates at the end of the simulation time. The variance and bias metrics are computed as follows, averaging over the initial values of θ.

$$(13) \quad \langle var \rangle_\theta = \left\langle \langle \hat{\theta}^2 \rangle_{\hat{\theta}} - \langle \hat{\theta} \rangle_{\hat{\theta}}^2 \right\rangle_\theta$$

$$(14) \quad \left\langle bias^2 \right\rangle_\theta = \left\langle \left(\langle \hat{\theta} \rangle_{\hat{\theta}} - \theta \right)^2 \right\rangle_\theta$$

Input control

Our approach of controlling the network's behavior with inputs relies on the ability to navigate a null space such that the dynamics governing the output change in a desired way. We achieve this by balancing out the input along a particular axis with an equivalent decay. This can be explained with some simple linear algebra.

First, consider a dynamical system with state vector *y*. As discussed previously, we can use the Jacobian matrix *J* to linearly approximate the system's behavior around some point. We can then express the Jacobian by its eigendecomposition.

$$\dot{\vec{y}} = J\vec{y}$$

$$\dot{\vec{y}} = U\Sigma U^T\vec{y}$$

If the local dynamics are of rank *m*, and the eigenvalues and eigenvectors are written as λ and $\hat{u}$ respectively, we can see that changes in *y* are essentially the sum of dynamics along separate eigenvectors:

$$\dot{\vec{y}} = \sum_{i \epsilon m} \lambda_i y_i \hat{u}_i$$

Where y_i refers to the projection of *y* onto the *i*th eigenvector. We now consider changes in a single dimension:

$$\dot{y}_i = \lambda y_i$$

The solution to this equation is simply an exponential function, where the eigenvalue determines the exponent.

$$y_i(t) = y_i(0)\ e^{\lambda_i t}$$

We will consider the case where the eigenvalue is a constant negative value. In that case, the system's projection in this dimension will decay towards zero. This means that zero is a stable fixed point for that dimension.

$$y_i(t) = y_i(0)\ e^{-at}$$

However, if we add a tonic input that projects along that dimension, we can change the system's behavior. Now, the differential equation is the following:

$$\dot{y}_i = -ay_i + I$$

The solution to this equation is still exponential decay, but if we solve for the fixed point there is now a different long-term behavior:

$$\dot{y}_i = 0 \quad \text{when} \quad I = a y_i$$

$$y_i(\infty) = I/a$$

This means that the stable fixed point along the dimension is now at *I/a*, rather than zero. This means that introducing a tonic input as described will cause the system to shift to a different region of state space where the projection onto the *i*th eigenvector is *I/a*.

We use this property to our advantage in the text. We define an "input dimension" that is orthogonal to the ring. This creates a cylinder-shaped manifold. Instead of just specifying the drift function around one ring, we define it for several rings that lie on the cylinder. Since the maximum drift speed smoothly increases as we move up the cylinder, tonic inputs that push the network state in that dimension increase the drift speed. The choice of increasing drift speed with the tonic input is arbitrary.

For our simulations, we set the eigenvalue corresponding to decay along the cylinder to -1. The rings were scaled to have a radius of 8, and rings corresponding to different input levels were 6 units of distance apart.

Constructing high-dimensional rings

We made several choices for how to embed a ring in a higher-dimensional space. As our goal was to compare how well EMPJ works for rings of different dimensionality and geometry, we decided to 1) keep total population activity constant across all conditions, and 2) use as few parameters as possible to define the ring.

To achieve the latter, we thought of the ring in terms of latent tuning curves that cause the ring to bend into different dimensions, and made the density and narrowness of these tuning curves the only parameters we could change. The latent tuning curves consisted of unnormalized von Mises functions that reach a maximum of ½. We defined the centers of the latent tuning curves so that they were evenly spaced around the ring. A single width parameter, κ, controlled the widths of all the tuning curves. Thus, for a ring with bends into *d* dimensions, the equation for the *j*th tuning curve is given by the following:

$$(15) \quad c_j(\theta) = \frac{1}{2}e^{\kappa(\cos(x-\frac{2\pi j}{d})-1)}$$

To keep the total population activity constant, we thought of the ring as lying on a hypersphere, meaning that the norm of the vector describing every point on the ring is constant. This allows us to consider latent tuning curves in terms of hyperspherical coordinates. An *n*-dimensional hypersphere is a manifold embedded in (*n*+1)-dimensional space (e.g. the 2-dimensional surface of a 3-dimensional ball). Any point on that manifold can be described by *n* coordinates: one planar angle that ranges from 0 to 2π and *n-1* elevation angles that range from 0 to π (Fig. S1a). The planar angle θ is the same as the parameter being "remembered" by the ring in the working memory task. We consider the latent tuning curves to be projections of the elevation angles onto an axis orthogonal to the plane corresponding to the planar angle (e.g. the vertical axis in Fig. S1a). The remainder of the magnitude of the sphere's radius is distributed to projections onto the two Euclidean axes defining the plane. The result is that there are always two latent curves related to the sine and cosine of θ, and then (*n-2*) latent tuning curves with a von Mises shape (Fig. S1b). The sphere radius we used in our simulations was 12, which we found causes only a few of the units in the network to have saturated firing rates when the ring coordinates were embedded in the high-dimensional network state using normally distributed vectors.

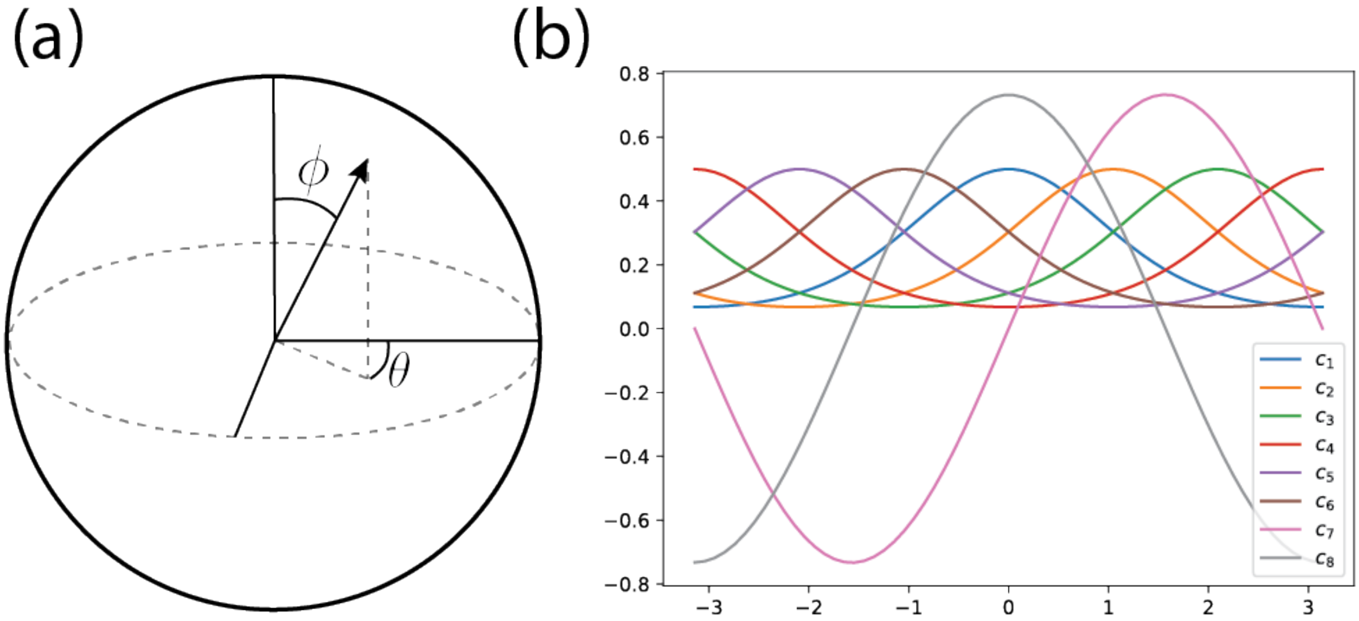


Figure S1 Rings on hyperspheres a) Illustration of a 2-sphere, for which the surface is parametrized by a planar angle θ and one elevation angle ϕ. Latent tuning curves describe the projection of the ring onto the Euclidean axes of the sphere. b) Example of latent tuning curves for a 7-sphere embedded in an 8-dimensional Euclidean space. The first six latent tuning curves contributing to the ring consist of equally spaced von Mises functions, while the last two are the sine and cosine of the planar angle θ, normalized to keep the norm constant at every point.

Ring capacity

To measure the ability of a network to approximate dynamics over a given ring, we defined an error metric we refer to as deviation. We defined deviation as the average Euclidean distance between the network state x and the network state $\hat{x}$ we would expect based on the decoded angle $\hat{\theta}$, averaged over time and initial conditions. This is expressed by the following equation:

$$(16) \quad deviation = \langle\langle\|x_t - \hat{x}_t\|\rangle_t\rangle_{x_0}$$

The first step for computing deviation is to decode the angle being represented by the network. This is done as described previously in (8) and (9). We then use the known latent tuning curves to generate a network state $\hat{x}$ corresponding to that angle. For our measurements of deviation, we did this for 24 different initial conditions on the ring and for 5 seconds of simulation time, sampling the trajectories every 0.1 seconds. It is worth noting that the exact value of deviation is not necessarily meaningful, but it is useful for comparing different networks.

When measuring network capacity as a function of network size, we measured deviation for 10 different networks. For each, we set the number of von Mises latent tuning curves to 4, the tuning curve width to 2, and the number of fixed points to 4. Another relevant parameter was the standard deviation of regularization noise added when finding the weight matrix, which we set to 1e-3. We tested network sizes of 100, 200, 300, 400, 500, and 600 units.

For measuring network capacity as a function of the number of the ring dimensionality, width of latent tuning curves, and number of fixed points, we used a similar procedure, this time keeping the network size fixed at 400 units and changing only the parameters of interest.

Code

Code for reproducing the results of this paper can be found at https://github.com/elipollock/EMPJ.

Acknowledgements

M.J. was supported by a CRCNS grant funded through the NIH (NIMH: 1R01MH122025-01) and the McGovern Institute. The authors wish to thank SueYeon Chung, Srdjan Ostojic, Ila Fiete, and Larry Abbott for helpful comments and discussions.

Competing Interests

The authors declare no financial or non-financial competing interests.

References

1. Stevenson IH, Kording KP. How advances in neural recording affect data analysis. Nat Neurosci. 2011 Feb;14(2):139–42.

2. Saxena S, Cunningham JP. Towards the neural population doctrine. Curr Opin Neurobiol. 2019 Mar 13;55:103–11.

3. Shadlen MN, Kiani R. Decision making as a window on cognition. Neuron. 2013 Oct 30;80(3):791–806.

4. Ratcliff R, McKoon G. The diffusion decision model: theory and data for two-choice decision tasks. Neural Comput. 2008 Apr;20(4):873–922.

5. Kato S, Kaplan HS, Schrödel T, Skora S, Lindsay TH, Yemini E, et al. Global brain dynamics embed the motor command sequence of Caenorhabditis elegans. Cell. 2015 Oct 22;163(3):656–69.

6. Sohn H, Narain D, Meirhaeghe N, Jazayeri M. Bayesian Computation through Cortical Latent Dynamics. Neuron [Internet]. 2019 Jul 15 [cited 2019 Jul 15];0(0). Available from: http://www.cell.com/article/S0896627319305628/abstract

7. Whiteway MR, Butts DA. The quest for interpretable models of neural population activity. Curr Opin Neurobiol. 2019 Aug 16;58:86–93.

8. Gao P, Trautmann E, Yu BM, Santhanam G, Ryu S, Shenoy K, et al. A theory of multineuronal dimensionality, dynamics and measurement [Internet]. bioRxiv. 2017 [cited 2017 Nov 6]. p. 214262. Available from: https://www.biorxiv.org/content/early/2017/11/05/214262

9. Doya K. Universality of Fully-Connected Recurrent Neural Networks. In: IEEE Transactions on Neural [Internet]. 1993 [cited 2017 May 9]. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.2137

10. Pandarinath C, O'Shea DJ, Collins J, Jozefowicz R, Stavisky SD, Kao JC, et al. Inferring single-trial neural population dynamics using sequential auto-encoders. Nat Methods [Internet]. 2018 Sep 17; Available from: https://doi.org/10.1038/s41592-018-0109-9

11. Rajan K, Harvey CD, Tank DW. Recurrent Network Models of Sequence Generation and Memory. Neuron. 2016 Apr 6;90(1):128–42.

12. Mante V, Sussillo D, Shenoy KV, Newsome WT. Context-dependent computation by recurrent dynamics in prefrontal cortex. Nature. 2013 Nov 7;503(7474):78–84.

13. Wang J, Narain D, Hosseini EA, Jazayeri M. Flexible timing by temporal scaling of cortical responses. Nat Neurosci. 2018 Jan;21(1):102–10.

14. Cueva CJ, Wei X-X. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization [Internet]. arXiv [q-bio.NC]. 2018. Available from: http://arxiv.org/abs/1803.07770

15. Sussillo D, Churchland MM, Kaufman MT, Shenoy KV. A neural network that finds a naturalistic solution for the production of muscle activity. Nat Neurosci. 2015 Jul;18(7):1025–33.

16. Barak O. Recurrent neural networks as versatile tools of neuroscience research. Curr Opin Neurobiol. 2017 Jun 29;46:1–6.

17. Russo AA, Bittner SR, Perkins SM, Seely JS, London BM, Lara AH, et al. Motor Cortex Embeds Muscle-like Commands in an Untangled Population Response. Neuron [Internet]. 2018 Jan 26; Available from:

http://dx.doi.org/10.1016/j.neuron.2018.01.004

18. Remington ED, Narain D, Hosseini EA, Jazayeri M. Flexible Sensorimotor Computations through Rapid Reconfiguration of Cortical Dynamics. Neuron. 2018 Jun 6;98(5):1005–19.e5.

19. Sussillo D, Barak O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. Neural Comput. 2013 Mar;25(3):626–49.

20. Eliasmith C, Anderson CH. Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. MIT Press; 2004. 356 p.

21. Mastrogiuseppe F, Ostojic S. Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks. Neuron [Internet]. 2018 Jul 26 [cited 2018 Jul 26];0(0). Available from: http://www.cell.com/article/S0896627318305439/abstract

22. Remington ED, Egger SW, Narain D, Wang J, Jazayeri M. A Dynamical Systems Perspective on Flexible Motor Timing. Trends Cogn Sci. 2018 Oct 1;22(10):938–52.

23. Strogatz S, Friedman M, Mallinckrodt AJ, McKay S. Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering. Computers in Physics. 1994 Sep 1;8(5):532–532.

24. Zhang K. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. J Neurosci. 1996 Mar 15;16(6):2112–26.

25. Panichello MF, DePasquale B, Pillow JW, Buschman TJ. Error-correcting dynamics in visual working memory. Nat Commun. 2019 Jul 29;10(1):3366.

26. Chaudhuri R, Gerçek B, Pandey B, Peyrache A, Fiete I. The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep. Nat Neurosci. 2019 Aug 12;1–9.

27. Yang GR, Joglekar MR, Song HF, Newsome WT, Wang X-J. Task representations in neural networks trained to perform many cognitive tasks. Nat Neurosci [Internet]. 2019 Jan 14; Available from: https://doi.org/10.1038/s41593-018-0310-2

28. Rigotti M, Barak O, Warden MR, Wang X-J, Daw ND, Miller EK, et al. The importance of mixed selectivity in complex cognitive tasks. Nature. 2013 May 19;497:585.

29. Zhang K, Sejnowski TJ. Neuronal tuning: To sharpen or broaden? Neural Comput. 1999 Jan 1;11(1):75–84.

30. Stroud JP, Porter MA, Hennequin G, Vogels TP. Motor primitives in space and time via targeted gain modulation in cortical networks. Nat Neurosci. 2018 Dec;21(12):1774–83.

31. Duncker L, Bohner G, Boussard J, Sahani M. Learning interpretable continuous-time models of latent stochastic dynamical systems [Internet]. arXiv [stat.ML]. 2019. Available from: http://arxiv.org/abs/1902.04420

32. DePasquale B, Cueva CJ, Rajan K, Escola GS, Abbott LF. full-FORCE: A target-based method for training recurrent networks. PLoS One. 2018 Feb 7;13(2):e0191527.

33. Recanatesi S, Ocker GK, Buice MA, Shea-Brown E. Dimensionality in recurrent spiking networks: Global trends in activity and local origins in connectivity. PLoS Comput Biol. 2019 Jul;15(7):e1006446.

34. Chung SY, Lee DD, Sompolinsky H. Classification and geometry of general perceptual manifolds. Physical Review X [Internet]. 2018; Available from: https://journals.aps.org/prx/abstract/10.1103/PhysRevX.8.031003

35. Stringer C, Pachitariu M, Steinmetz N, Carandini M, Harris KD. High-dimensional geometry of population responses in visual cortex. Nature. 2019 Jul;571(7765):361–5.