Reverse engineering recurrent neural networks with Jacobian switching linear dynamical systems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recurrent neural networks (RNNs) are powerful models for processing time-series data, but understanding how they function remains challenging, and improving this understanding is of substantial interest to both the machine learning and neuroscience communities. The framework of reverse engineering a trained RNN by linearizing around its fixed points has provided insight, but the approach has significant challenges. These include the need to perform a complex fixed point finding optimization after the RNN has been trained, difficulty choosing which fixed point to expand around when studying RNN dynamics, and error accumulation when reconstructing the nonlinear solution with the linearized solution. Here we present a new model that overcomes these limitations by co-training an RNN with a novel switching linear dynamical system (SLDS) formulation. The novel SLDS dynamics are governed by the first-order Taylor series expansion of the co-trained RNN. Further, it is equipped with an auxiliary function trained to pick out the fixed points of the RNN. The results are a trained SLDS variant that closely approximates the RNN, an auxiliary function that can produce a fixed point for each point in state-space, and a trained nonlinear RNN whose dynamics have been regularized such that its first-order terms perform the computation, if possible. This model removes the post-training fixed point optimization, and allows us to unambiguously study the learned dynamics of the SLDS at any point in state-space. It also generalizes SLDS models to continuous manifolds of switching points while sharing parameters across switches. We validate the utility of the model on two synthetic tasks relevant to previous work reverse engineering RNNs. We then show that our model can be used as a drop-in in more complex architectures, such as LFADS, and apply this LFADS hybrid to analyze single-trial spiking activity from the motor system of a non-human primate.

1 Introduction

Recurrent neural networks (RNNs) are a powerful and popular tool for modeling complex sequence data. They learn to transform input sequences into output sequences by using an internal state that allows data from the past to influence the current state. RNNs have been utilized in various applications such as speech recognition, sentiment analysis, music, and video [1–4]. In neuroscience, RNNs have been used for modeling large-scale neural recordings and as a generator of scientific hypotheses by studying the network's learned representations [5–10]. However, RNNs are generally viewed as black boxes. While there has been progress in understanding their operation on simple tasks, rigorously understanding how they solve complex tasks remains a significant challenge.

An important line of work to improve our understanding of RNN computations uses dynamical systems theory [11–16]. In particular, Sussillo and Barak [15] proposes reverse engineering a trained

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

RNN by using numerical optimization to find the RNN's fixed points. The RNN is linearized around the fixed points, and the dynamics of the resulting linear approximations are studied to draw insights into how the RNN solves the task. While there are no guarantees concerning the success of this approach, empirically, linearization around fixed points has led to insights in numerous applications [10, 17–23].

There are a few drawbacks of this method. First, it requires a separate numerical optimization routine after training the network. Second, after finding the fixed points, it can sometimes be ambiguous which fixed point to linearize around for any given point in state space. We may generally choose the closest fixed point, but this may not be the correct thing to do. Finally, simulating nonlinear RNNs with linearizations around fixed points can slowly accumulate significant error, forcing previous attempts to resort to one-step ahead dynamics generation [18]. These problems can lead to uncertainty of how well the linearized system describes the nonlinear dynamics. Understanding this uncertainty is important since if we observe that the linearized system can perform a task with similar accuracy as the nonlinear system, this is strong evidence that the nonlinear system can solve the task by switching linear dynamics around fixed points.

Here, we combine ideas from reverse engineering RNNs and switching linear dynamical systems (SLDS) [24–28] to address these challenges. Given an RNN we would like to train and analyze, we introduce a separate network consisting of a novel SLDS formulation based on the first-order Taylor series expansion of the RNN equation and a learnable auxiliary function that produces the RNN's fixed points. We then define a loss function that includes regularization terms to force the SLDS to approximate the RNN and switch about the RNN's fixed points. After co-training these three functions with standard RNN training methods, the result is an accurate switching linear approximation of the nonlinear RNN and a trained auxiliary function that provides approximate fixed points of the RNN.

This architecture and training procedure:

1. Eliminates the need for post-training fixed point finding.
2. Generalizes SLDS to be able to switch about continuous manifolds of fixed points.
3. Enables parameter sharing between the SLDS switches.
4. Allows the nonlinear RNN to be approximated by a SLDS, if possible.

The combination of these benefits significantly simplifies the process of reverse engineering an RNN using fixed points. We illustrate the benefits of this method on two synthetic tasks and a neural dataset.

2 Review of reverse engineering RNNs and SLDSs

2.1 Reverse Engineering RNNs

The idea of reverse engineering RNNs with fixed point analysis is based on the hypothesis that the mechanisms trained RNNs use to solve tasks are described well by the linearized dynamics around its fixed points and slow points [15]. These approximate fixed points are state vectors $\mathbf{h}^* \in \mathbb{R}^D$ that, given an input $\mathbf{u}^* \in \mathbb{R}^U$, do not significantly change when applying the RNN update function, $\mathbf{F} : \mathbb{R}^D \times \mathbb{R}^U \to \mathbb{R}^D$. That is, $\mathbf{h}^* \approx \mathbf{F}(\mathbf{h}^*, \mathbf{u}^*; \theta)$. We can find these points numerically by minimizing a loss function

$$\mathcal{L}(\mathbf{h}) = \|\mathbf{h} - \mathbf{F}(\mathbf{h}, \mathbf{u}^*; \theta)\|_2^2, \tag{1}$$

using auto-differentiation methods [29]. Once we have found the fixed points, we linearize the system around these fixed points and analyze the dynamics of the linearized system to conclude how the system computes.

2.2 Switching linear dynamical systems

Models based on linear dynamical systems (LDS) are often used to model multi-dimensional time series and lend themselves well to dynamical systems analyses. The basic LDS models time series data using a latent representation that follows linear dynamics. Switching LDS augment the basic model with a set of discrete states where each state corresponds to different linear dynamics. This

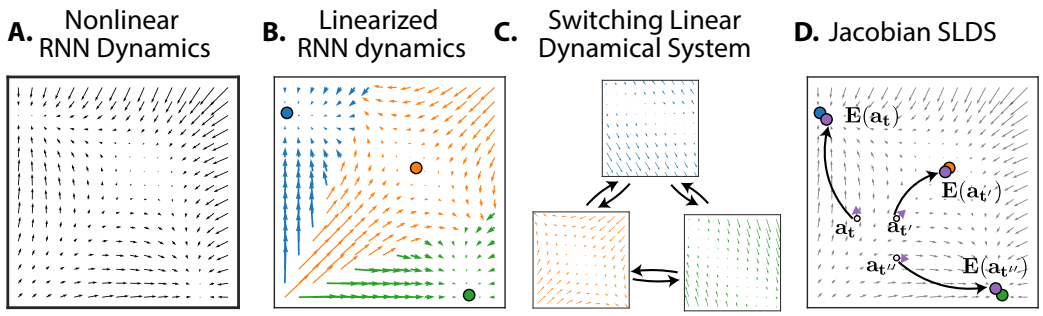

Figure 1: **A**. Example of a nonlinear dynamical system learned by a recurrent neural network (RNN). **B**. Linearization of the RNN dynamics around the nearest fixed point (dots). **C**. A switching linear dynamical system (SLDS) approximates the nonlinear dynamics by stochastically jumping between three linear regimes. **D**. The Jacobian SLDS co-trains an RNN (gray arrows) with an *expansion network* ($\mathbf{E}(\cdot)$), which maps the current JSLDS state (white dots, $\mathbf{a_t}$) to an expansion point (purple dots, $\mathbf{E}(\mathbf{a_t})$) near to a true fixed point of the RNN. The JSLDS linearizes the RNN dynamics around the expansion point to obtain a linear system (purple arrows) that approximates the local dynamics.

allows SLDS to model complex, nonlinear time series data by breaking the time series down into a sequence of simpler local linear dynamics. Explicitly, let $\mathbf{y}_t \in \mathbb{R}^N$ denote the observation data at time t and let $\mathbf{h}_t \in \mathbb{R}^D$ represent the continuous latent state at time t. SLDS model the expected observation value as $\mathbb{E}[y_t] = \mathbf{g}(\mathbf{h}_t)$ where $\mathbf{g}$ represents a mapping from $\mathbb{R}^D$ to $\mathbb{R}^N$. Given the current discrete latent state $z_t \in \{1, ..., K\}$ and input $\mathbf{u}_t \in \mathbb{R}^U$, the SLDS continuous latent states follow linear dynamics,

$$\mathbf{h}_t \sim \mathcal{N}\big(A^{(z_t)}\mathbf{h}_{t-1} + V^{(z_t)}\mathbf{u}_t + b^{(z_t)}, Q^{(z_t)}\big). \tag{2}$$

The current discrete state determines the current linear dynamics $A^{(z_t)} \in \mathbb{R}^{DxD}$, input matrix $V^{(z_t)} \in \mathbb{R}^{DxU}$, bias term $b^{(z_t)} \in \mathbb{R}^D$ and noise covariance $Q^{(z_t)} \in \mathbb{R}^{DxD}$. In the basic SLDS, a Markov transition matrix generally defines the discrete state switching probability.

An extension to the SLDS model is the recurrent switching linear dynamical system (RSLDS) which allows the discrete state transition to also depend on the previous continuous latent state [30–33]. This recurrent connection allows more expressivity in the model and corresponds to the idea that the current discrete state (and therefore the current dynamical regime) should depend on the current state space location. SLDS models can often offer a balance between interpretability and expressivity for many problems. The need to choose a finite number of discrete states and the increase in parameters as the number of discrete states increases are downsides that limit the model's expressivity.

3 Jacobian Switching Linear Dynamical System

We now present the Jacobian Switching Linear Dynamical System (JSLDS) model and training procedure. It combines ideas from reverse engineering RNNs and SLDS to allow for automated fixed point finding and accurate SLDS approximations of nonlinear RNNs.

3.1 Motivation

Let $\mathbf{F}$ denote a nonlinear RNN with previous state $\mathbf{h}_{t-1} \in \mathbb{R}^D$, input $\mathbf{u}_t \in \mathbb{R}^U$, and parameters θ. Writing the RNN update equation and its first order Taylor series expansion around points $\mathbf{h}^*$ and $\mathbf{u}^*$ we have

$$\mathbf{h}_t = \mathbf{F}(\mathbf{h}_{t-1}, \mathbf{u}_t; \theta) \approx \mathbf{F}(\mathbf{h}^*, \mathbf{u}^*; \theta) + \frac{\partial \mathbf{F}}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*; \theta)\,(\mathbf{h}_{t-1} - \mathbf{h}^*) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{h}^*, \mathbf{u}^*; \theta)\,(\mathbf{u}_t - \mathbf{u}^*)\,. \tag{3}$$

Here, $\frac{\partial \mathbf{F}}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*; \theta) \in \mathbb{R}^{DxD}$ is the **recurrent Jacobian** that determines the recurrent local dynamics and $\frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{h}^*, \mathbf{u}^*; \theta) \in \mathbb{R}^{DxU}$ is the **input Jacobian** that determines the system's input sensitivity. If

$\mathbf{h}^*$ are fixed points of $\mathbf{F}$ for a given $\mathbf{u}^*$, then $\mathbf{F}(\mathbf{h}^*, \mathbf{u}^*; \theta) = \mathbf{h}^*$ and equation (3) yields

$$\mathbf{h}_t - \mathbf{h}^* \approx \frac{\partial \mathbf{F}}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*; \theta)\,(\mathbf{h}_{t-1} - \mathbf{h}^*) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{h}^*, \mathbf{u}^*; \theta)\,(\mathbf{u}_t - \mathbf{u}^*)\,. \tag{4}$$

Eq. (4) gives a LDS that locally approximates $\mathbf{F}$ around $\mathbf{h}^*$. In principle, if one knew how to select the correct fixed point and there was always a fixed point nearby, then one could use eq. (4) to run a very accurate SLDS approximation of $\mathbf{F}$ by occasionally switching fixed points through time $\{\mathbf{h}_1^*, \mathbf{h}_2^*, \cdots, \mathbf{h}_t^*\}$.

Critically, the fixed point $\mathbf{h}_t^*$ indexes the recurrent Jacobian and input Jacobian in a manner analogous to how the discrete state z_t indexes the matrices $A^{(z_t)}$ and $V^{(z_t)}$ in the SLDS of eq. (2). In practice, selecting the correct fixed point in real-time is difficult to do, as mentioned above, which leads to the approach presented in the next section.

3.2 The JSLDS Model

Our approach is to co-train an RNN, $\mathbf{F}$, with a novel SLDS formulation based on the Jacobian of $\mathbf{F}$ in the spirit of equation (4). Specifically, we introduce a separate SLDS with its own hidden state $\mathbf{a}_t \in \mathbb{R}^D$ that switches around an expansion point $\mathbf{e}_t^* \in \mathbb{R}^D$

$$\mathbf{a}_t - \mathbf{e}_t^* = \frac{\partial \mathbf{F}}{\partial \mathbf{h}}(\mathbf{e}_t^*, \mathbf{u}^*; \theta)\,(\mathbf{a}_{t-1} - \mathbf{e}_t^*) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{e}_t^*, \mathbf{u}^*; \theta)\,(\mathbf{u}_t - \mathbf{u}^*)\,. \tag{5}$$

Note that eq. (5) shares its parameters θ with $\mathbf{F}$. We will generally take $\mathbf{u}^*$ to be either zero (the average value for examples) or the value of a context-dependent static input. The goal is for $\mathbf{e}_t^*$ to be an approximate fixed point. In order to accomplish this, we supplement the SLDS with a nonlinear auxiliary function $\mathbf{E}$ (the **expansion network**) with separate learned parameters ϕ

$$\mathbf{e}_t^* = \mathbf{E}(\mathbf{a}_{t-1}; \phi). \tag{6}$$

The expansion network $\mathbf{E}$ returns the learned expansion points and is co-trained with the nonlinear RNN and the JSLDS. Once trained, the goal is for this function to act as an approximate fixed point finder. For the experiments presented in this paper, we define $\mathbf{E}$ as a 2-layer multilayer perceptron (MLP) with the same dimension per layer as the state dimension of the RNN.

Eqs. (5-6) define the JSLDS model, which can be run forward in time independent of the original nonlinear RNN. Given the previous state $\mathbf{a}_{t-1}$, the expansion network $\mathbf{E}$ uses this state to select the next expansion point $\mathbf{e}_t^*$ the system should switch around. The system then uses $\mathbf{e}_t^*$ to define the current state update equation (eq. 5) by plugging its value into the recurrent Jacobian and input Jacobian of the original nonlinear RNN $\mathbf{F}$. Assuming the expansion network has learned to find fixed points, switching about the points $\mathbf{e}_t^*$ corresponds to reverse engineering nonlinear RNNs using fixed points. The dependence of the expansion point $\mathbf{e}_t^*$ exclusively on the previous state of the network $\mathbf{a}_{t-1}$ links to the recurrent connection in RSLDSs and the concept of using local linear dynamical systems to approximate a nonlinear system. Figure 1 illustrates this idea.

Note that eq. (5) is intended to closely follow the dynamics $\mathbf{F}$, and we will enforce this in the training procedure. We will refer to the system comprised of eqs. (5-6) as the JSLDS. We will refer to the combination of the JSLDS and the co-trained RNN as the JSLDS-RNN system. While a limitation of our method is that it does not currently lend itself to a stochastic formulation like the standard SLDS, it does allow for a potential continuum of different discrete switches using a constant number of parameters. It also automatically determines the number of discrete switches required to solve the task instead of the hand-tuning required in SLDS.

3.3 JSLDS Co-training Procedure

We co-train together the nonlinear RNN (eq. 3) and the JSLDS (eqs. 5-6). Each network can be run forward and solve the task independently. We pass each of their states through the same output activation function to create two loss functions, $\mathcal{L}_{\text{RNN}}$ and $\mathcal{L}_{\text{JSLDS}}$, for the RNN and JSLDS, respectively. In addition, the expansion points should approximate fixed points of $\mathbf{F}$ in order to achieve a good JSLDS approximation of the RNN. We also need to ensure the JSLDS states $\mathbf{a}_t$ approximate the RNN states $\mathbf{h}_t$. We achieve these goals by adding to the total loss function a fixed

point regularizer R_e and an approximation regularizer R_a defined as

$$R_e(\theta, \phi) = \sum_t \|\mathbf{e}_t^* - \mathbf{F}(\mathbf{e}_t^*, \mathbf{u}^*; \theta)\|_2^2 \tag{7}$$

$$R_a(\theta, \phi) = \sum_t \|\mathbf{a}_t - \mathbf{h}_t\|_2^2 . \tag{8}$$

Now we define the total training loss as

$$\mathcal{L}(\theta, \phi) = \lambda_{\mathsf{RNN}} \mathcal{L}_{\mathsf{RNN}}(\theta) + \lambda_{\mathsf{JSLDS}} \mathcal{L}_{\mathsf{JSLDS}}(\theta, \phi) + \lambda_e R_e(\theta, \phi) + \lambda_a R_a(\theta, \phi) \tag{9}$$

where $\lambda_{\mathsf{RNN}}, \lambda_{\mathsf{JSLDS}}, \lambda_e$ and λ_a control the strengths of the RNN loss, the JSLDS loss, the fixed point regularizer and the approximation regularizer, respectively. In practice, we have found these hyperparameters easy to select (see Section A.1 in the Appendix for a more detailed discussion). We optimize parameters θ and ϕ using standard backpropagation through time (BPTT) methods for RNNs [34]. Assuming the optimization goes well, the result will be two different systems trained to solve the task independently, with the JSLDS approximating the nonlinear RNN to first order.

4 Results

We evaluate and analyze the JSLDS-RNN system on three examples: a synthetic 3-bit memory task, a synthetic context-dependent integration task, and multineuronal population recordings from a monkey performing a reaching task.

4.1 3-bit discrete memory

We first show that JSLDS can automatically learn to switch about a discrete number of fixed points and significantly reduces linearized approximation error. We trained the JSLDS-RNN system to store and output three discrete binary inputs (Figure 2A) similar to the experiment described in [15]. For our purposes, the models receive three 2-dimensional input vectors where each input vector corresponds to a different channel.[1] Each input vector can take a value of $\{[1, 0], [0, 0], [0, 1]\}$ corresponding to a state of -1, 0, or +1, respectively. The models have three outputs, each of which needs to remember the last nonzero state of its corresponding input channel. The RNNs used in this experiment were GRUs [35] with a state dimension of 100. See Section A.2 of the Appendix for additional experiment details and hyperparameter settings.

Projecting the co-trained RNN and JSLDS dynamics into the readout space illustrates the close agreement between the networks for predictions on held-out trials (Fig. 2B). A benefit of JSLDS is that it allows us to find the approximate fixed points of the co-trained RNN by simply running the JSLDS forward in time. As a verification, we numerically found the fixed points of the co-trained RNN and projected both sets of fixed points into the readout space (Fig. 2E). We observe that the networks learned a fixed point solution consisting of 8 marginally stable fixed points (typically 3 eigenvalues within .025 of (1,0) in the complex plane). We then compared this solution to the fixed point solution found by a standard GRU (without co-training) (Fig. 2C). The difference between these two solutions shows that the JSLDS co-training procedure can change the fixed points of a nonlinear RNN for a particular task and suggests that it can provide regularization towards simpler solutions for some problems.

Next, we illustrate how JSLDS significantly improves the linear approximation of the RNN dynamics compared to linearizing a standard RNN (Figs. 2D,F). We compare the mean squared errors accumulated between the nonlinear RNN states and the approximate states for both methods on an evaluation batch of held-out trials. As noted previously, generating dynamics by linearizing a standard RNN presents difficulties. For each point in state space, we need to choose a numerical fixed point to linearize around and then use this linearized equation to update the state. However, there is no exact link between the numerical fixed points and points in state space, so we have to employ some decision criteria. Here we take the approach of simply choosing the closest fixed point (Euclidean distance). We also had to assist the standard RNN linearization by only performing one-step ahead dynamics generation (Fig. 2D). This was necessary because running the dynamics forward for many

[1] We reparameterized the inputs compared to [15]. This ensures the models do not have to act nonlinearly in the inputs and does not change the basic logic of the experiment since we are interested in nonlinear dynamics.

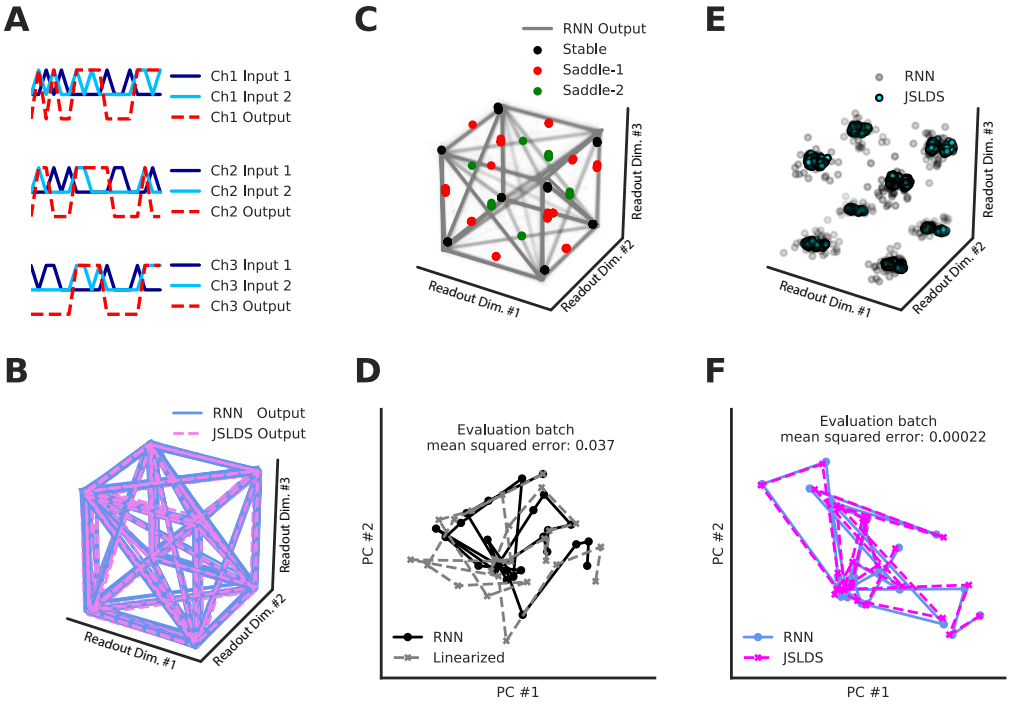

Figure 2: 3-bit memory. **A.** Two-dimensional inputs (dark blue and light blue) corresponding to input states of -1, 0, or 1 enter at random while the corresponding output (dashed red) has to remember the last non-zero input state. **B.** JSLDS closely approximates the co-trained RNN in readout space for held-out trial data. **C.** Standard GRU (no JSLDS co-training) outputs and numerical fixed points projected into readout space. **D**. Example PCA trajectory of standard GRU and linearized dynamics (one-step ahead dynamics generation) using numerically optimized fixed points. We also note the held-out MSE of this approximation. **E**. Comparison of co-trained RNN fixed points (found numerically) and JSLDS expansion points projected into readout space. The solution consists of 8 marginally stable fixed points. JSLDS has changed the fixed point solution compared to the standard GRU's solution (panel C). **F**. Example PCA trajectory of co-trained RNN and JSLDS dynamics (fully simulating dynamics forward) and the held-out MSE. JSLDS accumulates little approximation error.

timesteps accumulates substantial error and causes the trajectory to diverge. In contrast, this was not necessary for the JSLDS. Instead, we simulated the JSLDS forward for the entire trajectory (Fig. 2F). We observe that the JSLDS accumulates little approximation error for the full simulation of the dynamics, while the standard linearization method performs relatively poorly even when just performing one-step ahead dynamics generation.

4.2 Context-dependent Integration

We next present how JSLDS can learn to switch about multiple continuous manifolds of fixed points and also be used to perform a complex analysis similar to that performed in [17]. The experiment consists of training the model to contextually decide which of two white noise input streams to integrate (Fig 3A). The model receives two static context inputs, corresponding to motion and color contexts, and two time-varying white noise input streams and is trained to output the cumulative sum of the white noise stream specified by the active context input. We used a vanilla RNN with a state dimension of 128 for the co-trained RNN. See Section A.3 of the Appendix for more experiment details and hyperparameter settings.

After co-training the JSLDS-RNN system, we observed close agreement between the JSLDS and RNN on task performance for held-out trials (Fig. 3B). Next, we analyzed the dynamics of the JSLDS

for held-out trials under both contexts, set to different bias levels for both the color and motion input streams. We observed that for either context, the system integrates the relevant input using a single linear mode with an eigenvalue of 1, while the other eigenvalues decay rapidly (Fig. 3C).

From the analysis in [17], we expect a vanilla RNN to solve this type of task by representing the integration of relevant evidence as movement along an approximate line attractor (the **choice axis**) determined by the top right eigenvector. The solution consists of two separate line attractors that never exist together: one exists in the motion context and the other in the color context. For a given context (and therefore a specific line attractor), the top left eigenvector (the **selection vector**) determines the amount of evidence integrated. We expect the selection vector to project strongly onto the relevant input and be approximately orthogonal to the irrelevant input. See the Mathematical Supplement Section 10 in [17] for more details.

To verify this holds for the JSLDS, we produce a figure similar to Figures 5 and 6c from [17] by projecting the JSLDS states and expansion points into the 3-dimensional subspace meant to match the axes of choice, motion input and color input (Fig. 3D-I). This subspace was constructed by orthogonalizing the direction of the first right eigenvectors (averaged over expansion points) and the input weight vectors corresponding to the color and motion input streams, as in [17]. Panels D-F and G-I correspond to the motion and color contexts, respectively. In panels D and I, we see that for the relevant context input stream, the states move along the axis of choice and the relevant input axis in proportion to the strength of the input. Panels F and G show that the strength of the nonrelevant input stream does not affect the direction of choice. Finally, we analyze the global arrangement (Fig. 3J) of the motion and color context line attractors and selection vectors. We see that the selection vectors project strongly onto the relevant input axis but are approximately orthogonal to the irrelevant axis.

4.3 Monkey reach task with LFADS-JSLDS

Finally, we illustrate how JSLDS can be dropped in as a module to improve our understanding of more complex architectures that use RNNs such as LFADS [5]. LFADS is a sequential variational auto-encoder [36, 37] used to infer latent dynamics from single-trial neural spiking data. A criticism of LFADS has been that it is hard to interpret the RNN generator that produces the dynamics. Here, we use JSLDS to improve this understanding by substituting the combined JSLDS-RNN system for the standard GRU used in the LFADS generator (Fig. A.1 in the Appendix). We refer to this system as the LFADS-JSLDS. Once trained, we can use either the JSLDS or the co-trained RNN as the generator to produce the inferred firing rates.

We used the monkey J single-trial maze data from Churchland et al. [38] using the same setup as Pandarinath et al. [5] to train the LFADS-JSLDS model. The data consists of 2296 trials of spiking activity recorded from 202 neurons simultaneously while a monkey made reaching movements during a maze task [39, 38] across 108 reaching conditions. We used a GRU for the RNN generator. See Section A.4 of the Appendix for more details and hyperparameter settings.

We present the LFADS-JSLDS inferred rates (using the co-trained RNN generator) for several sample neurons in Fig. 4A. Next, we applied jPCA [38] to the RNN generator states (Fig. 4B) to confirm that the population dynamics inferred by LFADS-JSLDS on single trials exhibit the expected rotational dynamics [38, 5]. We also observed that the JSLDS generator closely approximates the RNN generator (Fig. 4C). Focusing on the JSLDS generator dynamics, we learned that despite minor variations in the expansion points, the JSLDS generator learned to represent the dynamics using a single, condition-independent linear system (Fig. 4D).

Sussillo et al. [21] noted that the dynamics found by linearizing around the fixed point of their regularized RNN model should roughly agree with the dynamics found by applying jPCA directly to the model. Along these lines, we validate the use of JSLDS by confirming that the JSLDS generator dynamics agree with the dynamics found by fitting jPCA to the RNN generator states (Fig. 4E-F). We see that the eigenvalues reported by jPCA (constrained to be purely imaginary) revealed four frequencies that closely agreed with the top four frequencies found by analyzing the JSLDS generator dynamics (Fig 4E). A subspace angle analysis confirms this correspondence by showing that the four corresponding planes in the JSLDS state space (defined by the eigenvectors) were similar to the first four jPCA planes (Fig. 4F). Having validated the use of JSLDS for this model, we see that LFADS-JSLDS learns to represent the dynamics for this task using a single, condition-independent linear system, a fact that was not obvious a priori.

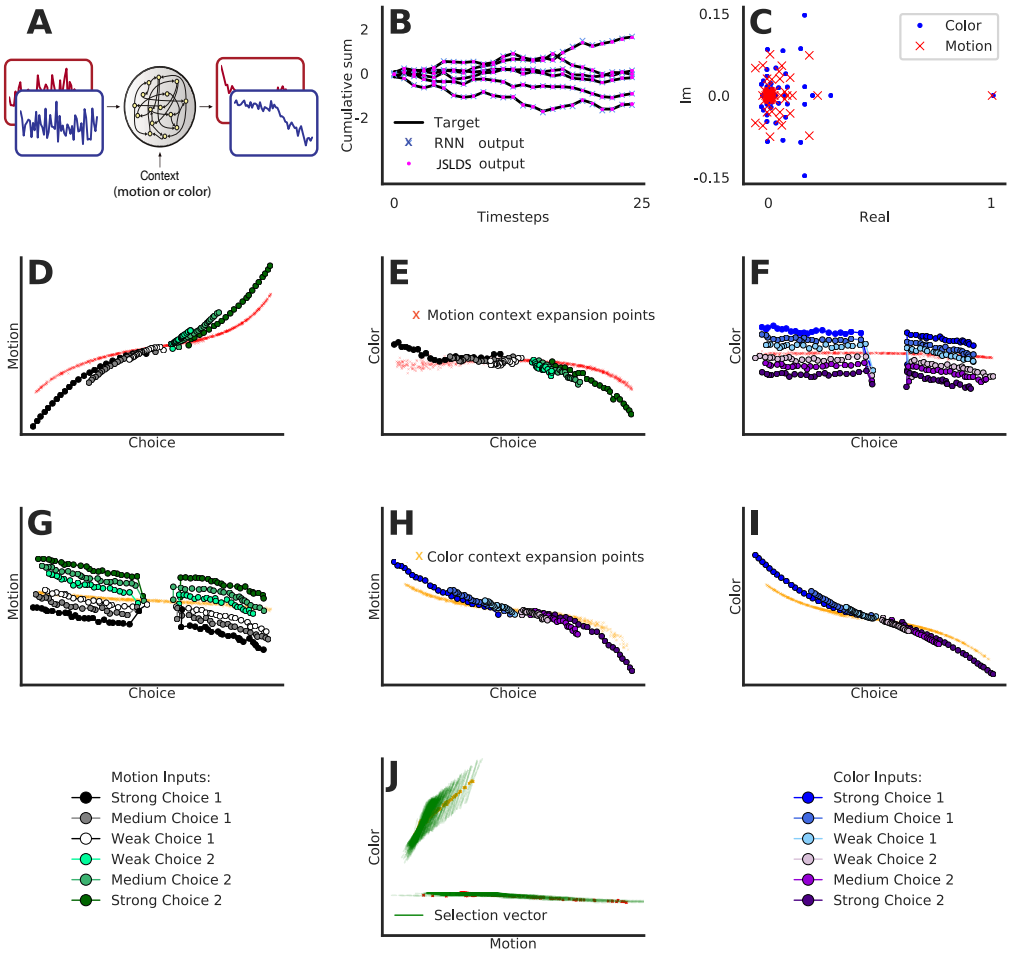

Figure 3: Context-dependent integration **A.** One of two white-noise input streams (motion or color) is selected to be integrated based on a static context input. The other stream is ignored. **B.** Sample held-out trial outputs show close agreement between JSLDS and RNN. **C.** Typical eigenvalues at a sample expansion point for motion (red x's) and color (blue dots) contexts. **D-J.** JSLDS has learned to switch between two continuous manifolds of fixed points and exhibits the solution behavior expected from [17]. JSLDS states (averaged) and expansion points are projected into the subspace spanned by the axes of choice, motion and color. Movement along the choice axis represents integration of evidence and the relevant input stream deflects along the relevant input axis. The input axes of **E,F,G,H** have been intensified. The trials used in **F** and **G** are the same trials as **D-E** and **H-I**, respectively, but re-sorted and averaged according to the direction and strength of the irrelevant input. The expansion points were computed separately for motion (red x's) and color contexts (orange x's). **J.** Global arrangement of the selection vectors (green lines) and line attractor expansion points for both contexts projected onto the input axes. Inputs are selected by the selection vector (which is approximately orthogonal to the contextually irrelevant input) and integrated along the line attractor.

5 Discussion

This work addresses the challenging problem of improving our understanding of how RNNs perform computations and was inspired by combining ideas from reverse engineering RNNs [15, 17–19, 21, 22, 10] and SLDS models [30–33]. We introduced a new model, the JSLDS, to improve our ability to reverse engineer RNNs. We applied it in a variety of settings and to different architectures: Vanilla RNNs, GRUs, and LFADS models. The JSLDS does not require post-training fixed point optimizations, significantly reduces the approximation error associated with reconstructing the

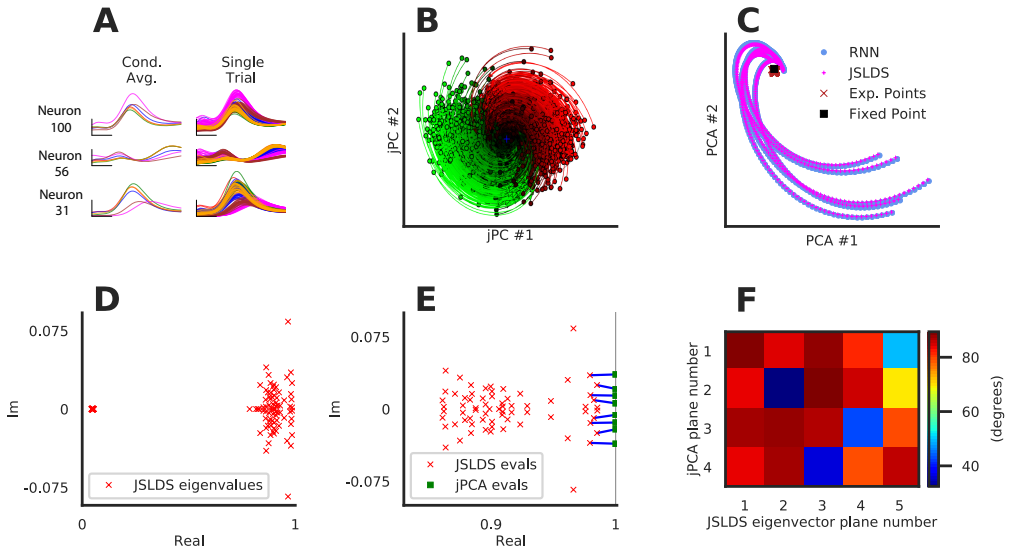

Figure 4: Monkey maze task. **A.** LFADS-JSLDS inferred firing rates for sample neurons. **B**. RNN generator exhibits the expected rotational dynamics. **C**. Sample trial dynamics show low approximation error between RNN and JSLDS. Note also the JSLDS expansion points and RNN numerical fixed point. **D**. JSLDS allows us to see LFADS-JSLDS has learned to organize the movement dynamics with this single condition-independent linear system. **E**. Top 70 eigenvalues from the same linear system shown in D along with the purely imaginary eigenvalues associated with the jPCA analysis (green squares). The jPCA eigenvalues are connected (blue line) to their corresponding JSLDS eigenvalues given by the subspace analysis in F. **F**. Subspace analysis comparing the jPC planes and JSLDS planes corresponding to the five slowest decaying oscillatory modes. Color indicates the minimum subspace angle between the corresponding planes. Angles of 30-40 degrees indicate highly overlapping subspaces. This comparison to jPCA validates the use of JSLDS.

nonlinear dynamics using the locally linearized solutions, and maps each point in state space to a fixed point. These benefits significantly improve our ability to reverse engineer RNNs, assuming a state-dependent SLDS can provide a good approximation of the original nonlinear RNN trained on a particular task. In situations where an SLDS does not provide a good approximation to a trained nonlinear RNN, a performance decrease using JSLDS can indicate this and serve as a potential measure of how "nonlinear" a task is. In addition to the above, JSLDS generalizes SLDS models to a potential continuum of discrete states with only a minor increase in parameters (ϕ of the shallow expansion network **E**) and automatically learns the number of discrete states.

In the 3-bit memory task, the JSLDS co-training changed the fixed point structure the co-trained RNN used to solve the task compared to the standard GRU solution. This suggests JSLDS can potentially regularize a nonlinear RNN towards a more simple switching linear dynamical solution. If there are multiple solutions to a task, knowing that one solution is described well by simpler switching linear dynamics could be valuable. This regularization towards a switching linear structure could also potentially have beneficial performance effects for robustness and generalization on held-out data. However, larger-scale studies are required to establish this.

Broader Impact While understanding how RNNs perform complex computations could eventually help bound expected model behavior, identify biases, improve robustness to adversarial inputs and suggest ways to improve performance, we foresee no immediate societal consequences of this work. Overall, this work makes it easier to reverse engineer RNNs and continue to gain insight into how these models work to benefit both neuroscience and machine learning.

References

[1] Rohit Prabhavalkar, Kanishka Rao, Tara N Sainath, Bo Li, Leif Johnson, and Navdeep Jaitly. A comparison of sequence-to-sequence models for speech recognition. In *Interspeech*, pages 939–943, 2017.

[2] Debidatta Dwibedi, Jonathan Tompson, and Pierre Sermanet. Temporal reasoning in videos using convolutional gated recurrent units. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[3] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2392–2396, 2017.

[4] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[5] Chethan Pandarinath, Daniel J O'Shea, Jasmine Collins, Rafal Jozefowicz, Sergey D Stavisky, Jonathan C Kao, Eric M Trautmann, Matthew T Kaufman, Stephen I Ryu, Leigh R Hochberg, et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 15(10):805–815, 2018.

[6] Christopher J. Cueva and Xue-Xin Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. In *International Conference on Learning Representations*, 2018.

[7] Ben Sorscher, Gabriel Mel, Surya Ganguli, and Samuel Ocko. A unified theory for the origin of grid cells through the lens of pattern formation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[8] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Modayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 2018.

[9] Ingmar Kanitscheider and Ila Fiete. Emergence of dynamically reconfigurable hippocampal responses by learning to perform probabilistic spatial reasoning. *bioRxiv*, 2017. doi: 10.1101/231159.

[10] Rylan Schaeffer, Mikail Khona, Leenoy Meshulam, Brain Laboratory International, and Ila Fiete. Reverse-engineering recurrent neural network solutions to a hierarchical inference task for mice. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4584–4596. Curran Associates, Inc., 2020.

[11] Fu-Sheng Tsung and Garrison Cottrell. Phase-space learning. pages 481–488, 01 1994.

[12] David Zipser. Recurrent network model of the neural mechanism of short-term active memory. *Neural Computation*, 3(2):179–193, 1991. doi: 10.1162/neco.1991.3.2.179.

[13] Michael Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8:1135–1178, 1996.

[14] Paul Rodriguez, Janet Wiles, and Jeffrey L Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

[15] David Sussillo and Omri Barak. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Comput.*, 25(3):626–649, March 2013. ISSN 0899-7667. doi: 10.1162/NECO_a_00409.

[16] I. Jordan, Piotr A. Sokól, and Il Memming Park. Gated recurrent units viewed through the lens of continuous time dynamical systems. *ArXiv*, abs/1906.01005, 2019.

[17] Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78–84, 2013.

[18] Niru Maheswaranathan, Alex Williams, Matthew Golub, Surya Ganguli, and David Sussillo. Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[19] Niru Maheswaranathan and David Sussillo. How recurrent networks implement contextual processing in sentiment analysis. *ArXiv*, abs/2004.08013, 2020.

[20] Federico Carnevale, Victor de Lafuente, Ranulfo Romo, Omri Barak, and Néstor Parga. Dynamic control of response criterion in premotor cortex during perceptual detection under temporal uncertainty. *Neuron*, 86(4):1067–1077, 2015. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2015.04.014.

[21] David Sussillo, Mark M Churchland, Matthew T Kaufman, and Krishna V Shenoy. A neural network that finds a naturalistic solution for the production of muscle activity. *Nature neuroscience*, 18(7):1025–1033, 2015.

[22] Niru Maheswaranathan, Alex Williams, Matthew Golub, Surya Ganguli, and David Sussillo. Universality and individuality in neural dynamics across large populations of recurrent networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[23] Arseny Finkelstein, Lorenzo Fontolan, Michael N. Economo, Nuo Li, Sandro Romani, and Karel Svoboda. Attractor dynamics gate cortical information flow during decision-making. *bioRxiv*, 2019. doi: 10.1101/2019.12.14.876425.

[24] G. Ackerson and K. Fu. On state estimation in switching environments. *IEEE Transactions on Automatic Control*, 15(1):10–17, 1970. doi: 10.1109/TAC.1970.1099359.

[25] C. B. Chang and M. Athans. State estimation for discrete systems with switching parameters. *IEEE Transactions on Aerospace and Electronic Systems*, AES-14(3):418–425, 1978. doi: 10.1109/TAES.1978.308603.

[26] Kevin P. Murphy. Switching Kalman filters. Technical report, 1998.

[27] Zoubin Ghahramani and Geoffrey E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):831–864, 2000. doi: 10.1162/089976600300015619.

[28] Emily Fox, Erik Sudderth, Michael Jordan, and Alan Willsky. Nonparametric Bayesian learning of switching linear dynamical systems. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2009.

[29] Matthew D. Golub and David Sussillo. Fixedpointfinder: A tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. *Journal of Open Source Software*, 3(31):1003, 2018. doi: 10.21105/joss.01003.

[30] Scott Linderman, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski. Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 914–922, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL `http://proceedings.mlr.press/v54/linderman17a.html`.

[31] Scott Linderman, Annika Nichols, David Blei, Manuel Zimmer, and Liam Paninski. Hierarchical recurrent state space models reveal discrete and continuous dynamics of neural activity in C. elegans. *bioRxiv*, 2019. doi: 10.1101/621540.

[32] Josue Nassar, Scott Linderman, Monica Bugallo, and Il Memming Park. Tree-structured recurrent switching linear dynamical systems for multi-scale modeling. In *International Conference on Learning Representations*, 2019.

[33] Joshua Glaser, Matthew Whiteway, John P Cunningham, Liam Paninski, and Scott Linderman. Recurrent switching dynamical systems models for multiple interacting neural populations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14867–14878. Curran Associates, Inc., 2020.

[34] P. Werbos. Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560, 1990.

[35] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179.

[36] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[37] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[38] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Justin D Foster, Paul Nuyujukian, Stephen I Ryu, and Krishna V Shenoy. Neural population dynamics during reaching. *Nature*, 487(7405):51–56, 2012.

[39] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Stephen I Ryu, and Krishna V Shenoy. Cortical preparatory activity: representation of movement or first cog in a dynamical machine? *Neuron*, 68(3):387–400, 2010.

Checklist

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

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

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

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] See broader impact statement.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [N/A]
(b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will include a Colab notebook with code to train the model for the synthetic tasks. We are unable to provide the monkey data as it is proprietary.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We did not conduct large scale experiments which would make this necessary.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]