Structured flexibility in recurrent neural networks via neuromodulation

Julia C. Costacurta [1,2,*], Shaunak Bhandarkar[3,*], David M. Zoltowski[1,4], and Scott W. Linderman[1,4]

Abstract

The goal of theoretical neuroscience is to develop models that help us better understand biological intelligence. Such models range broadly in complexity and biological detail. For example, task-optimized recurrent neural networks (RNNs) have generated hypotheses about how the brain may perform various computations, but these models typically assume a fixed weight matrix representing the synaptic connectivity between neurons. From decades of neuroscience research, we know that synaptic weights are constantly changing, controlled in part by chemicals such as neuromodulators. In this work we explore the computational implications of synaptic gain scaling, a form of neuromodulation, using task-optimized low-rank RNNs. In our neuromodulated RNN (NM-RNN) model, a neuromodulatory subnetwork outputs a low-dimensional neuromodulatory signal that dynamically scales the low-rank recurrent weights of an output-generating RNN. In empirical experiments, we find that the structured flexibility in the NM-RNN allows it to both train and generalize with a higher degree of accuracy than low-rank RNNs on a set of canonical tasks. Additionally, via theoretical analyses we show how neuromodulatory gain scaling endows networks with gating mechanisms commonly found in artificial RNNs. We end by analyzing the low-rank dynamics of trained NM-RNNs, to show how task computations are distributed.

∗ Denotes equal contribution
[1] Wu Tsai Neurosciences Institute, Stanford, CA, USA
[2] Department of Electrical Engineering, Stanford, CA, USA
[3] Department of Mathematics, Stanford, CA, USA
[4] Department of Statistics, Stanford University, Stanford, CA, USA
Correspondence should be addressed to: J.C.C. (jcostac@stanford.edu) or S.W.L. (scott.linderman@stanford.edu)

1 Introduction

Humans and animals show an innate ability to adapt and generalize their behavior across various environments and contexts. This suggests that the neural computations producing these behaviors must have flexible dynamics that are able to adjust to these novel conditions. Given the popularity of recurrent neural networks (RNNs) in studying such neural computations, a key question is whether this flexibility is (1) adequately and (2) accurately portrayed in RNN models of computation.

Traditional RNN models have fixed input, recurrent, and output weight matrices. Thus, the only way for an input to impact a dynamical computation is via the static input weight matrix. Prior work has shown that flexible, modular computation is possible with these models [1], but neurobiology suggests alternative mechanisms that may be at play. In particular, experimental neuroscience research has shown that synaptic strengths in the brain (akin to weight matrix entries in RNNs) are constantly changing — in part due to the influence of neuromodulatory signals [2].

Neuromodulatory signals are powerful and prevalent influences on neural activity and subsequent behavior. Dopamine, a well-known example, is implicated in motor deficits resulting from Parkinson's disease and has been the subject of extensive study by neuroscientists [3]. For computational study, neuromodulators are especially interesting because of their effects on synaptic connections and learning [4]. In particular, neuromodulators have been shown to alter synaptic strength between neurons, effectively reconfiguring circuit dynamics [5].

In this work, we seek to incorporate a neuromodulatory signal into task-trained RNN models. We propose a model consisting of a pair of RNNs: a small "neuromodulatory" RNN and a larger, low-rank "output-generating" RNN. The neuromodulatory RNN controls the weights of the output-generating RNN by scaling each rank-1 component of its recurrent weight matrix. This allows the network to produce flexible, yet structured dynamics that unfold over the course of a task. We first review background work in both the machine learning and computational neuroscience literature. Next, we introduce the model and provide some intuition for the impact of neuromodulation on the model's dynamics, relating it to the canonical long short-term memory (LSTM) network [6]. We end by presenting the performance and generalization capabilities of neuromodulated RNNs on a variety of neuroscience and machine learning tasks, showcasing the ability of a relatively simple, biologically-motivated augmentation to enhance the capacity of RNN models.

2 Background

First, we review related work in the theoretical neuroscience and machine learning literature.

2.1 Modeling neuromodulatory signals

Our work builds on a body of literature dating back to the 1980s, when pioneering computational neuroscientists added neuromodulation to small biophysical circuit models (for reviews, see [7–9]). These models consist of coupled differential equations whose biophysical parameters are carefully specified to simulate biologically-accurate spiking activity. As Marder [5] relates in her retrospective review, such neuromodulatory models were created in response to neuronal circuit models that viewed circuit dynamics as "hard-wired". Neuromodulatory mechanisms offered an answer to experimental observations that anatomically fixed biological circuits were capable of producing variable outputs [10, 11]. Of particular relevance to this work, Abbott [12] showed in his 1990 paper that adding a neuromodulatory parameter to an ODE model of spiking activity allows a network of neurons to display capacity for both long- and short-term memory and gate the learning process, anticipating the LSTMs that would become prominent a few years later. We also draw comparisons between our model and the LSTM in the sections that follow. However, our work does not aim to model any specific biophysical system;

rather, it aims to bridge the gap between these highly biologically-accurate models and general network models (i.e. RNNs) of neuronal activity by adding a biologically-motivated form of structured flexibility.

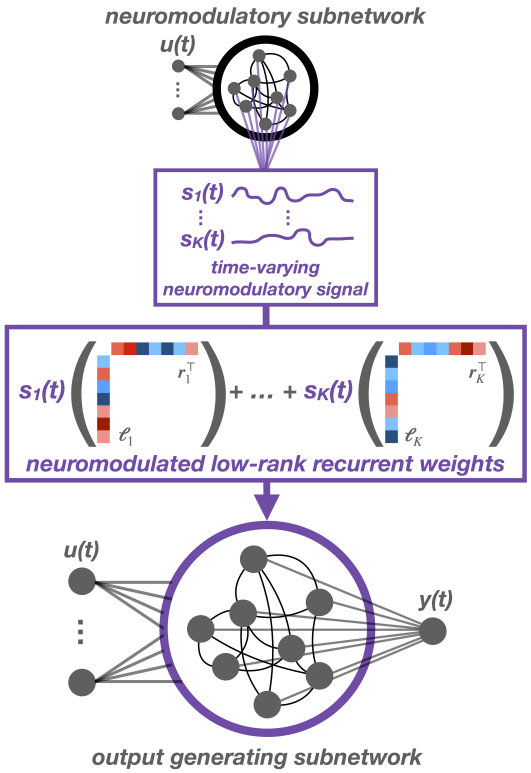

Figure 1: The NM-RNN consists of two subnetworks: a low-rank subnetwork that generates the output (bottom) and a smaller, full-rank neuromodulatory subnetwork (top).

More recent attempts to model neuromodulation have taken advantage of increased computational power. Kennedy et al. [13] modeled modulatory influence in spiking RNNs by linearly scaling the firing rates of a subset of neurons. Papadopoulos et al. [14] also use spiking RNNs, and incorporate arousal-mediated modulatory signals to induce phase transitions. Stroud et al. [15] use a balanced excitatory/inhibitory RNN to model motor cortex, and incorporate modulatory signals as constant multipliers on each neuron's activity. Liu et al. [16] study how neuromodulatory signals facilitate credit assignment during learning in spiking neural networks. Our work also aims to incorporate a modulatory signal into an existing neural network model, however, we specifically examine the potential of neuromodulation to augment task-trained low-rank RNNs.

The work of Tsuda et al. [17] is most similar to what we present here. The authors train RNNs using constant, multiplicative neuromodulatory signals applied to pre-specified subsets of the recurrent weights. They show that these neuromodulatory signals allow an otherwise fixed network to perform variations of a task. In contrast, we employ time-dependent neuromodulatory signals that allow dynamics to evolve throughout tasks. Instead of pre-specifying the values and regions of impact of neuromodulators, we allow the model to learn the time-varying neuromodulatory signal and what segment of the neuronal population it impacts.

2.2 Hypernetworks

Our approach is closely related to recent work using hypernetworks to enhance model capacity in machine learning. Ha et al. [18] use small networks (termed hypernetworks) to generate parameters for layers of larger networks. In their HyperRNN, a hypernetwork generates the weight matrix of an RNN as the linear combination of a learned set of matrices. We also allow our neuromodulatory network to specify the weight matrix of a larger RNN as a linear combination of a learned set of matrices; however, our learned matrices are rank-1 to facilitate easier dynamical analysis and faster training. It is also worth noting that in practice, Ha et al. [18] simplify their HyperRNN so that the hypernetwork scales the rows of a learned weight matrix, which could be seen as postsynaptic scaling in our model. Similarly, von Oswald et al. [19] study the ability of hypernetworks to learn in the multitask and continual learning setting. They find that hypernetworks trained to produce task-specific weight realizations achieve high performance on continual learning benchmarks. In exploring potential neuroscience applications of their work, they remark that while their approach might be unrealistic, a hypernetwork that outputs lower-dimensional modulatory signals could assist in implementing task-specific mode-switching. We seek to obtain similar performance gains with a more biologically plausible model of neuromodulation.

2.3 Low-rank recurrent neural networks

For a variety of tasks of interest, measured neural recordings are often well-described by a set of lower dimensional latent variables [20, 21] (although alternative views have been presented [22, 23]). Likewise, artificial neural networks trained to solve tasks that mimic those found in neural experiments also often exhibit low-rank structure. Based on these findings, recurrent neural networks with low-rank weight matrices (also called low-rank RNNs, LR-RNNs) have emerged as a popular class of models for studying neural dynamics [24–26]. Importantly, low-rank RNNs are able to model nonlinear dynamics that evolve in a low-rank subspace, offering potential for visualization and interpretation. Here, we leverage low-rank RNNs as ideal candidates for neuromodulation, with each factor of the low-rank recurrence matrix becoming a possible target for synaptic scaling.

3 Neuromodulated recurrent neural networks

Motivated by the appealing dynamical structure of low-rank RNNs and the ability of neuromodulation to add structured flexibility, we propose the *neuromodulated RNN* (NM-RNN). The NM-RNN consists of two linked subnetworks corresponding to neuromodulation and output generation. The output generating subnetwork is a low-rank RNN, which admits a natural way to implement neuromodulation. We allow the output of the neuromodulatory subnetwork to scale the low-rank factors of the output generating subnetwork's weight matrix. In particular, we propose incorporating neuromodulatory drive via a coupled ODE with neuromodulatory subnetwork state $\boldsymbol{z}(t) \in \mathbb{R}^M$ and output-generating state $\boldsymbol{x}(t) \in \mathbb{R}^N$:

$$\tau_z \frac{\mathrm{d}\boldsymbol{z}(t)}{\mathrm{d}t} = -\boldsymbol{z}(t) + \boldsymbol{W}_z\, \phi(\boldsymbol{z}(t)) + \boldsymbol{B}_z \boldsymbol{u}(t) \tag{1}$$

$$\tau_x \frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = -\boldsymbol{x}(t) + \boldsymbol{W}_x\big(\boldsymbol{z}(t)\big)\, \phi(\boldsymbol{x}(t)) + \boldsymbol{B}_x \boldsymbol{u}(t) \tag{2}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{d}, \tag{3}$$

where the dynamics matrix $\boldsymbol{W}_x\big(\boldsymbol{z}(t)\big)$ is a function of the neuromodulatory subnetwork state $\boldsymbol{z}(t)$ via a neuromodulatory signal $\boldsymbol{s}(\boldsymbol{z}(t)) \in \mathbb{R}^K$, which scales each low-rank component of $\boldsymbol{W}_x$:

$$\boldsymbol{s}\big(\boldsymbol{z}(t)\big) = \sigma(\boldsymbol{A}_z \boldsymbol{z}(t) + \boldsymbol{b}_z) \qquad \boldsymbol{W}_x\big(\boldsymbol{z}(t)\big) = \sum_{k=1}^{K} s_k\big(\boldsymbol{z}(t)\big)\, \boldsymbol{\ell}_k \boldsymbol{r}_k^\top. \tag{4}$$

The output-generating subnetwork is modeled as a size-N low-rank RNN where $\boldsymbol{u}(t) \in \mathbb{R}^P$ are the inputs, $\boldsymbol{B}_x \in \mathbb{R}^{N\times P}$ are the input weights, and $\phi(\cdot)$ is the tanh nonlinearity. The neuromodulatory subnetwork is modeled as a small "vanilla" RNN with its own time-constant τ_z, recurrence weights $\boldsymbol{W}_z \in \mathbb{R}^{M\times M}$, and input weights $\boldsymbol{B}_z \in \mathbb{R}^{M\times P}$. To limit the capacity of the neuromodulatory subnetwork, we take its dimension M to be smaller than the output-generating subnetwork's dimension N. We take $\tau_z \gg \tau_x$ since neuromodulatory signals are believed to evolve relatively slowly [27]. The neuromodulatory subnetwork state $\boldsymbol{z}(t)$ alters the rank-K dynamics matrix $\boldsymbol{W}_x \in \mathbb{R}^{N\times N}$ via a K-dimensional linear readout $\boldsymbol{s}(\boldsymbol{z}(t))$. The components of $\boldsymbol{s}$ act as linear scaling factors on each rank-1 component $\boldsymbol{\ell}_k \boldsymbol{r}_k^\top \in \mathbb{R}^{M\times M}$ of $\boldsymbol{W}_x$. For ease of notation, in the rest of the text we write $\boldsymbol{s}(t)$ to mean $\boldsymbol{s}(\boldsymbol{z}(t))$. The output $\boldsymbol{y}(t) \in \mathbb{R}^O$ of the NM-RNN is a linear readout of $\boldsymbol{x}(t)$.

This augmentation to the RNN framework allows for structured flexibility in computation. In traditional RNNs, the recurrent weight matrix is fixed, and thus the inputs to the system can only perturb the state of the network. In the NM-RNN, the neuromodulatory subnetwork can use information from the inputs to dynamically up- and down-weight individual low-rank components of the recurrent weight matrix, offering greater computational flexibility. As we will see below, this flexibility also allows the network to reuse dynamical components across different tasks and task conditions.

3.1 Mathematical intuition

To gain some intuition for the potential impacts of neuromodulation on RNN dynamics, first consider the case where $\boldsymbol{W}_x$ is symmetric (i.e., $\boldsymbol{\ell}_k = \boldsymbol{r}_k \, \forall k$) with the $\{\boldsymbol{\ell}_k\}_{1 \le k \le K}$ forming an orthonormal set, where the nonlinearity is removed (i.e., $\phi(x) = x$), and where there are no inputs (i.e., $\boldsymbol{u}(t) = 0 \, \forall t$). We can then reparameterize the system with a new hidden state $\boldsymbol{w}(t) = \boldsymbol{L}^\top \boldsymbol{x}(t)$, where $\boldsymbol{L} \in \mathbb{R}^{N \times K}$ is the matrix whose columns are $\boldsymbol{\ell}_k$ (so that $\boldsymbol{L}^T \boldsymbol{L} = \mathbf{I}$). This produces decoupled dynamics:

$$\tau_x \frac{\mathrm{d}\boldsymbol{w}(t)}{\mathrm{d}t} = -\boldsymbol{w}(t) + \mathbf{S}(t)\boldsymbol{w}(t) \tag{5}$$

where $\boldsymbol{S}(t) = \mathrm{diag}(\boldsymbol{s}(t))$. Solving this ODE gives an equation for the components of $\boldsymbol{w}(t)$:

$$w_k(t) = w_k(0) \exp\left(- \int_0^t \frac{\left(1 - s_k(t')\right)}{\tau_x} \, \mathrm{d}t' \right)$$

From this equation and the visualization in fig. 2A, we see that the decay rate of each component $w_k(t)$ is governed by its corresponding neuromodulatory signal $s_k(t)$. In this way, $\boldsymbol{s}(t)$ can effectively speed up or slow down decay of dynamic modes, similar to gating in an LSTM.

3.2 Connection to LSTMs

Having observed that neuromodulation can alter the timescales of dynamics, note further that the low-rank update for the linearized NM-RNN in eq. (5) mirrors the cell-state update equation for a long short-term memory (LSTM) cell [6]. Specifically, the neuromodulatory signal $\boldsymbol{s}(t)$ resembles the forget gate of an LSTM (fig. 2B). Indeed, as in eq. (5), if we linearize the output-generating subnetwork of the NM-RNN and assume that $\boldsymbol{L} = \boldsymbol{R}$ (so that $\boldsymbol{W}_x$ is symmetric), and $\boldsymbol{L}^T \boldsymbol{L} = I$,then for $\boldsymbol{w}(t) = \boldsymbol{L}^T \boldsymbol{x}(t)$ and $\tau_x = 1$, the discretized low-rank dynamics are given by

$$\boldsymbol{w}_t = \boldsymbol{s}_t \odot \boldsymbol{w}_{t-1} + \boldsymbol{L}^T \boldsymbol{B}_x \boldsymbol{u}_t \tag{6}$$

LSTMs have two states that recurrently update across each timestep t: a hidden state $\boldsymbol{h}_t \in \mathbb{R}^{N_{\mathrm{LSTM}}}$ and a cell state $\boldsymbol{c}_t \in \mathbb{R}^{N_{\mathrm{LSTM}}}$. Equation (6) closely mirrors the cell-state update of the LSTM:

$$\boldsymbol{c}_t = \mathbf{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{7}$$

Here, the forget gate $\mathbf{f}_t$ is a form of modulation that depends on the LSTM hidden state $\boldsymbol{h}_t$, much like the NM-RNN's neuromodulatory signal $\boldsymbol{s}(t)$ is a form of modulation depending on the NM-RNN's neuromodulatory subnetwork state $\boldsymbol{z}(t)$. The second term $\boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$ can be viewed as a gated transformation of the input signal $\boldsymbol{u}(t)$. In fact, under suitable assumptions, we show that the dynamics of an NM-RNN can be reproduced by those of an LSTM (see Theorem 1).

As a gated analog of the RNN, the LSTM has enjoyed greater success than ordinary RNNs in performing tasks that involve keeping track of long-distance dependencies in the input signal [28]. Thus, highlighting the connection between the NM-RNN and LSTM suggests NM-RNNs may be able to model long-timescale dependencies better than regular RNNs (see Section 6).

4 Time interval reproduction

To evaluate the potential of neuromodulated RNNs to add structured flexibility, we first consider a timing task since neuromodulators such as dopamine are implicated in time measurement and perception [29]. In the

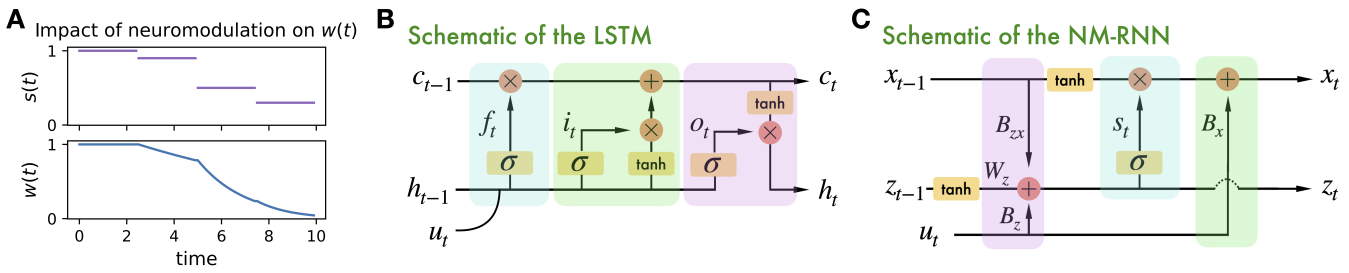

Figure 2: **A.** Illustration of how neuromodulatory signals $s(t)$ affect decay rates of the state $w(t)$ in a 1-D, simplified model. **B. & C.** Visual comparison of an LSTM and an NM-RNN. Corresponding parts of the networks are highlighted with shaded rectangles. Blue: a forget gate computation. Green: an input gate to recurrent dynamics. Purple: recurrent feedback onto the modulatory state variable.

Measure-Wait-Go (MWG) task (fig. 3A) [30], the network receives a 3-channel input containing the measure, wait, and go cues. The network must measure the interval between the measure and wait cues and reproduce it at the go cue by outputting a linear ramp of the same duration. Tasks such as this one are commonly used to study the neural underpinnings of timing perception in humans and non-human primates [31, 32].

4.1 Experiment matches theory for rank-1 networks

To investigate how the NM-RNN's neuromodulatory signal is constrained by the task requirements, we first consider a class of analytically tractable NM-RNNs: networks for which the output-generating subnetwork is linear (i.e., the tanh nonlinearity is replaced with the identity function) and rank-1. In this case, there is one pair of row and column factors, $\boldsymbol{\ell}$ and $\boldsymbol{r}$, respectively. If the target output signal is given by $f(t)$ and there are no inputs, then an NM-RNN that successfully produces this output signal will precisely have the neuromodulatory signal,

$$\boldsymbol{s}(t) = \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{(\boldsymbol{\ell}^\top \boldsymbol{r})\big(f(t) - \boldsymbol{c}^T \boldsymbol{w}^\perp(0) e^{-t/\tau_x} - \boldsymbol{d}\big) + (\boldsymbol{c}^\top \boldsymbol{\ell})\big(\boldsymbol{r}^\top \boldsymbol{w}^\perp(0) e^{-t/\tau_x}\big)},$$

where our readout is $y = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{d}$ and $\boldsymbol{w}^\perp(t) := \boldsymbol{x}(t) - \frac{1}{\|\boldsymbol{\ell}\|_2^2} \boldsymbol{\ell}\boldsymbol{\ell}^\top \boldsymbol{x}(t)$ is the component of $\boldsymbol{x}(t)$ evolving outside of the column space of $\boldsymbol{\ell}$ (viewed as a matrix in $\mathbb{R}^{N\times 1}$). For the full derivation, see Appendix B. If we assume further that $\boldsymbol{w}^\perp(0)$ is sufficiently small and τ_x is also sufficiently small, then we may make the approximation,

$$\boldsymbol{s}(t) \approx \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{\boldsymbol{\ell}^\top \boldsymbol{r}\,(f(t) - \boldsymbol{d})} = \frac{1}{\boldsymbol{\ell}^T \boldsymbol{r}} \left(1 + \frac{\tau_x f'(t)}{f(t) - \boldsymbol{d}}\right). \tag{8}$$

In the MWG task, no inputs are provided to the network during the ramping period following the go cue, so eq. (8) applies. In fig. 3B, we show that the neuromodulatory signal of a trained rank-1 NM-RNN during the ramping period matches closely with the theoretical prediction made by eq. (8) for both trained and extrapolated target intervals.

4.2 Improved generalization and interpretability on timing task for rank-3 networks

To continue our analysis of NM-RNNs on the MWG task, we increase the rank of the output-generating subnetwork to three. We do this to compare to the networks shown in Beiran et al. [30] and to showcase networks with more degrees of structured flexibility. In Beiran et al. [30], the authors show that rank-3 low-rank RNNs perform and extrapolate better on this task when provided a tonic context-dependent input, which varies depending on the length of the desired interval. As we have mentioned, such sensory inputs to the network may only alter the

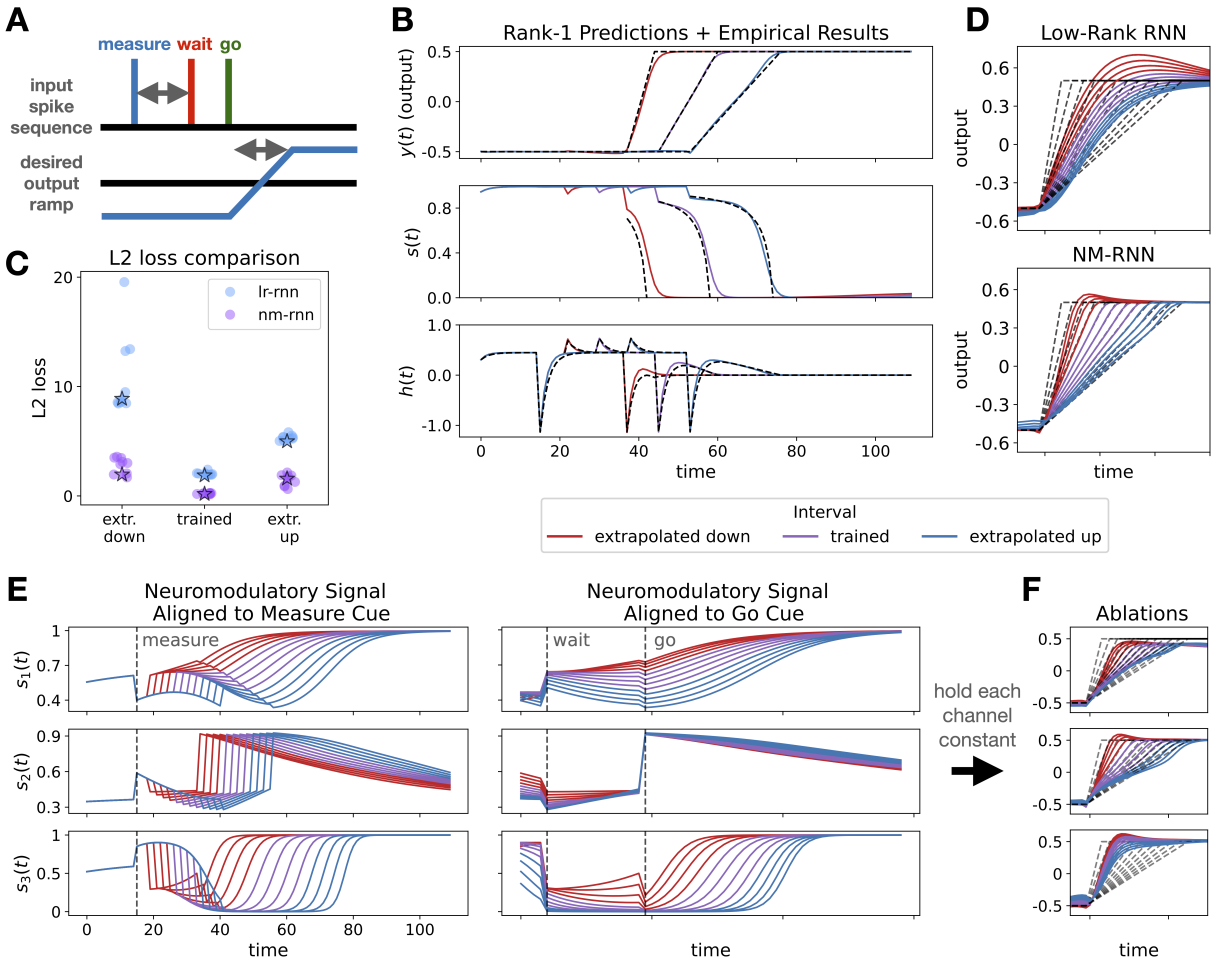

Figure 3: **A.** Visualization of Measure-Wait-Go task. **B.** Theoretical predictions (dashed lines) match closely with empirical results (solid lines) for rank-1 network. **C.** L2 loss comparsion for parameter-matched LR-RNNs and NM-RNNs. Performance of visualized models are starred. **D.** Comparison of model-generated output ramps for both trained and extrapolated intervals. **E.**Three-dimensional neuromodulatory signal $\boldsymbol{s}(t)$ for trained/extrapolated intervals. Left, traces are aligned to start of trial. Right, traces are aligned to 'go' cue. **F.** Resulting output traces when ablating each component of $\boldsymbol{s}(t)$. In all panels, colors reflect trained/extrapolated intervals (see legend). For output plots, dashed grey lines are targets. Additional model visualizations in figs. S1 and S2.

resulting dynamics by being passed through the input weight matrix. We propose the NM-RNN as an alternative mechanism by which inputs may alter the network dynamics.

We trained parameter-matched LR-RNNs ($N = 106,\ \tau = 10$) and NM-RNNs ($N = 100,\ M = 5,\ \tau_x = 10,\ \tau_z = 100$) to reproduce four intervals, then tested their extrapolation to longer and shorter intervals. In each of the networks, the low-rank matrix was chosen to have rank 3, as in Beiran et al. [30]. In fig. 3B, we plot the L_2 losses for ten instances of each model, showing that the NM-RNN consistently achieves a lower loss on both the trained and extrapolated intervals. In fig. 3C, we then show outputs for two typical networks (performance of these visualized networks indicated by stars in fig. 3B). The outputs of the NM-RNN have more accurate slope and shape for both trained and extrapolated intervals, with the LR-RNN struggling on shorter extrapolated intervals.

We then investigate how the neuromodulatory signal $\boldsymbol{s}(t)$ contributes to the task computation. Figure 3E shows the three dimensions of $\boldsymbol{s}(t)$ plotted over the full range of trained and extrapolated intervals. Each dimension

shows activity correlated to particular stages of the task. In fig. 3E (right), we see that $s_1(t)$ and $s_3(t)$ have activity highly correlated to the measured interval. In particular, after the go cue, $s_3(t)$ ramps at different speeds to saturate at 1, depending on the length of the interval. Figure 3F shows the result of ablating each dimension of $\boldsymbol{s}(t)$ by keeping that component fixed around its initial value. We see that performance suffers in all cases, especially when ablating the effect of $s_1(t)$ and $s_3(t)$. Most dramatically, ablating $s_3(t)$ destroys the ability of the network to change the slope of the output ramp appropriately. These results show how the network uses its neuromodulatory signal to generalize across task conditions.

5 Reusing dynamics for multitask learning

Next, we move beyond generalization within a single task to investigate the capabilites of the NM-RNN when switching between tasks. There has been recent interest in studying how neural network models might reassemble learned dynamical motifs to accomplish multiple tasks [1, 33]. Driscoll et al. [1] showed that an RNN trained to perform an array of tasks shares modular dynamical motifs across task periods and between tasks. With this result in mind, we were curious how the NM-RNN might use its neuromodulatory signal to flexibly reconfigure dynamics across tasks.

We performed our analysis using the four-task set from Duncker et al. [34], which includes the tasks DelayPro, DelayAnti, MemoryPro, and MemoryAnti illustrated in fig. 4A. In the DelayPro task, the network receives a three-channel input consisting of a fixation input and two sensory inputs which encode an angle $\theta \in [0, 2\pi)$ as $(\sin(\theta), \cos(\theta))$. The fixation input starts and remains at 1, then drops to 0 to signal the start of the *readout* period, when the network must generate its response. The sensory inputs appear after a delay, and persist throughout the trial. The goal of the network is to produce a three-channel output which reproduces the fixation and sensory inputs. In the MemoryPro task, the sensory inputs disappear before the readout period, requiring the network to store θ. In the 'Anti' tasks, the networks must instead produce the opposite sensory outputs, $(\sin(\theta + \pi), \cos(\theta + \pi))$, during the readout period. The task context is fed in as an additional one-hot input. These tasks are analogous to variants of the center-out reaching task, which has been used to study the neural mechanisms of arm movement in non-human primates [35].

To study the potential of NM-RNNs to flexibly reconfigure dynamics to perform a new task, we only fed the contextual inputs to the neuromodulatory subnetwork, and not to the output-generating subnetwork. This required the model to reuse the output-generating subnetwork's weights when adding a new task. We trained an NM-RNN to perform the first three tasks in the set (DelayPro, DelayAnti, MemoryPro), then froze the weights of the output-generating subnetwork and retrained only the neuromodulatory subnetwork's weights on the fourth task, MemoryAnti. We compared this to retraining the input weights of a LR-RNN, to investigate two strategies of processing context.

We trained parameter-matched LR-RNNs ($N = 100,\ \tau = 10$) and NM-RNNs ($N = 100,\ M = 20,\ \tau_x = 10,\ \tau_z = 100$) in this training/retraining framework. Figure 4B shows performance of example networks on the trained and retrained tasks, using the percent correct metric from Driscoll et al. [1]. While both models are able to learn the first three tasks, the NM-RNN more consistently performs the fourth task with high accuracy. This performance gain is not the result of retraining more parameters; in fact, due to the contrasting sizes of the neuromodulatory and low-rank subnetworks, the input weight matrix of the comparison low-rank RNN contains more parameters than the entire neuromodulatory subnetwork, since it must process all inputs (context, sensory, and fixation). To see exactly how the recurrent dynamics were rearranged for this new task, we plotted the neuromodulatory signal of an example network for learned and extrapolated tasks in fig. 4C.

We then analyzed the dynamical structure of one of the NM-RNNs by performing PCA on the output-generating subnetwork's state $\boldsymbol{x}(t)$ for a variety of input angles. Figure 4D (left) shows the first two PCs of the neural

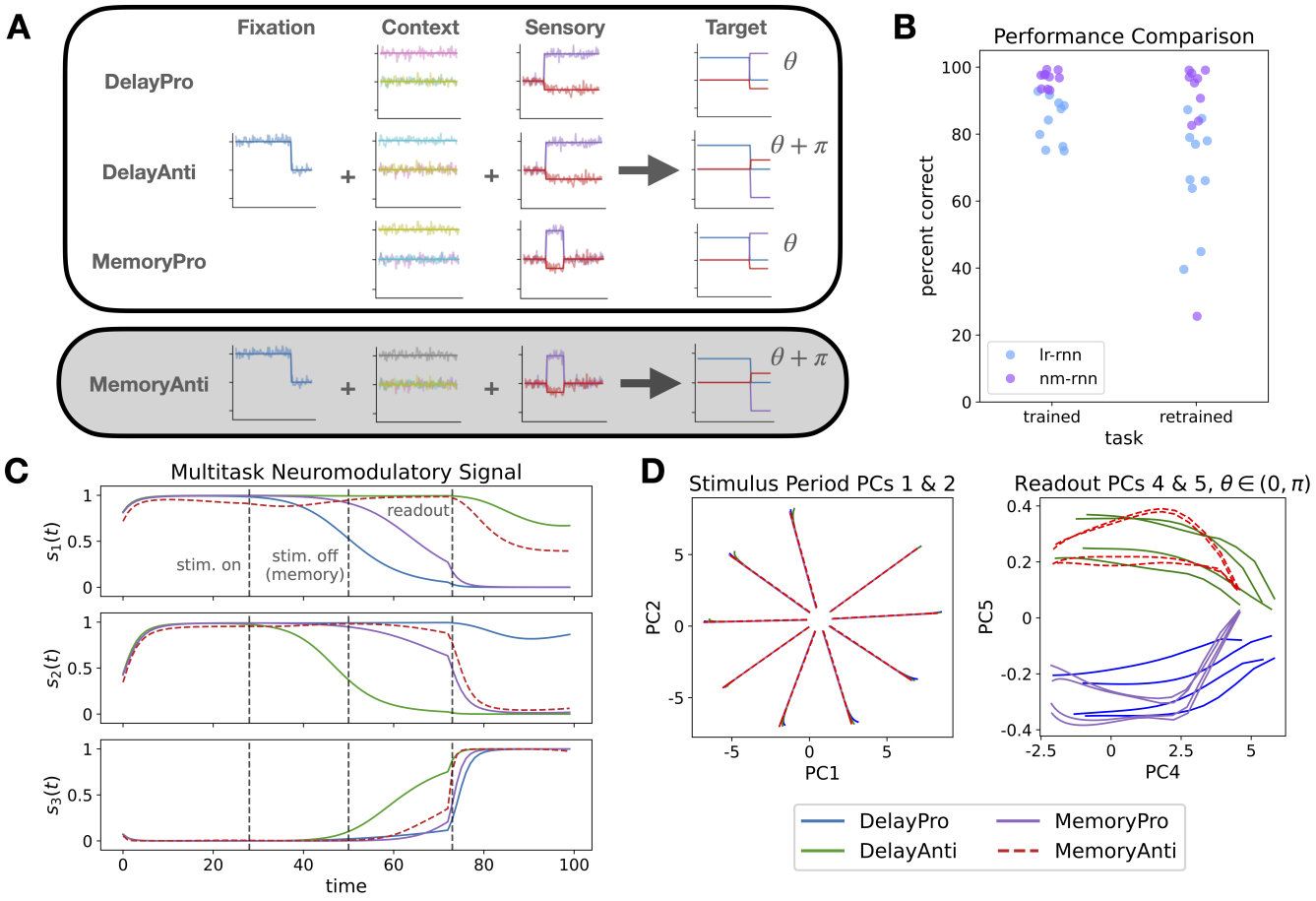

Figure 4: **A.** Depiction of inputs and targets for four tasks. Networks were trained on the first three tasks, then context-processing weights were retrained on the fourth task (in grey). **B.** Comparison of percent correct metric between parameter-matched low-rank and neuromodulated RNNs, for initial three tasks and retrained MemoryAnti task. **C.** Neuromodulatory signal for an example network. **D.** Dynamical analysis of network activity during different stages of the tasks. (Left) PCs 1&2 during the stimulus period show a ring attractor which stores the measured angle. (Right) Sign of PC5 during readout period corresponds to Pro/Anti. Additional model visualizations in figs. S3 and S4.

activity during the stimulus presentation period (before the stimulus shut off for Memory trials). During this period, the neural activity spreads out to arrange on a ring according to the measured angle. After the stimulus disappears in the MemoryPro/Anti tasks, the neural activity decays back along these axes, but it is still decodable based on its angle from the origin (see fig. S4). To find how this model encoded Pro/Anti versions of tasks, we performed another PCA on the neural activity during the readout period. As shown in fig. 4D (right), the sign of PC5 during this period is correlated with whether the task is Pro or Anti. Curiously, the positive/negative relationship flips for $\theta \in (\pi, 2\pi)$, likely relating to the symmetric structure of sine and cosine. These results show the ability of the NM-RNN to flexibly reconfigure the dynamics of the output-generating subnetwork, both to solve multiple tasks simultaneously and to generalize to a novel task.

6 Capturing long-term dependencies via neuromodulation

Inspired by the similarity between the coupled NM-RNN and the LSTM (see section 3.2), we designed a sequence-related task with long-term dependencies, called the *Element Finder Task* (EFT). On this task, gated models like the NM-RNN outperform ordinary RNNs. When endowed with suitable feedback coupling from the output-

generating subnetwork to the neuromodulatory subnetwork, the NM-RNN demonstrates LSTM-like performance on the EFT, while vanilla RNNs (with matched parameter count) fail to solve this task.

In the EFT (fig. 5A), the input stream consists of a query index, $q \in \{0, 1, \ldots, T-1\}$ followed by a sequence of T randomly chosen integers. The goal of the model is to recover the value of the element at index q from the sequence of integers. At each time for $t \geq 1$, the tth element of the sequence is passed as a one-dimensional input to the model. At time $t = T$, the model must output the value of the element at index q in the sequence. For our results (shown below), we took $T = 25$.

We trained several NM-RNNs, LR-RNNs, full-rank RNNs, and LSTMs on the EFT, conserving the total parameter count across networks. To emphasize its connection to the LSTM, each NM-RNN included an additional feedback coupling from $\boldsymbol{x}(t)$ to $\boldsymbol{z}(t)$:

$$\tau_z \frac{\mathrm{d}\boldsymbol{z}(t)}{\mathrm{d}t} = -\boldsymbol{z}(t) + \boldsymbol{W}_z \phi(\boldsymbol{z}(t)) + (\boldsymbol{B}_{zx}\, \phi(\boldsymbol{x}(t)) + \boldsymbol{b}_{zx}) + \boldsymbol{B}_z \boldsymbol{u}(t) \tag{9}$$

Each model used a linear readout with no readout bias. The resulting performances of each model tested are shown in fig. 5B. Figure 5C moreover illustrates the learning dynamics (as measured by MSE loss) for a single run of selected networks. Like LSTMs, NM-RNNs successfully perform the task, whereas low- and full-rank RNNs largely fail to do so.

To understand how a particular NM-RNN ($N = 18, M = 5, K = 8, \tau_x = 2, \tau_z = 10$) uses neuromodulatory gating to solve the EFT, we visualize the trial-averaged behavior of different components of $\boldsymbol{s}(t)$ across query indices ($q = 5, 10, 15, 20$), revealing that certain components of $\boldsymbol{s}(t)$ transition between 0 and 1 on a timescale correlated to the query index q (fig. 5D; left and right); while other components zero out (fig. 5D; middle). Visualizing a low-dimensional projection of $\boldsymbol{z}(t)$ across different query indices reveals that $\boldsymbol{z}(t)$ settles to a fixed point on an approximate line attractor encoding query index q (fig. 5E). These findings show that $\boldsymbol{z}(t)$ attends to the query index, facilitating gate-switching behavior in $\boldsymbol{s}(t)$ upon arrival of the queried element.

Next, we analyze $\boldsymbol{x}(t)$ by visualizing its top 2 principal components across each combination of the query indices $q = 5, 10$, and 15 and the target element values $-10, -5, 0, 5$, and 10 (fig. 5F). Trajectories with different element values but the same query index start at the same location. Each trajectory converges towards the origin, and upon arrival of the query timestep, rapidly moves to a fixed point on an approximate line attractor that encodes element value. The arrangement of fixed points along this line moreover preserves the ordering of their corresponding element values. In summary, these results show that the NM-RNN solves the EFT by distributing its computations across the neuromodulatory subnetwork, which attends to the query index, and the output-generating subnetwork, which retrieves the target element value.

7 Discussion

As we have shown, neuromodulated RNNs display an increased ability to both perform and generalize on tasks, both in the single-task setting and between different tasks. This performance is enabled by the structured flexibility neuromodulation adds to the dynamics of the network. Curiously, the gating-like dynamics introduced by adding neuromodulation create strong comparsions (and even equivalence) to the canonical LSTM.

Limitations. One limitation of this work relates to the scale of the networks tested. Our networks were on the scale of $N \approx 100$ neurons at their largest, as opposed to other related works which use neuron counts in the thousands. However, we found that this number of neurons was adequate to perform the tasks we presented. We also have yet to compare our results to neural data, limiting our ability to draw biological conclusions.

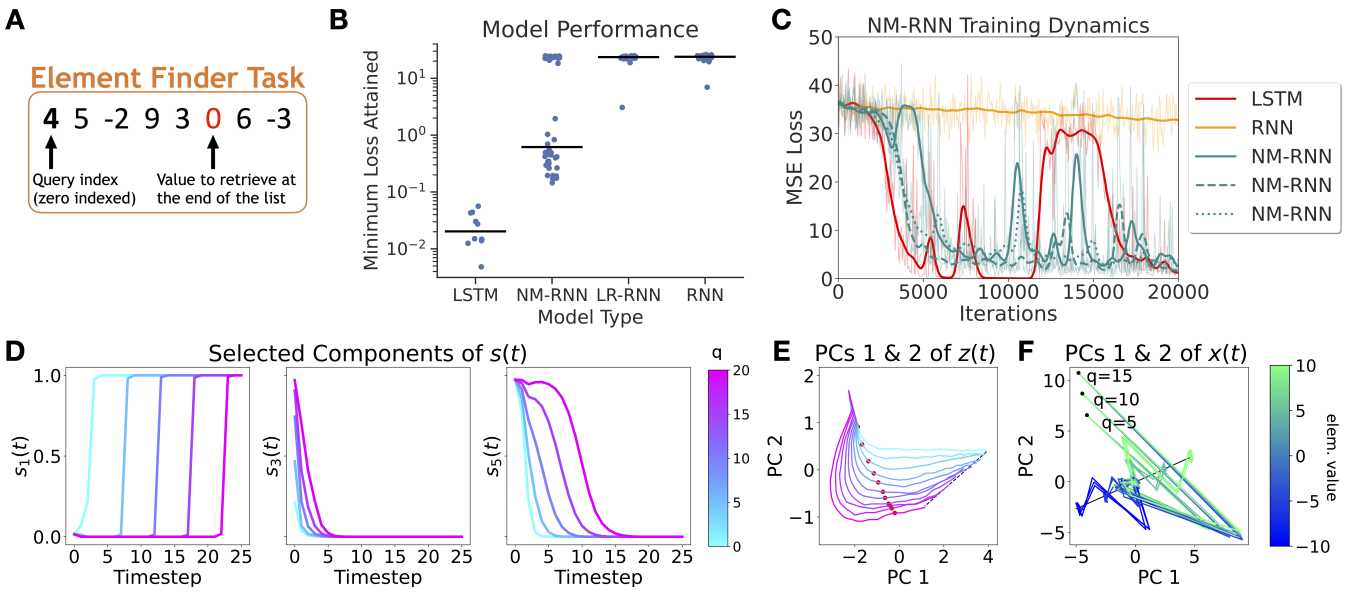

Figure 5: **A.** Visualization of the Element Finder Task. **B.** MSE losses attained across multiple runs in different classes of models trained on the EFT (median is indicated by black lines). **C.** Training loss curves for selected models. **D.** Visualization of selected components of $\boldsymbol{s}(t)$ for an example NM-RNN, shown across different query indices. **E.** Trajectories for the top two PCs of $\boldsymbol{z}(t)$ across different query indices. The different trajectories converge to an approximate line attractor (black) encoding query index. The time at which the queried element arrives is marked in red. **F.** Top two PCs of $\boldsymbol{x}(t)$, visualized for different query indices and target element values. Each trajectory converges to a fixed point on an approximate line attractor encoding element value. Each curve shown in **D**, **E**, and **F** is averaged over 100 trials. Additional visualizations are shown in figs. S5 and S6

Future Work. We are excited at the potential of future work to further bridge the gap between biophysical and recurrent neural network models. To expand on the NM-RNN model, we aim to embrace the broad range of roles neuromodulation can play in neural circuits. Potential future avenues include: (1) sparsifying the rank-1 components of the recurrent weight matrix to better imitate the ability of neuromodulators to act on spatially localized subpopulations of cells; (2) changing the readout function of $\boldsymbol{s}(t)$ to enable it to take both negative and positive values, in line with the ability of neuromodulators to act in both excitatory and inhibitory manners; and (3) investigating how different neuromodulatory effects may act on different timescales, both during task completion and learning over longer timescales [4, 5]. More generally, each neuron (or synapse) could have an internal state beyond its firing rate which is manipulated by neuromodulators, as in recent work investigating the role of modulation in generating novel dynamical patterns [36, 37]. Beyond neuromodulators, there exist a multitude of extrasynaptic signaling mechanisms in the brain, such as neuropeptides and hormones, which could function similarly in some cases, but also suggest new modeling directions.

In this work, we only analyzed networks post-training. We are also curious how our computational mechanism of neuromodulation impacts the network during learning. Prior work has modeled the role of neuromodulation in learning, for example, by augmenting the traditional Hebbian learning rule with neuromodulation to implement a three-factor learning rule [38], and by using neuromodulation to create a more biologically plausible learning rule for RNNs [39]. An interesting direction for future work is to study whether the neuomodulatory signal in the NM-RNN could produce similar learning dynamics.

Acknowledgements

We thank Laura Driscoll, Lea Duncker, Gyu Heo, Bernardo Sabatini, and the members of the Linderman Lab for helpful feedback throughout this project. This work was supported by grants from the NIH BRAIN Initiative (U19NS113201, R01NS131987, & RF1MH133778) and the NSF/NIH CRCNS Program (R01NS130789). J.C.C. is funded by the NSF Graduate Research Fellowship, Stanford Graduate Fellowship, and Stanford Diversifying Academia, Recruiting Excellence (DARE) Fellowship. D.M.Z. is funded by the Wu Tsai Interdisciplinary Postdoctoral Research Fellowship. S.W.L. is supported by fellowships from the Simons Collaboration on the Global Brain, the Alfred P. Sloan Foundation, and the McKnight Foundation. The authors have no competing interests to declare.

References

[1] Laura Driscoll, Krishna Shenoy, and David Sussillo. Flexible multitask computation in recurrent networks utilizes shared dynamical motifs. *bioRxiv*, pages 2022–08, 2022.

[2] Ami Citri and Robert C Malenka. Synaptic plasticity: multiple forms, functions, and mechanisms. *Neuropsychopharmacology*, 33(1):18–41, 2008.

[3] Marjan Jahanshahi, Catherine RG Jones, Jan Zijlmans, Regina Katzenschlager, Lucy Lee, Niall Quinn, Chris D Frith, and Andrew J Lees. Dopaminergic modulation of striato-frontal connectivity during motor timing in parkinson's disease. *Brain*, 133(3):727–745, 2010.

[4] Peter Dayan. Twenty-five lessons from computational neuromodulation. *Neuron*, 76(1):240–256, 2012.

[5] Eve Marder. Neuromodulation of neuronal circuits: back to the future. *Neuron*, 76(1):1–11, 2012.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[7] Jean-Marc Fellous and Christiane Linster. Computational models of neuromodulation. *Neural Computation*, 10(4):771–805, 1998.

[8] Ronald M Harris-Warrick and Eve Marder. Modulation of neural networks for behavior. *Annual Review of Neuroscience*, 14(1):39–57, 1991.

[9] Alex H Williams, Albert W Hamood, and Eve Marder. Neuromodulation in small networks. In *Encyclopedia of Computational Neuroscience*, pages 2300–2313. Springer, 2022.

[10] Eve Marder. Mechanisms underlying neurotransmitter modulation of a neuronal circuit. *Trends in Neurosciences*, 7(2):48–53, 1984.

[11] Eve Marder and Scott L Hooper. Neurotransmitter modulation of the stomatogastric ganglion of decapod crustaceans. In *Model Neural Networks and Behavior*, pages 319–337. Springer, 1985.

[12] Larry F Abbott. Modulation of function and gated learning in a network memory. *Proceedings of the National Academy of Sciences*, 87(23):9241–9245, 1990.

[13] Ann Kennedy, Prabhat S Kunwar, Ling-yun Li, Stefanos Stagkourakis, Daniel A Wagenaar, and David J Anderson. Stimulus-specific hypothalamic encoding of a persistent defensive state. *Nature*, 586(7831): 730–734, 2020.

[14] Lia Papadopoulos, Suhyun Jo, Kevin Zumwalt, Michael Wehr, David A McCormick, and Luca Mazzucato.

Modulation of metastable ensemble dynamics explains optimal coding at moderate arousal in auditory cortex. *bioRxiv*, April 2024.

[15] Jake P Stroud, Mason A Porter, Guillaume Hennequin, and Tim P Vogels. Motor primitives in space and time via targeted gain modulation in cortical networks. *Nature Neuroscience*, 21(12):1774–1783, December 2018.

[16] Yuhan Helena Liu, Stephen Smith, Stefan Mihalas, Eric Shea-Brown, and Uygar Sümbül. Cell-type–specific neuromodulation guides synaptic credit assignment in a spiking neural network. *Proceedings of the National Academy of Sciences*, 118(51):e2111821118, 2021.

[17] Ben Tsuda, Stefan C Pate, Kay M Tye, Hava T Siegelmann, and Terrence J Sejnowski. Neuromodulators generate multiple context-relevant behaviors in a recurrent neural network by shifting activity hypertubes. *bioRxiv*, pages 2021–05, 2021.

[18] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[19] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[20] John P Cunningham and Byron M Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, 2014.

[21] Saurabh Vyas, Matthew D Golub, David Sussillo, and Krishna V Shenoy. Computation through neural population dynamics. *Annual Review of Neuroscience*, 43:249–275, 2020.

[22] Carsen Stringer, Marius Pachitariu, Nicholas Steinmetz, Matteo Carandini, and Kenneth D Harris. High-dimensional geometry of population responses in visual cortex. *Nature*, 571(7765):361–365, 2019.

[23] Dean A Pospisil and Jonathan W Pillow. Revisiting the high-dimensional geometry of population responses in visual cortex. *bioRxiv*, pages 2024–02, 2024.

[24] Francesca Mastrogiuseppe and Srdjan Ostojic. Linking connectivity, dynamics, and computations in low-rank recurrent neural networks. *Neuron*, 99(3):609–623.e29, August 2018.

[25] Alexis Dubreuil, Adrian Valente, Manuel Beiran, Francesca Mastrogiuseppe, and Srdjan Ostojic. The role of population structure in computations through neural dynamics. *Nature Neuroscience*, 25(6):783–794, 2022.

[26] Manuel Beiran, Alexis Dubreuil, Adrian Valente, Francesca Mastrogiuseppe, and Srdjan Ostojic. Shaping dynamics with multiple populations in low-rank recurrent networks. *Neural Computation*, 33(6):1572–1615, 05 2021.

[27] Eve Marder and Ronald L Calabrese. Principles of rhythmic motor pattern generation. *Physiological Reviews*, 76(3):687–717, 1996.

[28] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, 07 2019.

[29] Allison E Hamilos, Giulia Spedicato, Ye Hong, Fangmiao Sun, Yulong Li, and John A Assad. Slowly evolving dopaminergic activity modulates the moment-to-moment probability of reward-related self-timed movements. *Elife*, 10, December 2021.

[30] Manuel Beiran, Nicolas Meirhaeghe, Hansem Sohn, Mehrdad Jazayeri, and Srdjan Ostojic. Parametric control of flexible timing through low-dimensional neural manifolds. *Neuron*, 111(5):739–753, 2023.

[31] Wilbert Zarco, Hugo Merchant, Luis Prado, and Juan Carlos Mendez. Subsecond timing in primates: comparison of interval production between human subjects and rhesus monkeys. *Journal of Neurophysiology*, 102(6):3191–3202, 2009.

[32] Akihisa Mita, Hajime Mushiake, Keisetsu Shima, Yoshiya Matsuzaka, and Jun Tanji. Interval time coding by neurons in the presupplementary and supplementary motor areas. *Nature Neuroscience*, 12(4):502–507, 2009.

[33] Guangyu Robert Yang, Madhura R Joglekar, H Francis Song, William T Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, 22 (2):297–306, 2019.

[34] Lea Duncker, Laura Driscoll, Krishna V Shenoy, Maneesh Sahani, and David Sussillo. Organizing recurrent network dynamics by task-computation to enable continual learning. *Advances in Neural Information Processing Systems*, 33:14387–14397, 2020.

[35] Cynthia A Chestek, Aaron P Batista, Gopal Santhanam, M Yu Byron, Afsheen Afshar, John P Cunningham, Vikash Gilja, Stephen I Ryu, Mark M Churchland, and Krishna V Shenoy. Single-neuron stability during repeated reaching in macaque premotor cortex. *Journal of Neuroscience*, 27(40):10742–10750, 2007.

[36] Kyle Aitken and Stefan Mihalas. Neural population dynamics of computing with synaptic modulations. *Elife*, 12:e83035, 2023.

[37] Kyle Aitken, Luke Campagnola, Marina E Garrett, Shawn R Olsen, and Stefan Mihalas. Simple synaptic modulations implement diverse novelty computations. *Cell Reports*, 43(5), 2024.

[38] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9:85, 2016.

[39] Yuhan Helena Liu, Stephen Smith, Stefan Mihalas, Eric Shea-Brown, and Uygar Sümbül. Biologically-plausible backpropagation through arbitrary timespans via local neuromodulators. *Advances in Neural Information Processing Systems*, 35:17528–17542, 2022.

[40] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[41] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/google-deepmind.

[42] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

[43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[44] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

A Formal connection between NM-RNNs and LSTMs

In this section, we mathematically formalize the connection between the NM-RNN and the LSTM. We show that, when considering suitable *linearized* analogs of the NM-RNN and the LSTM, the internal states and outputs of an NM-RNN may be reproduced by an LSTM. To that end, we begin by stating the necessary definitions.

First, we define a *linearized NM-RNN* via the following discretized dynamics:

$$\boldsymbol{x}_t = \frac{1}{N}\left(\boldsymbol{W}_x(\boldsymbol{z}_t)\right)\boldsymbol{x}_{t-1} + \boldsymbol{B}_x\boldsymbol{u}_t \tag{10}$$

$$\boldsymbol{z}_t = \left((1-\frac{1}{\tau_z})I + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\left(\boldsymbol{B}_{zx}\boldsymbol{x}_{t-1} + \boldsymbol{B}_z\boldsymbol{u}_t\right) \tag{11}$$

$$\boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{d} \tag{12}$$

As defined in the main text, we have that $\boldsymbol{W}_x(\boldsymbol{z}_t) = \boldsymbol{L}\boldsymbol{S}_t\boldsymbol{R}^T \in \mathbb{R}^{N\times N}$, where $S_t = \mathrm{diag}(\boldsymbol{s}_t)$ and $\boldsymbol{s}_t = \sigma(\boldsymbol{A}_z\boldsymbol{z}_t + \boldsymbol{b}_z) \in \mathbb{R}^K$. Note that we have added a $\frac{1}{N}$ prefactor to $\boldsymbol{W}_x$ in eq. (10). However, this does not fundamentally change the computation in Equation eq. (10) because we may imagine that this prefactor is absorbed by the matrices $\boldsymbol{L}$ and $\boldsymbol{R}$. Explicitly having the $\frac{1}{N}$ prefactor will be make the presentation of Proposition 1 more convenient.

Observe that the linearized NM-RNN is an NM-RNN with $\tau_x = 1$ and where all nonlinear transfer functions have been made to be the identity function. Notably, we have also added a *feedback coupling* term (given by the feedback weights $\boldsymbol{B}_{zx}$) from $\boldsymbol{x}(t)$ to $\boldsymbol{z}(t)$; this will serve to enhance the connection between this model and the LSTM.

Turning to the LSTM, we similarly define a suitable linearized relaxation – the *semilinear LSTM* – given by the following equations:

$$f_t = \sigma(W_f[h_{t-1}, u_t'] + b_f) \qquad i_t = \sigma(W_i[h_{t-1}, u_t'] + b_i) \tag{13}$$

$$\tilde{c}_t = W_c[h_{t-1}, u_t'] + b_c \qquad c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{14}$$

$$o_t = W_o[h_{t-1}, u_t'] + b_o \qquad h_t = o_t \odot c_t \tag{15}$$

Here, σ denotes the sigmoid nonlinearity, $u_t' \in \mathbb{R}^{N_{in}}$ denotes the input, and $h_t, c_t \in \mathbb{R}^{N_{LSTM}}$ are the hidden and cell states, respectively. The notation $[\cdot,\cdot]$ signifies concatenation of vectors. Each of the parameters W_k, b_k, for $k \in \{f, i, c, o\}$, are learnable.

A.1 Conditions leading to equivalence

Now, we formalize the connection between these two classes of models. It will be revealing to analyze the linearized NM-RNN in the case where $\boldsymbol{L} = \boldsymbol{R}$.

Proposition 1. *Consider a rank-K linearized NM-RNN, where the hidden size of the (linearized) output-generating network is N, and the hidden size of the neuromodulatory RNN is M. Moreover, assume that $\boldsymbol{L} = \boldsymbol{R}$ and that the columns of $\frac{1}{\sqrt{N}}\boldsymbol{L}$ are pairwise orthogonal, in the sense that $\frac{1}{N}\boldsymbol{L}^T\boldsymbol{L} = I_{K\times K}$. Finally, across all input sequences $\boldsymbol{u}_t$ on which the NM-RNN is tested, assume that the components of the states $\boldsymbol{x}_t$ and $\boldsymbol{z}_t$ are uniformly bounded over all timesteps. Then, this model's underlying K-dimensional dynamics (given by the low-rank variable $\boldsymbol{w}_t := \frac{1}{N}\boldsymbol{R}^T\boldsymbol{x}_t$), as well as its neuromodulatory states $\boldsymbol{z}_t$ and outputs $\boldsymbol{y}_t \in \mathbb{R}^O$, can be reproduced within a semilinear LSTM with a linear readout, whose inputs are $u_t' := \begin{bmatrix}\boldsymbol{u}_t \\ \boldsymbol{u}_{t-1}\end{bmatrix} \in \mathbb{R}^{2P}$. Here, $\boldsymbol{u}_t \in \mathbb{R}^P$ denotes the input to the NM-RNN at time t (with $\boldsymbol{u}_{-1} := 0 \in \mathbb{R}^P$ by convention). Moreover, such a semilinear LSTM can be made to have hidden size $K + M + P$ and utilize $O\left(\max\{K^2, M^2, PK, PM, OK, OP\}\right)$ learnable parameters.*

Proof of Proposition 1. Assuming $\boldsymbol{x}_t$ has uniformly bounded components, there exists some constant $H \in \mathbb{R}_+$ such that $|(\boldsymbol{x}_t)_i| < H$ for each $i \in \{1, \ldots, N\}$ and all t. Consequently, this means $\frac{1}{\sqrt{N}}||\boldsymbol{x}_t||_2 < H$ for all t. Now, consider the update equation for $\boldsymbol{x}_t$ in the linearized NM-RNN:

$$\boldsymbol{x}_t = \frac{1}{N}\boldsymbol{R}\boldsymbol{S}_t\boldsymbol{R}^T\boldsymbol{x}_{t-1} + \boldsymbol{B}_x\boldsymbol{u}_t.$$

For $\boldsymbol{R}_i$ the ith column of $\boldsymbol{R}$, the given orthogonality condition implies that $\frac{||\boldsymbol{R}_i||_2^2}{N} = 1$, or $\frac{1}{\sqrt{N}}||\boldsymbol{R}_i||_2 = 1$ in the large N limit. Defining $\boldsymbol{w}_t := \frac{1}{N}\boldsymbol{R}^T\boldsymbol{x}_t \in \mathbb{R}^R$ to the rank-K representation of $\boldsymbol{x}_t \in \mathbb{R}^N$, we have that $|(\boldsymbol{w}_t)_i| = \frac{1}{N}\left|\boldsymbol{R}_i^T\boldsymbol{x}_t\right| \leq \frac{1}{N}||\boldsymbol{R}_i||_2 \cdot ||\boldsymbol{x}_t||_2 = \frac{1}{\sqrt{N}} \cdot ||\boldsymbol{x}_t|| < H$. That is, the low-rank mode corresponding to $\boldsymbol{x}_t$ also has uniformly bounded components over time, i.e., the underlying K-dimensional dynamics of the output-generating network do not diverge.

Now, to show that the dynamics of a linearized NM-RNN satisfying the conditions given in the theorem statement can be replicated by a semilinear LSTM, we must effectively show that each of the update equations for the linearized NM-RNN (i.e., eqs. (10), (11) and (12)) can be suitably replicated through the semilinear LSTM architecture. In essence, we must suitably map the parameters of the given NM-RNN onto those of a semilinear LSTM.

First, we analyze the update equation for $\boldsymbol{w}_t$ in the output-generating network. Its corresponding K-dimensional (low-rank) update is

$$\boldsymbol{w}_t = \frac{1}{N}\boldsymbol{R}^T\boldsymbol{R}\boldsymbol{S}_t\boldsymbol{w}_{t-1} + \frac{1}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_t$$

$$\implies \boldsymbol{w}_t = \boldsymbol{S}_t\boldsymbol{w}_{t-1} + \frac{1}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_t.$$

Accordingly, define the cell state c_t of the corresponding semilinear LSTM to be

$$c_t = \begin{bmatrix} \boldsymbol{w}_{t-1} \\ \mathbf{1}_M \\ \mathbf{1}_P \end{bmatrix} \in \mathbb{R}^{K+M+P} \tag{16}$$

where $\mathbf{1}_\alpha$ denotes the vector of 1's in $\mathbb{R}^\alpha$ for $\alpha \in \{M, P\}$ (and, by convention, we set $\boldsymbol{w}_{-1} := 0 \in \mathbb{R}^K$). In particular, note that $\boldsymbol{x}_t$ can be recovered from the cell state via the relation $\boldsymbol{x}_t = \frac{1}{N}\boldsymbol{R}\boldsymbol{S}_t\boldsymbol{R}^T\boldsymbol{x}_{t-1} + \boldsymbol{B}_x\boldsymbol{u}_t = \boldsymbol{R}\boldsymbol{S}_t\boldsymbol{w}_{t-1} + \boldsymbol{B}_x\boldsymbol{u}_t = \boldsymbol{R}\boldsymbol{S}_t\text{Proj}_K c_t + \boldsymbol{B}_x\boldsymbol{u}_t$, where Proj_K denotes the projection matrix that sends c_t onto its first K components, namely, $\boldsymbol{w}_{t-1}$.

We may also set all the parameters in W_i and b_i (part of the input gate i_t) to be 0, so that $i_t = \frac{1}{2}\mathbf{1}_{K+M+P}$ (after applying the sigmoid nonlinearity). Then, define $\tilde{c}_t$ so that it equals $\begin{bmatrix} \frac{2}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \mathbf{0}_M \\ \mathbf{0}_P \end{bmatrix} \in \mathbb{R}^{K+M+P}$. Indeed,

$$\tilde{c}_t = W_c[h_{t-1}, u_t'] + b_c = W_c[h_{t-1}, \boldsymbol{u}_t, \boldsymbol{u}_{t-1}] + b_c$$

can be made to have its first K entries form the vector $\frac{2}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_{t-1}$ if we set $b_c = 0$ and zero out all columns of W_c that do not correspond to $\boldsymbol{u}_{t-1}$, and further by zeroing out the last $M + P$ rows of W_c. In block matrix form, we have defined

$$W_c = \left(\begin{array}{c|c|c} 0 & 0 & \frac{2}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \end{array}\right) \in \mathbb{R}^{(K+M+P)\times(K+M+3P)}.$$

Before explicitly computing the other gates, we first analyze how the semilinear LSTM might reproduce the update for the neuromodulatory state $\boldsymbol{z}_t \in \mathbb{R}^M$:

$$\boldsymbol{z}_t = \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\left(\boldsymbol{B}_{zx}\boldsymbol{x}_{t-1} + \boldsymbol{B}_z\boldsymbol{u}_t\right)$$

$$= \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{N\tau_z}\boldsymbol{B}_{zx}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{x}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{x}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t$$

$$= \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\boldsymbol{R}\boldsymbol{w}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{x}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t.$$

Note further that

$$\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{x}_{t-1} = \boldsymbol{x}_{t-1} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{x}_{t-1}$$

$$= \frac{1}{N}\boldsymbol{R}\boldsymbol{S}_{t-1}\boldsymbol{R}^T\boldsymbol{x}_{t-2} + \boldsymbol{B}_x\boldsymbol{u}_{t-1} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\left(\frac{1}{N}\boldsymbol{R}\boldsymbol{S}_{t-1}\boldsymbol{R}^T\boldsymbol{x}_{t-2} + \boldsymbol{B}_x\boldsymbol{u}_{t-1}\right)$$

$$= \frac{1}{N}\boldsymbol{R}\boldsymbol{S}_{t-1}\boldsymbol{R}^T\boldsymbol{x}_{t-2} + \boldsymbol{B}_x\boldsymbol{u}_{t-1} - \frac{1}{N}\boldsymbol{R}\boldsymbol{S}_{t-1}\boldsymbol{R}^T\boldsymbol{x}_{t-2} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_{t-1}$$

$$= \left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1}$$

where we have again used that $\frac{1}{N}\boldsymbol{R}^T\boldsymbol{R} = I_{K\times K}$. Thus, substituting this expression into our update equation for $\boldsymbol{z}_t$ above, we have

$$\boldsymbol{z}_t = \frac{1}{\tau_z}\boldsymbol{B}_{zx}\boldsymbol{R}\boldsymbol{w}_{t-1} + \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1} \quad (17)$$

Accordingly, define the output gate o_t of the semilinear LSTM to be

$$o_t := \begin{bmatrix} \frac{1}{2}\mathbf{1}_K \\ \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \boldsymbol{u}_{t-1} \end{bmatrix} \in \mathbb{R}^{K+M+P}.$$

Recalling that $o_t = W_o[h_{t-1}, \boldsymbol{u}_t, \boldsymbol{u}_{t-1}] + b_o$, the above gating is achieved by setting $b_o = 0$ and zeroing out the first K rows of W_o. The next M rows of W_o can be set to produce the middle entry in o_t shown above, and the last P rows of W_o can similarly be set so as to produce the vector $\boldsymbol{u}_{t-1}$. That is, in block matrix form, we may take W_o to be

$$W_o = \left(\begin{array}{c|c|c} 0 & 0 & 0 \\ \hline \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)W_{zh} & \frac{1}{\tau_z}\boldsymbol{B}_z & \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x \\ \hline 0 & 0 & I_{P\times P} \end{array}\right) \in \mathbb{R}^{(K+M+P)\times(K+M+3P)},$$

where $W_{zh} \in \mathbb{R}^{M\times(K+M+P)}$ is a suitable matrix (defined below) mapping the hidden state h_t of the semilinear LSTM to $\boldsymbol{z}_t$, so that $o_t = W_o[h_{t-1}, \boldsymbol{u}_t, \boldsymbol{u}_{t-1}]$. Then, computing $h_t = o_t \odot c_t \in \mathbb{R}^{K+M+P}$ gives

$$h_t = \begin{bmatrix} \frac{1}{2}\boldsymbol{w}_{t-1} \\ \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \boldsymbol{u}_{t-1} \end{bmatrix} \tag{18}$$

Now, observe that $\boldsymbol{z}_t = W_{zh}h_t$, where

$$W_{zh} = \left(\frac{2}{\tau_z}\boldsymbol{B}_{zx}\boldsymbol{R} \;\middle|\; I_{M\times M} \;\middle|\; 0\right) \in \mathbb{R}^{M\times(K+M+P)}.$$

(Thus, our earlier construction of output gate o_t as $o_t = W_o[h_{t-1}, \boldsymbol{u}_t, \boldsymbol{u}_{t-1}]$ is valid.) In particular, the equation $\boldsymbol{z}_t = W_{zh}h_t$ precisely corresponds to the update equation for $\boldsymbol{z}_t$ in the linearized NM-RNN given by eq. (17). Thus, at each time t, the hidden state h_t of the semilinear LSTM is a "deconstructed" version of $\boldsymbol{z}_t$, meaning that the model effectively reproduces $\boldsymbol{z}_t$ and its dynamics through h_t.

Finally, we set the forget gate so that it evaluates to $f_t = \begin{bmatrix} \boldsymbol{s}_{t-1} \\ \boldsymbol{1}_{M+P} \end{bmatrix}$. Recalling that $f_t = \sigma(W_f[h_{t-1}, u_t'] + b_f) \in \mathbb{R}^{K+M+P}$, we can achieve this gating by taking the first K entries of f_t to be $\boldsymbol{s}_{t-1} = \sigma(\boldsymbol{A}_z\boldsymbol{z}_{t-1} + \boldsymbol{b}_z) = \sigma(\boldsymbol{A}_z W_{zh}h_{t-1} + \boldsymbol{b}_z)$. We can also ensure that the last $M+P$ entries of f_t form the vector $\boldsymbol{1}_{M+P}$ by zeroing out the last $M+P$ rows of W_f and making the last $M+P$ entries of the bias vector b_f to be sufficiently large (so that applying the sigmoid function to these entries effectively yields 1). In other words, in block matrix form, we have

$$W_f = \left(\begin{array}{c|c|c} \boldsymbol{A}_z W_{zh} & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \end{array}\right) \in \mathbb{R}^{(K+M+P)\times(K+M+3P)}$$

and $b_f = \begin{bmatrix} \boldsymbol{b}_z \\ Q\boldsymbol{1}_{M+P} \end{bmatrix}$ where $Q \in \mathbb{R}_+$ may be chosen such that

$$Q >> M \max_{1\le i\le K, 1\le j\le M} \left|(\boldsymbol{A}_z)_{ij}\right| \sup_{1\le m\le M, t, \boldsymbol{u}_t} |(\boldsymbol{z}_t)_m| \ge ||\boldsymbol{A}_z\boldsymbol{z}_t||_\infty = \max_{1\le k\le K} \left|(\boldsymbol{A}_z\boldsymbol{z}_t)_k\right|.$$

(The above supremum is taken over all components of $\boldsymbol{z}_t$, all timesteps t – finitely or infinitely many – and all input sequences $(\boldsymbol{u}_t)_{t\ge 1}$ being considered. We then invoke uniform boundedness of the components of $\boldsymbol{z}_t$ to grant the existence of such a Q.)

Putting together our gate computations, the cell state update equation for the semilinear LSTM (eq. (14)) can be made to reproduce the low-rank update equation of the linearized NM-RNN's output-generating network:

$$\boldsymbol{w}_t = \boldsymbol{S}_t\boldsymbol{w}_{t-1} + \frac{1}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_t$$

$$\implies \begin{bmatrix} \boldsymbol{w}_{t-1} \\ \boldsymbol{1}_{M+P} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_{t-1} \\ \boldsymbol{1}_{M+P} \end{bmatrix} \odot \begin{bmatrix} \boldsymbol{w}_{t-2} \\ \boldsymbol{1}_{M+P} \end{bmatrix} + \frac{1}{2}\boldsymbol{1}_{K+M+P} \odot \begin{bmatrix} \frac{2}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \boldsymbol{0}_{M+P} \end{bmatrix}$$

$$\implies c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t.$$

Having seen that the corresponding semilinear LSTM is capable of replicating the low-rank update equation for $\boldsymbol{w}_t$ as well as (implicitly) reproducing the update equation for $\boldsymbol{z}_t$, we at last turn to analyzing the outputs $\boldsymbol{y}_t$ generated by the NM-RNN. The output readout for the NM-RNN (at time $t-1$, where $t \ge 2$) can be expressed as

$$\boldsymbol{y}_{t-1} = \boldsymbol{C}\boldsymbol{x}_{t-1} + \boldsymbol{d} = \boldsymbol{C}\left(\frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{x}_{t-1} + \left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{x}_{t-1}\right) + \boldsymbol{d}$$

$$= \boldsymbol{C}\boldsymbol{R}\boldsymbol{w}_{t-1} + \boldsymbol{C}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{x}_{t-1} + \boldsymbol{d}$$

$$\boldsymbol{C}\boldsymbol{R}\boldsymbol{w}_{t-1} + \boldsymbol{C}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1} + \boldsymbol{d} = C'h_t + d',$$

where we have used our earlier equation $\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{x}_{t-1} = \left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1}$. Furthermore, in the last equality, we have defined $d' = \boldsymbol{d}$ and $C' \in \mathbb{R}^{O\times(K+M+P)}$ that (in block matrix form) as

$$C' = \left(2\boldsymbol{C}\boldsymbol{R} \;\middle|\; 0 \;\middle|\; \boldsymbol{C}\left(I_{N\times N} - \tfrac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\right).$$

Therefore, we find that the outputs of the NM-RNN can be reproduced via a linear readout from the hidden state h_t of the semilinear LSTM.

Here, it should be noted that the semilinear LSTM we have constructed "lags" behind the NM-RNN by one timestep, i.e., the hidden state h_t is used to produce the $(t-1)$th output state $\boldsymbol{y}_{t-1}$. This is a consequence of the fact that at each timestep in the NM-RNN, the neuromodulatory state $\boldsymbol{z}_t$ is updated before $\boldsymbol{x}_t$, whereas in the semilinear LSTM, the cell state c_t (loosely corresponding to $\boldsymbol{x}_t$) is updated before the hidden state h_t (loosely corresponding to $\boldsymbol{z}_t$); as a result, the outputs of the semilinear LSTM are staggered by one timestep. In practice, this does not change the fact that the semilinear LSTM can replicate the outputs of the NM-RNN.

Having constructed a semilinear LSTM with hidden size $K+M+P$ that reproduces the states of the linearized NM-RNN, we finally turn to counting the total number of learnable parameters used by the semilinear LSTM (along with its linear readout):

1. f_t: W_f only transforms the hidden state h_t (i.e., $W_f[h_{t-1}, u_t'] = \boldsymbol{A}_z W_{zh} h_t$, where $\boldsymbol{A}_z W_{zh} \in \mathbb{R}^{K\times(K+M+P)}$), giving us $K(K+M+P)$ parameters. The bias vector b_f gives an additional $K+M+P$ parameters, for a total of $(K+1)(K+M+P)$ parameters.
2. i_t: We zero out all of these parameters, giving us a count of 0.
3. $\tilde{c}_t$: The only parameters used in this computation are those that linearly transform $\boldsymbol{u}_{t-1}$ into K-dimensional space (i.e., the elements of $\frac{2}{N}\boldsymbol{R}^T\boldsymbol{B}_x$), so the parameter count here is PK.
4. o_t: The bias term b_o was set to 0 and we defined the matrix W_o so that the middle M rows of W_o contained nontrivial entries, and the bottom P rows have P nontrivial parameters stemming from the identity matrix $I_{P\times P}$. Consequently, we obtain a contribution of $M(K+M+3P)+P$ parameters from the output gate computation.
5. Readout: We have that $d' \in \mathbb{R}^O$ and $C' \in \mathbb{R}^{O\times(K+M+P)}$ uses a total of $O(K+P)$ nontrivial parameters, giving us a total of $O(K+1+P)$ parameters used.

Thus, the number of nontrivial parameters used by this semilinear LSTM is

$$(K+1)(K+M+P) + PK + M(K+M+3P) + P + O(K+1+P)$$

$$= O\left(\max\{K^2, M^2, PK, PM, OK, OP\}\right).$$

□

B Derivation of exact neuromodulatory signal for rank-1 NM-RNNs

In this section, we precisely quantify how the neuromodulatory signal in a rank-1 NM-RNN is constrained by the target output signal. First, we derive the exact neuromodulatory signal in a general (possibly nonlinear) rank-1

NM-RNN that has successfully learned to produce a target output signal $f(t) \in \mathbb{R}$. Then, we specialize to the case in which the rank-1 NM-RNN has linear dynamics.

We start with a general rank-1 NM-RNN that produces the scalar output $\boldsymbol{y}(t) \in \mathbb{R}$ at each time t, and for which the output-generating network's nonlinearity is denoted by φ (which could be tanh, but also any other function). Because $K = 1$, we have $\boldsymbol{L} \in \mathbb{R}^{n\times 1}, \boldsymbol{R} \in \mathbb{R}^{n\times 1}$. Furthermore, treating $\boldsymbol{L}$ and $\boldsymbol{R}$ as vectors in $\mathbb{R}^n$, let $\boldsymbol{L} = ||\boldsymbol{L}||_2\hat{\boldsymbol{L}}, \boldsymbol{R} = ||\boldsymbol{R}||_2\hat{\boldsymbol{R}}$, where $\hat{\boldsymbol{L}}, \hat{\boldsymbol{R}}$ are unit vectors, and define $\tilde{s}(t) = ||\boldsymbol{L}||_2||\boldsymbol{R}||_2\boldsymbol{s}(t) \in \mathbb{R}$. Additionally, let $\boldsymbol{u}(t) \in \mathbb{R}^P$ denote the input signal over time. Then, for $t > 0$, the dynamics of the output-generating network read:

$$\begin{aligned}\tau_x \frac{d\boldsymbol{x}}{dt} &= -\boldsymbol{x} + \frac{1}{N}\boldsymbol{L}\boldsymbol{s}\boldsymbol{R}^T\varphi(\boldsymbol{x}) + \boldsymbol{B}_x\boldsymbol{u} \\ &= -\boldsymbol{x} + \tilde{s}\hat{\boldsymbol{L}}\hat{\boldsymbol{R}}^T\varphi(\boldsymbol{x}) + \boldsymbol{B}_x\boldsymbol{u}.\end{aligned}$$

We now define the rank-1 dynamics variable $\boldsymbol{w} := \hat{\boldsymbol{L}}^T\boldsymbol{x}$ and a residual mode $\boldsymbol{w}^\perp := (I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T)\boldsymbol{x}$, so that $\boldsymbol{x} = \boldsymbol{w}\hat{\boldsymbol{L}} + \boldsymbol{w}^\perp$ is a combination of a rank-1 mode and a residual component. This gives us the dynamics equations

$$\tau_x \frac{d\boldsymbol{w}}{dt} = \hat{\boldsymbol{L}}^T\left(-\boldsymbol{x} + \tilde{s}\hat{\boldsymbol{L}}\hat{\boldsymbol{R}}^T\varphi(\boldsymbol{x}) + \boldsymbol{B}_x\boldsymbol{u}\right) = -\boldsymbol{w} + \tilde{s}\hat{\boldsymbol{R}}^T\varphi\left(\boldsymbol{w}\hat{\boldsymbol{L}} + \boldsymbol{w}^\perp\right) + \hat{\boldsymbol{L}}^T\boldsymbol{B}_x\boldsymbol{u}$$

$$\text{and } \tau_x \frac{d\boldsymbol{w}^\perp}{dt} = \left(I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T\right)\left(-\boldsymbol{x} + \tilde{s}\hat{\boldsymbol{L}}\hat{\boldsymbol{R}}^T\varphi(\boldsymbol{x}) + \boldsymbol{B}_x\boldsymbol{u}\right) = -\boldsymbol{w}^\perp + \left(I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T\right)\boldsymbol{B}_x\boldsymbol{u}.$$

Using the basic theory of ordinary differential equations, we may solve the latter ODE to obtain

$$\boldsymbol{w}^\perp(t) = e^{-t/\tau_x}\left(\boldsymbol{w}^\perp(0) + \frac{1}{\tau_x}\int_0^t e^{s/\tau_x}\left(I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T\right)\boldsymbol{B}_x\boldsymbol{u}(s)ds\right) \tag{19}$$

As a special case, in the absence of any inputs, we would have $\boldsymbol{w}^\perp(t) = \boldsymbol{w}^\perp(0)e^{-t/\tau_x}$. For ease of notation, we define the function $J : \mathbb{R} \to \mathbb{R}^N$ by

$$J(t) = \frac{1}{\tau_x}e^{-t/\tau_x}\int_0^t e^{s/\tau_x}\left(I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T\right)\boldsymbol{B}_x\boldsymbol{u}(s)ds \tag{20}$$

Now, recall that our desired output signal is some prespecified function $f(t)$. Letting the linear readout (of the output-generating network) be given as $y = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d} = (\boldsymbol{c}^T\hat{\boldsymbol{L}})\boldsymbol{w} + \boldsymbol{c}^T\boldsymbol{w}^\perp + \boldsymbol{d} = (\boldsymbol{c}^T\hat{\boldsymbol{L}})\boldsymbol{w} + (\boldsymbol{c}^T\boldsymbol{w}^\perp(0))e^{-t/\tau_x} + \boldsymbol{c}^TJ + \boldsymbol{d}$, then solving for $\boldsymbol{w}$ and differentiating yields the equations

$$\boldsymbol{w}(t) = \frac{f(t) - \boldsymbol{c}^T\boldsymbol{w}^\perp(0)e^{-t/\tau_x} - \boldsymbol{c}^TJ(t) - \boldsymbol{d}}{\boldsymbol{c}^T\hat{\boldsymbol{L}}} \tag{21}$$

$$\tau_x\frac{d\boldsymbol{w}}{dt} = \frac{\tau_x f'(t) + \boldsymbol{c}^T\boldsymbol{w}^\perp(0)e^{-t/\tau_x} - \tau_x(\boldsymbol{c}^T\nabla J)}{\boldsymbol{c}^T\hat{\boldsymbol{L}}} \tag{22}$$

Now, observe that

$$\boldsymbol{s}(t) = \frac{\tilde{s}(t)}{||\boldsymbol{L}||_2 \cdot ||\boldsymbol{R}||_2} = \frac{\boldsymbol{w}(t) + \tau_x\frac{d\boldsymbol{w}}{dt} - \hat{\boldsymbol{L}}^T\boldsymbol{B}_x\boldsymbol{u}(t)}{||\boldsymbol{L}||_2 \cdot ||\boldsymbol{R}||_2 \cdot \hat{\boldsymbol{R}}^T\varphi\left(\boldsymbol{w}(t)\hat{\boldsymbol{L}} + \boldsymbol{w}^\perp(t)\right)},$$

meaning that

$$\boldsymbol{s} = \frac{\boldsymbol{w} + \tau_x\frac{d\boldsymbol{w}}{dt} - \hat{\boldsymbol{L}}^T\boldsymbol{B}_x\boldsymbol{u}}{||\boldsymbol{L}||_2 \cdot \boldsymbol{R}^T\varphi\left(\boldsymbol{w}\hat{\boldsymbol{L}} + \boldsymbol{w}^\perp\right)} \tag{23}$$

Substituting in our earlier expressions for $\boldsymbol{w}(t)$ and $\tau_x \frac{d\boldsymbol{w}}{dt}$ into the formula for $\boldsymbol{s}(t)$, we obtain the formula

$$\boldsymbol{s}(t) = \frac{f(t) + \tau_x f'(t) - \tau_x(\boldsymbol{c}^T \nabla J(t)) - \boldsymbol{d} - \hat{\boldsymbol{L}}^T \boldsymbol{B}_x \boldsymbol{u}(t)}{\boldsymbol{c}^T \boldsymbol{L} \cdot \boldsymbol{R}^T \varphi\left(\frac{f(t) - \boldsymbol{c}^T \boldsymbol{w}^\perp(0) e^{-t/\tau_x} - \boldsymbol{c}^T J(t) - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L}} \boldsymbol{L} + \boldsymbol{w}^\perp(0) e^{-t/\tau_x} + J(t)\right)}.$$

Moreover, in the absence of any inputs to the system, the neuromodulatory signal would be

$$\boldsymbol{s}(t) = \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L} \cdot \boldsymbol{R}^T \varphi\left(\frac{f(t) - \boldsymbol{c}^T \boldsymbol{w}^\perp(0) e^{-t/\tau_x} - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L}} \boldsymbol{L} + \boldsymbol{w}^\perp(0) e^{-t/\tau_x}\right)} \tag{24}$$

Furthermore, if we also know that that τ_x is sufficiently small and that $\boldsymbol{w}^\perp(0)$ has sufficiently small entries, then (owing to the exponential decay of $\boldsymbol{w}^\perp$) we may further approximate $\boldsymbol{s}$ as

$$\boldsymbol{s}(t) \approx \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L} \cdot \boldsymbol{R}^T \varphi\left(\frac{f(t) - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L}} \boldsymbol{L}\right)}.$$

Thus, for NM-RNNs in which the output-generating network's nonlinearity is removed (i.e., where $\varphi(\boldsymbol{x}) := \boldsymbol{x}$), eq. (24) implies that, in the absence of inputs, the neuromodulatory signal is

$$\boldsymbol{s}(t) = \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{(\boldsymbol{R}^T \boldsymbol{L})\left(f(t) - \boldsymbol{c}^T \boldsymbol{w}^\perp(0) e^{-t/\tau_x} - \boldsymbol{d}\right) + (\boldsymbol{c}^T \boldsymbol{L})\left(\boldsymbol{R}^T \boldsymbol{w}^\perp(0) e^{-t/\tau_x}\right)}.$$

If we assume further that $\boldsymbol{w}^\perp(0)$ is sufficiently small and τ_x is also sufficiently small, then we may make the approximation

$$\boldsymbol{s}(t) \approx \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{\boldsymbol{R}^T \boldsymbol{L}(f(t) - \boldsymbol{d})} = \frac{1}{\boldsymbol{R}^T \boldsymbol{L}}\left(1 + \frac{\tau_x f'(t)}{f(t) - \boldsymbol{d}}\right) \tag{25}$$

In particular, for $f(t)$ a linear ramp (such as during the ramping phase of the MWG task), eq. (25) implies that $\boldsymbol{s}(t)$ should follow a power law.

C Computing details

C.1 Code, data, and instructions

The code required to reproduce our main results is included in the `nm-rnn` GitHub repository: https://github.com/lindermanlab/nm-rnn. All code was written using Python, using fast compilation and optimization code from the JAX and Optax packages [40, 41]. Experiments and models were logged using the Weights and Biases ecosystem [42].

Instructions. To generate the results that we have presented, we have included the package folder "nmrnn" and folder of training scripts "scripts". The packages needed to replicate our coding environment are in "requirements.txt".

Each training script has a "config" dictionary near the top of the file that allows you to set hyperparameters, it is specifically set up to work with Weights and Biases. In the Jax framework, different initializations are set by changing the "keyind" value in the config dictionary.

C.2 Data generation

All data was generated synthetically.

Rank-1 Measure-Wait-Go. Rank-1 linearized NM-RNNs were trained on 40 trials, generated using the target intervals $[12, 14, 16, 18]$ and by setting the (integer-valued) timestep at which the measure cue appeared to be between 10 and 19, inclusive. The delay period was fixed to be 15 (timesteps). For any given target interval T, the target output ramp was the ramping function $f_T(t)$ given by

$$f_T(t) = \begin{cases} -\frac{1}{2} & t \leq t_{go} \\ \frac{1}{T}(t - t_{go}) - \frac{1}{2} & t_{go} \leq t \leq t_{go} + T \\ \frac{1}{2} & t \geq t_{go} + T \end{cases}.$$

The total number of timesteps was set to be 110. In generating Figure 3B of the main text, a trained network was tested on the intervals 7 (extrapolation below), 15 (interpolation of trained intervals), and 23 (extrapolation above). The theoretically expected neuromodulatory signal $\boldsymbol{s}(t)$ and rank-1 state $h(t)$ were computed using eqs. (8) and (21), respectively.

Rank-3 Measure-Wait-Go. All networks were trained on 40 trials, generated using desired intervals $[12, 14, 16, 18]$ and by setting the integer-valued delay period (interval between wait and go cues) between 10 and 19. Networks were tested on these trials plus corresponding extrapolated trials with interval lengths $[4, 6, 8, 10, 20, 22, 24, 26]$.

Rank-3 Multitask. Initial training was completed on 3000 randomly sampled trials from [DelayPro, DelayAnti, MemoryPro], with random angles and task period lengths. Retraining was completed on 1000 trials of MemoryAnti with random angles and task period lengths. Testing was completed on a different, fixed set of 1000 samples of each task.

Element Finder Task. All networks were trained on one-dimensional input sequences of length 26. For each input sequence used during training, the input at the first timestep was the query index q, which was uniformly sampled at random from $\{0, 1, \ldots, 24\}$. Each input in the proceeding sequence of 25 inputs was uniformly sampled at random from $\{-10, -9, \ldots, 9, 10\}$.

C.3 Training details

Training was performed via Adam with weight decay regularization and gradient clipping [43, 44]. Specific AdamW hyperparameters varied by task, see below for exact details.

Rank-1 Measure-Wait-Go. Training was performed via full-batch gradient descent for 50k iterations using the standard Adam optimizer with learning rate 10^{-3}. The linearized NM-RNN trained had the hyperparameters $N = 100$, $M = 20, K = 1, \tau_x = 2, \tau_z = 10$. Training took about 20 minutes. A single network was trained to produce the results shown in Figure 3B, and many more networks were trained while producing preliminary results.

Rank-3 Measure-Wait-Go. Training was performed with initial learning rate 10^{-2}. For NM-RNNs, we first trained the neuromodulatory subnetwork parameters and output-generating subnetwork parameters separately for 10k iterations each, followed by training all parameters for 50k iterations. We set specific hyperparameters as follows:

- **NM-RNN:** $N = 100$, $M = 5$, $K = 3$, $\tau_x = 10$, $\tau_z = 100$.
- **LR-RNN:** As above, except $N = 106$ for parameter matching.

Models were trained using at least 32 parallel CPUs (1G memory each) on a compute cluster. Training the NM-RNNs took about 15 minutes each, and training the LR-RNNs took about 9 minutes each. Ten of each model were used to produce the results shown in the paper; however, we trained many more while producing preliminary results.

Rank-3 Multitask. Training on first three tasks was performed with initial learning rate 10^{-3}, for 150k iterations. Retraining on the MemoryAnti task was performed with initial learning rate 10^{-2} for 50k iterations. We set specific hyperparameters as follows:

- **NM-RNN:** $N = 100$, $M = 20$, $K = 3$, $\tau_x = 10$, $\tau_z = 100$.
- **LR-RNN:** As above. In this case, since the LR-RNN receives contextual inputs as well as sensory/fixation inputs (compared to the NM-RNN's output generation subnetwork which only receives sensory/fixation inputs), the LR-RNN has more parameters.

Models were trained using at least 32 parallel CPUs (1G memory each) on a compute cluster. Training the NM-RNNs took about 2 hours each, and training the LR-RNNs took about 2.5 hours each. These times include both training and retraining. Ten of each model were used to produce the results shown in the paper; however, we trained many more while producing preliminary results.

Element Finder Task. For all networks, training was done over 20k iterations of gradient descent using a batch size of 128. Each batch consisted of newly randomly generated input sequences. The standard Adam optimizer was used, and the learning rate and hyperparameters were varied across the different models tested:

- **LSTM:** We trained LSTMs of hidden size $N = 10$ using a learning rate of 10^{-2}.
- **NM-RNN:** We trained multiple NM-RNNs across the hyperparameter combinations $(M, N, K) = \{(5, 18, 8), (5, 13, 12), (10, 14, 5), (10, 12, 7), (15, 6, 5)\}$, fixing $\tau_x = 10$ and $\tau_z = 2$, and using a learning rate of 10^{-2}.
- **LR-RNN:** We trained multiple LR-RNNs across the hyperparameter combinations $(N, K) = \{(23, 10), (31, 7), (50, 4), (83, 2)\}$, fixing $\tau_x = 10$, and using a learning rate of 10^{-2}.
- **(Full-rank) RNN:** We trained multiple RNNs with hidden size $N = 16$, fixing $\tau_x = 10$, and across the learning rates $\{10^{-3}, 10^{-2}, 10^{-1}\}$.

All trained models were parameter-matched (~ 500 total parameters). For each model type, hyperparameter combination, and learning rate, ten such models were trained; the resulting model performances are illustrated in Figure 5B of the main text. Figure 5C was illustrated using a single run of an LSTM ($N = 10$), a full-rank RNN ($N = 16$), and three NM-RNNs ($(M, N, K) = (5, 18, 8), (5, 13, 12), (10, 12, 7)$), where all models used a learning rate of 10^{-2}. Finally, Figure 5D-F show results for a single NM-RNN ($(M, N, K) = (5, 18, 8)$, $\text{lr} = 10^{-2}$). Training each network took roughly 5 minutes. Many more models were trained while producing preliminary results.

C.4 Metric for multitask setting

For the multitask setting, we used the percent correct metric from [1]. A trial was counted as correct if (1) the fixation output stayed above 0.5 until the fixation input switched off, (2) the angle read out at the final timestep was within $\pi/10$ of the desired angle.

Supplemental figures

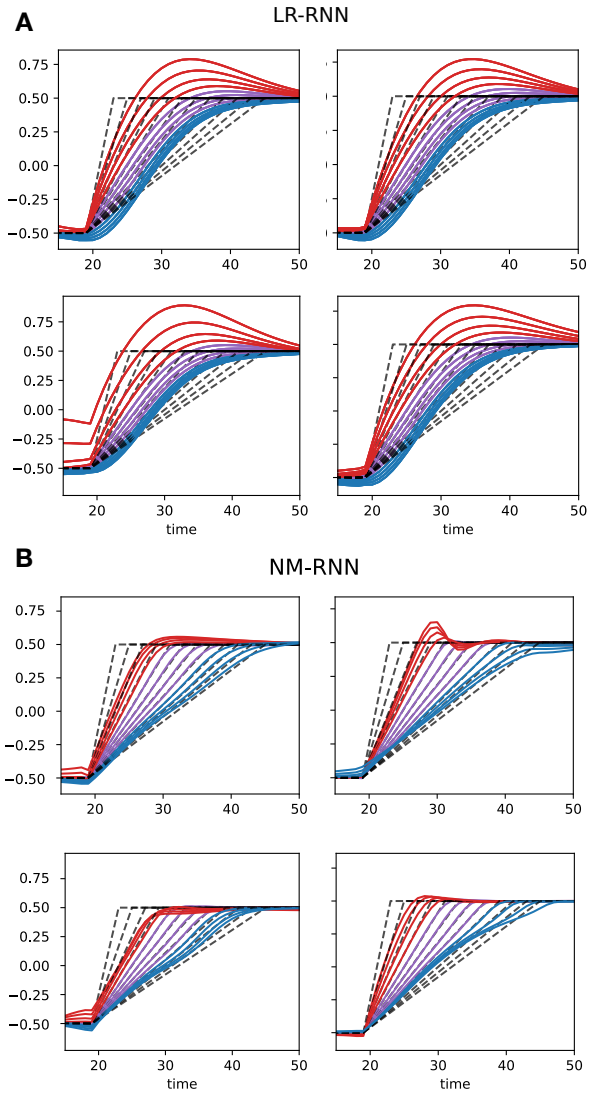

Figure S1: Example output comparison plots for the measure-wait-go task, as in fig. 3D in the main text, for four additional trained (**A**) low-rank RNNs and (**B**) NM-RNNs. Colors indicate extrapolated/trained intervals as in the main text.

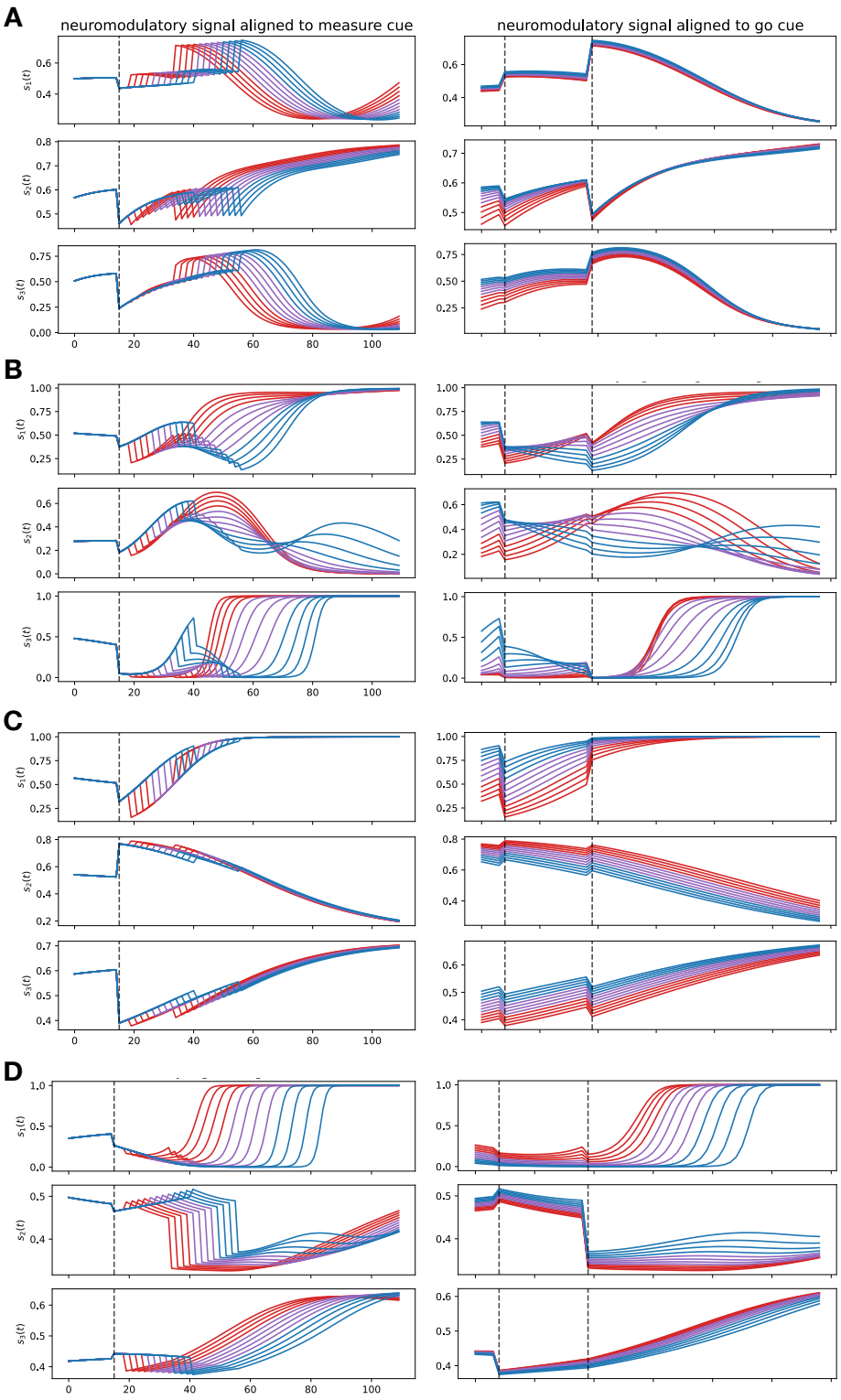

Figure S2: Example neuromodulatory signal plots for the measure-wait-go task, as in fig. 3E in the main text, for four additional trained NM-RNNs (same networks as shown in fig. S1B). Colors indicate extrapolated/trained intervals as in the main text.

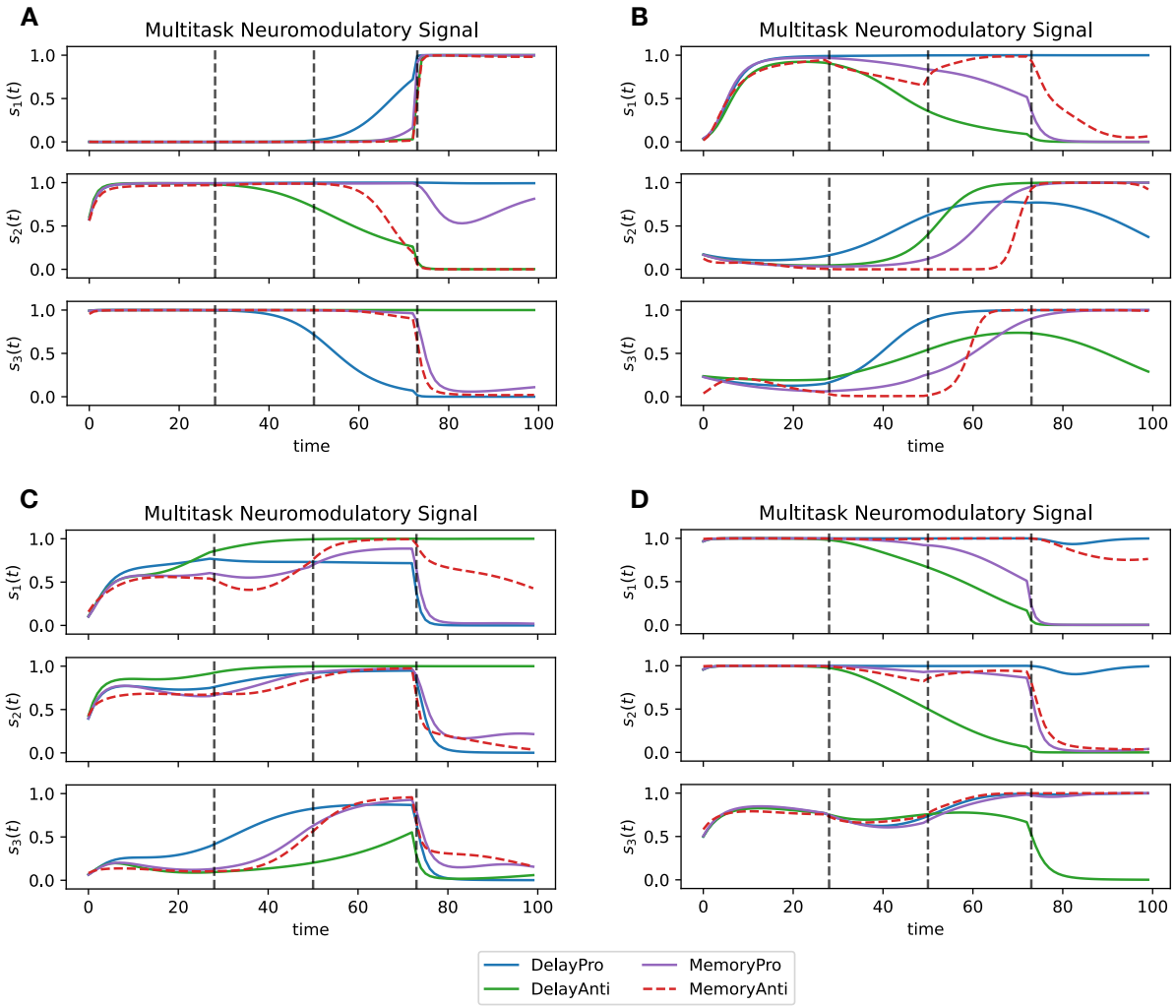

Figure S3: Example neuromodulatory signal plots for the multitask setting, as in fig. 4C in the main text, for four additional trained networks.

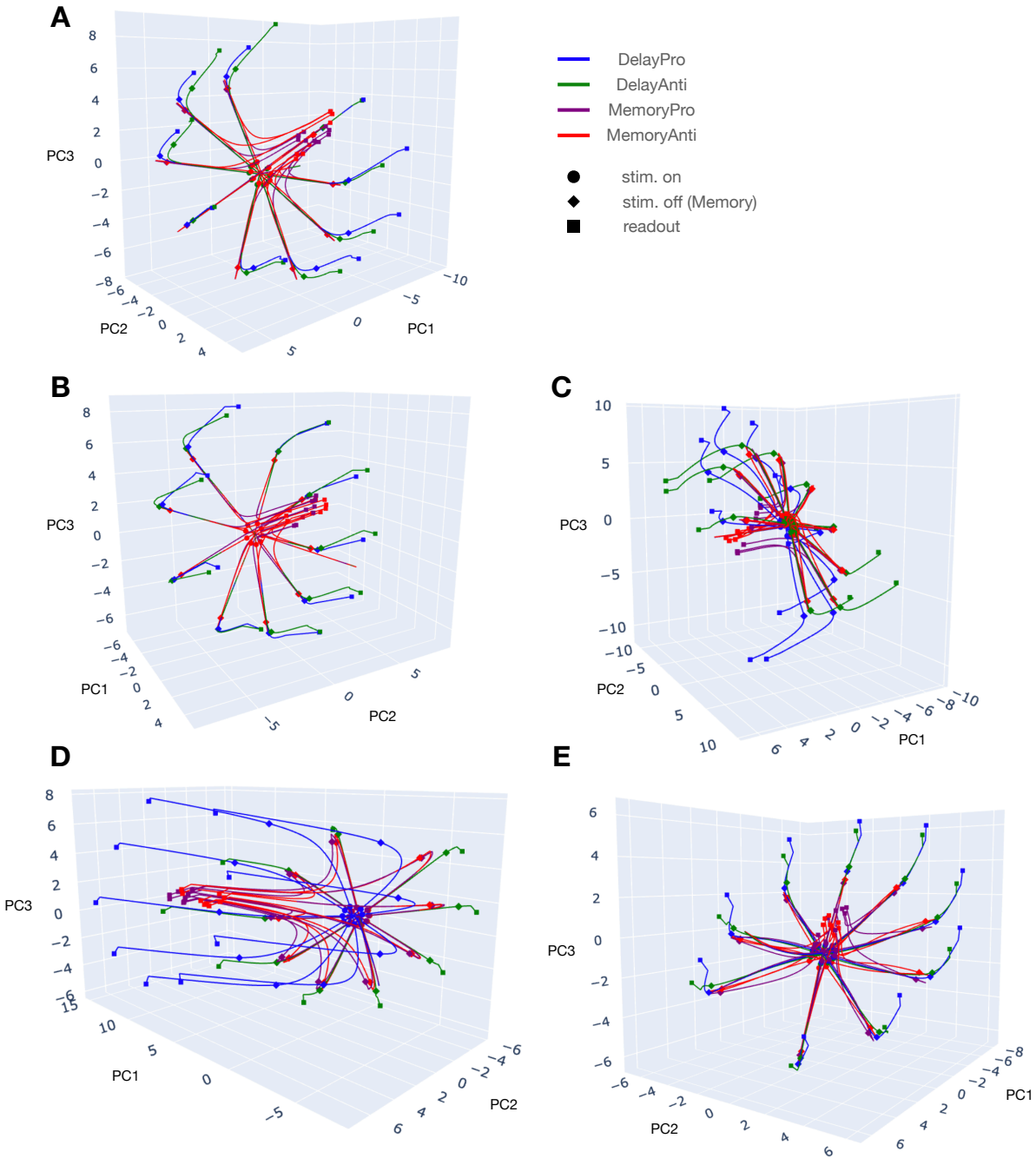

Figure S4: First three PCs of neural activity in multitask setting, plotted until readout period (for ease of visualization). **A.** Network visualized in main text, **B-E.** Four additional networks.

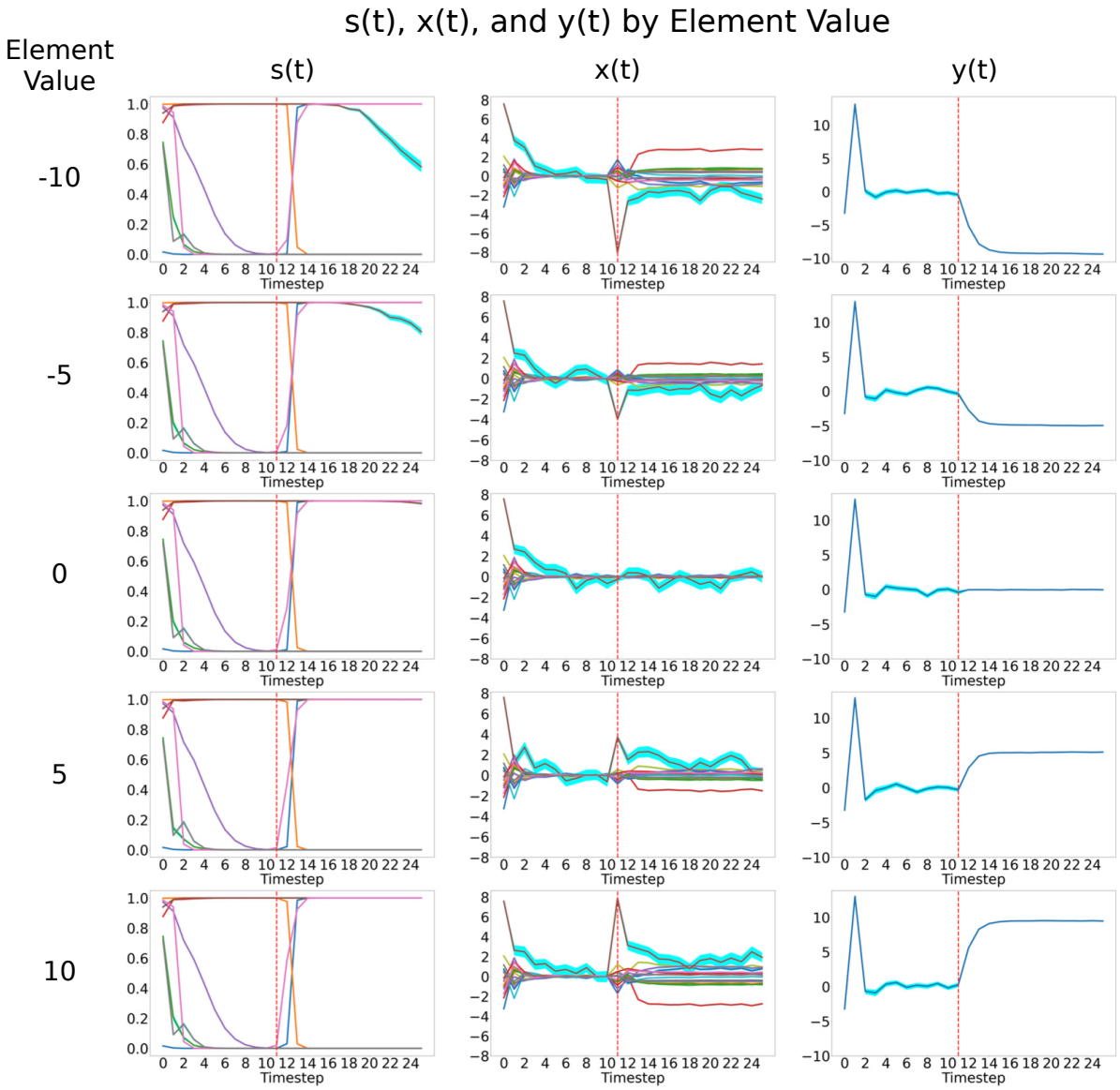

Figure S5: Sample internal states of an NM-RNN ($M = 5$, $N = 18$, $R = 8$) trained on the Element Finder Task, shown for 5 different element values (-10, -5, 0, 5, and 10). Each plot shows how all of the components of one of the vectors $s(t)$ (left), $x(t)$ (middle), $y(t)$ (right) vary through time. The query index is fixed to be 10, as indicated by the red dashed line in each plot. Each line shown is averaged over 100 independent runs of the model (standard error shown in cyan).

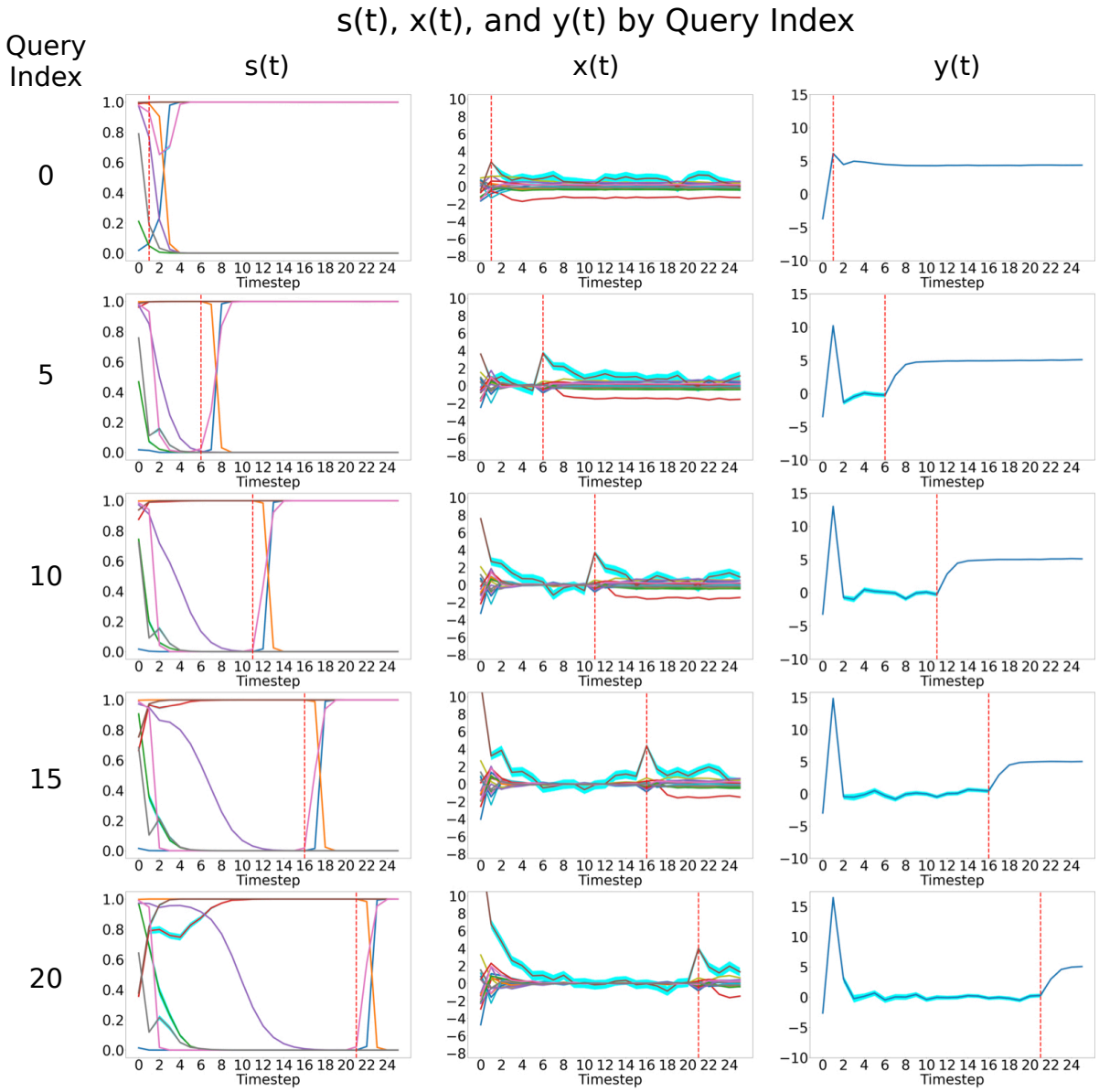

Figure S6: Sample internal states of an NM-RNN ($M = 5$, $N = 18$, $R = 8$) trained on the Element Finder Task, shown for 5 different query indices (0, 5, 10, 15, and 20), while fixing the target element value to be 5 in each case. Each plot shows how all of the components of one of the vectors $s(t)$ (left), $x(t)$ (middle), $y(t)$ (right) vary through time. In each plot, the onset of the query index is indicated by the red dashed line. Each line shown is averaged over 100 independent runs of the model (standard error shown in cyan).