# Supplementary Material: Structured flexibility in recurrent neural networks via neuromodulation

**Julia C. Costacurta***
Stanford University
jcostac@stanford.edu

**Shaunak Bhandarkar***
Stanford University
shaunakb@stanford.edu

**David Zoltowski**
Stanford University
dzoltow@stanford.edu

**Scott W. Linderman**
Stanford University
scott.linderman@stanford.edu

## Contents

## 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:

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

$$\boldsymbol{z}_t = \left((1-\frac{1}{\tau_z})I + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\left(\boldsymbol{B}_{zx}\boldsymbol{x}_{t-1} + \boldsymbol{B}_z\boldsymbol{u}_t\right) \tag{2}$$

$$\boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{d} \tag{3}$$

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 $S_t = \text{diag}(\boldsymbol{s}_t)$ 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 Equation 1. However, this does not fundamentally change the computation in Equation 1 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:

$$f_t = \sigma(W_f[h_{t-1}, u_t'] + b_f) \qquad i_t = \sigma(W_i[h_{t-1}, u_t'] + b_i) \tag{4}$$

$$\tilde{c}_t = W_c[h_{t-1}, u_t'] + b_c \qquad c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{5}$$

$$o_t = W_o[h_{t-1}, u_t'] + b_o \qquad h_t = o_t \odot c_t \tag{6}$$

Here, $\sigma$ 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 $i$th 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 $|(\boldsymbol{w}_t)_i| = \frac{1}{N}\left|\boldsymbol{R}_i^T\boldsymbol{x}_t\right| \leq \frac{1}{N}||\boldsymbol{R}_i||_2 \cdot ||\boldsymbol{x}_t||_2 = \frac{1}{\sqrt{N}} \cdot ||\boldsymbol{x}_t||_2 < H$. 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., Equations 1-3) 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

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

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

$$c_t = \begin{bmatrix}\boldsymbol{w}_{t-1}\\ \mathbf{1}_M \\ \mathbf{1}_P\end{bmatrix} \in \mathbb{R}^{K+M+P} \tag{7}$$

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 $\text{Proj}_K$ 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 $\boldsymbol{z}_t \in \mathbb{R}^M$:

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

Note further that

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

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

$$\begin{aligned}\boldsymbol{z}_t = \frac{1}{\tau_z}\boldsymbol{B}_{zx}\boldsymbol{R}\boldsymbol{w}_{t-1} &+ \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1}\\ &+ \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1}\end{aligned} \tag{8}$$

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 $o_t = W_o\left[h_{t-1}, \boldsymbol{u}_t, \boldsymbol{u}_{t-1}\right]$. Then, computing $h_t = o_t \odot c_t \in \mathbb{R}^{K+M+P}$ gives

$$h_t = \begin{bmatrix} \frac{1}{2}\boldsymbol{w}_{t-1} \\ \left((1-\frac{1}{\tau_z})I_{K\times K} + \frac{1}{\tau_z}\boldsymbol{W}_z\right)\boldsymbol{z}_{t-1} + \frac{1}{\tau_z}\boldsymbol{B}_z\boldsymbol{u}_t + \frac{1}{\tau_z}\boldsymbol{B}_{zx}\left(I_{N\times N} - \frac{1}{N}\boldsymbol{R}\boldsymbol{R}^T\right)\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \boldsymbol{u}_{t-1} \end{bmatrix} \tag{9}$$

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 Equation 8. 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 $f_t = \begin{bmatrix} \boldsymbol{s}_{t-1} \\ \mathbf{1}_{M+P} \end{bmatrix}$. 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 $\boldsymbol{s}_{t-1} = \sigma(\boldsymbol{A}_z\boldsymbol{z}_{t-1} + \boldsymbol{b}_z) = \sigma(\boldsymbol{A}_zW_{zh}h_{t-1} + \boldsymbol{b}_z)$. We can also ensure that the last $M+P$ entries of $f_t$ form the vector $\mathbf{1}_{M+P}$ 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

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

and $b_f = \begin{bmatrix} \boldsymbol{b}_z \\ Q\mathbf{1}_{M+P} \end{bmatrix}$ 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 $(\boldsymbol{u}_t)_{t\ge1}$ 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 (Equation 5) can be made to reproduce the low-rank update equation of the linearized NM-RNN's output-generating network:

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

$$\implies \begin{bmatrix} \boldsymbol{w}_{t-1} \\ \mathbf{1}_{M+P} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_{t-1} \\ \mathbf{1}_{M+P} \end{bmatrix} \odot \begin{bmatrix} \boldsymbol{w}_{t-2} \\ \mathbf{1}_{M+P} \end{bmatrix} + \frac{1}{2}\mathbf{1}_{K+M+P} \odot \begin{bmatrix} \frac{2}{N}\boldsymbol{R}^T\boldsymbol{B}_x\boldsymbol{u}_{t-1} \\ \mathbf{0}_{M+P} \end{bmatrix}$$
$$\implies c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t.$$

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 $t \geq 2$) 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 $\varphi$ (which could be tanh, but also any other function). Because $K = 1$, we have $\boldsymbol{L} \in \mathbb{K}^{n\times 1}, \boldsymbol{R} \in \mathbb{R}^{n\times 1}$. 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

$$\boldsymbol{w}^\perp(t) = e^{-t/\tau_x}\left(\boldsymbol{w}^\perp(0) + \frac{1}{\tau_x}\int_0^t e^{s/\tau_x}\left(I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T\right)\boldsymbol{B}_x\boldsymbol{u}(s)ds\right) \tag{10}$$

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

$$J(t) = \frac{1}{\tau_x}e^{-t/\tau_x}\int_0^t e^{s/\tau_x}\left(I - \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^T\right)\boldsymbol{B}_x\boldsymbol{u}(s)ds \tag{11}$$

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

$$\boldsymbol{w} = \frac{f(t) - \boldsymbol{c}^T\boldsymbol{w}^\perp(0)e^{-t/\tau_x} - \boldsymbol{c}^TJ - \boldsymbol{d}}{\boldsymbol{c}^T\hat{\boldsymbol{L}}} \tag{12}$$

$$\tau_x\frac{d\boldsymbol{w}}{dt} = \frac{\tau_x f'(t) + \boldsymbol{c}^T\boldsymbol{w}^\perp(0)e^{-t/\tau_x} - \tau_x(\boldsymbol{c}^T\nabla J)}{\boldsymbol{c}^T\hat{\boldsymbol{L}}} \tag{13}$$

Now, observe that

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

meaning that

$$\boldsymbol{s} = \frac{\boldsymbol{w} + \tau_x\frac{d\boldsymbol{w}}{dt} - \hat{\boldsymbol{L}}^T\boldsymbol{B}_x\boldsymbol{u}}{||\boldsymbol{L}||_2\cdot\boldsymbol{R}^T\varphi\left(\boldsymbol{w}\hat{\boldsymbol{L}} + \boldsymbol{w}^\perp\right)} \tag{14}$$

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

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

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

$$\boldsymbol{s} = \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L} \cdot \boldsymbol{R}^T \varphi\left(\frac{f(t) - \boldsymbol{c}^T \boldsymbol{w}^{\perp}(0) e^{-t/\tau_x} - \boldsymbol{d}}{\boldsymbol{c}^T \boldsymbol{L}} \boldsymbol{L} + \boldsymbol{w}^{\perp}(0) e^{-t/\tau_x}\right)} \tag{15}$$

Furthermore, if we also know that that $\tau_x$ 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 $\varphi(\boldsymbol{x}) := \boldsymbol{x}$), Equation 15 implies that, in the absence of inputs, the neuromodulatory signal is

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

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

$$\boldsymbol{s} \approx \frac{f(t) + \tau_x f'(t) - \boldsymbol{d}}{\boldsymbol{R}^T \boldsymbol{L}\,(f(t) - \boldsymbol{d})} = \frac{1}{\boldsymbol{R}^T \boldsymbol{L}} \left(1 + \frac{\tau_x f'(t)}{f(t) - \boldsymbol{d}}\right) \tag{16}$$

In particular, for $f(t)$ a linear ramp (such as during the ramping phase of the MWG task), Equation 16 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 a folder with the Supplementary Material. All code was written using Python, using fast compilation and optimization code from the Jax and Optax packages [1, 2]. Experiments and models were logged using the Weights and Biases ecosystem [3]. If published, all code will be posted on a public repository.

**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 Equations 16 and 12, 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 [4, 5]. Specific AdamW hyperparameters varied by task, see below for exact details. Training time varied based on model architecture and size, as we elaborate on below. However, we'd like to note that LR-RNNs and NM-RNNs with the same number of neurons as vanilla RNNs and LSTMs have comparatively fewer parameters due to their low-rank recurrence matrices, an appealing feature from a training standpoint (exact ratios depend on rank and neuromodulatory subnetwork size). In this work we made comparisons based on the number of parameters, not the number of neurons.

**Rank-1 Measure-Wait-Go.** Training was performed via full-batch gradient descent for 50k iterations using the standard Adam optimizer with learning rate $1e-3$. 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 $1e-2$. 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.
- **Vanilla RNN:** As above, except $N = 31$ for parameter matching.
- **LSTM:** As above, except $N = 15$ 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, training the LR-RNNs took about 9 minutes each, training the vanilla RNNs took about 2 minutes each, and training the LSTMs took about 4 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 $1e-3$, for 150k iterations. Retraining on the MemoryAnti task was performed with initial learning rate $1e-2$ 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.
- **Vanilla RNN:** As above, except $N = 18$ to match NM-RNN parameter count.
- **LSTM:** As above, except $N = 8$ to match NM-RNN parameter count.

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, training the LR-RNNs took about 2.5 hours each, training the vanilla RNNs took about 7 minutes each, and training the LSTMs took about 15 minutes 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 $1e-2$.
- **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 $1e-2$.
- **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 $1e-2$.
- **Vanilla RNN:** We trained multiple RNNs with hidden size $N = 16$, fixing $\tau_x = 10$, and across the learning rates $\{1e-3, 1e-2, 1e-1\}$.

All trained models were parameter-matched ($\sim$ 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 $1e-2$. Finally, Figure 5D-F show results for a single NM-RNN ($(M, N, K) = (5, 18, 8)$, lr = $1e-2$). 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 [6]. 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.

## D Societal impacts

As mentioned in the Paper Checklist, we don't anticipate any direct societal impacts from this work. In the long term, insights into neuromodulatory function provided by computational models have the potential to impact treatment of neurological diseases.

## E Supplemental figures

Included after References.

## References

[1] 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`.

[2] 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`.

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

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

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

[6] Laura N Driscoll, Krishna Shenoy, and David Sussillo. Flexible multitask computation in recurrent networks utilizes shared dynamical motifs. *Nature Neuroscience*, 27(7):1349–1363, 2024.

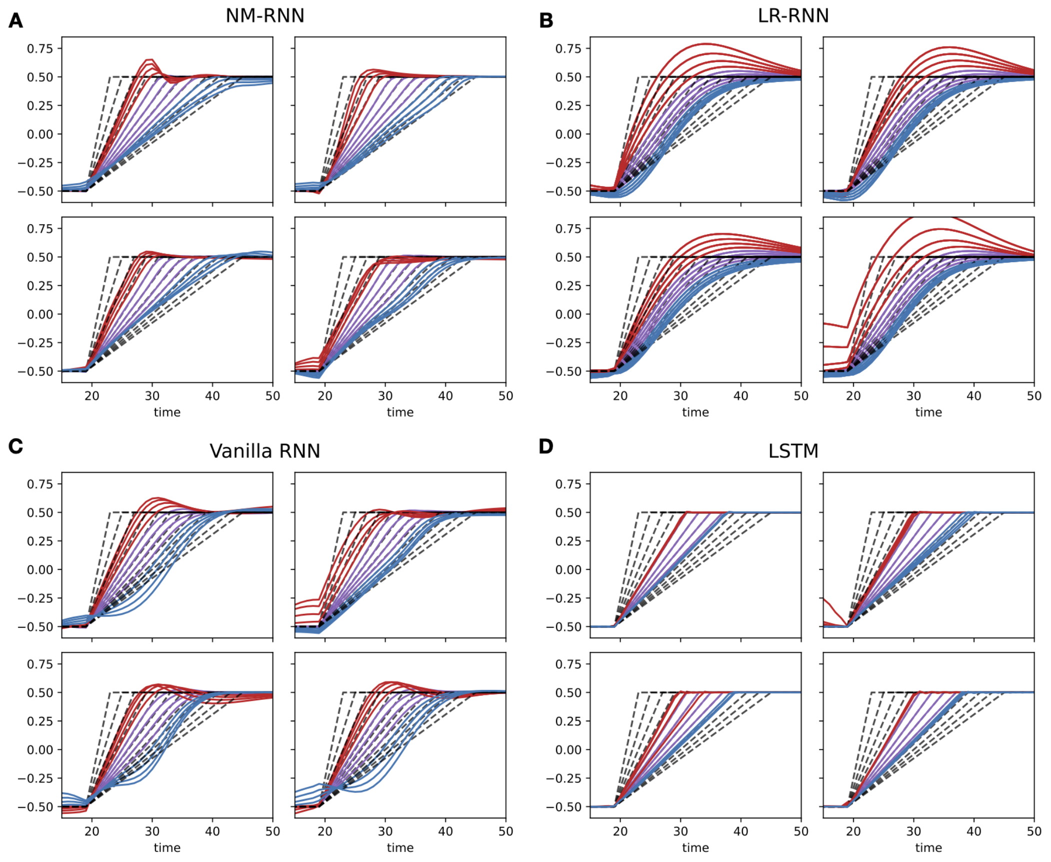

Figure 1: Example output comparison plots for the measure-wait-go task, as in Fig. 3 in the main text, for four trained parameter-matched NM-RNNs, LR-RNNs, vanilla RNNs, and LSTMs. Colors indicate extrapolated/trained intervals as in the main text. Model hyperparameters described in Supp. Section C.3.

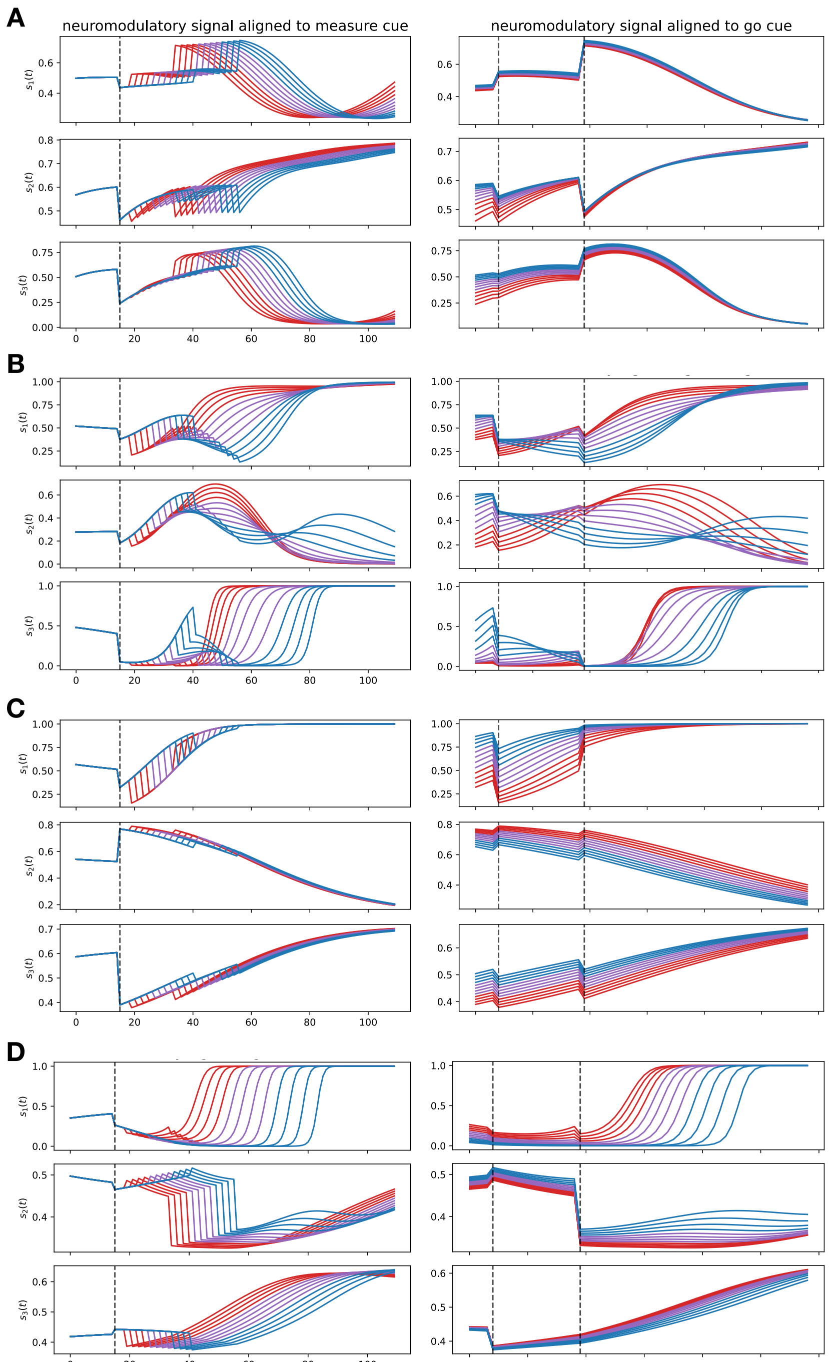

Figure 2: Example neuromodulatory signal plots for the measure-wait-go task, as in Fig. 3 in the main text, for four additional trained networks. Colors indicate extrapolated/trained intervals as in the main text.

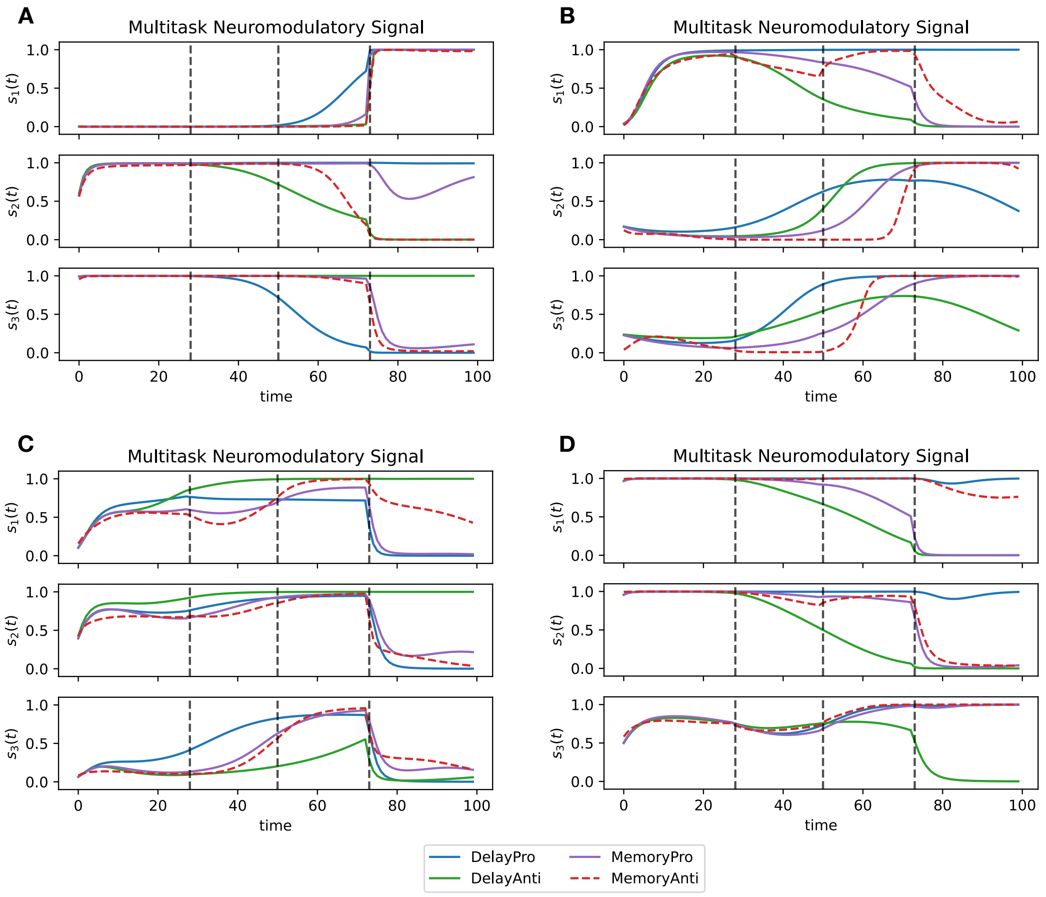

Figure 3: Example neuromodulatory signal plots for the multitask setting, as in Fig. 4 in the main text, for four additional trained networks.

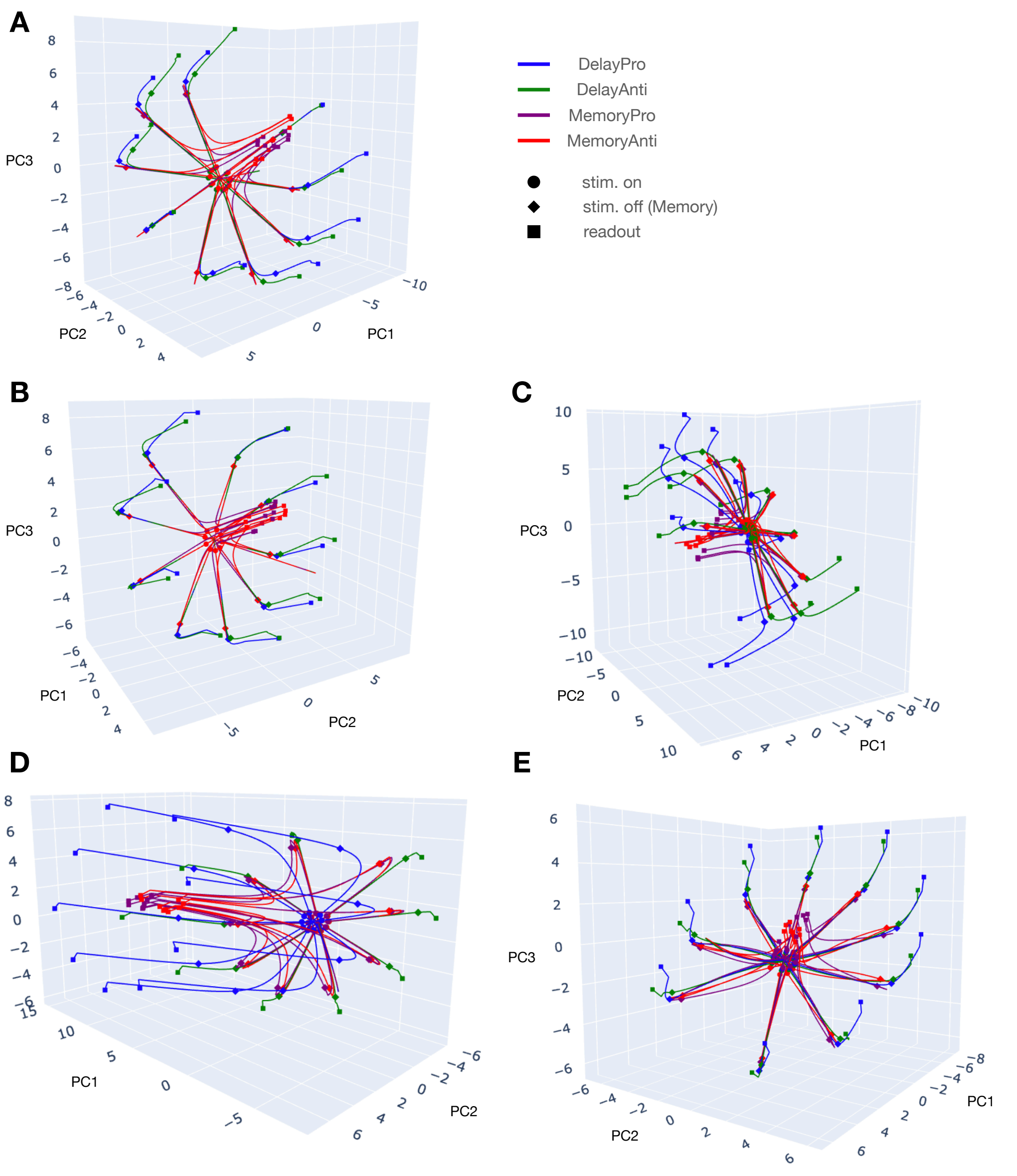

Figure 4: First three PCs of neural activity in multitask setting, plotted until readout period (for ease of visualization). **A**. Network visualized in main text, **B-E.** Four additional networks.

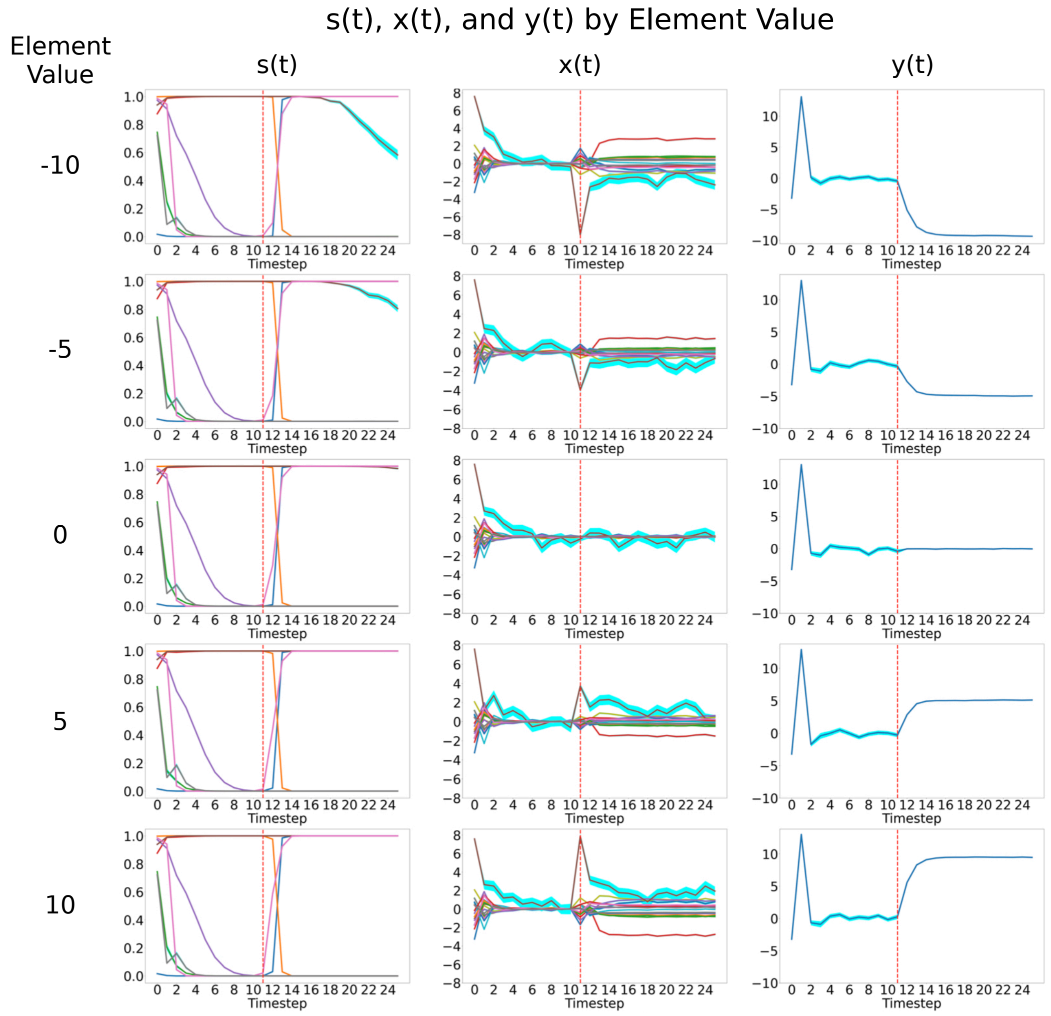

Figure 5: Sample internal states of an NM-RNN ($M = 5$, $N = 18$, $R = 8$) trained on the Element Finder Task, shown for 5 different element values ($-10$, $-5$, 0, 5, and 10). Each plot shows how all of the components of one of the vectors $s(t)$ (left), $x(t)$ (middle), $y(t)$ (right) vary through 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).

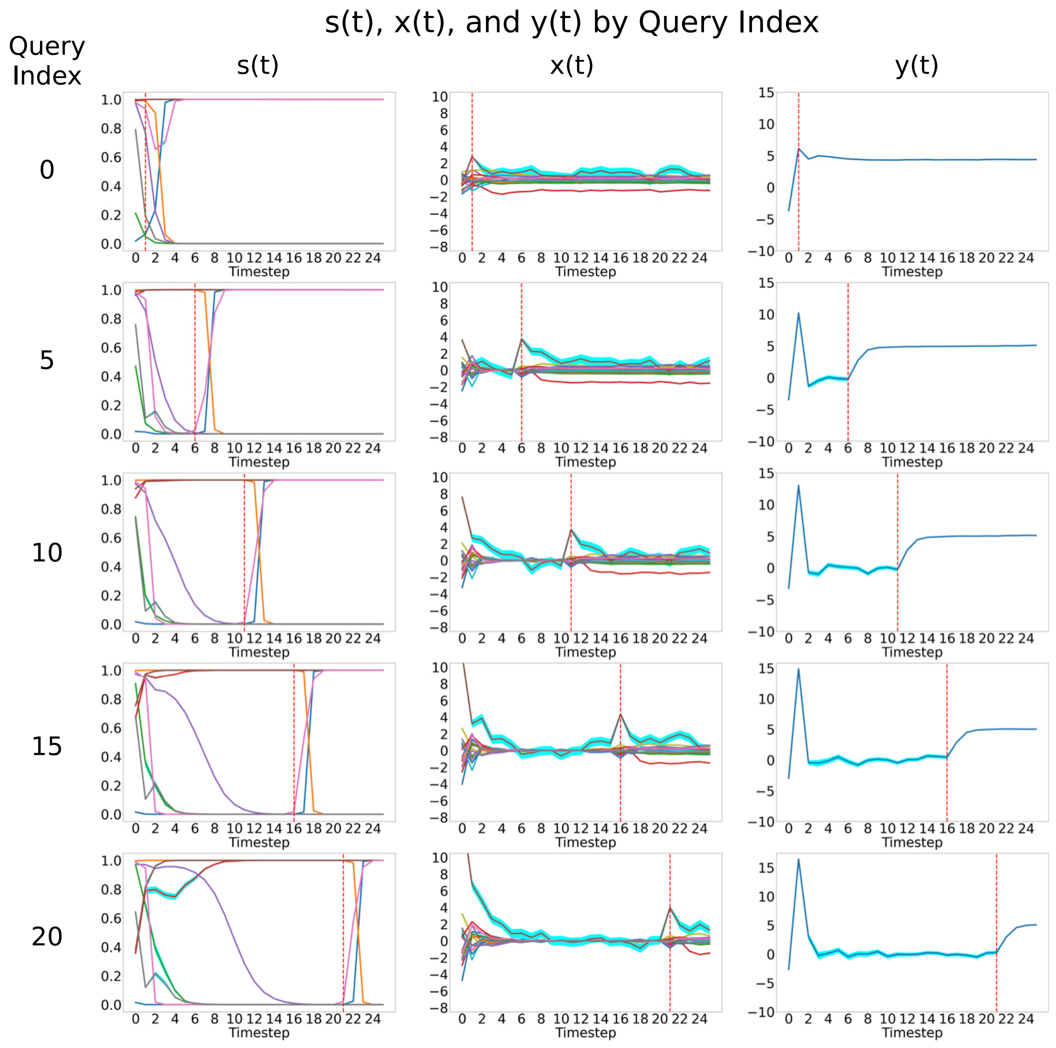

Figure 6: Sample internal states of an NM-RNN ($M = 5$, $N = 18$, $R = 8$) trained on the Element Finder Task, shown for 5 different query indices (0, 5, 10, 15, and 20), while fixing the target element value to be 5 in each case. Each plot shows how all of the components of one of the vectors $s(t)$ (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).