# A Choice of nonlinearity

If we consider a network with a rectified-linear instead of a $\tanh$ nonlinearity, the restrictions on the network's representational capacity in the absence of inputs are even more severe (Fig. 7). In this case, the basis functions are all scaled and axis-flipped *relu* functions that intersect the $x$ axis at $x = 0$. Thus they can only represent piecewise linear functions composed of two pieces with a knot at zero. Adding inputs (or per-neuron biases) allows the network to have universal approximation capabilities.

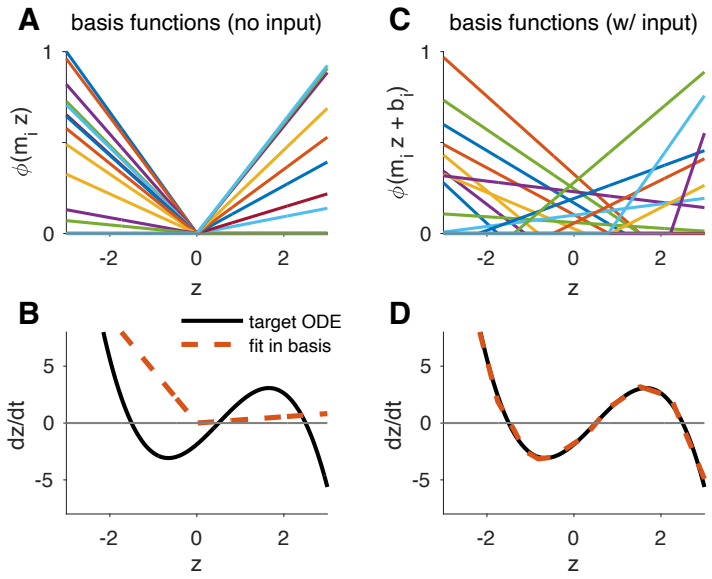


Figure 7: Representational capacity of a 1D low-rank RNN with rectified-linear (*relu*) nonlinearity. **(A)** Set of basis functions obtained by taking random coeffients $m_i \sim \mathcal{N}(0, 1)$ but without input ($\mathbf{v}_t = 0$). **(B)** Attempting to fit an example ODE using this basis recovers only a piecewise linear fit with a kink at zero. **(C)** By adding inputs, basis functions have random offset as well as slope. Here we set $\mathbf{v}_t = 1$ and sampled the input vector coefficients $\mathbf{b}_i \sim \mathcal{N}(0, 1)$. **(D)** Least squares fitting of $\mathbf{n}$ in the random basis from (C) provides a high-accuracy approximation to the target ODE.

## A.1 Comparison of activation functions in estimating ODEs

In this section, we explore the low-rank RNN's ability to approximate different types of dynamics (i.e function classes), with different activation functions (i.e basis functions). Our discussion above highlights how relu units can approximate functions through piecewise linear components. Non-zero inputs create basis functions which can be used to compose ODEs with "knots" at the shifted offsets. Alternatively, through our discussion in Section 3, we note tanh units provide smooth non-linear basis functions. The non-zero inputs create shifted basis functions, which perform a similar role, with smooth compositions. Following this intuition, if an ODE consists of smooth non-linear components it can be hypothesized that tanh units would have higher performance. Whereas, if the ODE consists of piecewise linear dynamics, relu units would prove to be more optimal. To validate this, we simulate two such ODEs in Fig. 8. Trivially, in the case of large enough number of basis functions, networks comprising of relu or tanh units can approximate any function (i.e they behave as universal approximators). However, to assess performance, we estimate the smallest networks in both cases that can fit the ODE within a pre-defined margin of error. As expected, the ODE with smoother non-linearities can be fit with smaller tanh networks than relu networks (the opposite is true for piecewise linear ODEs).

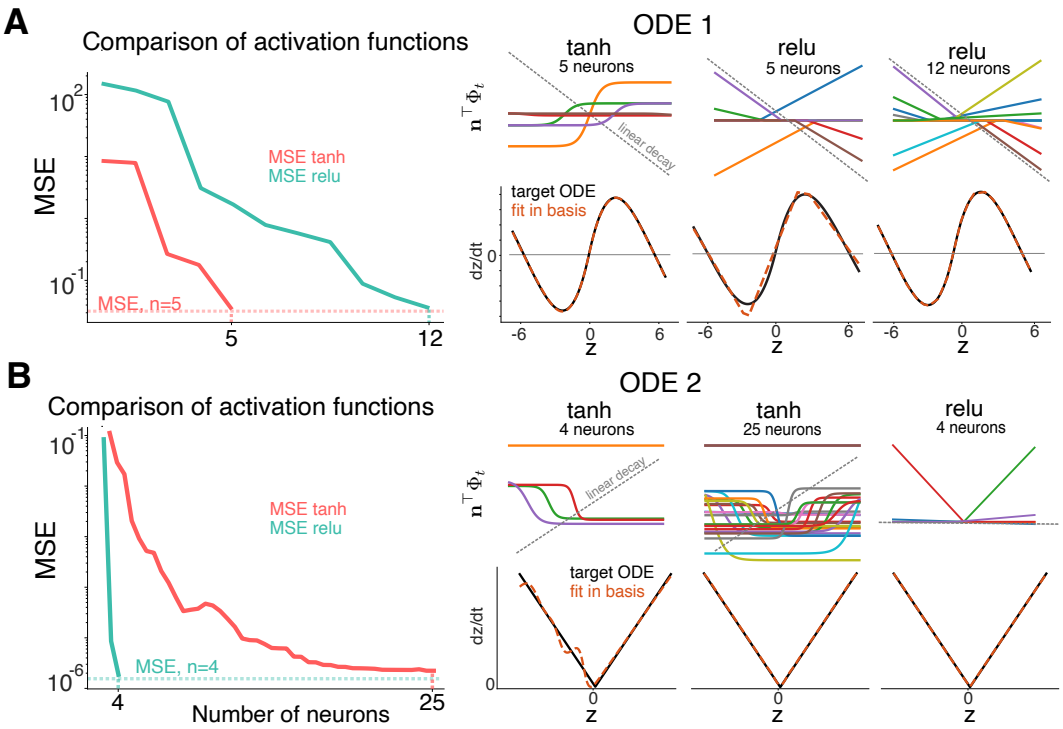


Figure 8: Performance comparison of tanh v/s relu in approximating different ODEs. **(A)** Depicts an ODE with two stable fixed points and one unstable fixed points (better fit with a tanh non-linearity). **(B)** Depicts an ODE with a shifted knot and two linear components (better fit with a RELU non-linearity). First column represents MSE (for a network with tanh and relu activations) as a function of the number of neurons in network. Neurons are added using OMP. The top row of the right column shows scaled basis functions selected via OMP. The bottom row shows fits (orange) of the target ODE (black) at the marked iterations of OMP.

### A.2 Absence of inputs for limit cycle

Section 3.1 depicts a limit cycle embedded into the low-rank RNN using our framework. The specific ODE of our non-linear and non-symmetric system is give as -

$$\frac{dx}{dt} = \left(\frac{1-\left(z_0^2+z_1^2\right)}{\sqrt{z_0^2+z_1^2+\epsilon}}\right) z_0 - z_1 - 0.35$$

$$\frac{dy}{dt} = \left(\frac{1-\left(z_0^2+z_1^2\right)}{\sqrt{z_0^2+z_1^2+\epsilon}}\right) z_1 + z_0 + 0.5$$

where $\epsilon$ is a small constant added for numerical stability. The constant values in each dimension make the underlying ODE non odd-symmetric.

In this section we show the inability of an RNN without inputs to appropriately approximate this function. In Fig. 9, the first column represents contour plots of the target ODE for each dimension. The overlayed vertical and horizontal dashed red lines depict $X = z_1 = 0, Y = z_2 = 0$ respectively. Note, there is a slight (left and upwards) shift in the contour plots, indicating the non-radial symmetry. This is introduced by adding a constant negative decay in $z_1$ and a positive correction in $z_2$. The second and third columns represents the fitted ODEs for an RNN with and without inputs respectively. It can be observed the RNN without inputs is unable to create offsets in any dimension, thus failing at recovering the underlying ODE. To further highlight this we simulate a sample trajectory from the polar coordinates of a limit cycle (detailed in Section 3.1) in the last row of Fig. 9. As expected, the low-rank RNN with inputs almost perfectly overlaps the trajectory, unlike the low-rank RNN without inputs.

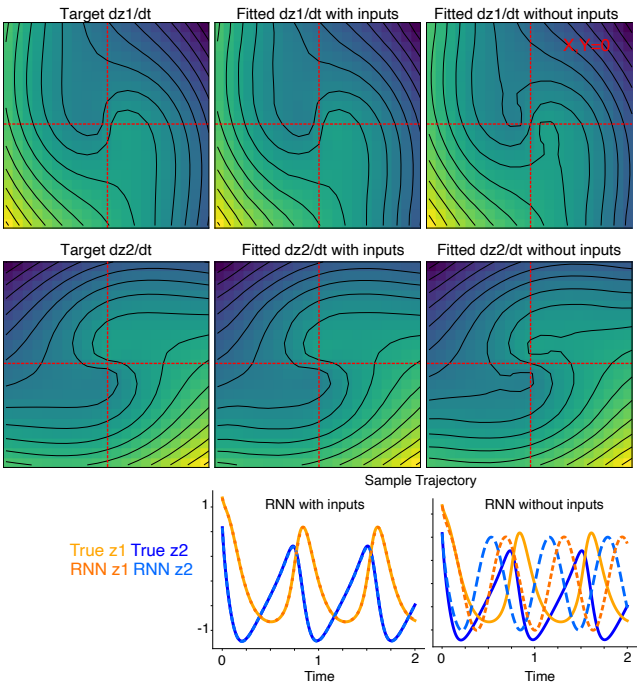


Figure 9: Influence of inputs in capturing non-symmetrical limit cycle

## B Online recursive least square (RLS) algorithm

Here we provide additional details on our recursive least-square algorithm. Specifically, for an observed target trajectory $\{\mathbf{z}_t\}_{t=1}^{T}$, with the initial state $\mathbf{z}_0$:

**Algorithm 1** Online RLS for Low-Rank RNNs

**Inputs:** target trajectory $\{\mathbf{z}_t\}_{t=0}^{T}$, step size $dt$.
**Initialize:** Basis parameters: $\mathbf{m}$, bias $\mathbf{b}$, weights $\mathbf{n} \leftarrow \mathbf{0}$, Precision $P \leftarrow \lambda^{-1}\mathbf{I}$
**for** $t = 1$ **to** $T$ **do**
  Basis vector $\phi_t \leftarrow \phi\big(\mathbf{m}\,\mathbf{z}_t + \mathbf{b}\big)$
  Target derivative $y_t \leftarrow \dfrac{\mathbf{z}_t - \mathbf{z}_{t-1}}{dt}$ (finite difference)
  Prediction $\hat{y}_t \leftarrow \mathbf{n}^\top \phi_t - \mathbf{z}_t$
  Error $e_t \leftarrow y_t - \hat{y}_t$
  Gain $k_t \leftarrow \dfrac{P\,\phi_t}{1 + \phi_t^\top P\,\phi_t}$
  Weight update $\mathbf{n} \leftarrow \mathbf{n} + k_t\,e_t$
  Covariance update $P \leftarrow P - k_t\,\phi_t^\top P$
**end for**

## C General formulation & application to binary decision making task

We apply our framework to a specific group of binary decision making tasks commonly observed in systems neuroscience. In this task, a rat accumulates evidence of auditory pulses over time from clicks on its left and right side. At the end of the stimulus period, the rat must turn to the side which produced more clicks, and is rewarded for inferring this correctly. It has been shown that multiple underlying dynamical portraits could represent this behavior [39]. We thus show applicability of our method by using it to recover the autonomous and input driven dynamics on four separate synthetically generated dynamic portraits linked to this task [39]. Here, intuitively, the input dynamics encode

for the accumulation of evidence based on the clicks, and a final decision to turn is made once the accumulation value reaches a specific attractor in the network. For instance, if the instrinsic dynamics encode a bi-stable attractor, each of the end points represent a specific decision, and the inputs move the dynamics along a line between them [38]. Additionally, consistent with previous studies, we model our simulations to provide equal weights to left and right clicks but with opposite magnitudes.

We model four flow fields representing intrinsic dynamics, namely a bi-stable attractor, a line attractor, a non-canonical line attractor and the flow field inferred from [39]. More formally, they are given as follows -

Bistable attractors:

$$\begin{aligned} dz_1 &= 10z_1(0.7+z_1)(0.7-z_1)dt + cudt \\ dz_2 &= -10z_2 dt \end{aligned}$$

Classic DDM - line attractor:

$$\begin{aligned} dz_1 &= \begin{cases} cudt & z_1 \in (-0.7, 0.7) \\ 10z_1(0.7-z_1)(0.7+z_1)dt & z_1 \notin (-0.7, 0.7) \end{cases} \\ dz_2 &= -30z_2 \end{aligned} \tag{17}$$

Non-canonical line attractor:

$$\begin{aligned} dz_1 &= 5z_2 \\ dz_2 &= -5z_2 dt + cudt \end{aligned}$$

Unsupervised model:

$$\begin{aligned} dz_1 &= 5z_1(0.85+z_1)(0.85-z_1)dt + cudt \\ dz_2 &= 5(0.5|z_1|+0.1)(z_1 - 1.2z_2) \end{aligned}$$

Here, $z_1, z_2$, represent the two latent dimensions, $u$ represents the magnitude of the input clicks, and $c$ represents if its positive or negative.
Critically, we observe the input dynamics lie in a dimension *parallel* to the recurrent activity. Or alternatively, drive the system in the dimensionality spanned by the recurrent activity. We thus present a general formulation of our equations to model these input dynamics. Following Eqn 5, for a scalar $\mathbf{z}$, we now not only observe orthogonal ($\mathbf{b} = \mathbf{b}_{perp}$) neuron specific inputs, but additional input dynamics that influence the recurrent activity ($\mathbf{b}_{par}$, spans the same direction as $\mathbf{m}$), thus updating Eqn 5 as :

$$\mathbf{x}(t) = \mathbf{m}\mathbf{z}(t) + \mathbf{b}_{par}\mathbf{v}_{par}(t) + \mathbf{b}_{perp}\mathbf{v}_{perp}(t), \tag{18}$$

where $\{\mathbf{v}_{par}(t), \mathbf{v}_{perp}(t)\}$ represent the low-pass filtered inputs which drive activity along and perpendicular to the recurrent dimensions respectively.

Our goal of embedding the ODE $g(\mathbf{z})$ into the network can now be viewed as setting the model parameters so that

$$g(\mathbf{z}) + \mathbf{z} \;\approx\; \mathbf{n}^\top \phi(\mathbf{m}\mathbf{z} + \mathbf{b}_{par}\mathbf{v}_{par}(t) + \mathbf{b}_{perp}v_{perp}(t)) \tag{19}$$

This allows us to follow a similar setup to our discussions in Sec. 3, with the exception that auditory inputs are applied along $\mathbf{b}_{par}$ or $\mathbf{b} = \mathbf{b}_{perp}$, or both.

As shown in Fig 10, each row represents one of the above dynamical regimes. The first column represents the dynamics along $z_1$, or $z_2$, and the RNN fitted version. Next, we model two right (or positive) clicks at $t = 0.5$ and $t = 1$ second and a single left (negative) click at $t = 2.5$ second. The second column represents the ODE when we start from ($z = 0$), pushed by these input dynamics, for our fitted RNN dynamics (Eqn 6) against the true ODE (computed using Euler method). Lastly, we also recover the underlying flow fields, as indicated by the last column. In Fig 11, we embed a non-canonical line-attractor in which input axis is perpendicular to the line attractor and non-normal dynamics give rise to movement along the line attractor. We successfully embedded all three of these systems with rank 1 RNNs. Lastly, we also embed a system with rotational dynamics between fixed points with integration along the diagonal between them. This is done through a rank 2 RNN with inputs along each of the directions spanned by $\mathbf{b}_{par}$(Fig. 11 B). This proves the flexibility of our framework in embedding dynamics associated with neuroscience tasks.

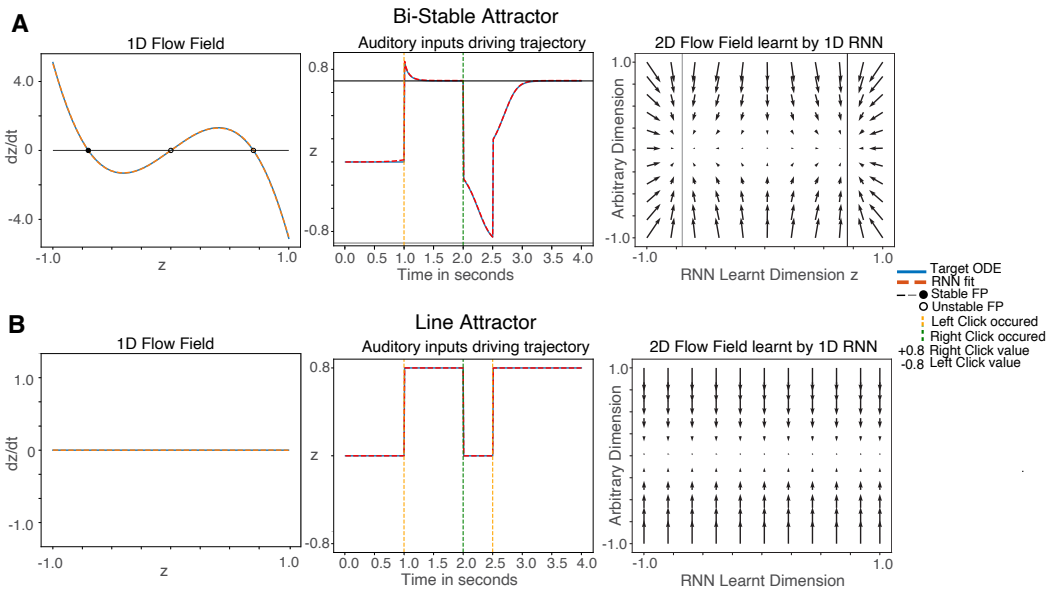


Figure 10: Two different dynamical portraits for binary decision making task: **(A)** bi-stable attractor ODE and **(B)** line attractor ODE. First column represents the true underlying ODE and the RNN estimate learned using least squares. Second column depicts a sample trajectory driven by momentary input clicks. A right click creates a drift towards the positive stable fixed point where as a left click, towards the negative stable fixed point for A. For B, accumulation along the line takes place with no diffusion. Third column represents the flow-field estimated by the RNN.

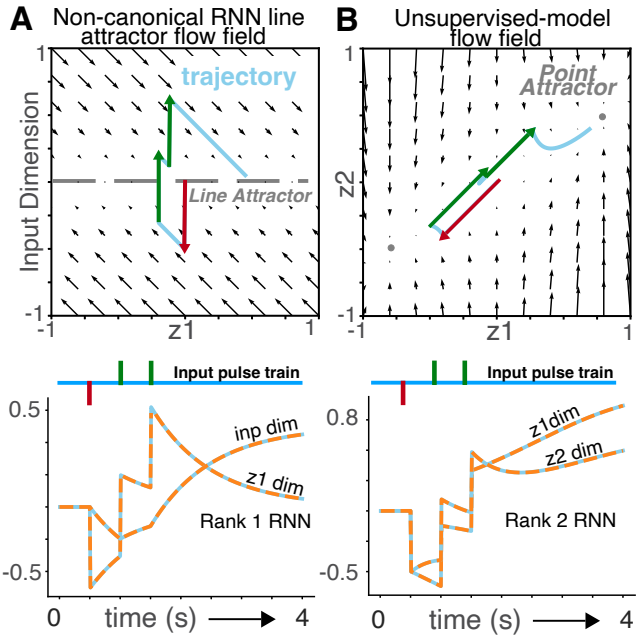


Figure 11: Two additional dynamical portraits for binary decision making task. Top: flow-field for each ODE, with input driven trajectory highlighted in blue. Bottom: true and fitted trajectories over time for each dimension.

# D OMP algorithm and it's application to a limit cycle ODE

The detailed algorithm for OMP is given here:

**Algorithm 2** OMP for finding smallest RNN

1: Select a grid of values $\mathbf{z}$.
2: Create global basis set $\Phi$ using uniformly sampled $\mathbf{m}, \mathbf{b}$ values.
3: Initialize $n$ weights using linear decay term only: $n_0 = -(\mathbf{z}^\top \mathbf{z})^{-1}\mathbf{z}^\top g(\mathbf{z})$.
4: Initialize residual: $r_0 = g(\mathbf{z}) - n_0 * (-\mathbf{z})$
5: Initialize solution basis set, $\Phi_0 = \emptyset$.
6: At each iteration $t$:

1. Find basis vector with highest correlation with residual:

$$i_t = \arg\max_i ||\Phi_i^T \mathbf{r}_{t-1}|||| \tag{20}$$

2. If entry $i$ is not in the solution basis:
   - Add new entry to the solution basis, $\Phi_t \leftarrow \Phi_i$
   - Solve to find new linear weights $[n_{0_t}\ \mathbf{n}'_t]$ using Eqn 8
   - Solve to find new linear weights $\mathbf{n}'_t$ using Eqn 8
   - Compute the updated residual:

$$r_t = g(\mathbf{z}) - \begin{bmatrix} n_{0_t}(-\mathbf{z}) \\ \mathbf{n}_t'^\top \Phi_t \end{bmatrix}, \tag{21}$$

3. Check for termination based on a predefined sparsity threshold

$$d' = \text{len}(\Phi_t) \tag{22}$$

## D.1 Smallest RNN for 2D Limit Cycle

Following our discussion on the multi-dimensional case and the smallest RNN (Sec. 3.1, 5 ), we apply our framework to learn the smallest number of neurons needed to fit an RNN for the limit cycle flow-field (equations provided in SI. A.2). Specifically, we apply both our OMP and COMP method to greedily add neurons that best approximate the target ODE. Fig 12 highlights the advantage of our COMP and OMP methods in designing small RNNs. Additionally we note, for our COMP method, with just 20 neurons the ODE fits are qualitatively similar to the true target, with much lower MSE values compared to OMP. This showcases the advantage of COMP as the basis parameter space increase.

## D.2 Parameter distribution of random basis for OMP

In this section we delve into the role of the distribution from which the random basis is sampled. As shown in Section 5, each basis function approximates the ODE ($g(\mathbf{z})$) over some finite domain ($\mathbf{z}$). Thus, first, it is critical the basis functions span the domain of the function being approximated. Second, these functions need not be odd symmetric and hence basis functions need to also be shifted to capture these movements. As shown in Fig 13, as long as these properties are met (i.e both the uniform grid and standard normal generate basis functions in the same domain, with the same offset ranges), the exact underlying distribution from which the basis functions are drawn does not play a critical role when finding the smallest low-rank RNN. This can be seen as the MSE values follow similar trends with greedy addition of basis functions (panel D). Qualitatively, this can also be observed via similar reconstruction of the ODE across iterations of OMP (panel A,B). Note however, by changing the distribution from which $\Phi$ is drawn the exact basis picked are different, as the global optimal basis are no longer the same (panel C).

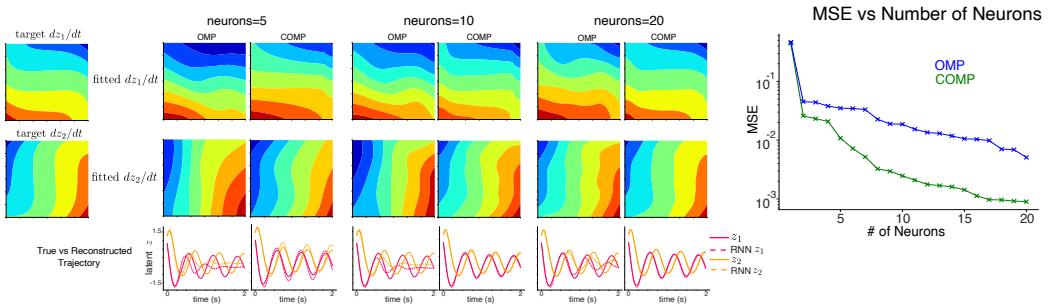


Figure 12: COMP and OMP for 2d Limit Cycle. The first row depicts the estimated flow-field for dimension one. The second row depicts the estimated flow-field for dimension two. Last row shows true vs the RNN reconstructed trajectory for both OMP and COMP. We show fits and reconstructions for $5, 10, 20$ neurons that are added via OMP and COMP respectively. Right most panel depicts the mean squared error (on log-scale) as a function of the number of neurons. Note COMP shows much steeper drops in MSE, compared to OMP.

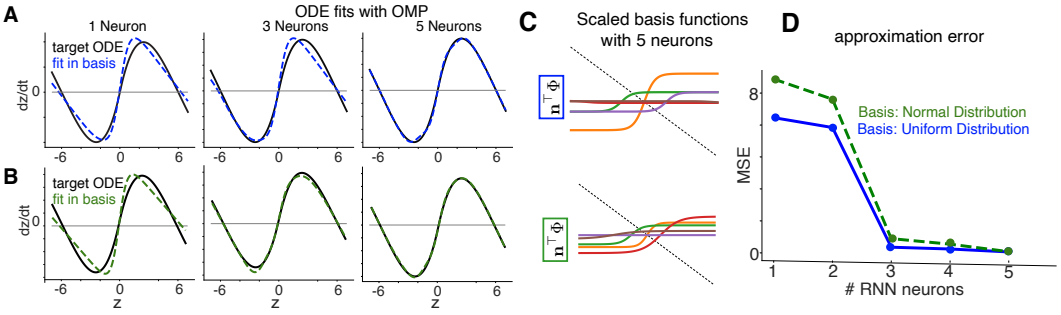


Figure 13: Influence of distribution of basis functions. **(A,B)** RNN estimated fit of Bi-stable attractor ODE the true underlying ODE over 1,3,5 iterations of OMP. In the top row basis functions (both $m_i$ and $\mathbf{b}_i$) are generated over a uniform grid spanning $+1$ to $-1$. Alternatively the bottom row consists of basis functions generated from a standard normal (i.e $m_i \sim \mathcal{N}(0,1)$, and $\mathbf{b}_i \sim \mathcal{N}(0,1)$). **(C)** Scaled basis functions selected after $5$ OMP iterations, with the top row drawn from the dniform distribution and the bottom row from the normal distribution. **(D)** Mean squared error (MSE) between target ODE and RNN approximation as a function of the number of RNN neurons added by OMP (blue: basis functions drawn from uniform grid, green: basis functions drawn from standard normal).

# E Additional training details

## E.1 Backprop comparison for a binary decision making ODE

In Section 4, both our framework and the networks trained with Backprop are trained and tested from a set of teacher trajectories, which are simulated from the underlying ODE (eg., in the binary decision-making task, trajectories originate from random initial conditions and evolve toward one of two fixed points). To generate each trajectory, we used Euler integration with a time step of $dt = 0.01$ over a duration of $4$ seconds, yielding $400$ time steps per trajectory. A total of $160$ unique initial conditions were uniformly sampled along the $z$-axis. Of these, $150$ were used for training and the remaining $10$ for testing. Each of the networks trained via BP in Fig 3 were trained for $15$ epochs, with a batch size of $10$ per epoch. Thus, a total of $150$ gradient steps were performed (performance plateaued at $100$ gradient steps). Additionally, each of the networks were trained over three random seeds of initializations, where parameters were initialized from standard normal distributions. The values reported in Fig 3 correspond to the best performing seed. Lastly, to compare performance we trained our networks via our online RLS implementation. Once training was complete (using the $150$ training trajectories), we used the final $\mathbf{n}$ weight to test performance.

We report training times in Table 1. Note, our framework provides significantly faster training.

Table 1: Training Time For Binary-Decision Making Task

| Model Type | Size | Time(s) |
|---|---|---|
| Low Rank ($r = 1$) | 5 | 62.419 |
| Low Rank ($r = 1$) | 10 | 62.468 |
| Full Rank | 2 | 29.161 |
| Full Rank | 3 | 29.209 |
| Full Rank | 5 | 29.025 |
| Full Rank | 10 | 29.392 |
| Full Rank | 50 | 28.998 |
| Our Model | 5 | 0.069 |

### E.2 Target tracking task: Compare networks with optimal bases vs standard normal bases

To compare performance for networks trained with optimized basis against those trained with the standard normal basis, we derive a target matching problem from the ODE presented in Figure 5 (Panels A and B). Specifically, we used a set of four starting locations in $\mathbf{z}$ given by: $[7.92, 4.72, 2.3, -7.36]$. We then rolled out the dynamics (as per the ODE) using an Euler integration for a total of a $1000$ time-steps, with $dt = 0.1$. Each of these trajectories (originating from each of the start locations) converged to a specific stable fixed point. We then compared rank $1$ RNNs trained via our Online Method (RLS) and those trained with BPTT. Our RLS method achieves near perfect performance after a single epoch, while BPTT trained networks were trained for $5$ epochs. MSE results presented are averaged over $5$ random seeds of initialization.

### E.3 FORCE Comparisons

#### E.3.1 Lorenz Attractor Trajectory

Following our discussion in Sec. 4, we present an additional evaluation using a more challenging target trajectory: the Lorenz attractor, a well-known chaotic system defined by the set of coupled nonlinear differential equations

$$\frac{dx}{dt} = \sigma(y - x), \quad \frac{dy}{dt} = x(\rho - z) - y, \quad \frac{dz}{dt} = xy - \beta z, \tag{23}$$

using standard parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. We simulated the trajectory using Euler integration with a time step of $dt = 0.01$ for a total of $T = 3000$ steps (30 seconds of data), and used the scaled $x(t)$ signal as the target output (as in [9]). While our method was trained on (and able to reconstruct) all three variables $Z = [x(t), y(t), \text{and } z(t)]$, only reconstructions of $x(t)$ are shown for comparison, as this is the only signal used in the FORCE training objective [9]. Specifically in our case, this can be formalized as a rank-3 RNN, written as the problem of fitting three different nonlinear functions $\frac{dx}{dt}$, $\frac{dy}{dt}$ and $\frac{dz}{dt}$ using three different linear combinations of the same 3D basis functions:

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix} \approx -\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \mathbf{n}_1^\top \phi(MZ + \mathbf{b}) \\ \mathbf{n}_2^\top \phi(MZ + \mathbf{b}) \\ \mathbf{n}_3^\top \phi(MZ + \mathbf{b}) \end{bmatrix}, \tag{24}$$

where $M = [\mathbf{m}_1 \mathbf{m}_2 \mathbf{m}_2]$ is a $d \times 3$ matrix, $\mathbf{b}$ is once again a column vector of offsets, and we have assumed a constant filtered input, $\mathbf{v} = 1$.

We compare networks trained using our online learning framework against those trained with the FORCE algorithm. For FORCE, we used a recurrent gain of $g = 1.5$ and trained each network across 10 random seeds; our method was similarly evaluated with the same number of seeds. Fig 14 reports mean squared error (MSE) averaged across seeds, along with representative trajectory reconstructions.

As in previous experiments, our method consistently achieves lower MSE with fewer neurons. Qualitatively, this is also evident in the trajectory reconstructions: our networks produce more

accurate fits at smaller sizes ($N = 16, 64$) than FORCE, which fails to reliably capture the signal until larger networks (for instance showed with $N = 1024$). Additionally, this further highlights that our networks converge more rapidly than those trained with FORCE.

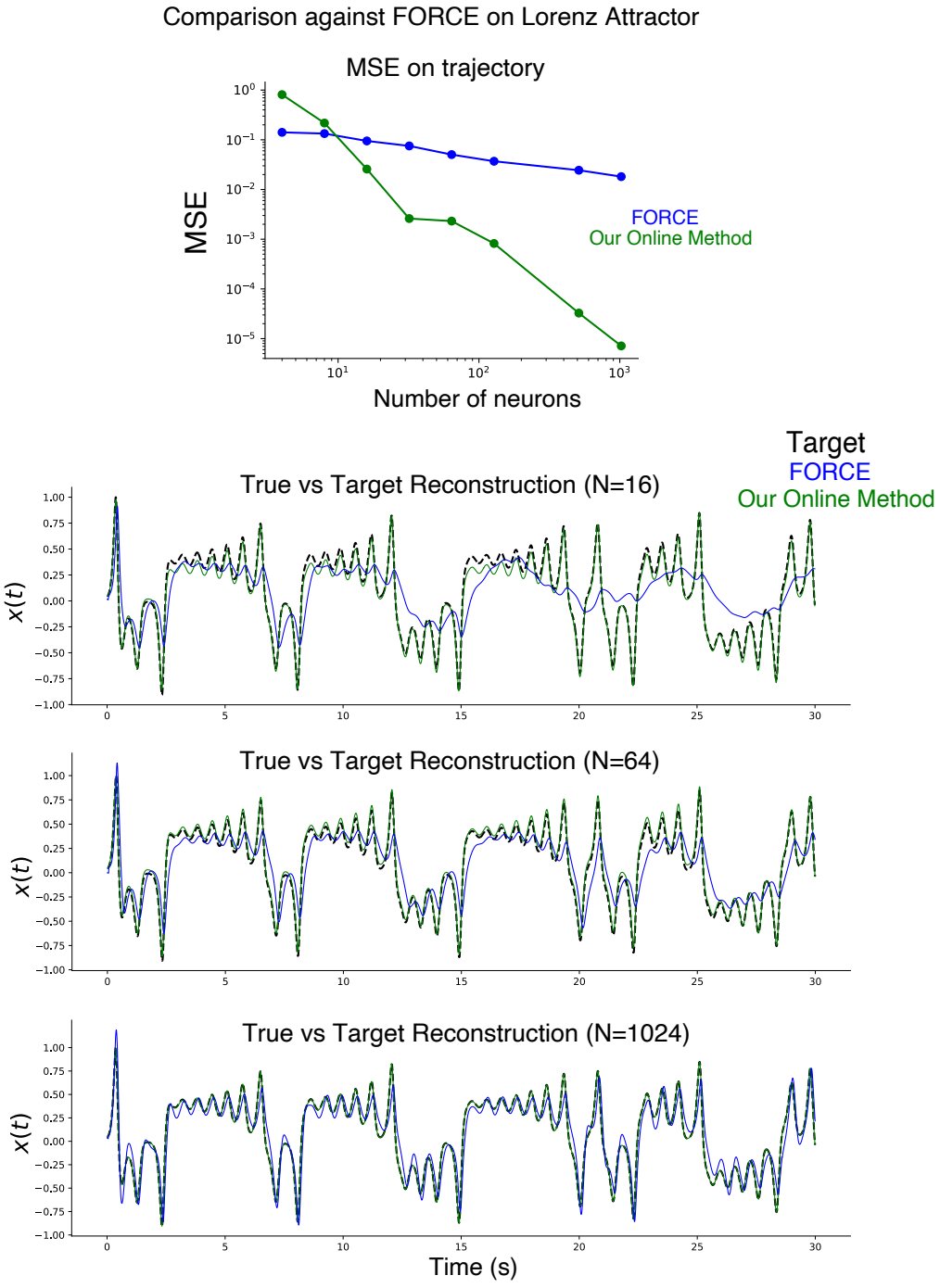


Figure 14: **Top Row:** Mean squared error (MSE) in log scale across networks of varying size (left). **Bottom Rows:** True vs reconstructed $x(t)$ trajectory from the Lorenz system, (green our method, blue FORCE) for networks of size $16, 64, 1024$.

### E.3.2 Application to noisy data

We further validate our framework by discussing it's application to noisy data. Specifically, we simulated a target signal for a total of $T = 1000$ seconds. This signal was sampled with a discrete time bin of $dt = 0.1$, thus 10,000 time steps of the target wave were obtained. We then added independent gaussian noise: $\mathcal{N} \sim (0, 0.05)$, at each time step of the trajectory to obtain the noisy target (also used in FORCE [9]). While the target output is a 1-d signal, we used the trajectory and it's two cumulative sums to fit our networks (thus needing rank 3 network). We trained RNNs using our RLS method, FORCE and BPTT (rank 3, 4). To ensure a consistent comparison, in Fig. 15 we report the training MSE (averaged across 5 seeds) after one epoch: a single pass over the $10,000$-step sequence, across all methods. We see highest performance with our RLS method. Additionally, both RLS and FORCE achieve near-perfect performance in a single pass, while BPTT typically requires 5–8 epochs to reach comparable errors.

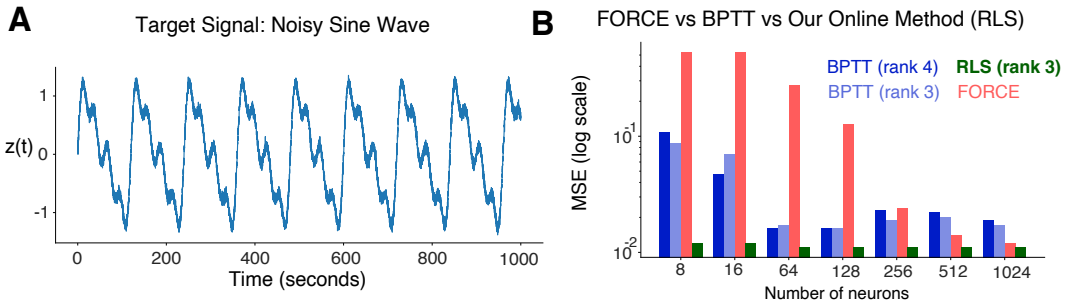


Figure 15: Networks fit on a noisy sum of four sinusoids target signal. **A**: Target trajectory sampled for $T = 1000$ seconds. **B**: Training Mean Squared Error (in log scale) of networks trained with FORCE, BPTT and our Online Method (RLS)

### E.3.3 Additional discussion on our online (RLS) method and the FORCE Framework

Below we discuss some ways in which our methodology differs from the FORCE/FULL-FORCE [9, 10] training schemes.

1. **Initialization with Full-Rank Weight Matrix:** FORCE/FULL-FORCE methods require full rank-initializations, and learn low-rank updates on this initilalization over time. This comes at the cost of interpretibility as the dynamics of such networks need to be analysed post training through methods such as PCA. On the other hand, our *offline and online* framework directly models the latent low-dimensional dynamic and doesn't ever need full-rank initializations.
2. **Senstive to initialization and requires multiple epochs:** One of the motivations of FORCE is that it introduces the benefits of training networks that exhibit chaotic activity prior to training. While powerful, it is observed this results in these networks needing multiple epochs/iterations. Additionally, these networks exhibit stochastic results based on initialization (e.g based on the parameter $g$). In contrast, our methods provide a deterministic closed form updates.
3. **Doesn't Directly Embed an ODE, but Produces a Set of Target Trajectories:** A stark difference between our *offline* framework is unlike other training methodologies similar to FORCE and FULL-FORCE that can only be trained against target trajectories, we can also directly model the underlying ODE. Thus, in cases where such a hypothesized ODE exists, we can represent the entire space of the low-dimensional dynamic.

## F Embedding higher dimensional ODEs

To highlight the ability of our framework to embed higher dimensional ODEs, we train a network to perform the n-bit flip flop task [12]. Specifically, we consider the case where $n = 10$. In this case,

the network receives $10$ binary inputs at random times, and produces $10$ binary outputs. The network is expected to maintain it's output if the inputs received are the same, or alternatively switch it's output if the opposite input pulse is presented. We successfully embed such a task using a rank $10$ network with $10$ neurons. We design our networks such that each neuron is required to keep track of a single input pulse, thus requiring $10$ neurons. Note, this could be arbitrarily scaled up to any number of input pulses, by simply scaling up the number of neurons in the network. In Fig. 16 each neuron is presented with random positive or negative input pulses in blue, to which it must respond appropriately. Our method generates the required output as seen via the activity ($x(t)$) of $4$ example neurons from this network. The network was simulated for $T = 50$ seconds, sampled at $dt = 0.01$. Concretely, this was achieved by rolling out the equivalent high-dimensional network dynamics given by:

$$\dot{\mathbf{x}} = -\mathbf{x} + MN^\top \phi(\mathbf{x}) + B\mathbf{u} \tag{25}$$

In our case, $M, N, B$ are randomly drawn from standard normal distributions. $\mathbf{u}$ represents the input pulse train seen in blue, and the vector of neural activity $x(t)$ represents the desired output.

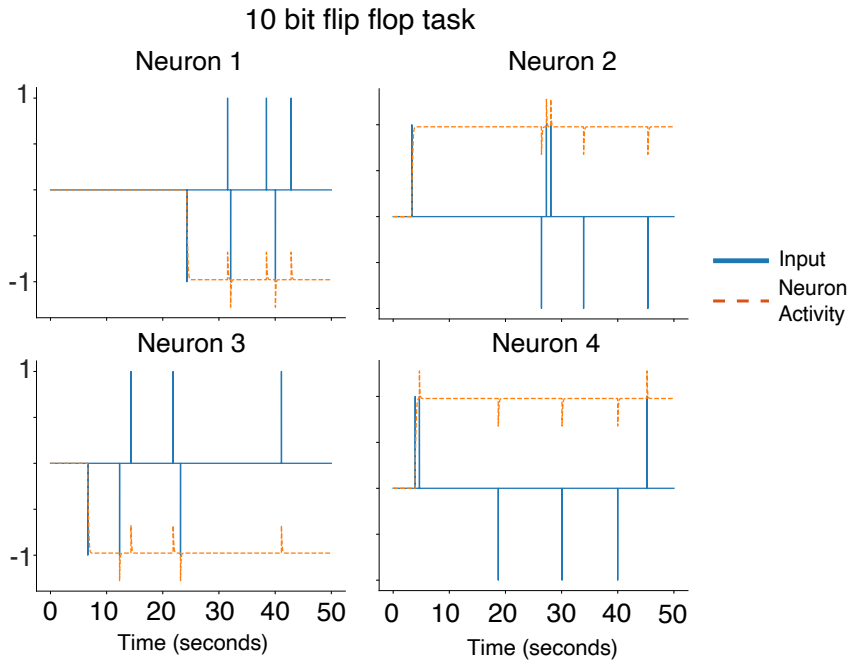


Figure 16: RNN generating appropriate response for the $10$ bit flip-flop task. Neuron Activity, $x(t)$ of $4$ example neurons is shown.