

## A APPENDIX 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. SI-1). 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.

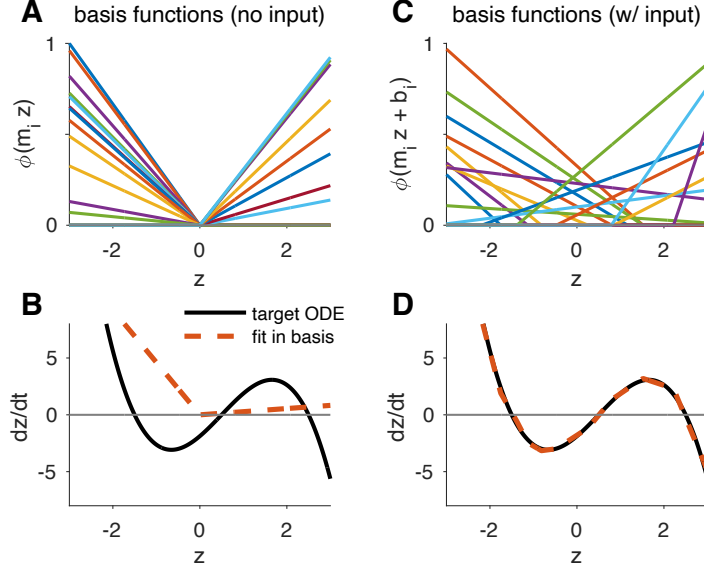


Figure SI-1: Representational capacity of a 1D low-rank RNN with rectified-linear (*relu*) nonlinearity. (A) Set of basis functions obtained by taking random coefficients  $m_i \sim \mathcal{N}(0, 1)$  but without input ( $\mathbf{v}_t = 0$ ). (B) Attempting to fit an example ODE using this basis recovers 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  $I_i \sim \mathcal{N}(0, 1)$ . (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. SI-2. 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).

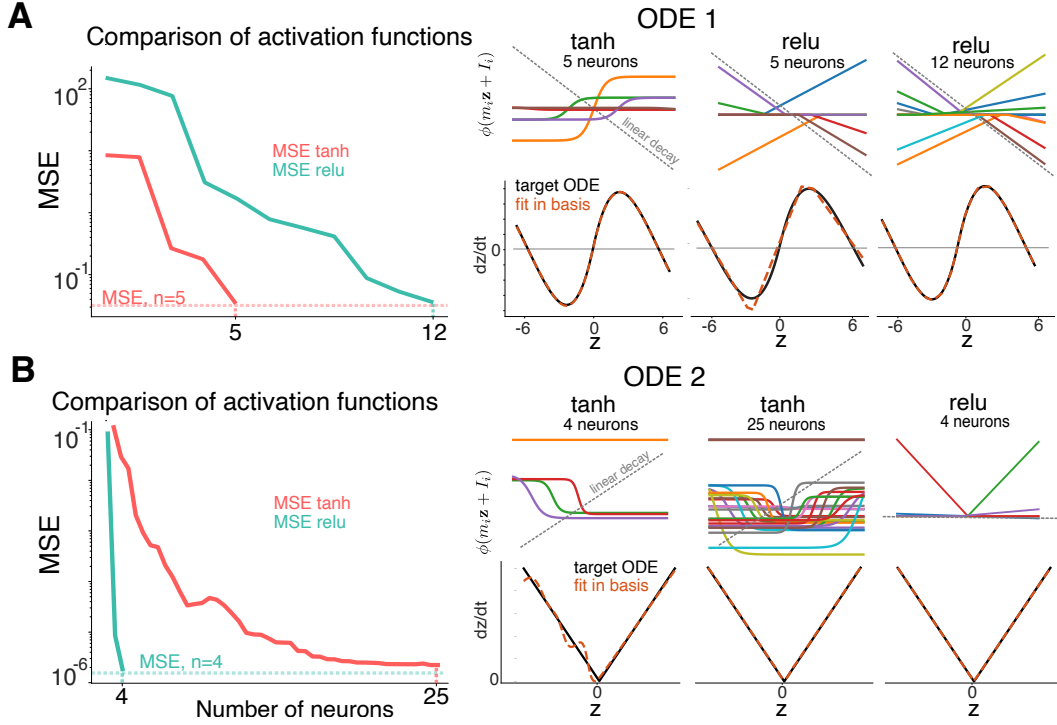


Figure SI-2: Performance comparison of tanh v/s relu in approximating different ODEs. (A) Depicts an ODE with two stable fixed points and one unstable fixed points. (B) Depicts an ODE with a shifted knot and two linear components. First column represents MSE (for a network with tanh and relu activations) as a function of the number of neurons in network. Neurons are added using OMP. Top row shows scaled basis functions selected after 1, 3, 5 iterations of OMP, along with the linear decay term  $-x$ . Bottom row shows target ODE (black) and RNN fit (orange) after each step 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 -

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

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

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. SI-3 the first column represents contour plots of the target ODE for each dimension. The overlaid 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 column represented the fitted ODEs for an RNN with inputs, while the last column represents fitted ODEs for an RNN without inputs. 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. SI-3. As expected, the low-rank RNN with inputs almost perfectly overlaps the trajectory, unlike the low-rank RNN without inputs.

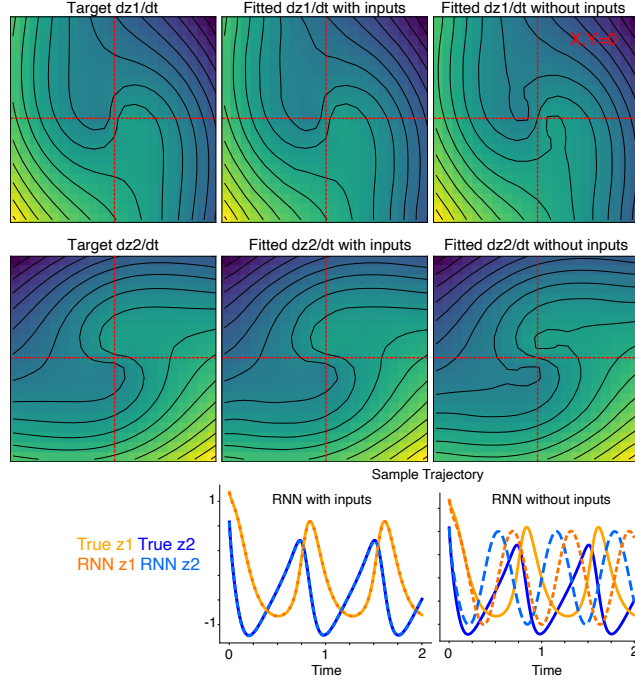


Figure SI-3: Influence of inputs in capturing non-symmetrical limit cycle

## B GENERAL FORMULATION AND 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 (Luo et al. (2023)). We thus show applicability of our method by using it to recover the intrinsic and input driven dynamics on four separate synthetically generated dynamic portraits linked to this task (Luo et al. (2023)). 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 intrinsic 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 (Wong & Wang (2006)). 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, an non-canonical line attractor and the flow field inferred from (Luo et al. (2023)). More formally, they are given as follows -

Bistable attractors:

$$\begin{aligned} dz_1 &= 10z_1(0.7 + z_1)(0.7 - z_1)dt + cudt \\ dz_2 &= -10z_2dt \end{aligned}$$

Classic DDM - line attractor:

$$\begin{aligned} dz_1 &= \begin{cases} cudt & z_1 \in (-0.7, 0.7) \\ 10z_1(0.7 - z_1)(0.7 + z_1)dt & z_1 \notin (-0.7, 0.7) \end{cases} \\ dz_2 &= -30z_2 \end{aligned} \quad (15)$$

Non-canonical line attractor:

$$\begin{aligned} dz_1 &= 5z_2 \\ dz_2 &= -5z_2dt + cudt \end{aligned}$$

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 [6] we now not only observe orthogonal ( $I = I_{perp}$ ) neuron specific inputs, but additional input dynamics that influence the recurrent activity ( $I_{par}$ , spans the same direction as  $\mathbf{m}$ ), thus updating Eqn [6] as :

$$\mathbf{x}(t) = \mathbf{m}\mathbf{z}(t) + I_{par}\mathbf{v}'(t) + I_{perp}\mathbf{v}(t), \quad (16)$$

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

$$g(\mathbf{z}) + \mathbf{z} \approx \mathbf{n}^\top \phi(\mathbf{m}\mathbf{z} + I_{par}\mathbf{v}'(t) + I_{perp}\mathbf{v}(t)) \quad (17)$$

where  $\mathbf{v}'(t)$  represents the low-pass filtered input which drives the system in the dimensions spanned by the recurrence ( $\mathbf{m}$ ). This allows us to follow a similar setup to our discussions in Sec. 3, with the exception that auditory inputs are applied along  $I_{par}$  or  $I = I_{perp}$ , or both.

As shown in Fig [SI-4], 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 [7]) against the true ODE (computed using Euler method). Lastly, we also recover the underlying flow fields, as indicated by the last column. In Fig [SI-5] 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  $I_{par}$  (Fig. [SI-5] B). This proves the flexibility of our framework in embedding dynamics associated with neuroscience tasks.

## C ADDITIONAL TASKS AND COMPARISONS

### C.1 DISCUSSION ON FORCE FRAMEWORK

Below we discuss some ways in which our methodology differs from the FORCE/FULL-FORCE (Sussillo & Abbott, 2009; DePasquale et al., 2018) training schemes.

1. **Initialization with Full-Rank Weight Matrix:** FORCE/FULL-FORCE methods require full rank-initializations, and learn low-rank updates on this initialization over time. This comes at the cost of interpretability as the dynamics of such networks need to be analysed post training through methods such as PCA. On the other hand, our framework directly models the latent low-dimensional dynamic and doesn't ever need full-rank initializations and is highly interpretable.



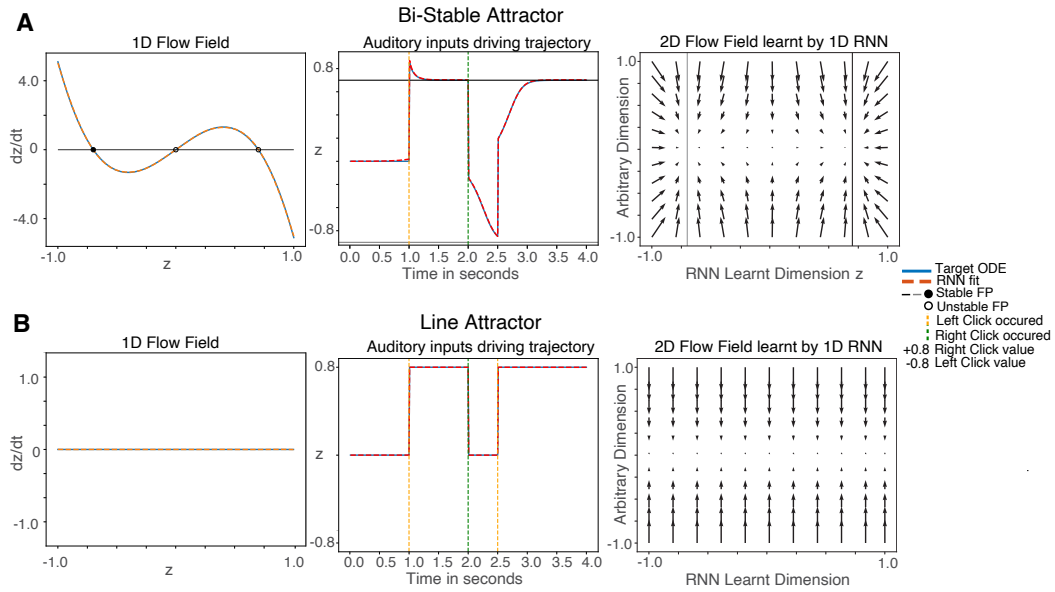


Figure SI-4: Two different dynamical portraits for binary decision making task: (A) bi-stable attractor ODE and (B) line attractor ODE. First column represents the true underlying ODE and the RNN estimate learned using least squares. Second column depicts a sample trajectory driven by momentary input 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.

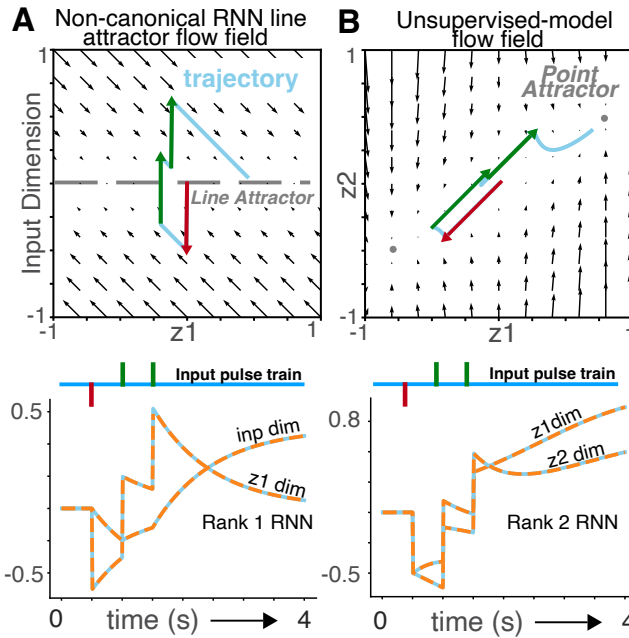


Figure SI-5: Two additional dynamical portraits for binary decision making task. Top: flow-field for each ODE, with input driven trajectory highlighted in blue. Bottom: true and fitted trajectories over time for each dimension.

2. **Sensitive 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. In contrast, our method provides a deterministic and single-step closed form solution.
3. **Doesn't Directly Embed an ODE, but Produces a Set of Target Trajectories:** A stark difference between our framework is unlike other training methodologies similar to FORCE and FULL-FORCE that are 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.

## C.2 3D LORENZ ATTRACTOR

In addition to the 1-dimensional and 2-dimensional ODEs described in the main text, we also scale our method to the Lorenz attractor. Specifically, we train our network on the Lorenz attractor over a set of 10 separate trajectories (start location denoted by red circles in Fig SI-6). Additionally, the error plots in Fig SI-6 indicates our network learns an accurate representation of this system. Our formulation follows from our descriptions in Sec. 3.1, with one extra dimension. This can be formalized as a rank-3 RNN, written as the problem of fitting three different nonlinear functions  $g_1(\mathbf{z})$ ,  $g_2(\mathbf{z})$  and  $g_3(\mathbf{z})$  using three different linear combinations of the same 3D basis functions:

$$g(\mathbf{z}) = \begin{bmatrix} g_1(\mathbf{z}) \\ g_2(\mathbf{z}) \\ g_3(\mathbf{z}) \end{bmatrix} \approx - \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} + \begin{bmatrix} \mathbf{n}_1^\top \phi(M\mathbf{z} + I) \\ \mathbf{n}_2^\top \phi(M\mathbf{z} + I) \\ \mathbf{n}_3^\top \phi(M\mathbf{z} + I) \end{bmatrix}, \quad (18)$$

where  $M = [\mathbf{m}_1 \mathbf{m}_2 \mathbf{m}_3]$  is a  $d \times 3$  matrix,  $I$  is once again a column vector of offsets, and we have assumed a constant filtered input,  $\mathbf{v} = 1$ .

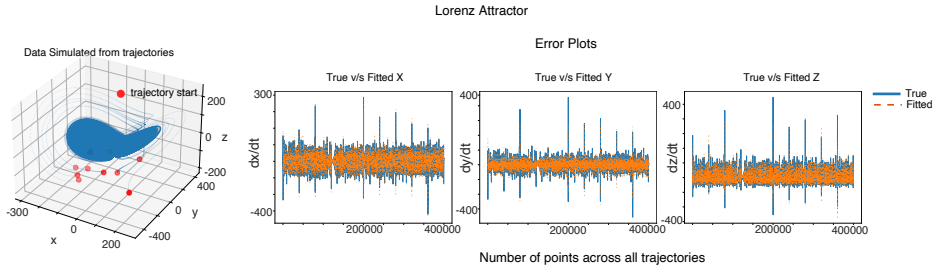


Figure SI-6: Lorenz attractor

Lastly, in Fig SI-7 we also show the efficacy of our model on a sample test trajectory. As shown, our low-rank RNN model recovers the time-traces with high accuracy.

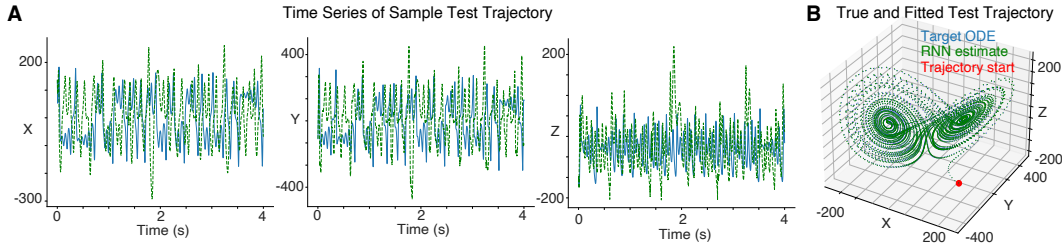


Figure SI-7: Performance on sample test trajectory

## D OMP FOR 2D LIMIT CYCLE

Following our discussion on the multi-dimensional case and OMP (Sec. 3.1 and 4), we apply our framework to learn the smallest number of neurons needed to fit an RNN for the limit cycle ODE. As depicted in Fig SI-8, with the addition of neurons we start noticing periodicity in trajectories with just 6 neurons. Finally, we obtain near perfect fits with 20 neurons.

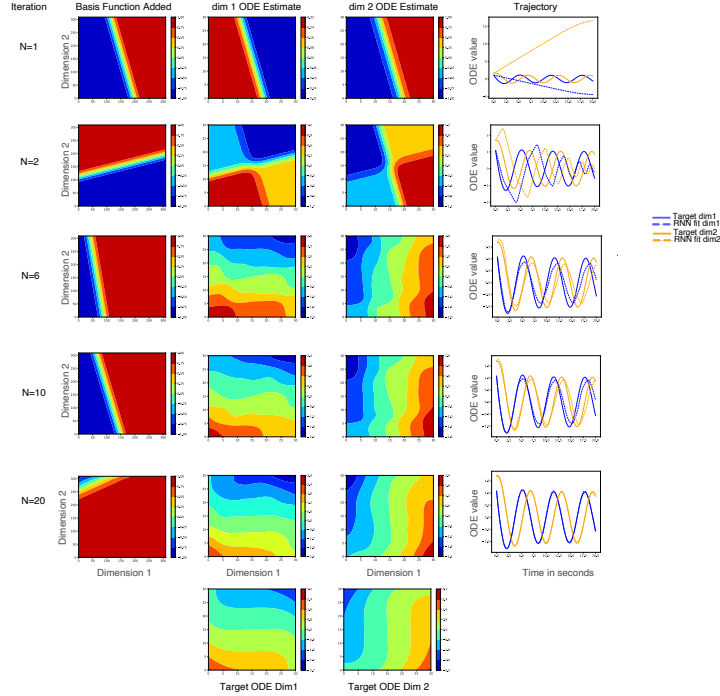


Figure SI-8: OMP for 2d Limit Cycle. Column 1 shows the single basis function added the corresponding iteration of OMP. Columns 2 and 3 represent the estimated flow-fields via a learnt linear weight, i.e two separate weights are learnt using the same basis function. Last column depicts a sample trajectory. Note, periodicity of the limit cycle starts appearing as more neurons are added.

## E PARAMETER DISTRIBUTION OF RANDOM BASIS

In this section we delve into the role of the distribution from which the random basis is sampled. As shown in Section 4, 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 SI-9, 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. This can be seen as the MSE values follow similar trends with greedy addition of basis functions (panel C). Qualitatively, this can also be observed via similar reconstruction of the ODE across iterations of OMP (panel A,B).

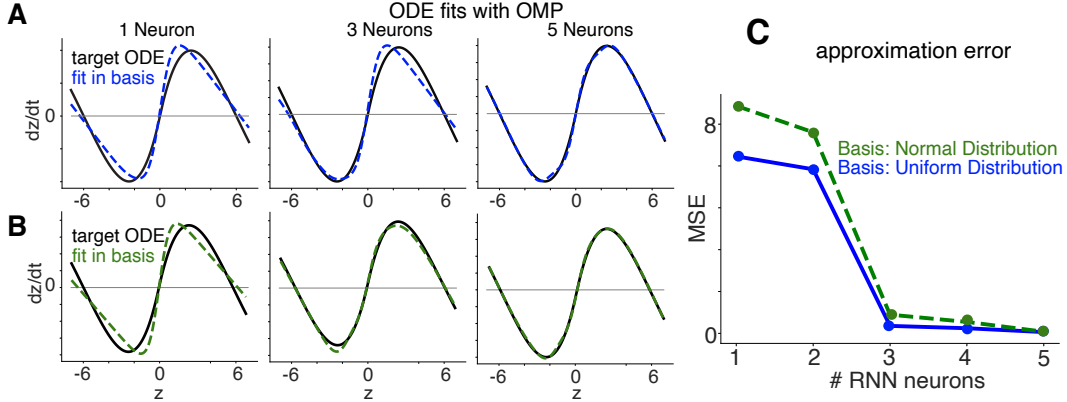


Figure SI-9: Influence of distribution of basis functions. **(A,B)** RNN estimated fit of Bi-stable attractor ODE the true underlying ODE over 1,3,5 iterations of OMP. In the top row basis functions (both  $m_i$  and  $I_i$ ) are generated over a uniform grid spanning  $+1$  to  $-1$ . Alternatively the bottom row consists of basis functions generated from a standard normal (i.e  $m_i \sim \mathcal{N}(0, 1)$ , and  $I_i \sim \mathcal{N}(0, 1)$ ). **(C)** 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).

## F ADDITIONAL TRAINING DETAILS

In Section 6 both our framework and the networks trained with BPTT are trained from a set of teacher trajectories, which are simulated from the underlying ODE (eg. for binary decision making - they start somewhere on the grid and eventually converge to one of the two fixed points). However, unlike previous sections, our method here can be broken into two main steps -

1. **Estimate ODE:** Through a finite differencing approach (euler), we compute the difference between every pair of points along the training trajectory. Thus, using a small time bin, we estimate the value of  $g(z)$  along each point of the trajectory. This is used to populate our target vector for regression.
2. **Perform Regression:** Our basis matrix is evaluated at grid points along these training trajectories. Given this design matrix, and target we compute the necessary weights through Eqn 9

Lastly, both our framework and the networks trained with BPTT must reproduce a set of target trajectories (eg. blue lines in In Fig. 5 panel A) from an input pulse train (eg. representing bit value for each channel over the time interval). Thus, both our method and models trained with gradient methods are trained to uncover underlying dynamics (from data points) that solve the task.

As shown in Table 1, 2 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

Table 2: Training Time For 3-Bit Flip Flop Task

Model Type	Size	Time(s)
Low Rank ( $r = 3$ )	5	305.256
Low Rank ( $r = 3$ )	10	303.559
Full Rank	3	170.512
Full Rank	5	169.076
Full Rank	10	170.239
Full Rank	50	173.590
Our Model	5	0.09

### F.1 APPLICATION TO HIGH DIMENSIONAL NOISY DATA

We further validate our framework by discussing its application to high-dimensional noisy data. In this case we assume we have access to high-dimensional noisy rate-based neural recordings. Specifically, this simulation is achieved by simulating a trajectory from the bi-stable ODE in Fig. SI-5B. This trajectory  $\mathbf{z}(t)$ , starts somewhere on the grid of  $\mathbf{z}$ , and is run forward until it converges to one of the fixed points. To convert this into a high-dimensional noisy rate based recordings we take the following steps. For each value along this trajectory  $\mathbf{z}(t)$  we -

1. Project it into a high dimensional space through a linear weight matrix (values are drawn between 0-1).
2. This linear mapping is made non-linear via the tanh activation
3. Finally, we independently add Gaussian noise to each of the dimensions ( $\mathcal{N}(0, 0.01)$ ).

We thus assume we have access to these high-dimensional recordings. We project the data onto the top PC's (since the underlying dynamic here is 1-d, the first PC dimension captures the dynamic). In Fig. SI-10 the low-dimensional trajectory was first projected into a 1000 dimensional space. PCA on this trajectory showed the single top PC captured most of the variance in the data. As shown, through this projection we recover a noisy trajectory that represents the true low-dimensional dynamic. The rank of the network will depend on the number of PC dimensions needed, in this case 1 suffices. Furthermore, as noise in the system increases this recovery will also become more noisy (or need more dimensions to capture it). Other confounding factors such as low-resolution recordings or sparse recordings could also influence recovery (although not explored here).

Finally, once we have access to these trajectories the approach discussed in Section F can once again be followed.

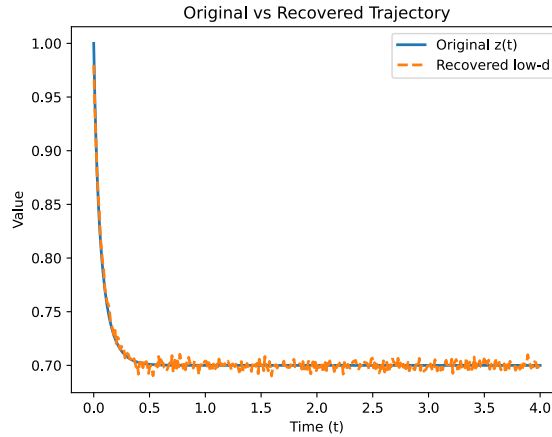


Figure SI-10: Recovering low-dimensional trajectory from noisy dimensional data