Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We mention that the mechanisms in the RNNs should be interpreted as hypothesis for biological mechanisms
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Since our work involves understanding deep neural networks, it inherits the broader positive and negative societal impacts of deep learning.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All details required to implement our experiments are described in the Appendix, and we have provided the code to our implementation.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 2.1 and appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See footnote on page 2
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

4.1 Clustering of attractor states

We constructed a similarity matrix, $\mathbb{S}$, to quantify the topological structure of attractor states. This section details how $\mathbb{S}$, was computed. Because all RNNs across all learning rules and hyperparameter choices studied in this work always instantiated a single attractor state for large input values, we focused on attractor states near zero as this is where we observed the greatest degree of variation between RNNs. Therefore, we compute all the attractor states for static input values between -0.02 and $+0.02$ (-0.03 and 0.03 for RDM task). Specifically we examined 20 non-zero static inputs in this range along with the zero input to give a total of 21 static input conditions, for each context. We examined only a subset of attractor states since different RNNs instantiate a variable number of attractor states for small inputs. Specifically, we chose to examine the $m = 10N$ attractors with the smallest absolute value, where N corresponds to the number of contextual inputs for the task. For the RDM task, we set $N = 1$.

We rank ordered the m attractors in order of increasing absolute value. For each of these attractor states, we solved for its nearest $n = 3N$ neighbors in RNN state space ($\mathbb{R}^{50}$) among all attractors found (not just the smallest m) using the L_2 distance. For each RNN we constructed a topological embedding vector, $\mathbf{v}$, based on the nearest neighbors for each of the m attractor states. The first element of the topological embedding vector, $\mathbf{v}_0$, is the static input associated with the first attractor examined (which is always zero). The next $n - 1$ elements of this vector are the static inputs associated with the n nearest neighboring attractors in RNN space. The next element at position n is the next attractor state examined, and the following $n - 1$ entries are the static inputs associated with its nearest neighboring attractor states. More generally, each element $\mathbf{v}_{ij}$ for $n \in \{0, 1, 2, ..., m\}$ will be the static input associated with the i^{th} attractor state examined and $\{\mathbf{v}_{in}, ..., \mathbf{v}_{(i+1)n-1}\}$ are the static inputs associated with the nearest neighboring attractors of the i^{th} attractor state. Therefore, for each RNN we constructed a topological embedding vector, $\mathbf{v}$ of length $mn = 30N^2$.

We then constructed the similarity matrix, $\mathbb{S}$, for a population of p RNNs by stacking the topological similarity vectors. Therefore, this similarity matrix was a $p \times mn$ matrix. Since each topological similarity vector contributed a row in the similarity matrix, if for all RNNs, the relative topological ordering of attractors by static input values was the same, then all rows would be the same and RNNs would be tightly clustered. Alternatively, if the topological arrangement of attractors varied greatly between RNNs, the similarity matrix would exhibit heterogeneity.

Silhouette cluster analysis was used to quantify the degree of clustering among attractor topologies from different RNNs. The ground truth lables were given by the RNNs learning rule (or specific hyperparameter being analyzed). MDS was used only for visualizing the amount of clustering among RNN attractor topologies.

Trained Models

Table 1 lists all the models that were trained for this study. We trained a total of $1,760$ RNNs across the study.

Integration strategies for CDI task

Figure 6 depicts the transient and stable integration strategies observed in RNNs trained to do the CDI task with either low (input variance set to 0.1) or high (input variance set to 1.0) input noise.

Visualization of Attractors for N-CDI Tasks

Figure 7 depicts the attractor structures for example RNNs trained on the 3,4,5, and 6-dimensional CDI tasks. All these attractor structures exhibit multiple attractors associated with the zero input and a single attractor associated with non-zero inputs. As additional input dimensions are added to the task, the network instantates additional attractor states associated with those contexts. Note that Figure 7 exaggerates attractor overlap when there are more contexts, since they are only visualized in 2D.

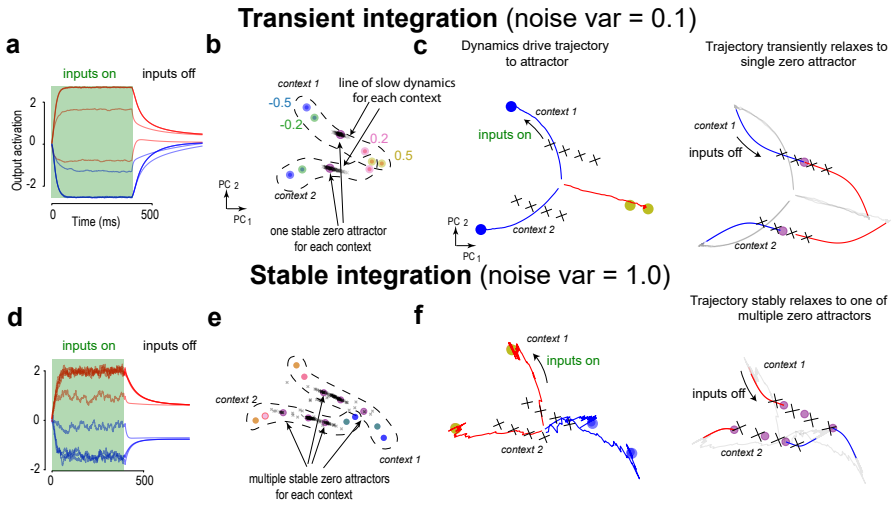

Figure 6: **Two different input strategies based on input noise for CDI-task.** Panels (a-c) illustrate transient integration, while (d-f) is for stable integration. **(a)** RNN output activation for positive (red) and negative (blue) inputs. After inputs turn off, the RNN output slowly decays. **(b)** Attractor topology for a transient RNN. Dots are attractors. Attractors corresponding to each of the two contexts are outlined. Numbers next to attractors in context one correspond to the setting of the input, $\mathbf{u}_t$. There was a single attractor for $\mathbf{u}_t = 0$ in both contexts. There was also a line attractor of slow dynamics under zero input, denoted by gray x's, for each context. **(c)** When the inputs turn on, the RNN state is driven towards attractors (blue, yellow dots). When the input turns off, the trajectories relax to a single attractor corresponding to zero input. This decay is relatively slow along a line attractor of slow dynamics. **(d)** RNN output activation for a RNN with stable integration. After the inputs turn off, the RNN output maintains a stable value. **(e)** Attractor topology for each context in a stable RNN. Now, at an input of zero (purple), there are multiple attractors in each context. **(f)** After the dots turn off, the trajectory stably relaxes to one of multiple attractors corresponding to zero input, thereby holding the memory of the input indefinitely.

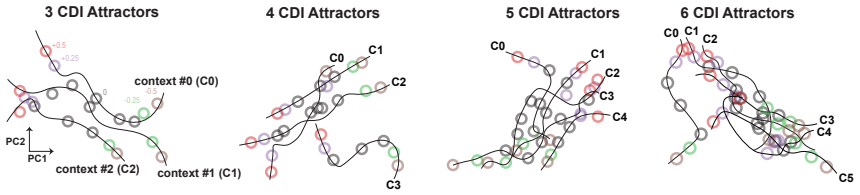

Figure 7: **Example attractor structures for an RNN trained on the N-CDI tasks.** For each N-CDI task, attractors are instantiated for each of the N contexts. Within each these contexts, attractors associated with zero input are duplicated.

Table 1: Breakdown of all RNNs traiend for this study

Task & Learning Rule	Number of RNNs	V=1.0	V=0.75	V=0.50	V=0.25	V=0.10
RDM BPTT	130	50	20	20	20	20
RDM GA	130	50	20	20	20	20
RDM H	130	50	20	20	20	20
RDM FF	130	50	20	20	20	20
CDI BPTT	160	50	20	50	20	20
CDI GA	160	50	20	50	20	20
CDI H	130	50	20	20	20	20
CDI FF	130	50	20	20	20	20
3-CDI BPTT	90	20	0	50	0	20
3-CDI GA	50	0	0	50	0	0
4-CDI BPTT	90	20	0	50	0	20
4-CDI GA	50	0	0	50	0	0
5-CDI BPTT	90	20	0	50	0	20
5-CDI GA	50	0	0	50	0	0
6-CDI BPTT	90	20	0	50	0	20
6-CDI GA	50	0	0	50	0	0
RDM BPTT (ReLU)	50	50	0	0	0	0
CDI BPTT (ReLU)	50	50	0	0	0	0
DNMS BPTT	20					
DNMS GA	20					
DNMS H	20					
DNMS FF	20					
RDM BPTT (single neuron)	20					
RDM GA (single neuro)	20					
CDI BPTT (single neuro)	20					
CDI GA (single neuro)	20					
Total	1920					

Delayed Non-Match To Sample Task

In addition to the integration tasks discussed in main text, we also investigated the attractor topologies and network representations following training on a working memory task. To do this, we trained RNNs to perform the delayed non-match to sample (DNMS) task as described in [18] using the four different learning rules. The network was presented with two input channels which are set to zero at all times except for two stimulus periods (stimulus 1: $0 - 200$ms and stimulus 2: $400 - 600$ms). During the stimulus periods, one of the two input channels is set to one. The stimuli may occur in the same channel or different channels. The RNN is trained to output $+1$ when both stimuli occur in the same channel and -1 when the stimuli occur in different channels.

Consistent with our findings for the integration tasks, we found that RNNs trained with different learning rules may result in networks with distinct representations (Figure 8b) but similar attractor structures (Figure 8c). In contrast to the integration tasks, which had a continuous space of physiological attractors, this task only has three attractor states: two attractors each corresponding to one of the channels being turned on and one attractor corresponding to both channels off. We noticed three clusters in the attractor MDS because our clustering methods are sensitive to slight variations in distances between these three neighboring attractors. This does not mean there are three distinct dynamical mechanisms. All these networks exhibited a similar dynamical motif during the task, depicted in Figure 8e. Rather, the attractor MDS method will find distinct clusters depending on which of the attractors are closest to each other, which may vary even within a learning rule as in Figure 8c. This is a limitation of the attractor MDS clustering we used. Nevertheless, we see there is no significant clustering of attractor structure by learning rule (silhouette score: 0.01 for attractor MDS).

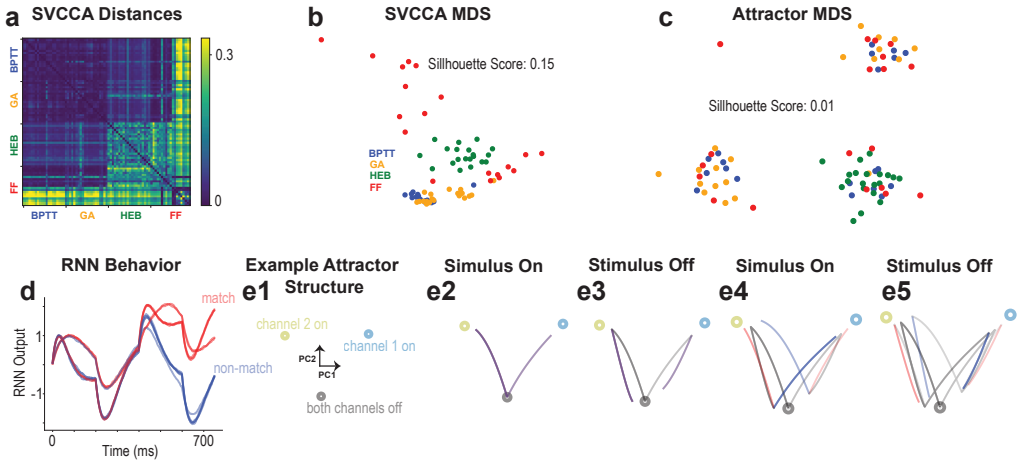

Figure 8: **Representations and attractor structures for RNNs trained on the DNMS task. (a)** SVCCA distances between RNNs on the DNMS task following training with one of the four learning rules. **(b)** MDS clustering of representations and **(c)** and attractor structures reveals that there are some differences in the representations learned by networks trained with different learning rules but attractor structures remain similar (SVCCA silhouette: 0.15, attractor silhouette: 0.01). **(d)** After training, RNNs are able to succesfully discriminate between match and non-match trials. **(e1)** Example attractor structure for a network trained on the DNMS task. **(e2)** During the first stimulus, the RNNs state moves towareds one of two attractors depending on which channel is turned on. **(e3)** When the stimulus is turned off, RNNs decay back to a common attractor. **(e4)**, during the second stimulus, RNNs take different trajectories depending on which channel gets turned on relative to which channel was on during the first stimulus. **(e5)** After the second stimulus ends, RNNs decay back to a common fixed point, however non-match and match conditions start their decay from different regions of state space, yielding different decay trajectories.

Activation function affects dynamical mechanism for simple but not complex tasks

Because our results suggested RNNs become more invariant to design choice as task complexity grows, we also tested the effect of activation function on attractor topology and representational geometry following training with the RDM and CDI tasks. Specifically, we trained RNNs with BPTT using either a saturating $(1 + \tanh)$ activation function or a non-saturating (ReLU) activation function.

Following training on either the RDM or CDI task, we tested RNNs by providing them a pulse of inputs. We found that RNNs trained to perform the RDM task using the $\tanh$ activation function consistently instantiated the stable integration mechanism while RNNs that used a ReLU activation consistently demonstrated transient integration dynamics (Fig 9a,c). However, networks trained to perform the CDI task maintained a bistable integration mechanism regardless of the choice of activation function (Fig 9b,d). To quantify the similarity of dynamical mechanism between RNNs, we clustered the network attractor states, where the activation function was used as the labels. Networks that used $\tanh$ and ReLU formed distinct attractor clusters for the RDM task but not the CDI task (9e,f), consistent with the observed differences in dynamical mechanism. Representations also formed distinct clusters for different activation functions following training on the RDM but not the CDI task. Therefore, we find that this is further evidence that for more simple tasks RNNs are likely more sensitive to design choice while for more complex tasks RNNs become more invariant to design decisions. Namely, it appeared the activation function was sufficient to impact the dynamical mechanism in the case of a simple task (RDM) but not a more complicated decision-making task (CDI).

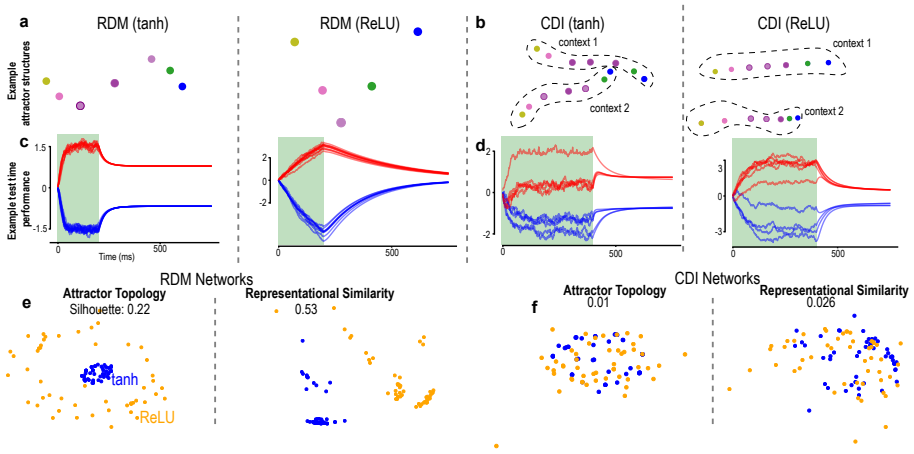

Figure 9: **Effect of Activation Function on Dynamical Mechanism. (a)** Topology of attractors for RNNs trained on RDM task with $tanh$ (left) and ReLU (right) activation functions. **(b)** Attractors for RNNs trained on the CDI task are also depicted with a $tanh$ (left) and ReLU (right) activation function. **(c, d)** Network outputs during pulsed experiments reveals different activation functions lead to different integration mechanisms on the RDM task but not the more complex CDI task. **(e, f)** Attractor topologies and representational geometries form distinct clusters based on activation function (attractor sillhouette score: 0.22, representational geometry sillhouette score: 0.53) for the RDM task. However, RNNs trained on the CDI task have overlapping attractor structure (sillhouette score: 0.01) and representaions (sillhouette score: 0.026).

Figure 10: **Representational similarities between RNNs before and after training on the RDM task.** For all four learning rules, we observe that there is a signficiant degree of clustering between trained (yellow) and untrained (blue) RNNs. In particular, trained RNNs have distinct representations from untrained RNNs as evidenced by a silhouette score greater than 0.

SVCCA of untrained RNNs

As discussed by [14], canonical correlation analysis can sometimes suggest that trained and untrained RNNs have more similar representations than RNNs trained with different design choices. We investigated the representational similarities detected with SVCCA among RNNs before and after training on the RDM task. We found that untrained RNNs had representations that were distinct form those of trained RNNs on the RDM task across all four learning rules (Figure 10). Therefore, the RNN representations we inferred with SVCCA are influenced by the training process and unlikely to be similar to each other due to initialization.

Single neuron output RNNs

One potential concern is that Hebbian trained RNNs used a single recurrent neuron as output while the other learning rules used a linear readout over all of the neurons as output. To control for this, we trained an additional 20 RNNs using both BPTT and GA learning rules to perform the RDM and CDI tasks using a single neuron as output (Figure 11). We excluded FF from this analysis because we observed training difficulties for this analysis, likely due to the additional soft constraints that FF places on recurrent neurons. When comparing RNNs from these three learning rules following training, we find that our results are consistent with those originally reported in Figure 1. In particular,

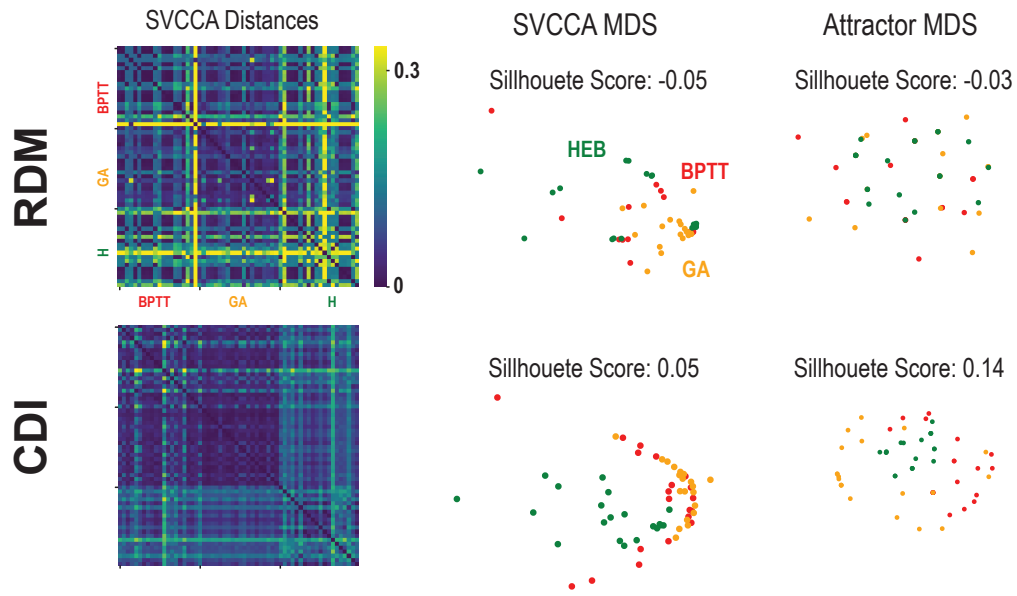

Figure 11: **Representations and attractor structures for RNNs that use a single neuron as output.** For the RDM task, we see that both the representational geometries and attractor topologies do not show any significant clustering by learning rule.

we observed that attractor topologies are largely invariant to the choice of learning rule for both the RDM and CDI tasks. Additionally, we obtained similar Silhouette scores for both representations and attractor structures.