Why do recurrent neural networks suddenly learn? Bifurcation mechanisms in neuro-inspired short-term memory tasks

Udith Haputhanthri [*1] Liam Storan [*1] Yiqi Jiang [*1] Adam Shai [2] Orhun Akengin [1] Mark J. Schnitzer [†13]
Fatih Dinc [†1] Hidenori Tanaka [†45]

Abstract

Recurrent neural networks (RNNs) are regularly studied as in silico models of biological and artificial computation. Training RNNs requires updating many synaptic weights, making the learning process complex and high-dimensional. In order to uncover learning mechanisms, we investigated the sudden accuracy jumps in RNNs' loss curves. Across several short-term memory tasks, we identified an initial search phase with accuracy plateaus, followed by rapid acquisition of skills. Studying attractor landscapes during learning revealed high-dimensional bifurcations as the links between these phases. Next, we introduced the temporal consistency regularization (TCR), a biologically plausible learning rule that incentivizes formation of memory-subserving attractors. In diverse short-term memory tasks, TCR accelerated (online) training, promoted robust attractors, and enabled networks initialized in a chaotic regime to train efficiently. Our analyses lead to testable predictions for system neuroscientists and highlight the need to study high-dimensional dynamical system theory to uncover learning mechanisms in biological and artificial networks.

1. Introduction

Recurrent neural networks (RNNs) play a crucial role in both biological and artificial neural systems, enabling complex associative computations and memory formation [1, 2, 3, 4, 5, 6]. To study their computational principles, dynamical systems theory has emerged as an indispensable tool [7, 8, 9, 10, 11, 12], allowing a large body of neuroscience work to utilize RNNs as trainable dynamical systems and study attractor formation in neural circuits [2, 4, 5, 13, 14, 15, 16, 17, 18, 19, 20, 21]. However, given the high-dimensional and often chaotic nature of brain dynamics [22, 23], the dynamical system modeling of biological RNNs is often a challenging task [24, 25] (though see [26, 27, 28] for promising developments). Therefore, systematic and carefully designed studies are needed to extract the fundamental principles of learning processes and computational mechanisms.

A recent promising direction has focused on modeling nonlinear dynamical systems via interpretable, mathematically tractable RNN architectures [29, 30, 31, 32]. This approach primarily relies on the assumption that many real-world tasks can be performed with relatively simple attractor landscapes [7, 8] (though also see [33]), which can often be visualized to explain the substrate of computations in RNNs [34, 35, 36]. Yet, one of the primary challenges in training RNNs lies in the fact that weight initialization often fails to place the network in a weight subspace with the desired attractor landscapes. Consequently, the learning process must navigate through bifurcations [37, 38], which are critical yet underexplored events characterized by qualitative changes in the network's dynamics. These bifurcations often manifest as spikes or jumps in learning curves [37] and are essential for the network to converge to functional solutions. To date the harmful effects of bifurcations, such as exploding gradients, have received considerable attention [39, 40, 41, 42]. However, the constructive role of bifurcations in guiding high-dimensional RNNs to train effectively in a general setting remains largely unexplored.

In this work, we conduct a thorough scientific investigation on the sudden learning phenomenon observed in RNNs and identify the crucial and constructive role of necessary bifurcations in learning dynamics. We identify two distinct regimes emerging during the training of RNNs, search and comprehension. The transitions between these phases, *i.e.*, bifurcations, emerge as sudden changes in the loss functions and are characterized by changes in the distinct qualitative properties of attractor landscapes, which are the building blocks of computation in short-term memory tasks. Inspired by these observations, we propose the temporal consistency regularization (TCR), a spatiotemporally local mechanism incentivizing attractor formation. We find that

[*]Equal contribution †Supervisors. [1]Stanford University, Stanford, CA, USA [2]Principles of Intelligent Behavior in Biological and Social Systems, Berkeley, CA, USA [3]Howard Hughes Medical Institute, Stanford, CA, USA [4]Harvard University, Cambridge, MA, USA [5]NTT Research, Sunnyvale, CA, USA. Correspondence to: Hidenori Tanaka <hidenori_tanaka@fas.harvard.edu>.

Mechanistic Interpretability Workshop 2024, Vienna, Austria. Copyright 2024 by the author(s).

TCR accelerates training by shortening the search phase, and can facilitate online learning of cue-associated fixed-points. Surprisingly, TCR enables training of chaotic networks, a feat that was considered rather infeasible with the back-propagation through time algorithm and led to the birth of new paradigms such as reservoir computing [43, 44] and FORCE [26, 27].

Though prior work studied the bifurcation mechanisms during learning [37, 45] (though still in the context of how to avoid them [37, 46]) our analysis extends beyond the limited architectures and specific bifurcations studied by earlier studies and offers a more nuanced interpretation, *e.g.*, bifurcations are not simple inconveniences that need to be mitigated. Overall, our work constitutes a rigorous systematic study of learning and computation in high-dimensional dynamical systems, connects formations of attractors to computation in artificial neural networks, and has direct implications for learning in biological networks and explaining skill acquisition in artificial networks.

2. Results

2.1. Training recurrent neural networks on a short-term memory task

To illustrate the emergence of bifurcations during network training with a simple example, we start our analysis with the piece-wise linear recurrent neural networks (PLRNNs):

$$x[t] = Ax[t-1] + W\phi(x[t-1]) + Cs[t] + h, \quad (1)$$

where $x[t] \in \mathbb{R}^N$ are the activities of N neurons, $r[t] = \phi(x[t])$ their firing rates with the $\phi(.)$ non-linearity, $A \in \mathbb{R}^{N\times N}$ a diagonal matrix of neuronal decay rates, $W \in \mathbb{R}^{N\times N}$ the recurrent connectivity matrix, $s[t] \in \mathbb{R}^K$ is the external input, $C \in \mathbb{R}^{N\times K}$ input weights, and $h \in \mathbb{R}^N$ the biases. Here, piece-wise linear specifically refers to the RELU non-linearity, *i.e.*, $\phi(z) = \max(0, z)$.

The delayed addition task is a simple short-term memory task that produces a sudden jump in the accuracy during training (Fig. 1, Appendix S1.2). In the delayed addition task, there are two channels providing inputs to the network (Fig. 1**A**). The first channel, $u_1(t)$, contains continuously valued "cues", whereas the second channel is mainly zero ($u_2(t) = 0$), except for two pulses at distinct times ($u_2(t_1) = u_2(t_2) = 1$). The output of the task corresponds to the the sum of the two cues at times t_1 and t_2, *i.e.*, $\hat{o}(t) = u_1(t_1) + u_1(t_2)$, which should be returned at the end of the trial $t = T$. Notably, in order to perform the task accurately, the networks should be able to hold two numbers in short-term memory.

When we trained PLRNNs to perform this task, we observed two phases of learning, namely, the search phase and the rapid comprehension phase (Fig. 1**B**). During the search phase, the network did not show a significant performance improvement in either the training or the test sets. Yet, at epoch 120, the network entered the rapid comprehension phase, in which the training performance abruptly improved. Our goal for the rest of this work is to understand and facilitate the mechanisms behind such abrupt transitions.

2.2. Bifurcations emerge during the learning

To study the mechanism of abrupt changes during training, we turn to the dynamical system theory, which establishes that the parameter space is divided into several subspaces with qualitatively distinct properties [36]. Going from one subspace to another requires the system to go through qualitative changes in its attractor landscape, *i.e.*, bifurcations. Traditionally, bifurcations are studied in the context of low-dimensional dynamical systems, often as the few number of system parameters is varied, not trained [36]. In contrast, in recurrent neural networks, the high-dimensional weight parameters, $W \in \mathbb{R}^{N\times N}$, establish the attractor landscapes, which in turn define the breadth of computations that can be carried out by the state variables, $x(t) \in \mathbb{R}^N$.

To operationalize the identification of attractor landscapes supported by a particular weight matrix, W, we utilized a slightly modified version of the energy minimization approach [8]:

$$x^* \in \arg\min_{x\in\mathcal{N}(x_0)} E(x) = \arg\min_{x\in\mathcal{N}(x_0)} \left|\left|\frac{\partial x[t]}{\partial t}\right|\right|_2^2, \quad (2)$$

in which the "kinetic energy" of neural trajectories is minimized in the neighborhoods, $\{\mathcal{N}(x_0)\}$, of few pre-defined state variables, $\{x_0\}$, to identify states with locally minimal speeds (x^*, Figure S1). In this work, we sample x_0 from transient neural activities (Appendix S1.5). This architecture and attractor agnostic approach is well-suited for identifying not only fixed points and the centers of limit cycles – elements of traditional bifurcations with specific names (*e.g.*, Hopf bifurcation) [36] – but a general class of slow points in RNNs with thousands of parameters [8].

In this work, we define "high-dimensional bifurcations" as qualitative changes in the arrangement of these slow-point landscapes. This definition includes broader sets of phenomena compared to the traditional definition of bifurcations [36]. Yet, it captures the essence of the phenomena we study in this work, since (approximate) attractors are slow points by definition and their re-arrangement qualitatively changes the network dynamics. New terminology may need to be developed in future studies of high-dimensional dynamical systems, which is out of scope here.

Using this approach, we identified the slow points of the PLRNNs and visualized their time evolution as the training progressed (Fig. 1**C**). As expected, we observed a sudden shift in the neural activities and the overall attractor structure

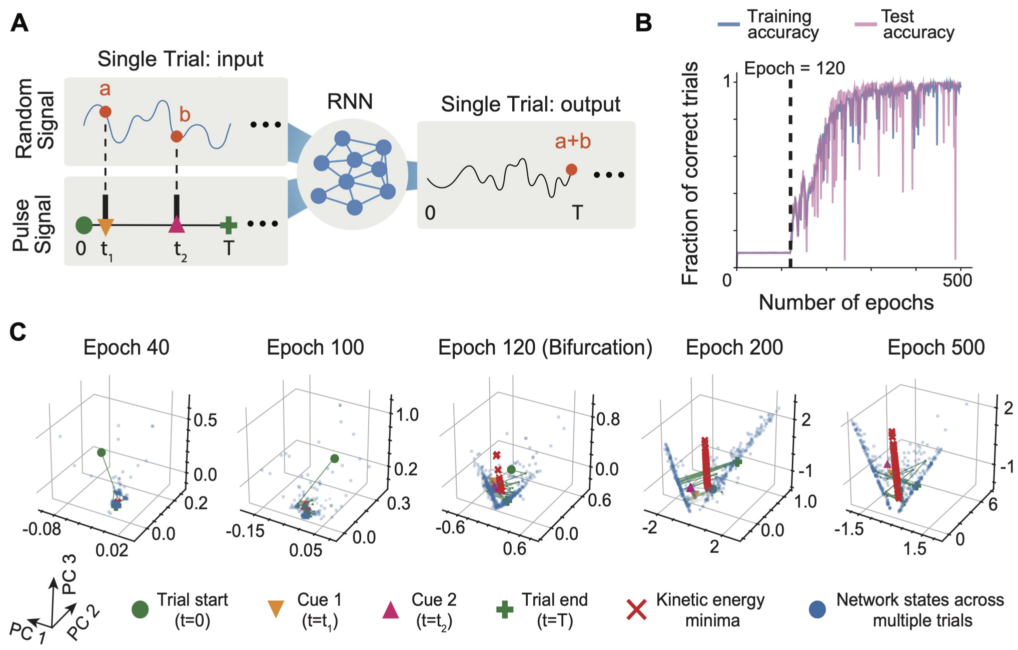


Figure 1: **Bifurcations subserve the sudden accuracy jumps during the learning of the delayed addition task.** **A** In the delayed addition task, the network receives signals from two channels: the first channel has continuously valued random signals, the second channel is zero for almost all times, but has two binary pulses at times t_1 and t_2. During these pulse times, the network needs to memorize the "cue" values presented in the first channel and must output their sum at the end of the trial ($t = T$). **B** The fraction of correct trials as a function of number of training epochs. The training and testing curves both show two phases, with a sudden transition at the epoch 120. **C** Top three principal components of the network's activations over multiple trials (blue), neural trajectories during a single trial (green), and kinetic energy minima corresponding to slow points (red). A bifurcation at epoch 120 is marked by the emergence of a new attractor structure, which causes a structural and qualitative change in the slow point landscape and separates the two phases. For **B-C**, we used a representative unregularized network with $N = 40$ neurons trained over 500 epochs to perform the delayed addition task with $T = 40$.

around epoch 120 (Fig. 1**C**), signalling a bifurcation. This shift was then followed by a rapid increase in both training and testing accuracies. Notably, the slow point landscape right after the bifurcation exhibited the same qualitative behavior as the further trained network (Fig. 1**C**, epochs 120 vs 500), suggesting that the bifurcation propelled the network into the "right" weight subspace. Though this observation alone cannot explain why the network performance suddenly started improving, as we discuss in Section 2.4, it is an important step forward.

2.3. Hidden and/or explicit bifurcations lead to irregularities in the training curves

In the previous section, we considered a network that was already initialized close to the bifurcation boundary. Consequently, the search phase was relatively brief (120 epochs). The bifurcation corresponded to a jump in the loss function and had lasting effects on training performance, resulting in a sustained decrease. We categorize such bifurcations as "explicit bifurcations." However, as we demonstrate in this section, not all bifurcations lead to sustained decreases in the loss function, *i.e.*, may be "hidden" due to the lack of sustained effects on the training curves, for example, emerging as small spikes that are quickly recovered.

To study the hidden bifurcations, we considered a second network that spent around 2500 epochs in the search phase and had a small spike in the learning curve which did not improve or hurt the network performance (Fig. 2**A**, red rectangle). Yet, it was not clear whether this was a self-correcting behavior or a necessary event towards the final solution. Therefore, as a first test, we computed the singular value decomposition of the instantaneous gradient[1] and the current weight matrix, $W(t)$, at the training epoch t. As expected, although there was no visible change in the network performance after recovering from the spike (Fig. 2**A**, red rectangle), the instantaneous gradient became unstable, exhibiting sudden changes (Fig. 2**B**, red rectangle). Surprisingly, the weight matrix had rapid, but continuous and sustained, changes (Fig. 2**C**, red rectangle); ruling out a self-recovery explanation.

Next, we verified that the spike in the training curve indeed was linked to a *hidden* bifurcation by studying the qualitative changes in the slow point landscape (Fig. 2**D**, Epoch 800). Interestingly, even though this hidden bifurcation

[1]Throughout this work, we use the word "instantaneous gradient" to refer to the quantity $W(t) - W(t-1)$, which may be different from the exact gradient as we use the ADAM optimizer.

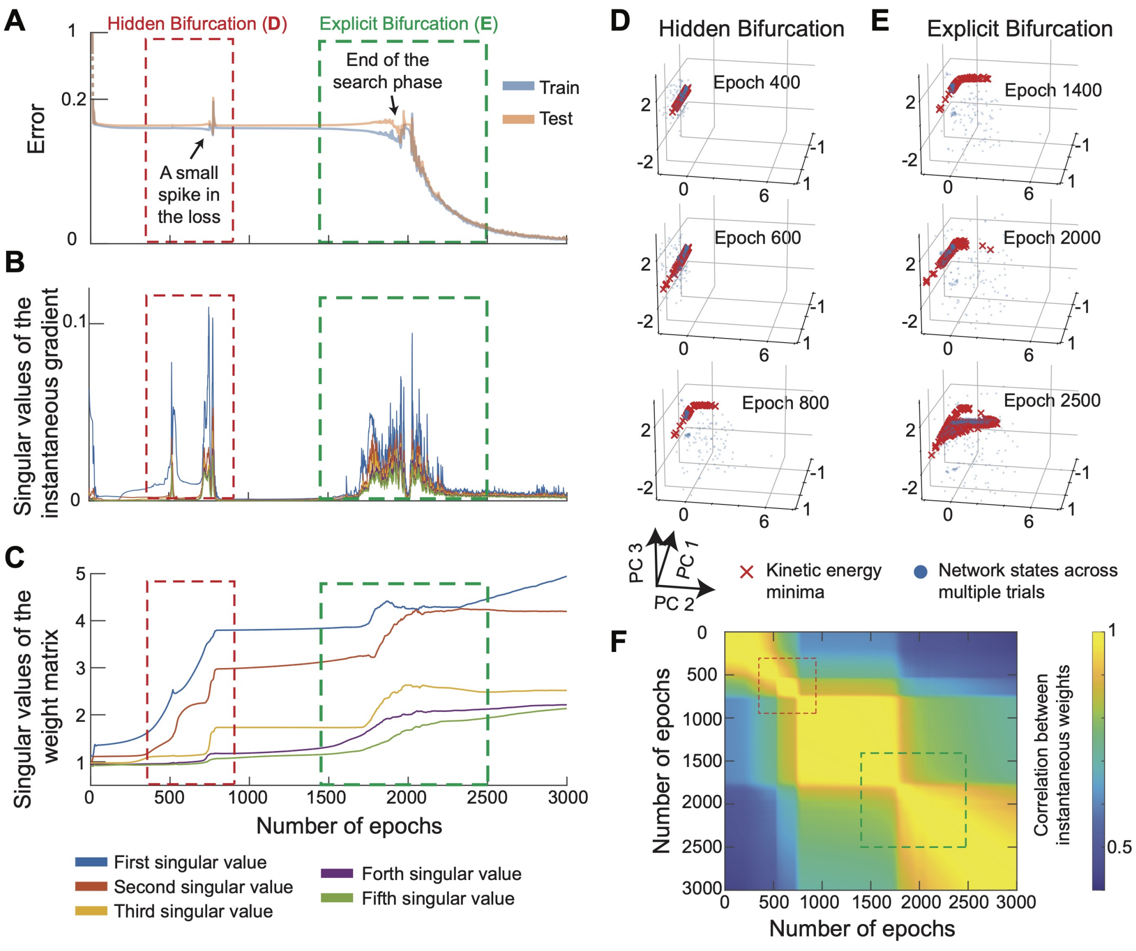


Figure 2: **Hidden and explicit bifurcations drive weight restructuring and attractor formation during learning. A** Small spikes during the search phase may indicate a hidden bifurcation, followed by a sharp decrease in train/test loss post-search phase. **B,C** Instability in the singular values of the gradient (**B**) and the weight matrix (**C**) is an indicator of the bifurcation process. **D,E** New attractor geometries form in activity space after hidden (Epoch 800) and explicit (Epoch 2500) bifurcations. **F** Correlation between weights at two distinct time points during training shows high correlation until bifurcations, indicating significant restructuring of the weight matrix during the hidden and explicit bifurcations. For all panels, we used an unregularized network ($N = 40$) performing the delayed addition task with $T = 40$.

did not immediately improve the performance, it served as a step towards finding the desired attractor structure (See green rectangles in Figs. 2**A-C**, and Fig. 2**E**), and *not as a detour that needed to be traced back*. Specifically, in this case, the bifurcation at epoch 800 marked the emergence of a second "arm" of slow points (Fig. 2**D**) that subsequently bifurcated into a plane-like structure (Fig. 2**E**). Common to both bifurcations were the sudden emergence of instabilities in the instantaneous gradient and the continual changes in the weight matrices (Figs. 2**B, C**), but the former was hidden in the sense that no immediate sustained changes were visible in the training curve (Fig. 2**A**). When we visualized the changes in the weight matrix throughout the training (Fig. 2**F**), we observed the emergence of a block diagonal structure: The network experienced minimal changes in weight values between bifurcations, but instantaneous adjustments during bifurcation events. After the final bifurcation was reached, the weights were gradually (but still rapidly) updated, corresponding to the jump in the loss values.

Thus far, we have demonstrated on two illustrative examples (Figs. 1 and 2) that passing through the bifurcations may be reflected by the sudden drops in the training curves, with a transition enabled by a qualitative shift in the networks' attractor landscapes. This is signalled by the rapid fluctuations of the gradient and sudden changes in the weight matrix (Figs. 2**B, C**). To generalize beyond these examples, we confirmed the sustained changes in the weight matrices and the rapid fluctuations in the gradients by averaging over several networks (Fig. S2). We also visually confirmed the qualitative changes in the slow point landscapes during the loss jumps for multiple networks (data not shown).

Overall, our analysis in this section provided a more nuanced view of our discussion in Fig. 1, where the search phase had concluded swiftly. Specifically, a network initialized far from the desired bifurcation boundary can undergo

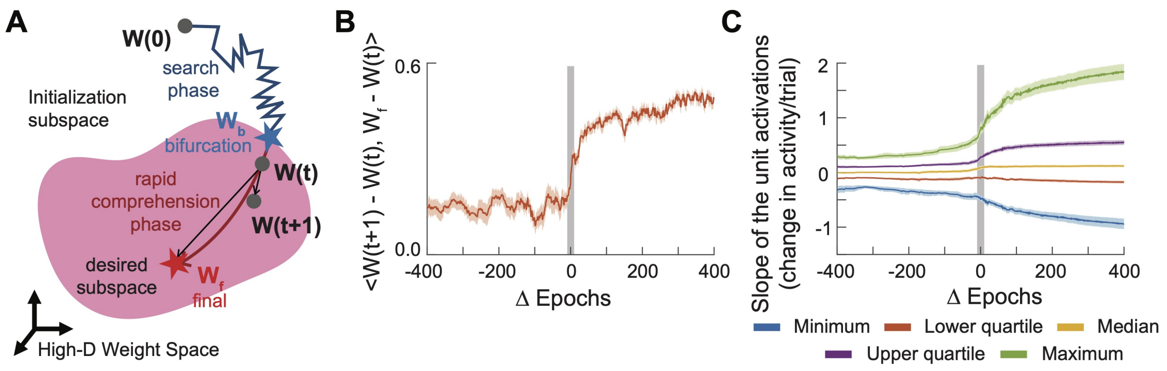


Figure 3: **Bifurcations are gateways between weight subspaces with distinct attractor landscapes.** **A** Schematic of the training process as a dynamical trajectory in the weight space. The training passes through several weight subspaces separated by bifurcations and settles in one with a desirable attractor landscape. **B** We computed the cosine similarity between the instantaneous update and the optimal update directions, which is used to measure the quality of the learning signal at a given epoch. The curve shows the better alignment of the model weight updates towards the final weight matrix after the bifurcation. **C** The slope of the neural activities, calculated as the change between the activities at the end and the start of the trials, as a function of training epochs. During the bifurcation, the fraction of ramping units with high absolute slopes increased dramatically. In these plots, $\Delta\text{Epochs} = 0$ corresponds to the final bifurcation, where the search phase concludes and the comprehension phase begins. For **B** and **C**, we trained 20 networks on the delayed addition task with $N = 40$ and $T = 40$ for 3000 epochs, all but one successfully solved the task. Lines: means. Shaded regions: s.e.m. over 19 networks.

intermediary hidden bifurcations (Fig. 2), traversing the solution landscape until the desired attractor landscape is reached. Although the loss function may appear flat, the gradient can experience regions of instability, signaling qualitative changes in the slow point landscape and indicating bifurcations. Notably, after bifurcations, the weight matrices did not revert back to their original states; rather, the networks progressed through the bifurcations.

2.4. Characterizing learning dynamics in search and comprehension phases

The training process of the PLRNNs often starts from a randomly initialized weight subspace, which does not necessarily include the "right" attractors that can facilitate solving the task at hand. During training, the weights are updated such that the RNN bifurcates into a particular weight subspace with desired attractor landscapes (Fig. 3**A**). In this section, we return to investigate the differences in the learning speeds between the search and comprehension phases (Fig. 1**B**) by studying a key change during bifurcations: the increased quality of the learning signal.

To assess the quality of the gradient-based learning signals, we analyzed the weight changes in the PLRNNs at each training step. Specifically, we measured the alignment of the instantaneous weight change, $W(t+1) - W(t)$, with the optimal learning direction, $W_f - W(t)$, where W_f represents the final weights of the fully trained network (Fig. 3**A**). We computed this alignment, quantified via the cosine similarity, over twenty networks and plotted as a function of number of training epochs in Fig. 3**B**. Notably, though the alignment was small earlier in the training, it was above chance level across the both training phases (data not shown). Thus, as a first step, we found that the learning signal from the ADAM-based gradient descent was beneficial in both phases.

Next, since the gradient alignment remained low before the bifurcation, we inferred that the weight updates were suboptimal for driving the network through the beneficial bifurcations into the desired weight subspace (Fig. 3**A** and **B**). In contrast, the learning signal quality was much higher in the comprehension phase (Fig. 3**B**), implying that gradient is effective for fine-tuning the task-subserving structures once a desired attractor landscape is reached (Fig. 3**B**). Taken together, the facts that gradient is an effective fine-tuner, but not necessarily a good "attractor finder," ties back to our question at the end of Section 2.1 and provides an explanation for the different training speeds observed in search vs rapid comprehension phases.

2.5. Memory neurons with slowly varying activities shorten the search phase

As the gradient remained suboptimal during the search phase, we next investigated what factors played a role in the learning efficiency. Specifically, we searched for the factors that predicted the onset of the beneficial bifurcations.

The most consistent and effective predictor we observed was the emergence of ramping "memory neurons" (Fig. 3**C**). This observation aligned with our expectations, as neurons with steadily increasing or decreasing activity have been known to be associated with short-term memory processes and time-keeping [30, 47, 48]. Our goal, then, is to design a learning algorithm that incentivizes the emergence of these neurons, thereby transforming these correlational observations (Fig. 3**C**) into a causal relationship.

Towards this goal, we first investigated the emergence of

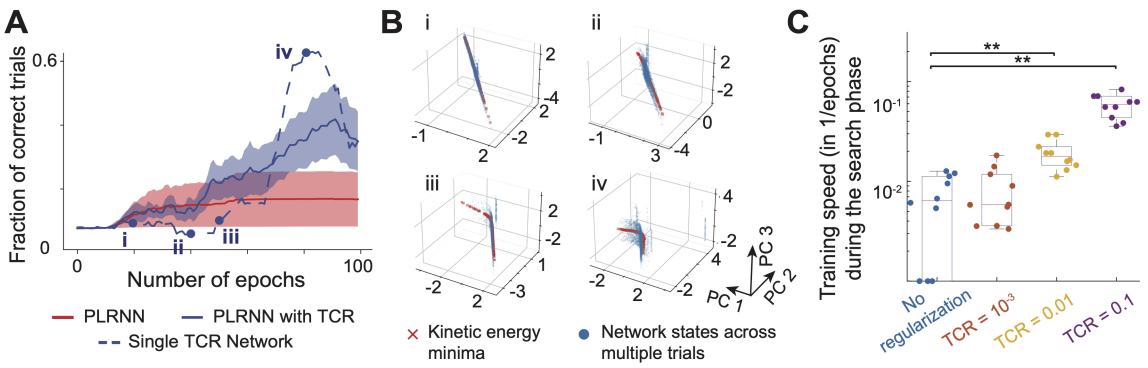


Figure 4: **Enforcing time consistency in memory neurons promotes attractor formation.** We trained 10 PLRNNs on the delayed addition task with $T = 40$, where $N_{\text{reg}} = 20$ out of $N = 40$ neurons were regularized with $\lambda_{\text{TCR}} = 1$. **A** TCR increased the number of networks passing the search phase. Dotted line: an example network. Solid lines: means. Shaded areas: s.e.m. over 10 networks. **B** We plotted the attractor landscape of an example network, corresponding to the (i) - (iv) stages in **A**. The network undergoes a bifurcation facilitated by TCR, in which a second arm is created on the slow-point landscape (red crosses). **C** Increasing TCR regularization speeds up the training during the search phase. The comparisons were performed with two-sided Wilcoxon signed-rank tests ($^{**}p < 10^{-2}$).

slowly varying ramping units in greater detail by studying a recently proposed manifold attractor regularization (MAR). Although MAR has been developed specifically for PLRNNs and incentivizes only a certain type of attractor structure [30], we were able to extract generalizable insights (Appendix S1.3 for methodological details). Specifically, designing a plane attractor (by hand) in a subset of units abolished the search phase for the delayed addition task (Fig. S3), in which networks may use plane attractors for keeping the memory of the two cues. The units in this subset, termed the "memory neurons" (the rest termed the "computation neurons"), had the same ramping properties we observed in Fig. 3**C** (Fig. S4). As expected, explicitly enforcing the presence of such units enhanced the memory of cues in PLRNNs (Fig. S5) and promoted faster training by selecting a closer bifurcation point to the initial weights (Fig. S6). Consequently, we were able to establish a causal *and* explainable relationship between the slowly varying ramping units and the enhanced memory capabilities in PLRNNs, confirming and expanding the prior literature with a mechanistic explanation [30]. Though prior work has shown that slow-units arise due to MAR and lead to better training, our analysis here illuminates the reason why: Enforcing slow units accelerate the search phase by incentivizing faster bifurcations towards the desired attractor landscapes.

In addition to memory neurons, we tested different values of the weight decay and the training batch sizes as potential predictors of shorter search phases. Orders of magnitude variations in weight decay had little-to-no effects on the duration of the search phase (Fig. S7), yet suboptimal weight decay values led to decreased final performances of the learned networks (Fig. S7). Therefore, even though it is an important part of the optimization process, weight decay did not provide an actionable insight to further probe the learning signal quality in the search phase. In contrast, when we trained PLRNNs with different numbers of batch sizes, we observed a clear trend of decreased search phase with lower batch sizes, *i.e.*, increased stochasticity (Fig. S8). Yet, perturbing the synaptic weights randomly before the bifurcation without the aid of the gradient did not lead to the beneficial bifurcations (data not shown). Thus, bringing these observations together, we concluded that introducing greater variability to the training process, *e.g.*, addition of a secondary objective instead of an absolute minimization of the task loss, may facilitate faster transitions to the desired bifurcation boundaries.

2.6. Temporal consistency incentivizes rapid formation of robust attractors

So far, we established that the emergence of slow units increases the memory capabilities, which can be explicitly enforced to improve training efficiency in PLRNNs performing the delayed addition task (Fig. S6, [30]). Inspired by this observation, we now propose the temporal consistency regularization (TCR), a (potentially) biologically plausible mechanism that is agnostic to the network architecture and the specific properties of the desired attractor landscapes:

$$\mathcal{L}_{\text{TCR}} = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N_{\text{reg}}} (x_i[t] - x_i[t-1])^2 \tag{3}$$

In words, TCR encourages slow time dynamics for a subset (N_{reg} out of N) of the neurons, which we term as "memory neurons." As we show in the rest of this section, regularizing the task training with TCR, *i.e.*, $\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{TCR}}\mathcal{L}_{\text{TCR}}$, incentivizes attractor formation and leads to *superior* and *robust* short-term memory capabilities.

As a first step, we once again considered the PLRNNs in Eq. (1), which were trained to perform delayed addition

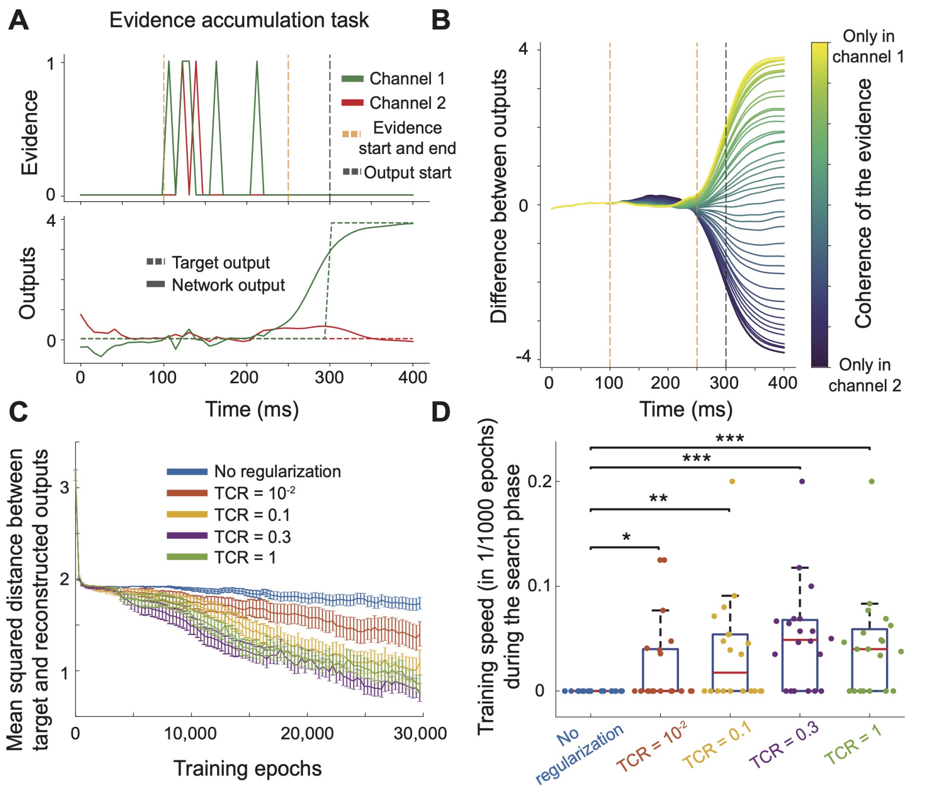


Figure 5: **Enforcing temporal consistency allows training chaotically initialized lfRNNs to perform an evidence accumulation task. A** An illustration of the evidence accumulation task. **B** The differences between network outputs are modulated based on the coherence of the evidence. **C-D** We trained chaotically initialized lfRNNs ($N = 50$, $N_{\text{reg}} = 25$, $\tau = 10ms$, $\Delta t = 8ms$, and $g = 4$) for $30,000$ epochs to solve the evidence accumulation task. **C** The evolution of the mean squared error between the (scaled) coherence of trials and the differences in output channels plotted as a function of training epochs. Solid lines: means. Error bars: s.e.m. over 20 networks. **D** The training speed during the search phase as a function of temporal consistency regularization strength. The comparisons were performed with two-sided Wilcoxon signed-rank tests ($^{*}p < 0.05$,$^{**}p < 10^{-2}$, and $^{***}p < 10^{-3}$).

tasks with $T = 40$ (Fig. 4). Compared to unregularized networks, training PLRNNs with TCR led to shortened search phase and faster convergence (Fig. 4**A**). Using the energy minimization analysis (following Eq. (2)), we confirmed that the faster convergence was indeed due to the emergence of attractor structures (Fig. 4**B**). Moreover, notably, higher values of λ_{TCR} resulted in faster training during the search phase (Fig. 4**C** and Fig. S9), signalling the direct involvement of TCR in shortening the search phase.

As a second step, we asked whether TCR can achieve rapid convergence on par with enforcing optimal solutions. Given that MAI/MAR establishes a plane attractor in the network (Appendix S1.3), a feasible solution for the delayed addition task, we can consider the search phase durations of networks with MAR as approximate upper bounds. As expected, the convergence speeds with TCR were generally slower; however, the difference decreased with increasing values of λ_{TCR} (Fig. S9). Notably, following a grid search over hyperparameters, we found that 6 out of 10 networks could complete the search phase in fewer than 100 epochs of training (Fig. S10), indicating that TCR may perform almost as well as MAR when optimized (Fig. S9).

As the final step, to test the robustness of the attractors incentivized by TCR, we trained PLRNNs with and without TCR, otherwise using the same hyperparameters, on delayed addition tasks. Then, during test trials, we injected random and constant inputs to memory and computation neurons of the regularized networks, and to the same number of randomly selected neurons of the unregularized networks (See Appendix S1.9). We plotted the changes in the performances in Fig. S11. Our results indicated that the regularized networks were robust to random noise injected into memory neurons (Fig. S11A) and constant noise injected to either types (Fig. S11B). In contrast, the performance of the unregularized networks abruptly degraded in both cases (Fig. S11). Thus, we confirmed that TCR not only leads to faster search phase, but also identifies *robust* attractor landscapes.

2.7. Rapid learning in chaotic leaky firing rate RNNs with fixed time-scales

So far, we have shown that networks trained with temporal consistency regularization can be as effective as those trained with an optimal regularization for the particular architecture and task [30]. To illustrate the network- and task-agnostic nature of TCR, we next focused on leaky firing rate RNNs (lfRNNs) [28, 49]:

$$\tau\frac{\mathrm{d}r(t)}{\mathrm{d}t} = -r(t) + \tanh(Wr(t) + Cs(t) + h). \quad (4)$$

Here, $\tau \in \mathbb{R}$ is the pre-defined, non-trainable, neuronal decay time, other components are defined similar to Eq. (1), and with $x(t) = Wr(t) + Cs(t) + h$. For our purposes, we set $h = 0$, as our tasks of interest can be trained without biases. In practice, we discretize Eq. (4) with a time step Δt (Appendix S1.1).

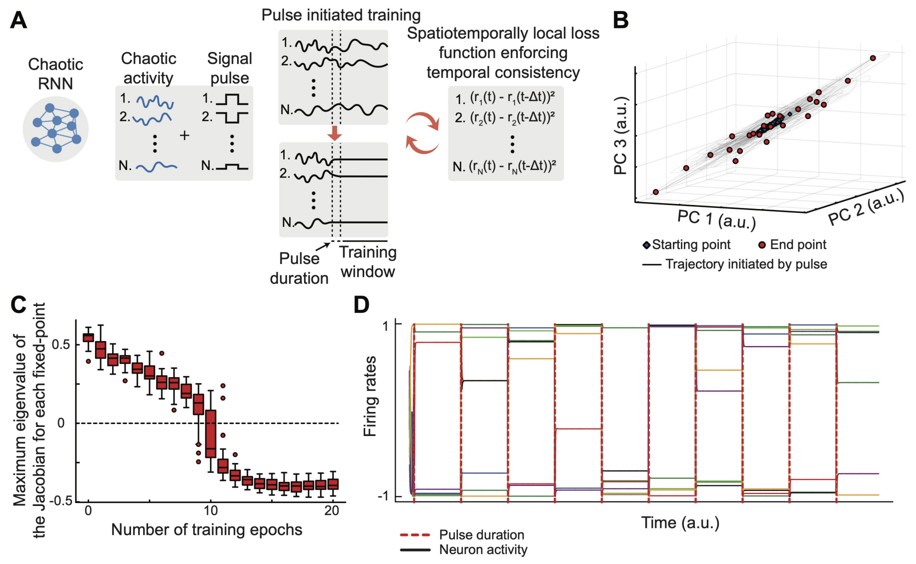


Figure 6: **Temporal consistency facilitates online learning of cue-associated fixed-points. A** The training procedure starts with a randomly initialized chaotic RNN. A rectangular pulse is applied through a randomly initialized, non-trainable input weight matrix to the network. For a pre-defined duration after the pulse offset, the freeze-in training takes place. **B** We visualized the final output of a fully trained network with 25 distinct, cue-associated fixed-point attractors. **C** The maximum eigenvalues of the Jacobian matrices became non-positive (attractive fixed-points) after as little as 12 epochs of training. **D** We visualized 10 out of 25 cue-associated fixed-points. Pulses are shown during a very short duration (red dotted lines), and drive the transitions between fixed-points.

Next, we trained lfRNNs to perform the evidence accumulation tasks (Figs. 5 and S12). In the evidence accumulation task, there are two input and two output channels to the network (Fig. 5**A**). While the pulses in the input channels are sampled from a binomial distribution for each time point, the network is tasked to indicate which input channel received more pulses, *i.e.*, evidence, by outputting a pulse in the respective output channel. Higher coherence of the evidence, *i.e.*, when higher fraction of the total evidence was presented predominantly in one of the channels, corresponds to an easier task (Fig. 5**B**).

As a first test, we confirmed that even unregularized lfRNNs can solve this task when initialized with Xavier initialization, though TCR still increased the speed with which the search phase was concluded (Fig. S12). Surprisingly, TCR showed its true value when lfRNNs were initialized in a chaotic regime (controlled by a parameter, g, see Appendix S1.1), where recurrent connections are notoriously hard to train [26, 27, 43, 44]. Specifically, even after 30,000 epochs of training, none of the 20 unregularized networks were able to undergo the right bifurcations (Fig. 5**C**). For these networks, the errors plateaued on a particular value, similar to the loss curve plateaus of the delayed addition task (Figs. 1 and 2). This was only overcome by networks trained with TCR (applied to $N_{\text{reg}} = 25$ out of $N = 50$ neurons), where 14 out of 20 networks bifurcated and were able to solve the task. Moreover, increasing the TCR strength, λ_{TCR}, increased the training speed (until saturated) during the search phase (Fig. 5**D**), similar to the delayed addition task (Fig. 4**C**).

Finally, we provided additional evidence for the generality of our findings so far with two experiments: i) we trained long short-term memory (LSTM) networks on the delayed addition tasks (Fig. S13), and ii) we trained chaotically initialized lfRNNs on the 3-bit flip flop tasks (Fig. S14). In both cases, in line with our results so far, TCR led to faster training during the search phases.

2.8. Cue-associated fixed points with online freeze-in learning

In essence, TCR forces the derivative of network activities towards zero for a subset of neurons. Discretization of this term leads to an error signal that is simply the difference between current and previous neural activities, which can be applied as a local and online rule (Appendix S1.11):

$$\Delta W[t] = -\lambda_{\text{TCR}} \left(r[t] - r[t-\Delta t]\right) r[t-\Delta t]^T, \qquad (5)$$

where $\Delta W[t]$ stands for the changes in the recurrent weights. We applied this rule to learn cue-associated fixed-points, which arose rapidly through online training starting with a randomly connected chaotic lfRNNs (Fig. 6A). Since online learning freezes the network in time, but only locally, we named this paradigm *freeze-in training*.

Using the freeze-in training, we trained 25 distinct fixed-point attractors, each of which was associated with a cue (Fig. 6B). Notably, training converged quickly, leading to an attractive dynamics around the fixed-points (Appendix S1.11, Fig. 6C). Though the attractive fixed-points were achievable by cue inputs, their existence was not dependent on them since the network was trained to remain in the fixed-points *long after* the cue offset (Fig. 6D). Here, the cues were used only to create the associations.

Overall, using an online version of temporal consistency, we were able to train cue-associated fixed-point attractors into the network without making use of any global learning signals or explicit supervision.

3. Discussion

In this work, we performed high-dimensional bifurcation analyses of RNNs during learning and introduced the temporal consistency as a biologically plausible and local learning rule. These findings lead to several biological and computational insights that we summarize below.

Analysis of learning mechanisms in task-trained RNNs: In this work, we studied the bifurcation mechanisms as RNNs were learning to perform short-term memory tasks. Prior work performed similar analyses in dynamical system reconstruction problems by primarily focusing on specific attractor structures [30, 37]. Here, we studied short-term memory tasks that had sparse learning signals due to the delay period with little-to-no meaningful input or output signals. In our studies, we had not assigned any particular attractor structure or architectures to the networks. Instead, our methodology allowed the attractor landscapes to be found during the task-training, which we reverse-engineered to extract mechanistic insights on the emergence of attractors subserving short-term memory.

Use of energy minimization for detecting bifurcations: Our goal in this work has been to study bifurcations in a generalized setting, which stands in contrast to previous work that used semi-analytical methods to study the (traditional, *e.g.*, fixed-points and k-cycles) bifurcations in PLRNNs [37]. To study bifurcations in high-dimensional systems, we operationalized slow-point analysis (Eq. (2)) to study the qualitative changes in the kinetic energy landscape, a well-suited method for exploring a broad range of bifurcations in RNNs. With the help of this methodology, we were able to identify the bifurcations in the delayed addition task (Figs. 1 and 2), which did not manifest through changes in the number of fixed points or limit cycles [36, 37, 45]. Therefore, our findings illustrate the importance of studying a broad class of *high-dimensional* bifurcations to reverse-engineer the learning dynamics in RNNs.

New mechanistic insights into the learning signal quality: Our findings provided evidence for the common sense assumption, a generalization of intuition gained from studying low-dimensional dynamical systems [36], that the weight space consists of multiple subspaces with different geometries of the attractor structure (Fig. 3). We found that gradient updates drive the network towards the optimal weights rapidly only if the network dynamics are already in the right subspace with desired attractor structures, but are suboptimal as learning signals otherwise. To enter the optimal attractor subspace, the network might have to successfully traverse one or more subspaces with very little, yet non-zero, help from the gradient signals. Thus, the solution to increasing the training speed in large scale foundational models may lie, beyond developing new optimizers [50], in designing loss functions whose gradients incentivize beneficial bifurcations and are optimal during the search phase.

Introduction of the temporal consistency regularization: In this work, we assigned a subset of neurons the role of "memory neurons," which were encouraged to have slow-time dynamics (Fig. 4). The resulting temporal consistency regularization allowed training leaky firing rate RNNs, which do not have a RELU non-linearity or trainable neuronal time scales, under even chaotic initialization and potentially in an online manner (Figs. 5 and 6). Unlike prior solutions to training RNNs despite chaos [26], temporal consistency regularization is compatible with back-propagation through time and allowed using established algorithms (here, ADAM optimizer [50]) for training RNNs.

Implications for learning in biological systems: Our work may explain the mechanism of a widely observed phenomenon in systems neuroscience: slow-time dynamics and ramping neural activities play a vital role in memory and associative computations [30, 47, 48]. As we have shown in this work, a plausible explanation for this phenomenon may be that directly encouraging slow-time dynamics in biological networks can promote attractor formation, thereby shortening the search phase via beneficial bifurcations. Moreover, the temporal consistency may carry long-term dependencies in an online scenario (Fig. 6), without requiring explicit calculation of the gradients of a global loss function, *e.g.*, via backpropagation through time, which often fails in chaotic networks to begin with [43, 44]. Consequently, a testable biological prediction of our work is the putative existence of a very specific form of short-term synaptic plasticity mechanism that implements TCR biologically.

4. Conclusion

In this work, we studied learning dynamics, bifurcations, and attractor formation in recurrent networks. We showed how different regions of weight space correspond to different attractor landscapes, and uncovered that gradients alone often struggle to find the "right" weight subspaces with desired attractors. We re-analyzed manifold attractor regularization proposed specifically for PLRNNs [30] and discovered that enforcing slow-time dynamics explicitly may induce attractor formation, giving rise to enhanced short-term memory capabilities. Inspired by these observations, we proposed *temporal consistency regularization*, an attractor structure-agnostic, architecture-agnostic, and biologically plausible learning mechanism for incentivizing attractor formation. We found that TCR led to faster search phase by driving beneficial bifurcations, and allowed training chaotic networks in offline and online scenarios. Overall, our work is a significant leap for understanding the learning mechanisms in biological and artificial networks, emphasizing the importance of studying currently underexplored topics in high-dimensional dynamical system theory.

Acknowledgements

We thank Dr. Itamar Landau, Dr. Surya Ganguli, Dr. Nina Miolane, Mert Yuksekgonul, and Abby Bertics for their helpful feedback on the manuscript and valuable interactions throughout the project. MJS gratefully acknowledges funding from the Simons Collaboration on the Global Brain, the Vannevar Bush Faculty Fellowship Program of the U.S. Department of Defense, and Howard Hughes Medical Institute. FD receives funding from Stanford University's Mind, Brain, Computation and Technology program, which is supported by the Stanford Wu Tsai Neuroscience Institute. FD expresses gratitude for the valuable mentorship he received at PHI Lab during his internship at NTT Research.

Contribution statement

FD proposed and designed the project under the supervision of HT and MJS. UH, LS, YJ, and FD performed the experiments, whereas AS and OA wrote the code for the flip flop and evidence accumulation tasks, respectively. All authors contributed to the writing of the manuscript.

Code and data availability

The reproduction code for the experiments performed in this work will be made public upon publication in an archival venue. Until then, they are available in a private repository and can be shared upon reasonable request.

Limitations

We note that implementing TCR required using PyTorch's *RNNCell* module, which is significantly slower compared to the *RNN* module. Thus, training speed is significantly suboptimal without targeted development of modules that allow direct access to internal RNN states.

References

[1] Daniel Durstewitz, Georgia Koppe, and Max Ingo Thurm. Reconstructing computational dynamics from neural measurements with recurrent neural networks. *bioRxiv*, 2022.

[2] Matthew G Perich, Charlotte Arlt, Sofia Soares, Megan E Young, Clayton P Mosher, Juri Minxha, Eugene Carter, Ueli Rutishauser, Peter H Rudebeck, Christopher D Harvey, et al. Inferring brain-wide interactions using data-constrained recurrent neural network models. *bioRxiv*, pages 2020–12, 2021.

[3] David Sussillo, Mark M Churchland, Matthew T Kaufman, and Krishna V Shenoy. A neural network that finds a naturalistic solution for the production of muscle activity. *Nature neuroscience*, 18(7):1025–1033, 2015.

[4] Adrian Valente, Jonathan W Pillow, and Srdjan Ostojic. Extracting computational mechanisms from neural data using low-rank rnns. *Advances in Neural Information Processing Systems*, 35:24072–24086, 2022.

[5] Arseny Finkelstein, Lorenzo Fontolan, Michael N Economo, Nuo Li, Sandro Romani, and Karel Svoboda. Attractor dynamics gate cortical information flow during decision-making. *Nature Neuroscience*, 24(6):843–850, 2021.

[6] Christopher Langdon, Mikhail Genkin, and Tatiana A Engel. A unifying perspective on neural manifolds and circuits for cognition. *Nature Reviews Neuroscience*, pages 1–15, 2023.

[7] Niru Maheswaranathan, Alex Williams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics, 2019.

[8] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.

[9] Matthew G Perich and Kanaka Rajan. Rethinking brain-wide interactions through multi-region 'network of networks' models. *Current opinion in neurobiology*, 65:146–151, 2020.

[10] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, March 2016.

[11] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[12] Brian M. de Silva, Kathleen Champion, Markus Quade, Jean-Christophe Loiseau, J. Nathan Kutz, and Steven L. Brunton. Pysindy: A python package for the sparse identification of nonlinear dynamics from data, 2020.

[13] Mikail Khona and Ila R. Fiete. Attractor and integrator networks in the brain. *Nature Reviews Neuroscience*, 23(12):744–766, Dec 2022.

[14] Daniel Levenstein, Veronica A. Alvarez, Asohan Amarasingham, Habiba Azab, Zhe S. Chen, Richard C. Gerkin, Andrea Hasenstaub, Ramakrishnan Iyer, Renaud B. Jolivet, Sarah Marzen, Joseph D. Monaco, Astrid A. Prinz, Salma Quraishi, Fidel Santamaria, Sabyasachi Shivkumar, Matthew F. Singh, Roger

Traub, Farzan Nadim, Horacio G. Rotstein, and A. David Redish. On the role of theory and modeling in neuroscience. *Journal of Neuroscience*, 43(7):1074–1088, 2023.

[15] Elia Turner and Omri Barak. The simplicity bias in multi-task rnns: Shared attractors, reuse of dynamics, and geometric representation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 25495–25507. Curran Associates, Inc., 2023.

[16] KiJung Yoon, Michael A. Buice, Caswell Barry, Robin Hayman, Neil Burgess, and Ila R. Fiete. Specific evidence of low-dimensional continuous attractor dynamics in grid cells. *Nature Neuroscience*, 16(8):1077–1084, Aug 2013.

[17] Rishidev Chaudhuri, Berk Gerçek, Biraj Pandey, Adrien Peyrache, and Ila Fiete. The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep. *Nature Neuroscience*, 22(9):1512–1520, Sep 2019.

[18] A David Redish, Adam N Elga, and David S Touretzky. A coupled attractor model of the rodent head direction system. *Network: Computation in Neural Systems*, 7(4):671, nov 1996.

[19] Kanaka Rajan, Christopher D Harvey, and David W Tank. Recurrent network models of sequence generation and memory. *Neuron*, 90(1):128–142, 2016.

[20] Lea Duncker and Maneesh Sahani. Dynamics on the manifold: Identifying computational dynamical activity from neural population recordings. *Current opinion in neurobiology*, 70:163–170, 2021.

[21] Zach Cohen, Brian DePasquale, Mikio C Aoi, and Jonathan W Pillow. Recurrent dynamics of prefrontal cortex during context-dependent decision-making. *bioRxiv*, pages 2020–11, 2020.

[22] Philippe Faure and Henri Korn. Is there chaos in the brain? i. concepts of nonlinear dynamics and methods of investigation. *Comptes Rendus de l'Académie des Sciences - Series III - Sciences de la Vie*, 324(9):773–793, 2001.

[23] Christine A Skarda and Walter J Freeman. Chaos and the new science of the brain. *Concepts in neuroscience*, 1(2):275–285, 1990.

[24] Jonas Mikhaeil, Zahra Monfared, and Daniel Durstewitz. On the difficulty of learning chaotic dynamics with rnns. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 11297–11312. Curran Associates, Inc., 2022.

[25] Y. Sato and S. Nagaya. Evolutionary algorithms that generate recurrent neural networks for learning chaos dynamics. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 144–149, 1996.

[26] David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

[27] Brian DePasquale, Christopher J Cueva, Kanaka Rajan, G Sean Escola, and LF Abbott. full-force: A target-based method for training recurrent networks. *PloS one*, 13(2):e0191527, 2018.

[28] Fatih Dinc, Adam Shai, Mark Schnitzer, and Hidenori Tanaka. Cornn: Convex optimization of recurrent neural networks for rapid inference of neural dynamics. *Advances in Neural Information Processing Systems*, 36:51273–51301, 2023.

[29] Manuel Brenner, Florian Hess, Jonas M Mikhaeil, Leonard F Bereska, Zahra Monfared, Po-Chen Kuo, and Daniel Durstewitz. Tractable dendritic RNNs for reconstructing nonlinear dynamical systems. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2292–2320. PMLR, 17–23 Jul 2022.

[30] Dominik Schmidt, Georgia Koppe, Zahra Monfared, Max Beutelspacher, and Daniel Durstewitz. Identifying nonlinear dynamical systems with multiple time scales and long-range dependencies. In *International Conference on Learning Representations*, 2021.

[31] Georgia Koppe, Hazem Toutounji, Peter Kirsch, Stefanie Lis, and Daniel Durstewitz. Identifying nonlinear dynamical systems via generative recurrent neural networks with applications to fmri. *PLoS computational biology*, 15(8):e1007263, 2019.

[32] Dhruvit Patel and Edward Ott. Using machine learning to anticipate tipping points and extrapolate to post-tipping dynamics of non-stationary dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(2), February 2023.

[33] Z. Monfared and D. Durstewitz. Existence of n-cycles and border-collision bifurcations in piecewise-linear continuous maps with applications to recurrent neural networks. *Nonlinear Dynamics*, 101(2):1037–1052, Jul 2020.

[34] Kenji Doya et al. Bifurcations in the learning of recurrent neural networks 3. *learning (RTRL)*, 3:17, 1992.

[35] Roberto Marichal, Jose Piñeiro, and Gonzalez E. Study of fold bifurcation in a discrete recurrent neural network. *Lecture Notes in Engineering and Computer Science*, 2179, 10 2009.

[36] Steven H Strogatz. *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[37] Lukas Eisenmann, Zahra Monfared, Niclas Alexander Göring, and Daniel Durstewitz. Bifurcations and loss jumps in rnn training. *arXiv preprint arXiv:2310.17561*, 2023.

[38] Robert Haschke and Jochen J. Steil. Input space bifurcation manifolds of recurrent neural networks. *Neurocomputing*, 64:25–38, 2005. Trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004.

[39] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[40] Sekitoshi Kanai, Yasuhiro Fujiwara, and Sotetsu Iwamura. Preventing gradient explosions in gated recurrent units. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[41] Alexander Rehmer and Andreas Kroll. The effect of the forget gate on bifurcation boundaries and dynamics in recurrent neural networks and its implications for gradient-based optimization. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08, 2022.

[42] António H. Ribeiro, Koen Tiels, Luis A. Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2370–2380. PMLR, 26–28 Aug 2020.

[43] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[44] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[45] Peter DelMastro, Rushiv Arora, Edward Rietman, and Hava T Siegelmann. On the dynamics of learning time-aware behavior with recurrent neural networks. *arXiv preprint arXiv:2306.07125*, 2023.

[46] Florian Hess, Zahra Monfared, Manuel Brenner, and Daniel Durstewitz. Generalized teacher forcing for learning chaotic dynamics. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 13017–13049. PMLR, 23–29 Jul 2023.

[47] Christopher J Cueva, Alex Saez, Encarni Marcos, Aldo Genovesio, Mehrdad Jazayeri, Ranulfo Romo, C Daniel Salzman, Michael N Shadlen, and Stefano Fusi. Low-dimensional dynamics for working memory and time encoding. *Proceedings of the National Academy of Sciences*, 117(37):23021–23032, 2020.

[48] Nandakumar S Narayanan. Ramping activity is a cortical mechanism of temporal control of action. *Current opinion in behavioral sciences*, 8:226–230, 2016.

[49] Nicolas Y Masse, Guangyu R Yang, H Francis Song, Xiao-Jing Wang, and David J Freedman. Circuit mechanisms for the maintenance and manipulation of information in working memory. *Nature neuroscience*, 22(7):1159–1167, 2019.

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

S1. Methods

S1.1. Network architectures

Piece-wise linear recurrent neural networks

We considered piece-wise linear recurrent networks (PLRNNs) [31, 30] for most of our experiments unless otherwise specified. PLRNN dynamical systems equation is as follows.

$$x[t] = Ax[t-1] + W\phi(x[t-1]) + Cs[t] + h \tag{S1}$$

Here, $x[t] \in \mathbb{R}^N$ is the network currents/ states. N number of neurons, $\phi(x[t])$ is the network firing rates with ReLU non-linearity; $\phi(x[t])_i = \max(0, x_i[t]), i \in 1, ..., N$. $A \in \mathbb{R}^{N\times N}$ is a diagonal matrix encoding decay time constants of neurons. $W \in \mathbb{R}^{N\times N}$ is the recurrent connectivity matrix. $s[t] \in \mathbb{R}^K$ is the external input with shape K. $C \in \mathbb{R}^{N\times K}$ injects inputs into neurons. $h \in \mathbb{R}^N$ is the bias.

To obtain the predictions $\hat{o}[T] \in \mathbb{R}$, we linearly project the currents at the end of the trials $x[T]$.

$$\hat{o}[T] = W_{\text{out}} x[T] + b_{\text{out}} \tag{S2}$$

Here, $W_{\text{out}} \in \mathbb{R}^{1\times N}$ projects the currents onto the final predictions. $b_{\text{out}} \in \mathbb{R}$ is the output bias. Unless otherwise specified, we allow the network to learn $A, W, C, h, W_{\text{out}}$ and b_{out} to minimize the mean squared error between the target ($o_T = u^1_{t_1} + u^1_{t_2}$ for the delayed addition task) and the network predictions $\hat{o}[T]$.

Leaky firing rate recurrent neural networks

For the analysis on networks with fixed-time scales, we used the leaky firing rate RNNs:

$$\tau \frac{\mathrm{d}r(t)}{\mathrm{d}t} = -r(t) + \tanh(Wr(t) + Cs(t)), \tag{S3}$$

where τ is the fixed neuronal time scales and the rest is defined as above. In practice, we used a discretized version of these equations:

$$r[t + \Delta t] = (1-\alpha)r[t] + \alpha \tanh(Wr[t] + Cs[t]), \tag{S4}$$

where $\alpha = \Delta t/\tau$ is the discretization constant. We used these equations to perform forward and backwards propagation in all experiments. For all experiments, $\tau = 10ms$ and $\Delta t = 8ms$, leading to $\alpha = 0.8$. When lfRNNs are initialized in chaotic regime, we sampled the weight matrices from a Gaussian distribution with zero mean and standard deviation of $g/\sqrt{N}$ with $g \geq 1$.

S1.2. Task details

Delayed addition task

The delayed addition task consists of several train and test trials. Input to each trial, $U = \{(u^1_1, u^2_1), (u^1_2, u^2_2), ..., (u^1_T, u^2_T)\}$, has the shape of $T \times 2$, where T is the trial length. Here,

$$u^1_t \sim \mathcal{U}[0, 1) \tag{S5}$$

is sampled randomly from a uniform distribution between 0 and 1, whereas

$$u^2_t = \begin{cases} 1 & \text{if } t = t_1 \text{ or } t = t_2 \\ 0 & \text{otherwise} \end{cases} \tag{S6}$$

is the cue signal that is mostly zero except for times $t = t_1$ and $t = t_2$. Here, t_1 and t_2 are randomly sampled such that $t_1 < 10$ and $t_2 < T/2$. The goal of the task is to output $u^1_{t_1} + u^1_{t_2}$ at the final time step T. This requires the network to have a memory that can store a number for at least $T/2$ time steps. To measure the network performances, we calculated the fraction of correct trials. Specifically, we counted the trials where $|o_T - \hat{o}[T]| < 0.04$ and presented it as a fraction. The task was used in previous work to identify memory capabilities in PLRNNs [30].

EVIDENCE ACCUMULATION TASK

In the evidence accumulation task, the network receives two inputs and has two corresponding outputs. Two inputs contain transient pulses during the evidence interval, at random time points. The task is to output *high* after a certain time point, only in the output channel corresponding to the input channel with the higher number of transient pulses, *e.g.*, evidence. The network should sustain this output until the end of the trial. See Fig. 5**A** for an illustration.

3-BIT FLIP FLOP TASK

In the 3-bit flip flop task, the RNN has 3 outputs corresponding to the state of 3 memory bits and receives transient ± 1 pulses from the corresponding 3 inputs. The task is to output in each channel the latest sign of the corresponding input channel, leading to a total of 2^3 memory states. The network is required to sustain that value until the corresponding input changes. See Fig. S14**A** for an illustration.

S1.3. Manifold attractor regularization

Recent work has demonstrated the promise in promoting attractor formation [30]. In this framework, memory units are regularized to form a line-attractor subspace, while computation units remain unregularized, resulting in enhanced memory capacity:

$$\mathcal{L}_{MAR} = \lambda_{\text{MAR}} \left[\sum_{i=1}^{N_{reg}} (A_{ii} - 1)^2 + \sum_{i=1}^{N_{reg}} \sum_{\substack{j=1 \\ j \neq i}}^{N} W_{i,j}^2 + \sum_{i=1}^{N_{reg}} h_i^2 \right]. \tag{S7}$$

Here, MAR enforces the diagonal time scales to be 1 (perfect memory of previous state), and the weights and biases to be zero. If this is exactly enforced on the inputs ($\lambda_{MAR} \to \infty$), we are left with:

$$x[t+1] = x[t] + Cs[t], \tag{S8}$$

which is a plane attractor and performs integration. In the case of manifold attractor initialization (MAI), only a subset of neurons are initialized with this plane attractor, creating a sub-circuit capable of performing integration of inputs.

Overall, the slow time dynamics induced by MAI/MAR directly stem from the fact that they incentivize explicit plane attractor formation, which is specifically beneficial to the delayed-addition tasks. Moreover, MAI/MAR require the dynamical system equation to be in a specific format, making them inapplicable to general architectures.

S1.4. Hyperparameter selection and details on figures

Unless otherwise specified, we trained the networks on the delayed addition task with Adam optimizer [50], a train batch size of 500, a test batch size of 100, and 100,000 train and 10,000 test trials. We ran most of the experiments with PyTorch framework on a desktop computer with two GeForce RTX 4090 GPUs.

Figs. 1 and S1: In these figures, we trained an unregularized PLRNN on a delayed addition task with $T = 40$. We initialized all the weights (i.e. A, W, C, h) using PyTorch's default recurrent network initialization method, by sampling from a uniform distribution of $[-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}]$. We kept the b_{out} learnable and initialized $x[0]$ from a standard normal distribution. The network was trained for 500 epochs, with 10.0 gradient clipping. To optimize the learning rate, we repeated the experiment 3 times with a learning rate from $\{0.0015, 0.001\}$. A learning rate of 0.0015 was picked.

Figs. 2, 3, and S2: To further analyze the gradient alignment and bifurcations, we repeated the same experiments in Fig. 1 with $T = 40$ and $3,000$ epochs with a stable 0.001 learning rate. To make the training harder, the networks were shown $1,000$ training trials and $10,000$ test trials.

Fig. 4: To find out the effect of TCR (Fig. 4), we conducted experiments with unregularized PLRNNs, PLRNNs with MAR, and PLRNNs with TCR. We considered the same network implementations, initialization method, and hyper-parameters used in the previous work [30], $T = 40$, 100 epochs of training, and a learning rate of 0.001 for all the networks. A gradient clipping of 3.0 was applied for all the networks. For TCR, we used $\lambda_{\text{TCR}} = 1.0$ as the regularization strength. For all the networks with TCR, we regularized 50% of the neurons, *i.e.* $N_{reg} = 40 \times 0.5 = 20$. We repeated the experiments 10 times and reported the mean and the standard error (Fig. 4**A, B**). For Fig. 4**C**, we summarized the search phase duration of the networks presented in Fig. S9 (See below).

Fig. S3: This figure compares the performances of unregularized PLRNNs vs those trained with MAR and MAI from the experiments in Fig. S9 (See below).

Fig. S4: To reproduce MAR results from [30] (Fig. S4**A**), we considered three networks; unregularized PLRNN, PLRNN with MAI and PLRNN with MAI and MAR. We followed hyper-parameters used in [30] for $T \in \{20, 40, 60, 100, 200, 300, 400, 500\}$. Experiments were repeated across 10 networks per condition.

Fig. S5: For this experiment, picked 10 unregularized PLRNNs and 10 PLRNNs trained with MAR and MAI for the delayed addition task ($T = 60$) from the networks shown in Fig. S4**A**.

Fig. S6: We trained PLRNNs using the same configuration as in Figs. 2, 3, and S2, but on harder delayed addition tasks ($T = 40, 60, 100, 200$). For MAR, we used $\lambda_{\text{MAR}} = 0.1$ and regularized half of the $N = 40$ neurons.

Figs. S7 and S8: These figures analyzed the effect of weight decay and batch size respectively. All the other hyper-parameters were consistent with experiments from Fig. S9 (see below).

Fig. S9: Here, we conducted experiments with the same hyperparameters used for bifurcation analysis for unregularized PLRNNs (Fig. 1). In addition to unregularized PLRNNs, here we considered PLRNNs with MAI, PLRNNs with MAR ($\lambda_{\text{MAR}} = 5.0$), and PLRNNs with TCR ($\lambda_{\text{TCR}} \in \{0.001, 0.01, 0.1\}$). For all the networks with MAR/ TCR/ MAI, we regularized only half of the $N = 40$ neurons. We repeated all the experiments 10 times.

Fig. S10: We performed a grid search over TCR parameters, otherwise using the same parameters as in Fig. 4**A, B**.

Fig. S11: For noise-injection experiments, we considered the same configurations used for Fig. 1 bifurcation analysis, except with a learning rate of 0.001 and a weight decay of 10^{-8}. Experiments were repeated 3 times per condition.

Figs. 5 and S12: For the evidence accumulation task, we initialized the RNNs with $g = 4$ for Fig. 5 and with Xavier initialization for Fig. S12. For illustrations in Fig. 5**A-B**, we used an unregularized lfRNN with $N = 50$ neurons, which was initialized with Xavier initialization and trained for $3,000$ epochs. For all other experiments, we trained 20 lfRNNs per condition, with Adam optimizer and 0.001 learning rate. We used 100 training trials, each trial was 400ms in duration. The evidence was presented between $[100, 300]$ms intervals. Expected number of flips per channel was 4. The coherence of each task was sampled randomly from a uniform distribution on $[-1, 1]$.

Fig. S13: For the LSTM experiments, we trained 50 networks for 2000 epochs with Adam optimizer, gradient clipping parameter of 5, weight decay rate of 10^{-8}, and 0.001 learning rate. We varied $\lambda_{\text{TCR}} \in \{0, 10^{-3}, 10^{-2}, 0.1, 1\}$, which was applied to the cell states of $N_{\text{reg}} = 20$ out of $N = 40$ neurons. Both the training and test datasets contained $1,000$ trials each.

Fig. S14: For flip-flip experiments, we initialized the lfRNNs with $g = 3.0$. We trained 20 networks for 2000 epochs with Adam optimizer and 0.001 learning rate. The training dataset contained 100 trials with a duration of 400ms. Expected number of flips per channel was 8.

Fig. 6: See Appendix S1.11. In Fig. 6**C**, boxes contain data points from lower to upper quartiles, with the middle line denoting the median. The upper and lower whiskers are 1.5 times the interquartile distance.

S1.5. Energy minimization

To extract the slow points of the network, the core implementation is based on previously published work [8], though we modified it slightly by replacing the minimization algorithm with a Pytorch based solver to speed up the process. We first picked a data batch with 100 trials from the test set. We obtained currents for all the trials. This resulted in $100 \times T$ total number of states. 500 states were randomly picked ($x[0]$) as the initialization points for energy minimization. Starting from each $x[0]$, we minimized energy using stochastic gradient descent in a recursive manner:

Algorithm S1 Energy minimization

```
$x^e[0] \leftarrow x[0]$
for $t \in 0, \ldots,$ number of steps -1 do
    Generate $x[t+1]$ from $x[t]$ using one step of our model in eq. 1
    $E[t] \leftarrow ||x[t+1] - x[t]||_2^2$
    $x^e[t+1] \leftarrow x[t] - \alpha \frac{\partial E(t)}{\partial x[t]}\Big|_{x[t]}$
    $x[t+1] \leftarrow x^e[t+1]$
end for
```

Here, $\alpha = 0.1$ and the number of steps is 1000. $[x^e[0], x^e[1], ..., x^e[1000]]$ gives a single energy minimization trajectory. Final states $x^e[1000]$ for each initialization were visualized as red crosses using PCA in Fig. 1, 2, 4, and S1.

S1.6. Visualization analysis

To visualize attractor manifolds, we reduced the dimensionality of $x[t]$ and $x^e[t]$ into 3 with Principle Component Analysis (PCA). To fit the data to PCA, we first fed 100 trials from the test set to the *fully-trained PLRNN* and obtained the corresponding currents $x[t]$s. All the states in $x[t]$s are used to fit the PCA. The resultant PCA model transformed all the $x^e[t]$s and $x[t]$s obtained from checkpoint models during the training.

S1.7. Stimulus Decoding Experiments

To test the memory capabilities of attractor-incentivized networks compared to unregularized PLRNNs, we conducted cue and stimulus decoding experiments. We picked 10 unregularized PLRNNs and 10 PLRNNs trained with MAR and MAI for the delayed addition task from the networks shown in Fig. S4 A: $T = 60$.

We first created a new addition dataset with predetermined cue positions (i.e. $t_1 = 5$ and $t_2 = \frac{T}{2} - 1$). The goal was to quantify networks ability to predict $u^1_{t_1}, u^1_{t_2}$, and $u^1_{t_1} + u^1_{t_2}$ from instantaneous currents of all the neurons ($x_S[t] = x_i[t]_{i=\{1,...,N\}}$), memory neurons ($x_{S_1}[t] = x_i[t]_{i=\{1,...,N_{reg}\}}$), and computational neurons ($x_{S_2}[t] = x_i[t]_{i=\{N_{reg}+1,...,N\}}$) separately. Analysis was conducted for all the time steps, $t \in 1, ..., T$.

To predict cues and/or targets from neuronal activity, we used linear regression models. For a given network and a time step t, 9 linear regression models were independently fitted to map 3 types of instantaneous inputs (i.e. $x_S[t], x_{S_1}[t], x_{S_2}[t]$) into 3 types of decoding targets (i.e. $u^1_{t_1}, u^1_{t_2}$, and $u^1_{t_1} + u^1_{t_2}$). We repeated the procedure for $t \in \{1, ..., T\}$ and all 20 networks (*i.e.*, 10 unregularized PLRNNs and 10 regularized PLRNNs). All the fitting was done on the training dataset with $10,000$ trials and testing was done on the testing dataset with $1,000$ trials.

S1.8. Frequency Analysis

We analyzed frequency distributions of $x[t]$ belonging to unregularized PLRNNs, PLRNNs with MAI, and PLRNNs with MAI+MAR. To conduct this analysis, we selected 3 representative networks from Fig. S4**A**, with $T = 20$. We first concatenated 50 trials with $T = 20$ from the test set and obtained a long trial with a temporal length of $T = 1000$. Currents $x[t]$s were obtained for this long input trial. Obtained currents for 10 neurons are shown in Fig. S4**C**.

To obtain the frequency plots, we separately considered activities, $x[t]$, of computational and memory neurons. We first divided the trajectories of the neuronal currents by the maximum current of each trajectory. Fast Fourier transform followed by frequency shift was then applied. We then obtained absolute values from the resultant frequency sequence. The resultant frequency sequences were divided again by the maximum frequency component of the corresponding frequency sequence. Finally, we took the mean frequency sequence over neurons. Resulting frequency plots are shown in Fig. S4**C**.

S1.9. Intervention experiments

We performed intervention experiments (Fig. S11) by injecting noise into the networks that were trained with and without TCR. Networks had $N = 40$ neurons and were performing an addition task with $T = 40$.

In the first case (Fig. S11**A**), we injected Gaussian noise with ϵ standard deviation ($\sim \mathcal{N}(0, \epsilon^2)$) during the first half of the task duration $t \in \{1, ..., 20\}$, to 2 sets of neurons; $S_1 = 1, ..., N_{reg}$ and $S_2 = N_{reg}, ..., N$ separately. In the second case (Fig. S11**B**), we injected a constant input with amplitude γ during the first half of the task duration to neurons; S_1 and S_2. For networks trained with TCR, the set S_1 and S_2 corresponded to memory and computational neurons respectively. The procedure was repeated for 6 networks (3 unregularized PLRNNs and 3 PLRNNs with TCR).

S1.10. Training speed during the search phase

We calculated the training speed during the search phase by taking $\frac{1}{\text{epoch}_b}$. epoch_b is computed based on the stalling points in the loss function during training. Specifically, we computed the first epoch that achieves a particular value in the loss or accuracy levels, after which rapid learning took place. For the delayed addition task (Figs. 4 and S6), this was ~ 0.08 for the fraction of correct trials. For the K-bit flip flop task (Fig. S14), this was ~ 0.5 in the correlation values between the target and the outputs. For the evidence accumulation task (Figs. 5 and S12), this was ~ 0.3 in the mean squared distance between target and reconstructed coherence. We performed visual inspection in all cases to ensure that we captured the onset of the rapid comprehension phase.

S1.11. Cue-associated fixed-point generation (online and local)

For our online experiments, we once again used leaky firing rate RNNs:

$$\begin{aligned}\tau\frac{dr_i(t)}{dt} &= -r_i(t) + f(z_i(t)),\\ z_i(t) &= \sum_{j=1}^{N_{\text{rec}}} W_{ij} r_j(t) + \sum_{j=1}^{N_{\text{in}}} C_{ij} s_j(t) + \epsilon_i(t),\ \forall i \in [N_{rec}].\end{aligned} \tag{S9}$$

Here, r_i is the activity or firing rate of neuron i and z_i is the total input current to neuron i. N_{rec}, the number of neurons, is 400 and N_{in}, the number of input channels. The rest is defined as before. Furthermore, the time constant $\tau = 10ms$, and $f(\cdot) = \tanh(\cdot)$. Each element of W was initially drawn from a normal distribution with mean 0 and standard deviation $1.8/\sqrt{N_{rec}}$ while each element of C was drawn from a standard normal. To prevent self-excitation, we also enforced $W_{ii} = 0,\ \forall i \in [N_{rec}]$.

While the continuous time formulation provides a theoretically motivated window into the time dynamics of the network, the simulations were performed via discretization with Δt time steps. The discretized network dynamics followed:

$$r_i[t + \Delta t] = (1 - \alpha) r_i[t] + \alpha f(z_i[t + \Delta t]), \tag{S10}$$

where $\alpha = \frac{\Delta t}{\tau}$ is the unitless normalized discretization time, here chosen to be $\alpha = 0.1$.

The learning process began after $s[t]$ becomes nonzero, *i.e.*, the arrival of the rectangular pulse. After completion of this pulse, we enforced that the network aims to produce its previous activity $r(t - \Delta t)$. Namely, we performed a local gradient step on the loss function:

$$\lambda(r_i[t] - r_i[t-1])^2 \tag{S11}$$

for all neurons, where λ determines the strength of this derivative regularizer - we set $\lambda = 1/1000$. A step of this update rule corresponded to

$$W[t + \Delta t] = W[t] - \lambda_{\text{TCR}}\left(r[t] - r[t - \Delta t]\right) r[t - \Delta t]^T. \tag{S12}$$

We repeated this update several times, in a randomized manner across cues, as shown in Fig. 6. Each pass across all cues was considered a single epoch. For Fig. 6**C**, we computed the Jacobian J at some given r_* by taking partial derivatives of the right-hand side in Eq. (S9).

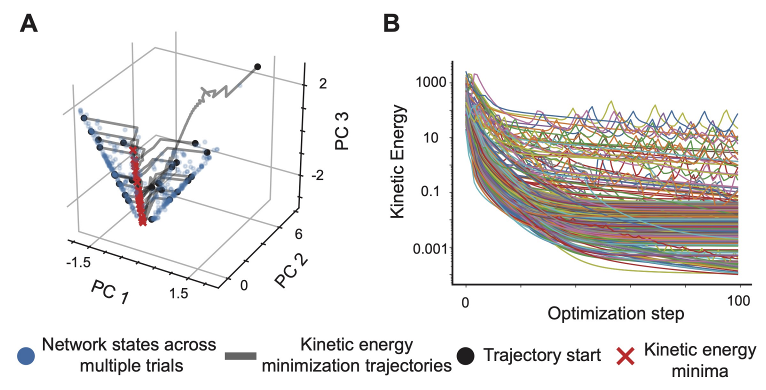


Figure S1: **Minimizing the kinetic energy of the neural activities unravels the slow point landscape.** **A** Since the kinetic energy minimization is a non-convex problem, the optimization trajectories (black lines) and the converged slow points (red crosses) depend on the initial points, which are chosen randomly from the neural states visited during the trials (blue dots). **B** The kinetic energy is plotted over optimization step, each line corresponds to a different trajectory from a distinct initialization. Though the kinetic energy decreases over iterations, it does not reach zero (or close to the machine precision), indicating the existence of slow, but not necessarily fixed, points.

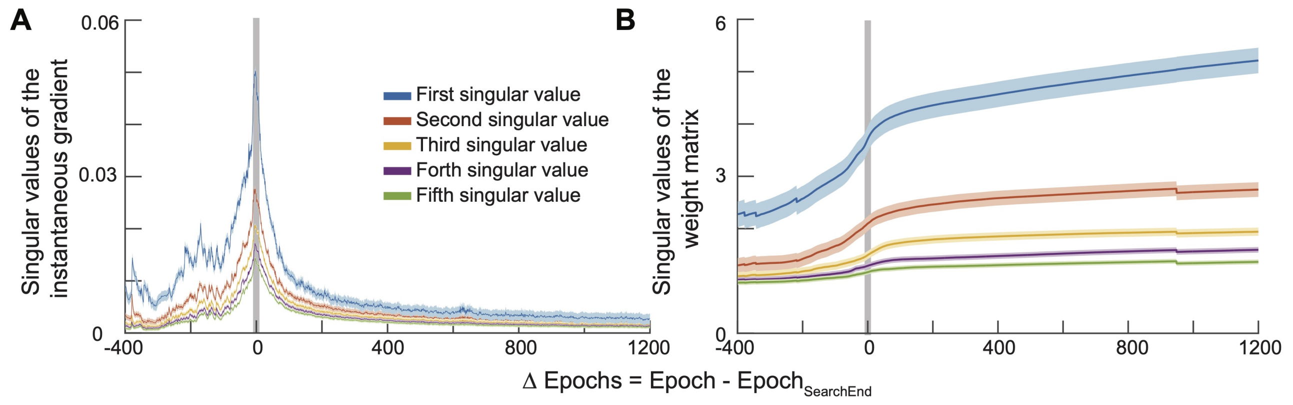


Figure S2: **The gradients first destabilize then recover at a bifurcation, but the weight changes are permanent.** **A** Right around the bifurcations, the weights undergo large updates, in line with the prior work [39]. **B** The weight matrices evolve rapidly during the bifurcation and settle into a new equilibrium once the bifurcation is complete. Solid lines: means. Shaded areas: s.e.m. over 19 networks ($N = 40$ neurons, $T = 40$ time steps, 3000 epochs training, no regularization).

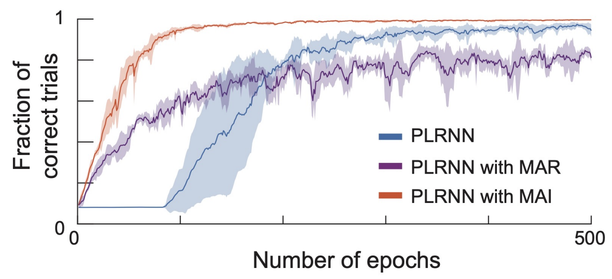


Figure S3: **Manifold attractor regularization and initialization lead to negligible search phase.** As discussed in the Appendix S1.3, MAI creates a plane attractor using a small subset of units in the PLRNN, which gives the network enhanced memory capabilities. Therefore, MAI practically skips the search phase while having an extremely rapid learning phase. In comparison, MAR encourages the plane attractor formation during the learning with a regularization term, leading to undetectable fast search phase. Solid lines: means. Shaded areas: s.d. over 5 networks that are trained on the delayed addition task with $T = 40$ and $N = 40$, with 20 regularized memory neurons whenever applicable. MAR regularization weight was $\lambda_{\text{MAR}} = 5$.

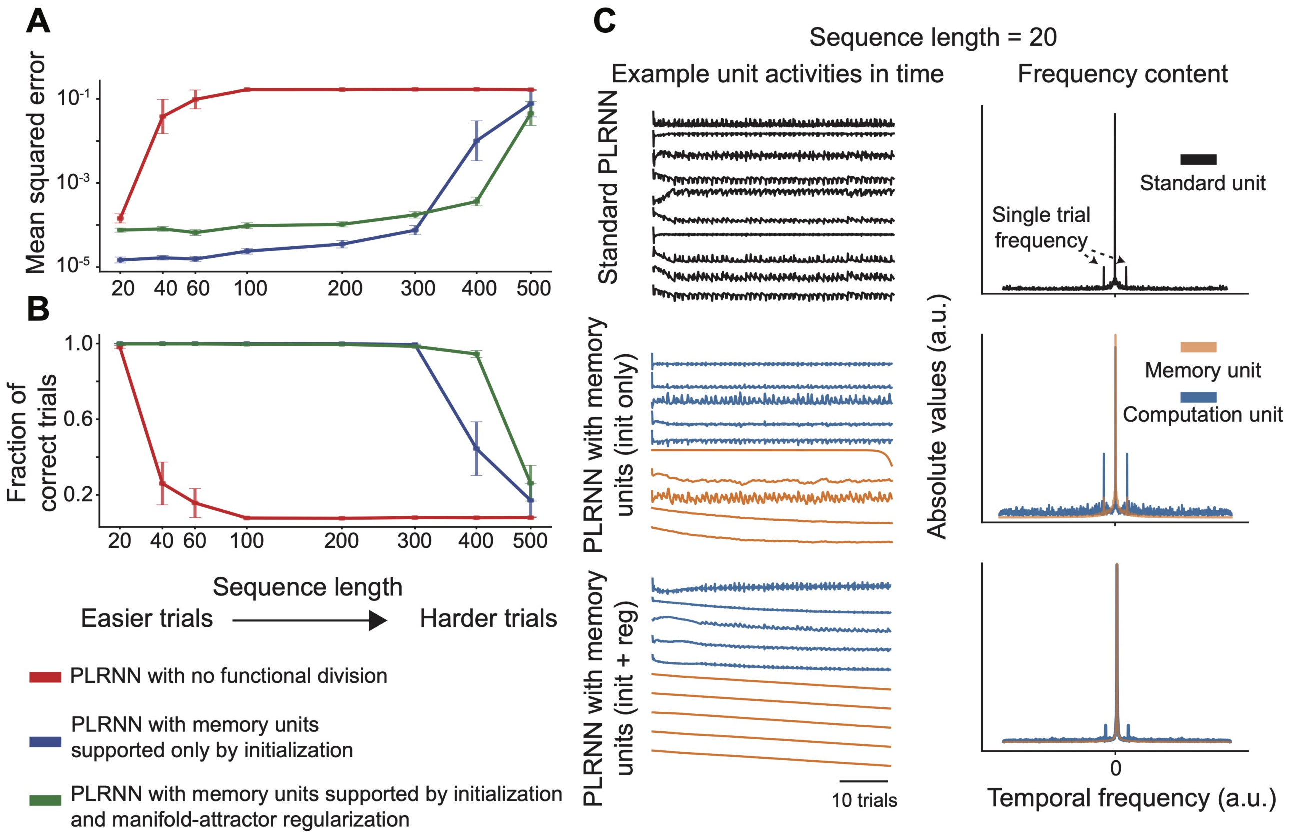


Figure S4: **Manifold attractor initialization and regularization incentivize slow time dynamics**. **A-B** We trained unregularized and regularized PLRNNs on the delayed-addition task with varying levels of sequence length. We plotted (**A**) mean squared errors and (**B**) the fractions of correct trials. Solid lines: means. Error bars: s.e.m. over 10 runs. **C** Analysis of frequency content in PLRNNs' computation and memory units. Memory units in networks with MAR/MAI show slow time dynamics, whereas the neural activities of the computation units have high frequency components. For all experiments, MAR/MAI was applied to 20 out of 40 neurons. MAR regularization weight was $\lambda_{\mathrm{MAR}} = 5$.

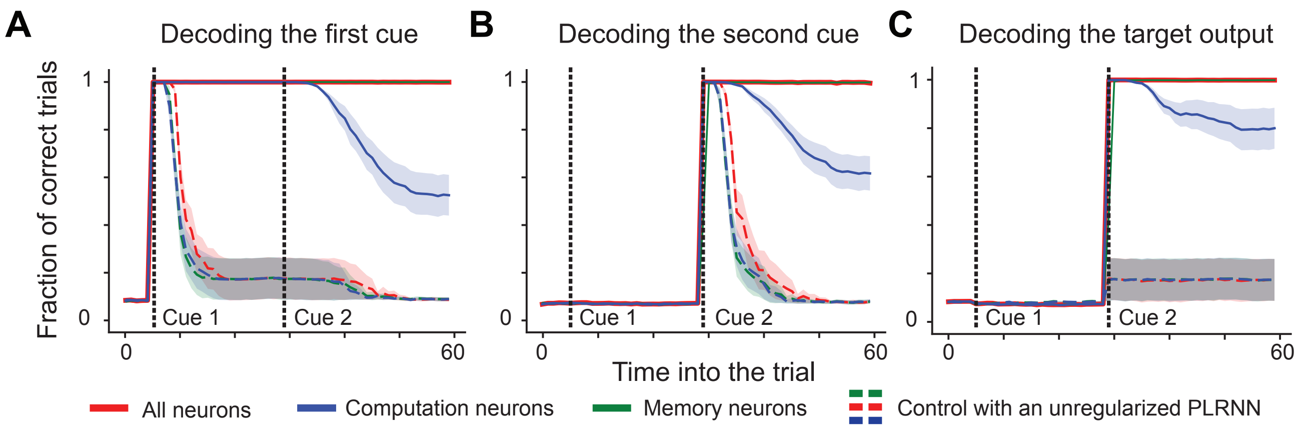


Figure S5: **Manifold attractors enhance the memory capabilities of PLRNNs.** To test the memorization of the task-relevant information in the unit activations, we trained PLRNNs on delayed addition tasks with $T = 60$, which were initialized with manifold attractors and encouraged to retain them with a regularization term. Here, 20 out of $N = 40$ neurons were regularized with $\lambda_{\mathrm{MAR}} = 5$. During testing only, we focused on a variation of the delayed addition task, in which the input times for the cues were fixed (See Appendix S1.7). Using the activities of memory, computation, and all neurons, we trained linear estimators for (**A**) the first cue, (**B**) the second cue, and (**C**) the target output, *i.e.*, the sum of the two cues. Computation, but not the memory, neurons have eventually forgotten cue 1, cue 2, and the output, emphasizing the role of the manifold attractors for enhancing the short-term memory capabilities. As a control (dashed lines), we also trained unregularized PLRNNs, which were not able to retain the memories for either of the cues to begin with. Solid and dashed lines: means. Shaded areas: s.e.m. over 10 networks.

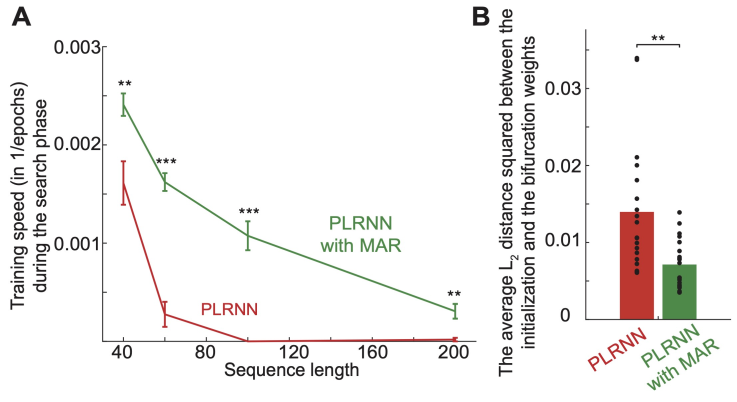


Figure S6: **MAR shortens the search phase by choosing a closer bifurcation point**. **A** MAR leads to a faster training speed during the search phase. The comparisons are performed with two-sided Wilcoxon signed-rank tests ($^{***}p < 10^{-3}$, $^{**}p < 10^{-2}$). Solid lines: means. Error bars: s.e.m. over 20 networks. **B** PLRNNs trained with MAR show lower L_2 distances between initialization and bifurcation weights, indicating that regularized networks cross the desired weight subspace boundary at a closer bifurcation point. The comparison is performed with a two-sided rank sum test ($^{**}p < 10^{-2}$).

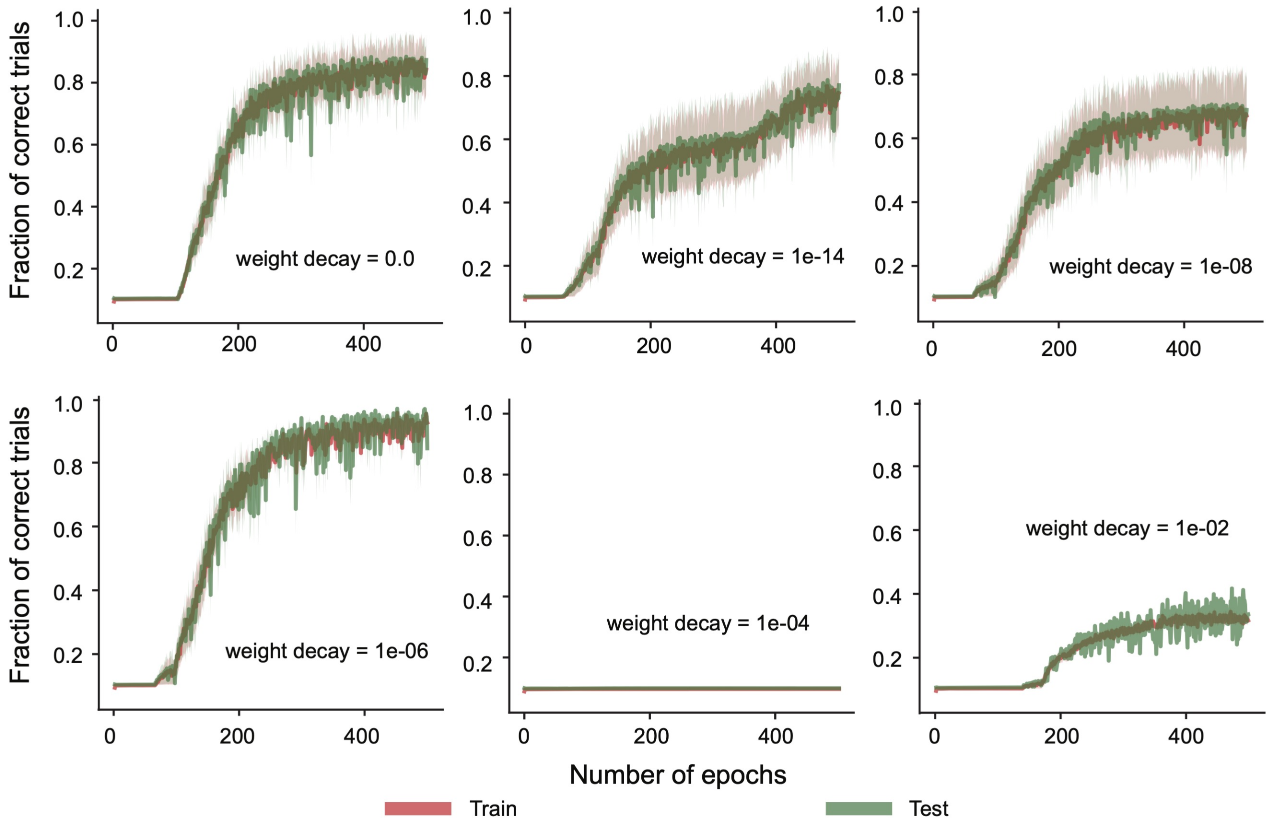


Figure S7: **Weight decay does not have a consistent effect on the duration of the search phase.** To test whether weight decay can facilitate beneficial bifurcations, we trained PLRNNs ($N = 40$) regularized with varying levels of weight decay values on delayed addition tasks with $T = 40$. We had not observed any consistent affect of the weight decay values on the duration of the search phase. Solid lines: mean. Shaded areas: s.e.m. over 10 networks.

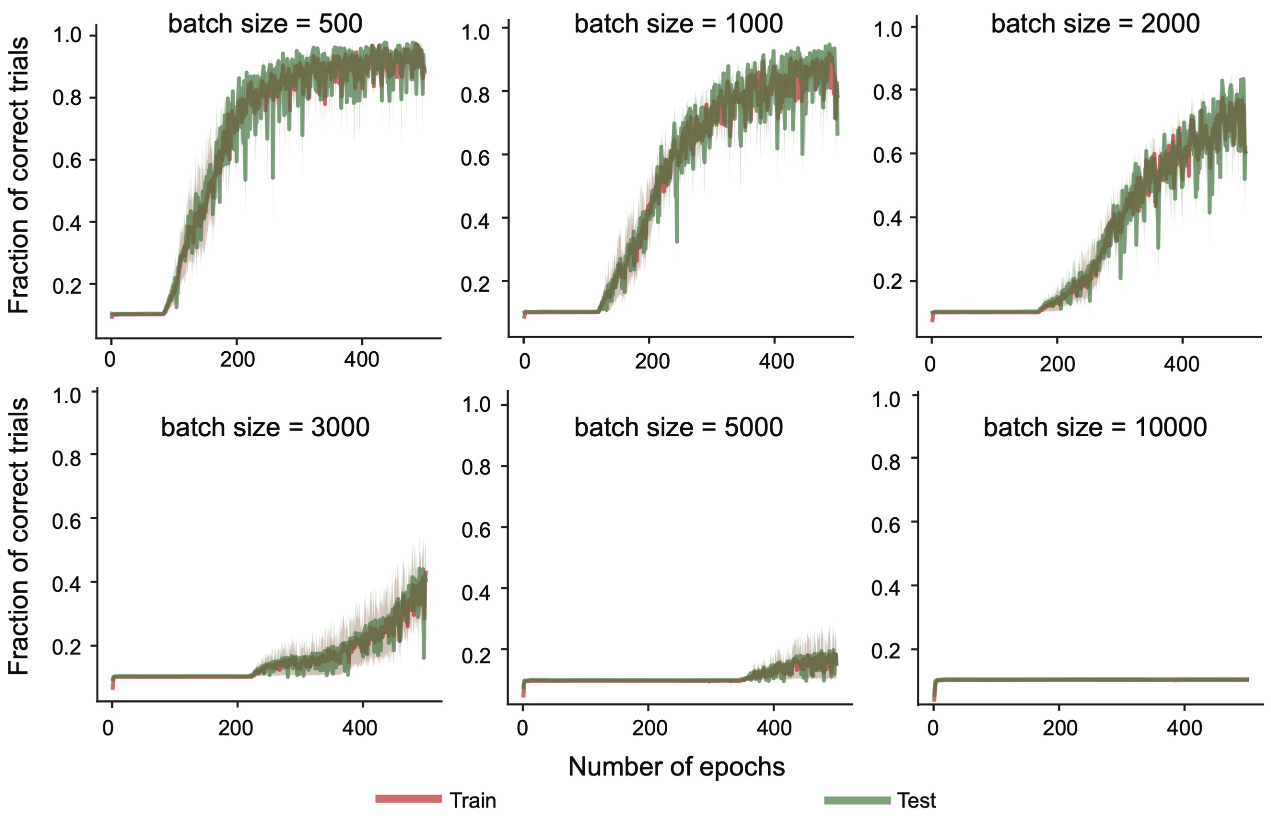


Figure S8: **Increasing the stochasticity of the gradient shortens the search phase.** Similar to Fig. S7, we tested the effects of stochasticity in training by varying the batch sizes while training unregularized PLRNNs ($N = 40$) to perform delayed addition tasks with $T = 40$. Solid lines: means. Shaded areas: s.e.m. over 5 networks.

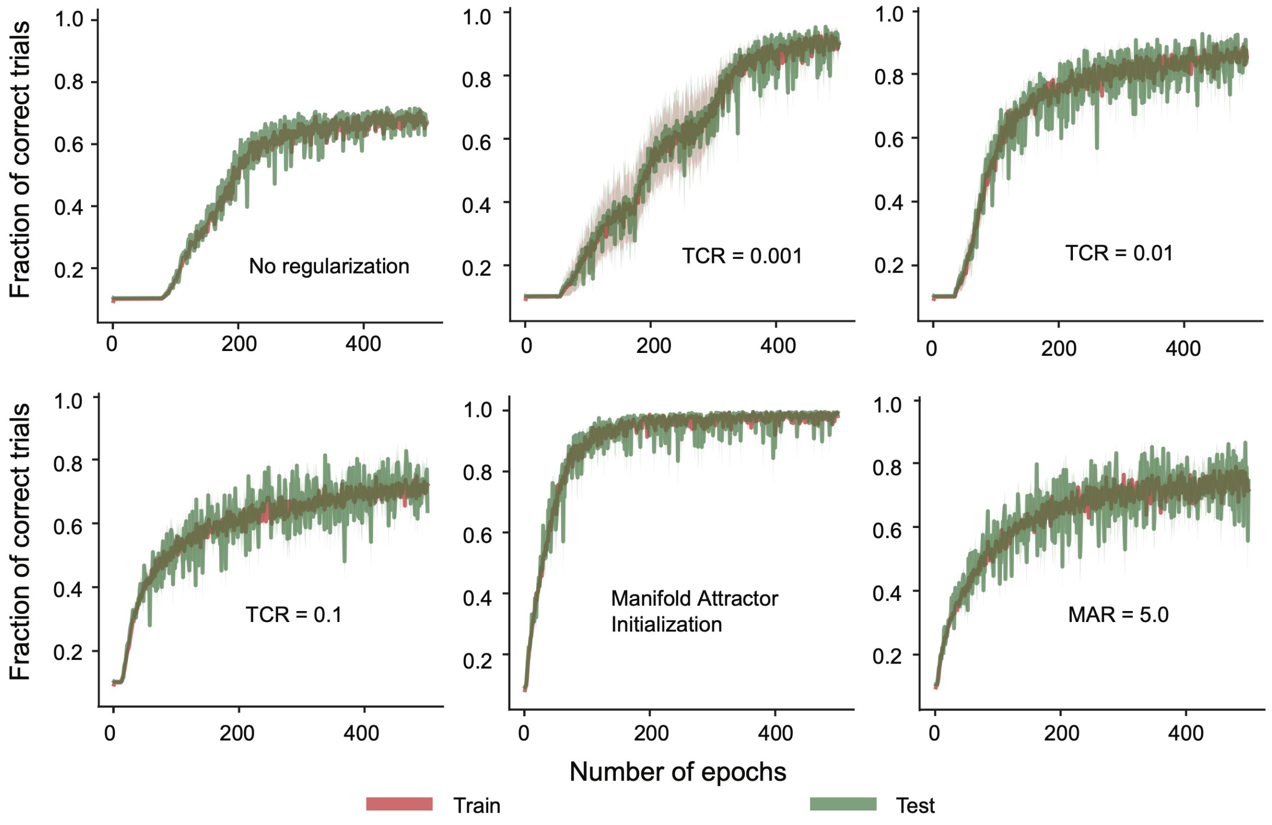
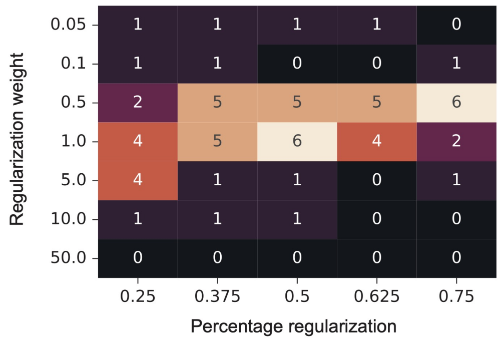


Figure S9: **Enforcing temporal consistency on subset of neurons shortens the search phase.** To test the efficacy and efficiency of TCR, we trained PLRNNs (regularized $N_{\text{reg}} = 20$ out of $N = 40$ neurons) to perform delayed addition tasks with $T = 40$. Higher TCR strength led to faster conclusion of the search phase, whereas PLRNNs trained with either MAR or MAI practically skipped the search phase. Solid lines: means. Shaded areas: s.e.m. over 10 networks.



Regularization weight	0.25	0.375	0.5	0.625	0.75
0.05	1	1	1	1	0
0.1	1	1	0	0	1
0.5	2	5	5	5	6
1.0	4	5	6	4	2
5.0	4	1	1	0	1
10.0	1	1	1	0	0
50.0	0	0	0	0	0

Figure S10: **In a narrow range of parameters, TCR can lead to a rapid conclusion of the search phase.** We re-analyzed the experiments in Fig. 4**A**, but with varying levels of hyperparameter configurations for TCR. Extreme cases, *i.e.*, $\lambda_{\text{TCR}} = 0.05$ and $\lambda_{\text{TCR}} = 50$, show that the networks can barely pass the search phase without or with extreme levels of TCR, whereas moderate levels of regularization significantly increased the chances of rapid training. Similar to Fig. 4**A**, we trained PLRNNs ($N = 40$) on delayed addition tasks with $T = 40$ for 100 epochs.

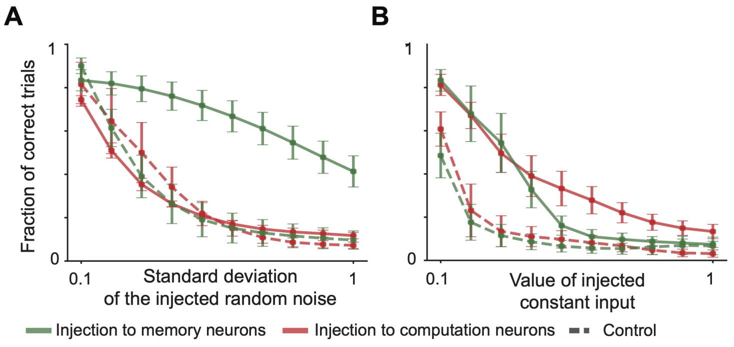


Figure S11: **Enforcing temporal consistency led to the formation of robust attractors.** As before, we trained PLRNNs on delayed addition tasks with $T = 40$. During test trials, we injected random (**A**) and constant (**B**) inputs to memory and computation neurons in the regularized networks, and to randomly selected (control) neurons in the unregularized networks (See Appendix S1.9). All networks had $N = 40$ neurons, regularization ($\lambda_{\mathrm{TCR}} = 0.5$) was applied to 20 neurons whenever applicable. Solid lines: means. Error bars: s.e.m. over 3 networks.

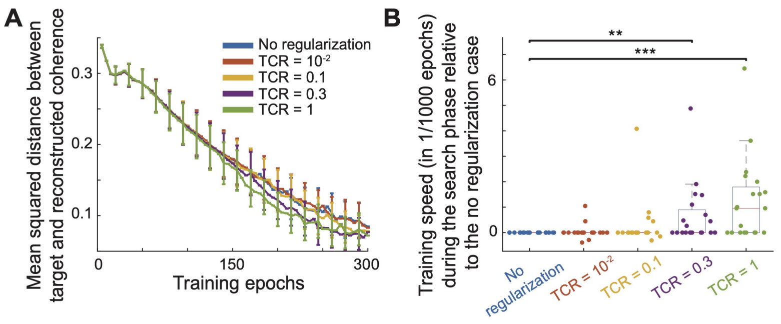


Figure S12: **Even when initialized in a non-chaotic regime, TCR improves the convergence on the evidence accumulation task.** We trained the networks on the same task in Fig. 5 but with Xavier initialization, for 300 epochs. Similar to before, TCR resulted in slightly lower loss (**A**) and faster training speed (**B**) during the search phase. The lfRNNs had $N = 50$ neurons, $N_{\mathrm{reg}} = 25$ were regularized with TCR whenever applicable, the neuronal decay time was $\tau = 10ms$, and the lfRNNs were discretized with $\Delta = 8ms$. The comparisons were performed with two-sided Wilcoxon signed-rank tests ($^{**}p < 10^{-2}$ and $^{***}p < 10^{-3}$).

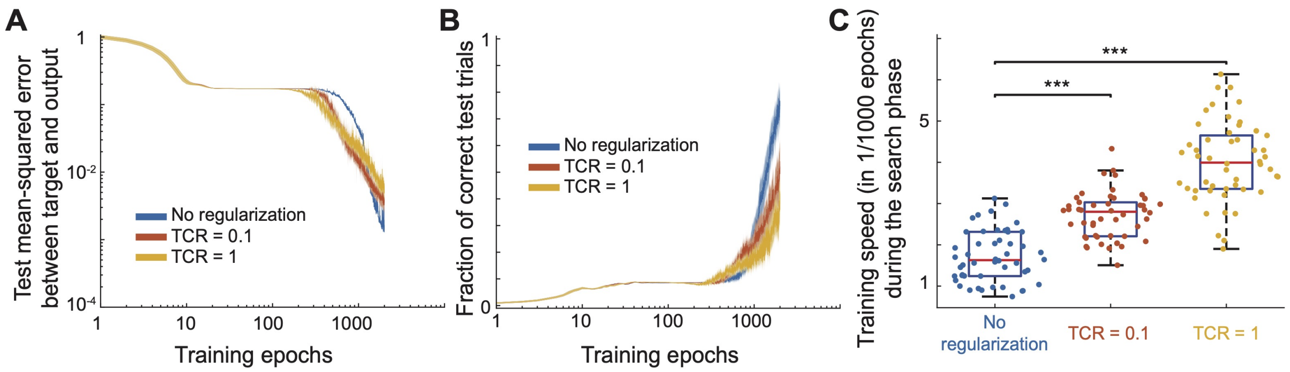


Figure S13: **Time consistency regularization improves training speed of LSTMs during the search phase.** We trained 50 LSTMs on the delayed addition task with $T = 40$, where $N_{\mathrm{reg}} = 20$ out of $N = 40$ neurons were regularized with TCR (See Appendix S1.4). The error values (**A**) and the accuracies (**B**) over $1,000$ test trials as a function of number of training epochs. Solid lines: means. Shaded regions: 95% confidence intervals across 50 networks. **C** Once again, TCR led to faster search phase. The comparisons were performed with two-sided Wilcoxon signed-rank tests ($^{***}p < 10^{-3}$).

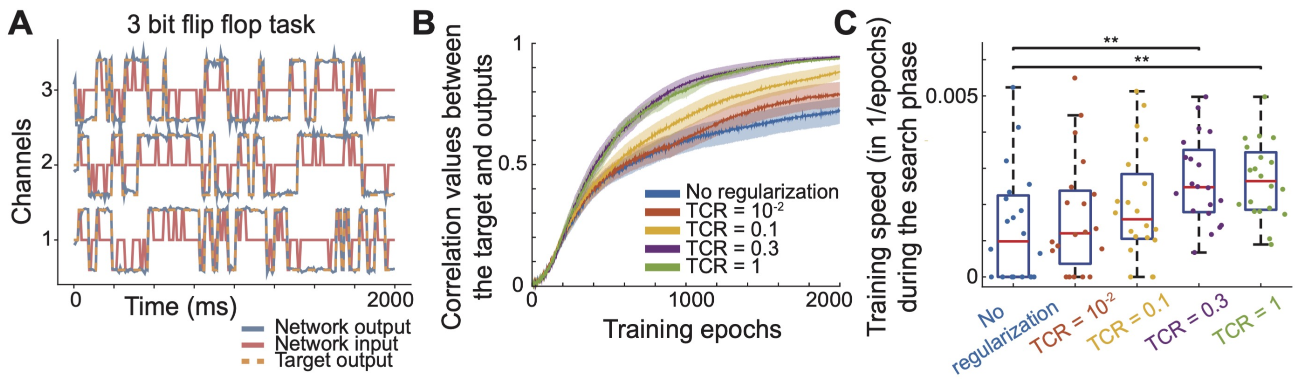


Figure S14: **Time consistency regularization improves training of fixed-points in chaotic networks.** We trained 20 chaotically initialized lfRNNs ($N = 50$, $N_{\mathrm{reg}} = 25$, $\tau = 10ms$, $\Delta t = 8ms$, and $g = 3$) on a 3-bit flip-flop task for $2,000$ epochs (See Appendix S1.4). **A** In the flip-flop task, the network receives three (mostly zero) inputs, with occasional positive and negative pulses signalling state changes. The network should adapt its three outputs to the latest state presented by the input, which requires switching its internal dynamics between $2^3 = 8$ distinct states. As in Fig. S13, TCR enhanced the learning speed and performance (**B**) and accelerated training during the search phase (**C**).