Enhanced cue associated memory in temporally consistent recurrent neural networks

Udith Haputhanthri[*]
Center for Advanced Imaging
Harvard University
Cambridge, MA 02138

Liam Storan[*]
Department of Applied Physics
Stanford University
Stanford, CA 94305

Adam Shai[*]
CNC Program
Stanford University
Stanford, CA 94305

Surya Ganguli
Department of Applied Physics
Stanford University, Stanford, CA 94305

Mark J. Schnitzer
Howard Hughes Medical Institute
Stanford University, Stanford, CA 94305

Hidenori Tanaka[†]
Physics & Informatics Laboratories, NTT Research, Inc.
Sunnyvale, CA 94805
Center for Brain Science, Harvard University
Cambridge, MA 02138

Fatih Dinc[†]
Department of Applied Physics,
Stanford University
Stanford, CA 94305
fdinc@stanford.edu

Abstract

Recurrent connections are instrumental in creating memories and performing time-delayed computations. During their training, networks often explore distinct topological regions across the parameter space, each with unique attractor structures that serve specific computational purposes. However, the mechanisms that facilitate these topological transitions, so called bifurcations, toward an optimal parameter space configuration remain poorly understood. In this workshop paper, we investigated the learning process of recurrent neural networks in memory-assisted computation and developed a regularization strategy to encourage bifurcations that enhance memory formation capacity. To begin, we examined a delayed addition task that required the network to retain cue-associated memories for an extended duration. We observed two distinct phases during the learning of recurrent neural networks, separated by a bifurcation. In the initial *search phase*, both train and test loss values remained stable as the network searched for beneficial bifurcations leading to optimal parameter configurations. In the subsequent *rapid comprehension phase*, the loss values rapidly decreased, and the network quickly learned the task while preserving its topology but updating its geometry. During our analysis, we observed that the gradient direction, *i.e.*, learning signal, was aligned with the optimal descent direction in the second but not the first phase. To aid learning in the search phase, we developed a temporal consistency regularization that incentivized a subset of neurons to have slow time dynamics, which subsequently decreased the duration of the search. Next, we tested the stability of the learned attractors with and without the temporal consistency regularization, via noise injection experiments, where we uncovered a more robust attractor subspace formation in the former. Finally, we enforced temporal consistency in a randomly initialized chaotic recurrent neural network to obtain several cue-associated fixed points in an unsupervised, online, and biologically plausible manner. Our results provide a deeper understanding of the role of bifurcations in enhancing associative memory by driving networks toward the desired attractor formation.

[*]These authors contributed equally to this work.
[†]These authors co-supervised this work.

Associative Memory & Hopfield Networks in 2023. NeurIPS 2023 workshop.

1 Introduction

Creating associations in recurrent architectures often requires finding an attractor subspace that can support the desired computations and hold relevant memories [1]. From a dynamical systems perspective, the transitions between topologically distinct regions towards a memory-friendly parameter space are driven by desired bifurcations [2]. Although undesired bifurcations have largely been explored in the context of exploding gradients, where sudden qualitative changes in the loss landscape can hurt learning [3, 4], driving the network into desired bifurcations remains largely unexplored in both neuroscience and machine learning literature.

A hint may lie in the mammalian brain recordings, where ramping neural activities with slow-time dynamics play a crucial role in the dynamic timing processes, supporting memory storage and associative computations [5, 6]. Slow networks dynamics near attractors [2, 1] is well known to play a central role in computation and memory [1, 7–10]. Thus, enforcing slow-time dynamics can be a reasonable way to incentivize attractor formation.

To promote slow-time dynamics in piece-wise linear recurrent neural networks (PLRNNs), a recent study [11] proposed manifold-attractor regularization (MAR). In this paradigm, a subset of neurons, termed memory units, are enforced to have weights that form a line-attractor subspace, whereas the rest, termed computation units, are not regularized. As expected, [11] observed increased memory capacity in regularized PLRNNs. However, the mechanism underlying this newfound ability and potential connections to incentivizing desired bifurcations were unexplored.

In this work, we start by reverse-engineering the dynamics of piece-wise recurrent neural networks (PLRNNs) trained on delayed addition tasks, where the networks are required to hold cue-associated memories for long time intervals. Our novel contributions include introducing our architecture-agnostic temporal-consistency regularization (TCR) approach, providing evidence that i) it promotes bifurcations towards the optimal state-space topology, ii) networks trained with TCR form robust memory subspaces compared to their counterparts, and finally iii) devising an online, spatiotemporally local learning rule to train cue-associated memories into randomly initialized chaotic recurrent neural networks with non-trainable time-constants and $\tanh(.)$ non-linearity. While similar forms of temporal-consistency loss have been utilized in computer vision research [12–16], our work represents the first application of this regularization technique to incentivize attractor formation in recurrent neural networks.

2 Results

2.1 Bifurcations separate two distinct learning phases when training recurrent networks

To understand the role of bifurcations in the learning of recurrent neural networks, we first trained an unregularized PLRNN on the delayed addition task and reverse-engineered the state-space geometry during learning by finding slow-points via energy minimization [1] (Fig. 1). The training curve showed two distinct regions, separated by epoch 120 (Fig. 1A). We hypothesized that a bifurcation divided the training into these phases. To test this hypothesis, we first confirmed that the alignment of the instantaneous gradient direction with the optimal parameter update direction had a distinct jump at this point (**Methods**, Figs. 1B, C), hinting at a change in the loss landscape.

Next, we performed energy minimization [1] and plotted low-dimensional neural trajectories alongside the slow points (Fig. 1D). The visualization plots confirmed a qualitative change in the attractor topology. During the early training phase (epochs 40 and 100), all data points were clustered together with no noticeable structure. Remarkably, by epoch 120, the network states spanned in two distinct directions, and neural activation trajectories transitioned between these lines in a zig-zag pattern. A pivotal observation is the bifurcation, occurring at around epoch 120, where energy minima formed a pronounced line in the principal component space. Later training epochs kept the same state-space structure, while fine-tuning the location of the minima (See epoch 500, Fig. 1D).

Bringing it all together, we concluded that the training dynamics showed two distinct phases, where the network searched for the optimal state-space topology in the first phase and fine-tuned the geometry in the second, rapid comprehension, phase. Surprisingly, we also observed that the gradient was weakly aligned, if at all, with the set of recurrent weights that supported the desired attractor subspace right after bifurcation (Fig. S1).

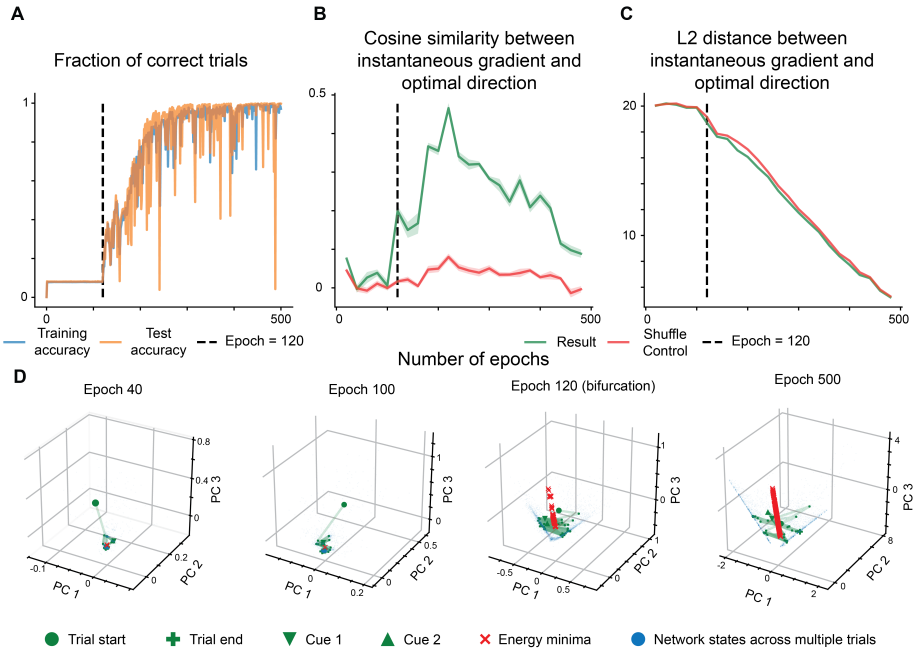

Figure 1: **Learning dynamics during the delayed addition task show two distinct phases, separated by a bifurcation. A** Percentage of correct trials plotted against the number of training epochs revealed two phases of training. **B, C** The instantaneous gradient direction was aligned to the optimal learning direction during the second, but not the first, phase (**B**, **Methods**), which was also echoed by the L2 distance (**C**). Solid line: mean. Shaded areas: standard errors of the mean (s.e.m.) across 10 runs. **D** Next, to gain insight into the sudden change in Epoch 120, we visualized the network states over multiple trials (blue), neural trajectories during a single trial (green), and energy minima (red) within the top three principal component spaces. A bifurcation, signaled by a qualitative change in the energy landscape (red markers), separates the two learning phases.

2.2 Temporal consistency in recurrent networks incentivizes attractor subspace formation

Next, we first reproduced the results of [11] with PLRNNs initialized and/or regularized with the MAR (Fig. S2). We observed the emergence of memory neurons that had ramping, slow time dynamics, in line with the biological insights [6].

To better understand the internal workings of the regularized networks, we tested which subsets of neurons kept cue-associated numbers in memory, through cue decoding (**Methods**). Unsurprisingly, we confirmed that regularized networks had enhanced memory capacity and that memory neurons memorized the cue-associated numbers more faithfully than the computation neurons (Fig. S3).

Inspired by this observation, we devised a network-agnostic regularization procedure, termed temporal consistency regularization (TCR, **Methods**). We observed that networks trained with TCR had a higher chance of convergence compared to unregularized networks (Fig. S4A), with robust memory subspaces to noise injections (Fig. S4B). While TCR strength modulated phase 1 length, MAR was capable of overcoming phase 1 completely (Figs. S5, S6), likely because the specific line-attractor enforced by MAR was already a suitable candidate to solve the delayed addition task.

2.3 Online and local learning of cue-associated distinct memory states

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 in a local and online learning rule (**Methods**). Next, we applied this rule to learn cue-associated fixed-points, which arose rapidly through online training starting with a randomly connected chaotic network (Fig. 2A). Since online learning freezes the network in time, but only locally, we named this paradigm “freeze-in training".

Using the freeze-in training, we were able to train 25 distinct fixed-point attractors, each of which was associated with a cue (Fig. 2B). Notably, training converged quickly, leading to a negative maximum

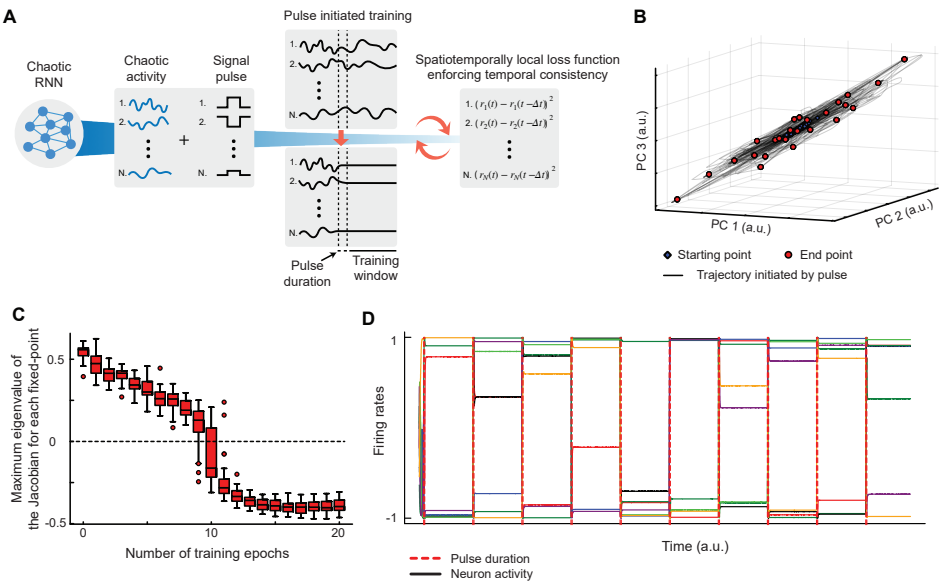

Figure 2: **Temporal consistency enables online learning of cue-associated fixed-points. A** The training procedure starts with a randomly initialized chaotic RNN. A fixed, rectangular pulse is applied through a randomly initialized non-trainable input weight matrix to the network (**Methods**). For a pre-defined duration after the pulse offset, the network is forced to become temporally consistent via spatiotemporally local consistency signals. **B** We visualized the final output of a fully trained network with 25 distinct, cue-associated fixed-point attractors. **C** As little as 12 instances of repeated cue presentation were sufficient to train the attractive fixed-points, as measured by the maximum eigenvalue of the Jacobian (**Methods**). 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. **D** We visualized 10 out of 25 cue-associated fixed-points. Pulses are shown during a very short duration, and drive the transitions between fixed-points.

eigenvalue of the Jacobian (**Methods**, Fig. 2C), hinting at the formation of several attractor subspaces. Though the attractive fixed-points were gated by cues, their existence was not dependent on them since the network was trained to remain in the fixed-points long after the cue offset (Fig. 2D), using the cue to build the association. Hence, we were able to train, in an unsupervised manner, several cue associations into the network without making use of any global signal or loss function.

3 Discussion

The study of bifurcations during learning has been limited in the recent literature, with some toy examples discussing the difficulty of learning during changes in the state-space topology [3] (Though see [17, 18] for excellent recent examples). In this work, we showed that a systematical study of learning in recurrent networks from the perspective of dynamical systems theory can lead to new insights into how the memory capacity of networks can be enhanced by incentivizing beneficial bifurcations.

To better understand the emergence of attractors subserving memory, we studied learning during a delayed addition task, which requires the network to learn a substantial memory capacity. Our results indicated that the training had two distinct phases: search and rapid comprehension. Notably, gradient was well aligned with the optimal direction in the second, but not the first phase, whereas enforcing time-consistency had the opposite behavior. Finally, we applied the idea of enforcing time-consistency in an online manner and obtained multiple cue-associated fixed-points, which were trained rapidly in few successive cue representations. Hence, our results provided a first confirmation that when the optimal topology for the parameter space is not yet represented in the loss function, *i.e.* the network has not gone through the right bifurcations yet, the gradient carries at most a weak learning signal, and simple yet efficient learning rules can drive the network through desirable bifurcations.

References

[1] 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.

[2] Steven H Strogatz. Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering. CRC press, 2018.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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. URL `https://openreview.net/forum?id=_XYzwxPIQu6`.

[12] Dayan Guan, Jiaxing Huang, Aoran Xiao, and Shijian Lu. Domain adaptive video segmentation via temporal consistency regularization, 2021.

[13] Kwanyong Park, Sanghyun Woo, Dahun Kim, Donghyeon Cho, and In So Kweon. Preserving semantic and temporal consistency for unpaired video-to-video translation. In Proceedings of the 27th ACM International Conference on Multimedia. ACM, oct 2019. doi: 10.1145/3343031.3350864.

[14] Dongdong Chen, Jing Liao, Lu Yuan, Nenghai Yu, and Gang Hua. Coherent online video style transfer, 2017.

[15] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[16] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In Lecture Notes in Computer Science, pages 26–36. Springer International Publishing, 2016. doi: 10.1007/978-3-319-45886-1_3.

[17] 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.

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

[19] 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.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

A Methods

A.1 Task description

We conducted all our task-based experiments on the delayed addition task. The task consisted of 100,000 training and 10,000 testing data samples. Each data sample $U = \{(u_1^1, u_1^2), (u_2^1, u_2^2), ..., (u_T^1, u_T^2)\}$ had a size of $T \times 2$. Here, the entries $u_t^1 \in [0, 1]$ were sampled from a uniform random distribution, while the entries $u_t^2 \in \{0, 1\}$ formed binary series. The values of u_t^2 were sampled such that $\forall t, u_t^2 = 0$, except for $t = t_1$ and $t = t_2$. The values of t_1 and t_2 were randomly sampled with the constraint $t_1 < 10$ and $t_2 < T/2$. The target output at the final time step T should be $o_T = u_{t_1}^1 + u_{t_2}^1$. This approach of modeling the task requires the network to maintain long-term memory of at least $T/2$ time steps and was used in [11] for benchmarking manifold-attractor regularization.

A.2 Model description

We considered PLRNNs [19, 11] for all the experiments with delayed addition tasks. The dynamical system equations were:

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

Here, $x[t] \in \mathbb{R}^N$ is the network currents with N number of neurons, $\phi(x[t])$ is the network firing rates with ReLU non-linearity $\phi(.)$ (i.e. $\phi(x[t])_i = \max(0, x_i[t]), i \in 1, ..., N$), $A \in \mathbb{R}^{N \times N}$ is a diagonal matrix which encodes decay time constants of neurons, $W \in \mathbb{R}^{N \times N}$ is the recurrent weight matrix, $s[t] \in \mathbb{R}^K$ is the external input with dimension K, $C \in \mathbb{R}^{N \times K}$ is the input mapping, and $h \in \mathbb{R}^N$ is a bias.

To obtain the output of the network, we linearly projected the final currents $x[T]$ at T^{th} time-step, where T is the temporal length of the input/ task:

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

$\hat{o}[T] \in \mathbb{R}$ is the output of the network, $W_{out} \in \mathbb{R}^{1 \times N}$ is the output mapping, $b_{out} \in \mathbb{R}$ is a output bias.

Unless otherwise specified, the following were learned: A, W, C, h, W_{out} and b_{out}. For all the experiments with delayed addition tasks, we considered $N = 40$ neurons.

We trained all our networks to minimize the mean squared error between the target (o_T) and the network output $\hat{o}[T]$. Network performances were quantified by the percentage of predictions that have < 0.04 error with the target output.

A.3 Regularization schemes

Throughout the paper, we considered two types of regularization, proposed temporal consistency regularization (TCR) and manifold attractor regularization (MAR) [11]. Whenever applicable, we regularized N_{reg} neurons with $N_{reg} < N$, unless otherwise specified.

Temporal Consistency Regularization The proposed Temporal Consistency Regularization (TCR) loss function is defined as:

$$\mathcal{L}_{TCR} = \frac{\tau}{T} \sum_{t=1}^{T} \sum_{i=1}^{N_{reg}} (x_i[t] - x_i[t-1])^2 \tag{S3}$$

Manifold Attractor Regularization Manifold attractor regularization (MAR) [11] is defined as:

$$\mathcal{L}_{MAR} = \tau \sum_{i=1}^{N_{reg}} (A_{i,i} - 1)^2 + \tau \sum_{i=1}^{N_{reg}} \sum_{\substack{j=1 \\ j \neq i}}^{N} W_{i,j}^2 + \tau \sum_{i=1}^{N_{reg}} h_i^2 \tag{S4}$$

For networks which trained with MAR, W was taken as an off-diagonal matrix to be consistent with [11]. We used an off-diagonal W for other networks as well when we performed direct comparisons with MAR.

Manifold Attractor Initialization Manifold Attractor Initialization (MAI) is an initialization method [11]. In this initialization, the networks are initialized such that $\mathcal{L}_{MAR} = 0$ at the beginning.

A.4 Hyperparameter selection

We trained all the networks with Adam optimizer [20], with a train batch size of 500 and a test batch size of 100.

For bifurcation analysis (Fig. 1), we considered an unregularized PLRNN 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 $[-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}]$ [21]. 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. We analyzed gradients during the learning (Fig. 1B, C, Fig. S1) and visualized the attractor formations (Fig. 1D).

To find out the effect of TCR (Fig. S4), 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 [11], $T = 40$, 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 MAR, we used $\tau = 5.0$ as the regularization strength [11]. For TCR, we used $\tau = 1.0$ as the regularization strength. For all the networks with TCR/ MAR, we regularized 50% of the network, *i.e.* $N_{reg} = 40 \times 0.5 = 20$. We repeated the experiments 10 times and reported the mean and the standard error of the mean (Fig. S4A).

To reproduce MAR results (Fig. S2A), we considered three networks; unregularized PLRNN, PLRNN with MAI and PLRNN with MAI and MAR. We followed hyper-parameters used in [11] for $T \in \{20, 40, 60, 100, 200, 300, 400, 500\}$. Experiments were repeated 10 times. Mean and standard errors were reported (Fig. S2A). We analyzed the frequency components of the networks (Fig. S2C), and memory capabilities (Fig. S3).

For noise-injection experiments (Fig. S4B,C), we considered the same configurations used for Fig. 1 bifurcation analysis with a learning rate of 0.001 and a weight decay of 10^{-8}. Experiments were repeated 3 times. Mean and standard errors were reported.

To analyze the impact of TCR for the bifurcation point (Figs. S5, S6), 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 ($\tau = 5.0$), and PLRNNs with TCR ($\tau \in \{0.001, 0.01, 0.1\}$). For all the networks with MAR/ TCR/ MAI, we regularized only the half of the network. We repeated all the experiments 5 times and reported the results in Figs. S5 and S6.

A.5 Energy minimization

To extract the slow points of the network, we followed energy minimization described by [1]. 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 as the initialization points for energy minimization (i.e. $x^e[0]$). For time step t, we computed the energy $E[t]$ as follows.

$$E[t] = ||x^e[t+1] - x^e[t]||_2^2 \tag{S5}$$

Starting from each $x^e[0]$, we minimized the energy using stochastic gradient descent:

$$x^e[t+1] = x^e[t] - \alpha \frac{\partial E[t]}{\partial x^e[t]}\bigg|_{x^e[t]} \tag{S6}$$

with $\alpha = 0.1$ and performed 1000 iterations. This resulted in 500 trajectories. Final states $x^e[1000]$ for each initialization were visualized as red crosses using PCA in Fig. 1.

A.6 Visualization analysis

To visualize attractor manifolds, we applied the dimensionality reduction method, Principle Component Analysis (PCA), and reduced the dimensionality of $x(t)$ into 3.

To fit the PCA, we first picked a data batch with 100 trials from the test set. Second, we obtained corresponding currents $x[t]$s for trials from a *fully-trained* network. PCA is fitted for all the states of $x[t]$s. The resultant fitted PCA model is then applied to reduce dimensions of $x^e[t]$s and $x[t]$s obtained from models that were not fully trained (Fig. 1D).

A.7 Gradient similarity analysis

We performed gradient similarity analysis over the training to measure how alignment between instantaneous gradients and the ideal gradient changes with respect to the bifurcation point.

To compute this, we picked trained models after each 20 epochs. For W, ground truth and instantaneous gradients are generated as follows:

$$dW_{inst} = W_{epoch} - W_{epoch}^{tuned}, \tag{S7a}$$

$$dW_{gt} = W_{epoch} - W_{final} \tag{S7b}$$

dW_{inst} and dW_{gt} are instantaneous and ideal gradients. We computed W_{epoch}^{tuned} by optimizing weights for 10 epochs starting from W_{epoch}, using the Adam optimizer with a learning rate of 0.0015. We repeated the optimization 10 times with multiple starting seeds and averaged over to obtain W_{epoch}^{tuned}.

Different distance measures between the ideal and the instantaneous gradient are plotted over the epochs:

$$d = D(dW_{inst}, dW_{gt}) \tag{S8}$$

Here, $D(.)$ is the cosine similarity or L2 distance. For the comparison, we also obtained a shuffled control:

$$d_{control} = D(Perm(dW_{inst}), dW_{gt}) \tag{S9}$$

$Perm(.)$ randomly shuffles the values of the array given.

A.8 Stimulus Decoding Experiments

To test the memory capabilities of attractor-incentivized networks compared to unregalarized PLRNNs, we conducted cue/ stimulus decoding experiments. For this, we picked 10 unregularized PLRNNs and 10 PLRNNs trained with MAR and MAI for the addition task with $T = 60$ from Fig. S2A.

We first created an 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 $u_{t_1}^1, u_{t_2}^1$, and $u_{t_1}^1 + u_{t_2}^1$ from $x_S[t] = x_i[t]_{i=\{1,...,N\}}$, $x_{S_1}[t] = x_i[t]_{i=\{1,...,N_{reg}\}}$, and $x_{S_2}[t] = x_i[t]_{i=\{N_{reg}+1,...,N\}}$ separately. Analysis was conducted for all the time steps, $t \in 1, ..., T$.

To predict cues/ targets from neuronal activity, we considered linear regression models. For a given network and for a given t, 9 linear regression models were independently fitted to map 3 types of

currents (i.e. $x_S[t], x_{S_1}[t], x_{S_2}[t]$) into 3 types of 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 plane PLRNNs and 10 memory-incentivized PLRNNs). All the fitting was done on the training dataset with 10000 trials and testing was done on the testing dataset with 1000 trials. We plotted the mean performance curves on test datasets across 10 networks in Fig. S3.

A.9 Frequency Analysis

We analyzed frequency distributions of currents of PLRNN, PLRNN with MAI, and PLRNN with MAR and MAI. To conduct this, we selected 3 networks from Fig. S2A, 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. We showed currents for 10 neurons in Fig. S2C (left).

To obtain frequency plots, we separately considered currents from computational and memory neurons. Trajectories of the neuronal currents were first divided by the maximum current of the corresponding trajectory. Fast Fourier transform followed by frequency shift was then applied. We obtained absolute values from the resultant frequency sequence. The resultant frequency sequences are then divided again by the maximum frequency component of the corresponding frequency sequence. Finally, we took the mean frequency sequence over neurons. Resultant frequency plots are shown in Fig. S2C (right).

A.10 Intervention experiments

We performed intervention experiments 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, 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, we injected a constant noise/ 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. We considered a range of ϵ and γ for the experiment. The procedure was repeated for 6 networks (3 plane PLRNNs and 3 PLRNNs with TCR). We plotted the mean performance fluctuation under noise strength with standard error in Fig. S4B, C.

A.11 Quantification of phase durations and learning speeds

We summarized the learning curves in Fig. S5 in terms of the epochs taken to achieve bifurcation, convergence speed, and maximum training performance (Fig. S6).

Given a converged network, we obtained the convergence speed s_{conv} by:

$$s_{conv} = \frac{P_{90\%} - P_{10\%}}{e_{P_{90\%}} - e_{P_{10\%}}} \tag{S10}$$

Here, $P_{\mu\%} = 0.01\mu P_{max}$ is the $\mu\%$ of the maximum training performance P_{max}. $e_{P_{\mu\%}}$ was the lowest epoch that gives performance greater than $P_{\mu\%}$. Duration of the phase 1 was $e_{P_{10\%}}$. Fig. S6A, B and C show $e_{P_{10\%}}$, s_{conv} and P_{max} respectively.

A.12 Online learning rules

The RNNs for our online experiments obeyed the "neural inspired" differential equations

$$\begin{aligned} \tau_{\mathrm{c}} \frac{dr_i(t)}{dt} &= -r_i(t) + f(z_i(t)), \\ z_i(t) &= \sum_{j=1}^{N_{\mathrm{rec}}} W^{\mathrm{rec}}_{ij} r_j(t) + \sum_{j=1}^{N_{\mathrm{in}}} W^{\mathrm{in}}_{ij} u_j(t) + \epsilon_i(t). \end{aligned} \tag{S11}$$

Here, r_i is the activity or firing rate of neuron i and z_i is the total input current to neuron i. $N_{rec} = 400$ and $N_{in} = 1$. The input vector is denoted by $\vec{u}$ (rectangular pulse) and $\epsilon_i(t)$ is i.i.d Gaussian noise. Furthermore, the time constant $\tau_c = 10$, and $f(\cdot) = \tanh(\cdot)$. Each element of W_{rec} was initially drawn from a normal distribution with mean 0 and standard deviation $1.8/\sqrt{N_{rec}}$ while each element of W_{in} was drawn from a standard normal. We utilized multiple W_{in}s to train multiple fixed points. To prevent self-excitation, we also enforced $W_{ii}^{rec} = 0$.

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:

$$r_i[t + \Delta t] = (1 - \alpha)r_i[t] + \alpha f(z_i[t + \Delta t]), \tag{S12}$$

where $\alpha = \frac{\Delta t}{\tau_c}$ is the unitless normalized discretization time, commonly chosen a small value such as $\alpha = 0.1$.

The learning process began after $\vec{u}$ 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 step of gradient descent on

$$\lambda(r_i[t] - r_i[t - 1])^2 \tag{S13}$$

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

$$W^{rec}[t + \Delta t] = W^{rec}[t] - \lambda(\vec{r}[t] - \vec{r}[t - \Delta t])\vec{r}[t - \Delta t]^T. \tag{S14}$$

We repeated this update several times, in a randomized manner across cues, as shown in Fig. 2. Each pass across all cues was considered a single epoch.

A.13 Computation of the Jacobian

We computed the Jacobian at some given $\vec{r}_*$ by taking partial derivatives of equation (S11). The resulting matrix was given by

$$J = \frac{1}{\tau_c}\left(W^{rec}(1 - \tanh^2(W^{rec}\vec{r}_*) \otimes M - I_{nxn})\right) \tag{S15}$$

where $\otimes$ is element-wise multiplication and M is a $n\ x\ n$ matrix of 0's on the diagonal and 1's everywhere else.

A.14 Details on figure reproduction

We plan to release the code to reproduce each figure in our GitHub.

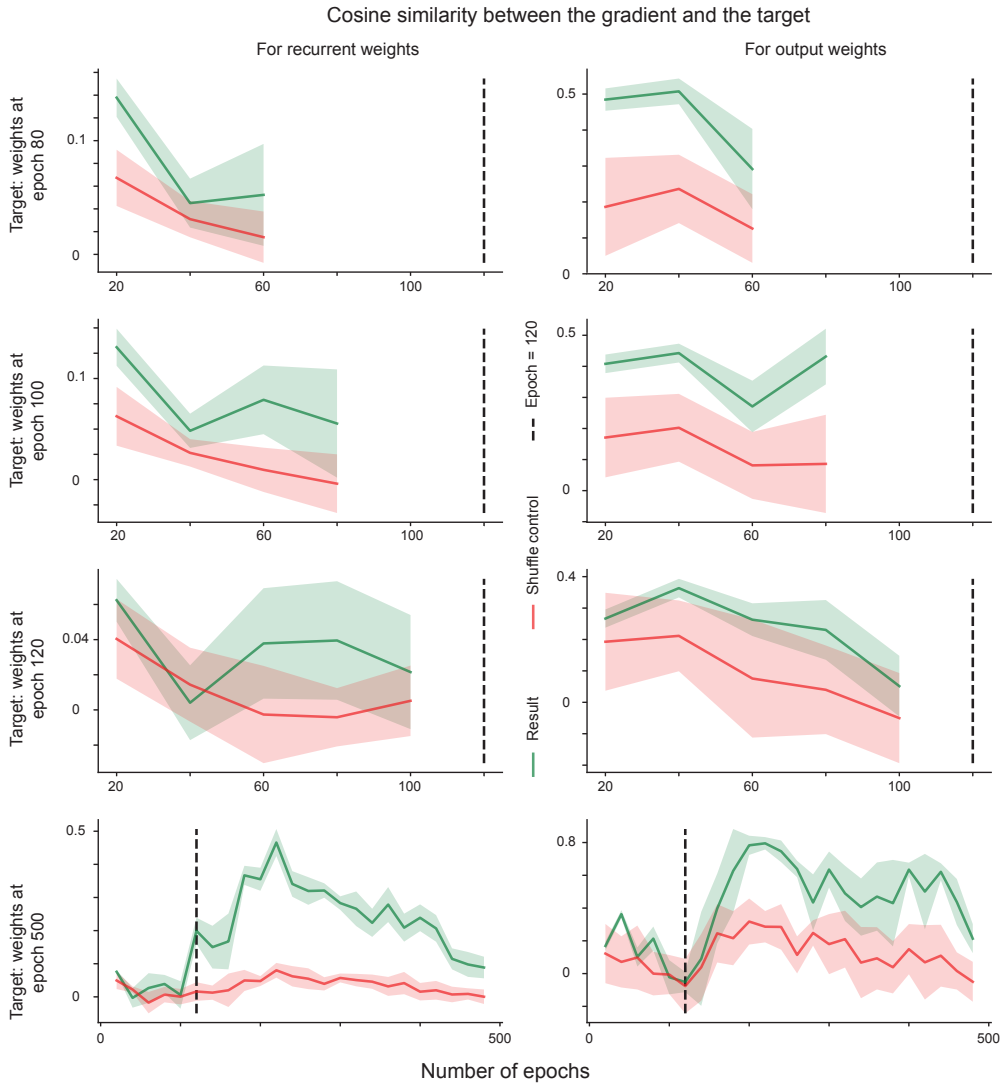

Figure S1: **During the search phase, gradients contribute at most weakly, if at all, to the search.** We computed the cosine similarity of the gradient with the optimal direction for the recurrent weight matrix W, column 1, and the output weight matrix W_{out}, column 2. Each row corresponds to a different target weight for the similarity analysis. Before the bifurcation, the gradients lack a clear direction towards the final weights. Surprisingly, they also fail to update the weights efficiently toward the bifurcation point. After the bifurcation, gradients show an alignment with the ideal gradient direction (row 4). Solid line: mean. Shaded area: s.d. over 10 runs.

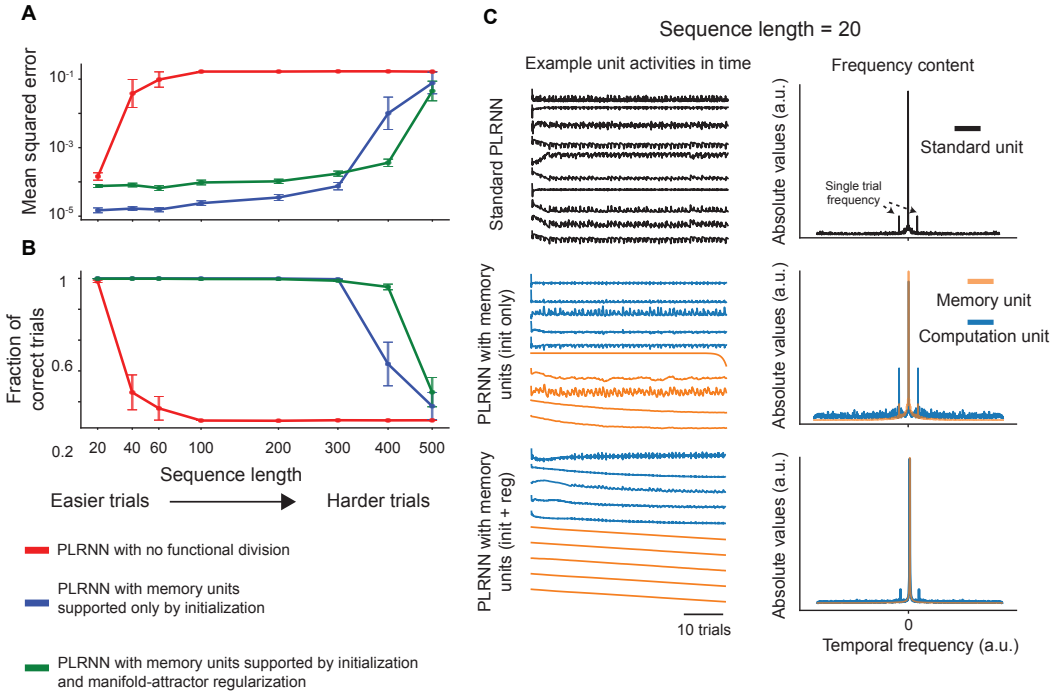

Figure S2: **Manifold attractor regularization incentivizes slow-time dynamics. A, B** We plotted the mean-squared error (**A**) and the fraction of correct trials (**B**) for plain PLRNNs, as well as manifold-attractor initialized (MAI, **Methods**, [11]) and regularized (MAR, **Methods**, [11]). Both MAI and MAR significantly improve the convergence capabilities of PLRNNs by initializing (MAI) network dynamics near a plane attractor and enforcing (MAR) it during training. Solid lines: mean. Error bars: s.e.m over 10 runs. **C** Activity frequencies of networks performing the addition task (T=20) show that MAI and MAR incentivize slow-time dynamics in a subset of neurons, termed memory units.

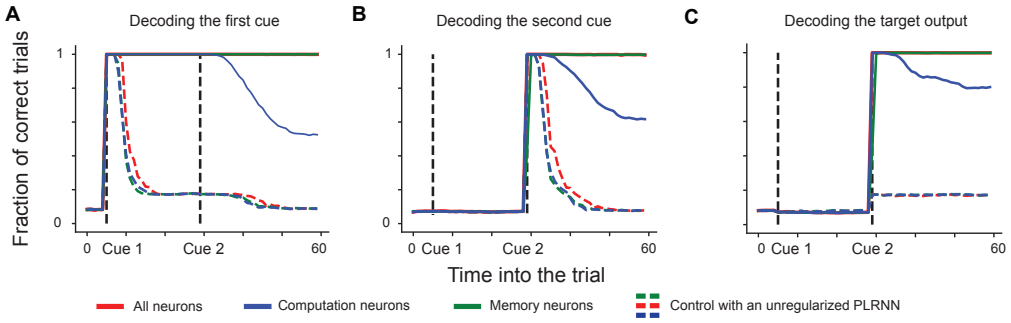

Figure S3: **Stimulus decoding experiments show that MAR forms memory neurons that are capable of keeping cues and outputs for longer periods of time**. A. Decoding cue 1 from currents from all the neurons ($x_S[t]$), currents from memory neurons ($x_{S_1}[t]$), and currents from computation neurons ($x_{S_2}[t]$). Plane PLRNNs fail to keep cue 1 in the memory. Memory neurons of MAR-PLRNNs keep cue 1 until the end of the trial while computational neurons of MAR-PLRNNs forget cue 1 after the network receives cue 2. B. Decoding cue 2 from $x_S[t], x_{S_1}[t]$ and $x_{S_2}[t]$. Plane PLRNNs and computational units of MAR-PLRNNs forget cue 2. Memory neurons of MAR-PLRNNs remember cue 2. C. Decoding addition targets from $x_S[t], x_{S_1}[t]$ and $x_{S_2}[t]$. Computational neurons of MAR-PLRNNs generate targets immediately after the network receives cue 2. However, they eventually forget it. Memory neurons start to generate the target after the computational neurons and remember it until the trial ends. Solid lines: mean over 10 runs.

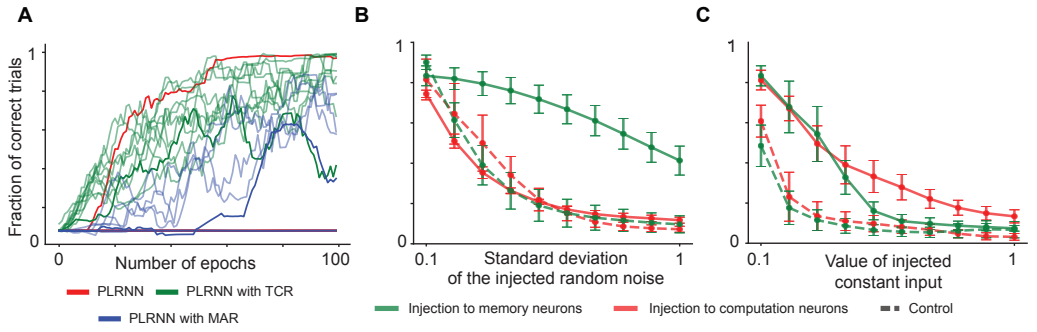

Figure S4: **Temporally consistent recurrent neural networks converge faster with a robust memory subspace. A** We trained 10 PLRNNs on the delayed addition task (T=40). All manifold-attractor regularized networks [11]) unsurprisingly skipped the search phase. Enforcing temporal consistency increased the number of networks reaching the second phase from one to five, out of ten total networks. **B, C** Next, we fine-tuned the network hyperparameters to find a configuration where PLRNNs with no regularization converge (**Methods**). In this case, temporal consistency, but not the unregularized network, led to the formation of a memory subspace that was robust to both random (**B**) and constant (**C**) noise injections.

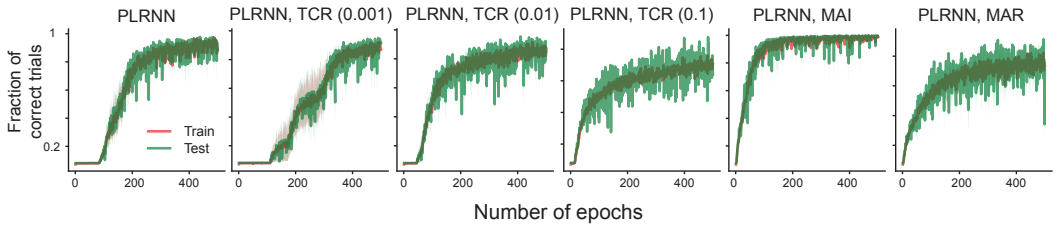

Figure S5: **Enforcing temporal consistency results in shorter phase-1.**. The training curves of PLRNNs with various forms of regularizations. Solid lines: mean. Shaded areas: s.e.m. over 5 runs.

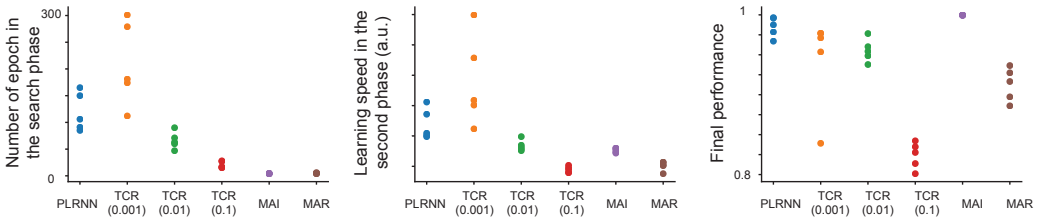

Figure S6: **Temporal consistency enforcement strength correlates negatively with the length of the search phase and the speed in the second phase.**. Phase one duration (*left*) and the learning speed in the second phase (*middle*) decreases with increased regularization. *Right.* Once the network is able to enter the second phase, final training performance saturates rapidly.