Neuron

Article

A tale of two algorithms: Structured slots explain prefrontal sequence memory and are unified with hippocampal cognitive maps

Highlights

- A duality between the prefrontal (working) and hippocampal (episodic) sequence memory algorithms
- Mechanistic understanding of prefrontal working memory as controllable activity slots
- Recurrent neural networks learn activity slots on a wide array of sequence memory tasks
- Prefrontal working memory representations explained as controllable activity slots

Authors

James C.R. Whittington, William Dorrell, Timothy E.J. Behrens, Surya Ganguli, Mohamady El-Gaby

Correspondence

jcrwhittington@gmail.com

In brief

The algorithm of the prefrontal working memory system on sequence memory tasks is not well understood, whereas it is well understood for the hippocampal episodic memory system. This work shows a mathematical duality between working and episodic sequence memory. This leads to a computational understanding of sequence working memory—as structured activity slots. This theory algorithmically explains recurrent neural network (RNN) and prefrontal representations during sequence memory tasks.

Whittington et al., 2025, Neuron *113*, 321–333
January 22, 2025 © 2024 The Author(s). Published by Elsevier Inc.
https://doi.org/10.1016/j.neuron.2024.10.017

CellPress

Neuron

CellPress
OPEN ACCESS

Article

A tale of two algorithms: Structured slots explain prefrontal sequence memory and are unified with hippocampal cognitive maps

James C.R. Whittington,[1,2,5,*] William Dorrell,[3] Timothy E.J. Behrens,[2,4] Surya Ganguli,[1] and Mohamady El-Gaby[2]
[1]Department of Applied Physics, Stanford University, Palo Alto, CA, USA
[2]Wellcome Centre for Integrative Neuroimaging, University of Oxford, Oxford, UK
[3]Gatsby Computational Neuroscience Unit, University College London, London, UK
[4]Sainsbury Wellcome Centre for Neural Circuits and Behaviour, University College London, London, UK
[5]Lead contact
*Correspondence: jcrwhittington@gmail.com
https://doi.org/10.1016/j.neuron.2024.10.017

SUMMARY

Remembering events is crucial to intelligent behavior. Flexible memory retrieval requires a cognitive map and is supported by two key brain systems: hippocampal episodic memory (EM) and prefrontal working memory (WM). Although an understanding of EM is emerging, little is understood of WM beyond simple memory retrieval. We develop a mathematical theory relating the algorithms and representations of EM and WM by unveiling a duality between storing memories in synapses versus neural activity. This results in a formalism of prefrontal WM as structured, controllable neural subspaces (activity slots) representing dynamic cognitive maps without synaptic plasticity. Using neural networks, we elucidate differences, similarities, and trade-offs between the hippocampal and prefrontal algorithms. Lastly, we show that prefrontal representations in tasks from list learning to cue-dependent recall are unified as controllable activity slots. Our results unify frontal and temporal representations of memory and offer a new understanding for dynamic prefrontal representations of WM.

INTRODUCTION

Predicting what will happen next in novel environments is a fundamental feature of intelligent cognition. However, due to the one-dimensional (1D) and irreversible nature of time itself, we can only learn about structured environments through sequential experience. Thus, brains[1–3] and machines[4–6] must convert sequential experience into internal models of the world to remember and exploit structured relationships. When such an internal model, or cognitive map, emerges, it can allow additional flexibility beyond next-step prediction, such as inferring new routes to goals[7] or simulating counterfactual scenarios.[8] An algorithmic understanding of how sequential experience is converted into a rich cognitive map capable of predicting future counterfactual consequences of diverse potential actions remains a major aim of cognitive neuroscience.

Two key brain regions build cognitive maps from sequential experience: the episodic memory (EM) system in the medial temporal lobe and the working memory (WM) system in the prefrontal cortex (PFC).[9–11] However, it is not clear how these brain systems are related in representation or underlying algorithm.

For sequential EM in hippocampus (HPC), ideas are emerging on the underlying algorithms and representations.[4,12–14] In these models, memories are stored in HPC by updating synaptic connections (e.g., Hopfield networks[15]), with cortical recurrent neural networks (RNNs) controlling which memory to store or retrieve by tracking position (ordinal, spatial, or otherwise) within the sequence. These models explain many cellular recordings for both spatial and non-spatial tasks: hippocampal cells, such as place cells,[16] landmark cells,[17] and splitter cells,[18] are explained as memory representations, whereas entorhinal cells, such as grid cells,[19] object-vector cells,[20] border-vector cells,[21] non-spatial grid cells,[22,23] and non-spatial sound frequency cells,[24] are explained as RNN position representations. Importantly, the hippocampal literature shows us that sequence memory is not just remembering sequences exactly as they were presented but rather using structured knowledge (e.g., position) to recall the right memory at the right time. Successful sequence memory goes beyond rote memorization of experience but instead reflects a cognitive map-building process that learns systematic relationships between events induced by the underlying task structure.[4,5,12–14]

For sequence WM in PFC, our understanding is limited to remembering sequences exactly as they were presented. Here, findings from both artificial[25–27] and biological networks suggest memories of items are organized into,[28] and moved between,[29,30] neural subspaces according to their ordinal position. Crucially, unlike HPC, storing these memories requires no

This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

synaptic plasticity because recurrent connections maintain memories in neural activity. Although this is revealing, PFC is implicated in tasks beyond remembering sequences exactly as they were presented, i.e., tasks that required flexible control of memory retrieval.[29,31,32] Because we understand how HPC EM models allow such flexible control, if we could relate PFC WM and HPC EM, then we could understand how the PFC WM system flexibly controls memories to solve tasks requiring cognitive map-like knowledge of structured relationships.

In this work, we provide an understanding of sequence WM in PFC and how it relates, in representation and algorithm, to HPC EM. In particular, we (1) develop a unifying theory of storing sequence memories in synapses (EM) and neural activity (WM); (2) demonstrate that the two algorithms differ in their ability to scale to larger task sizes; (3) demonstrate how the different algorithms utilize different neural representations, with EM using abstract positions, whereas WM uses slots (distinct neural subspaces) in neural activity; (4) demonstrate that WM slot representation affords "scene algebra"; (5) demonstrate how internally computed velocity signals control the contents of PFC slots; and (6) show that our theory of controllable activity slots provides a common explanation for PFC data from several disparate studies. Overall, a main dividend of our derived duality between HPC EM and PFC WM is a new, unifying theoretical framework for how PFC WM networks could implement and control dynamic cognitive maps of the environment or task through recurrent updates of neural activity alone, without any synaptic plasticity.

Theory: Unifying memories stored in synapses and neural activity

Sequence memory is a structure learning problem

Sequence memory tasks depend on underlying task structures. For example, in immediate serial recall[33] (ISR; a series of observations must be recalled in order) the underlying structure is an ordinal line. Other sequences have different underlying structures, e.g., a sequence drawn from navigating two-dimensional (2D) space has embedded 2D structure. Internal representations that include this structural knowledge dramatically facilitate recall and prediction as different structures have different transition functions. For example, in a 2D navigation task, correct predictions rely on knowing you have returned to a previous position, which itself requires knowledge of the structure of 2D space (i.e., how recent velocities integrate). Exploiting knowledge of task structure facilitates recall and prediction in problems with a common structural constraint, even when each problem consists of sequences with different observations (therefore, one problem's sequence cannot be memorized and used for another problem). Here, the underlying task structure must be meta-learned across problems (learning to learn; Figure 1A left for example ISR task).

Task and problem formalism

Formally, for each task, we consider a dataset $\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1, \cdots, \mathcal{D}_N\}$. Each $\mathcal{D}_i$ is one of N problem instances of the common task and is a sequence consisting of vectors of observations, $\boldsymbol{o}$, of dimension d_o (termed $\boldsymbol{o} \in \mathbb{R}^{d_o}$), targets, $\boldsymbol{t} \in \mathbb{R}^{d_o}$, and (allocentric) velocities, $\boldsymbol{v} \in \mathbb{R}^{d_v}$, i.e., $\mathcal{D}_i = \{(\boldsymbol{o}_0, \boldsymbol{t}_0, \boldsymbol{v}_0), (\boldsymbol{o}_1, \boldsymbol{t}_1, \boldsymbol{v}_1), \cdots, (\boldsymbol{o}_K, \boldsymbol{t}_K, \boldsymbol{v}_K)\}$ for a sequence of length K. Although velocities reflect an agent's actions (conceptual or physical), we provide them externally. Importantly, the underlying structure of the task is determined by how successive actions cumulatively change a latent task state, which is never directly observed by the agent.

Concretely, we assume an underlying latent variable, $\boldsymbol{z} \in \mathbb{R}^{d_z}$, corresponding to a latent agent/task state (e.g., position in spatial contexts). Furthermore, we assume each latent state, or position, $\boldsymbol{z}$, is associated with an observation, $\boldsymbol{o}$, which the agent sees if it has the corresponding latent state/position $\boldsymbol{z}$. For any ordered pair of neighboring source and target latent states, there is an action, or velocity, $\boldsymbol{v}$, that modifies the agent's latent state from the source to the target (e.g., Figure 1A left for a loop structure with one action). The task structure is encoded by how actions/velocities, change the latent state/position. For example, when navigating in a 2D grid, there are four elementary actions/velocities—one step north, south, west, or east—with each step moving the latent 2D grid position $\boldsymbol{z}$ by one step in the corresponding direction. When a sequence of velocities additively cancel (e.g., north + east + south + west = 0 for a 2D grid), the agent returns to the same latent position and encounters the same observation as before.

For a single task $\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1, \cdots, \mathcal{D}_N\}$, each individual problem instance $\mathcal{D}_i$ assumes a different and random pairing of observations $\boldsymbol{o}$, to latent states $\boldsymbol{z}$. However, all N problem instances share the *same* underlying task structure. The aim of the task is to predict a target, $\boldsymbol{t}$, at each timestep: either an upcoming observation (i.e., what you will see after going north), or a past observation (i.e., what you saw 5 steps ago). Importantly, neither latent states nor how actions affect latent states are directly observed. Instead, the agent experiences a sequence of observations and actions. Solving the task, across all problems, requires building a cognitive map capturing how actions affect latent states, and how latent states are bound to observations. Only then, can the agent predict what it will see next when first returning to a previous position. We note that the task formalism (and subsequent model formalism) is general to both discrete and continuous tasks, although we primarily consider discrete tasks here.

We now develop a theory for a simple network realization of an EM solution and from that derive a simple network realization of a WM solution. Crucially, we later show that many features of these simple solutions hold in more general scenarios, both in artificial and biological networks that successfully solve structured sequence memory tasks.

Solving sequence memory tasks with EM

The hippocampal literature has developed EM models that solve these tasks.[4,14] These models have two components (Figure 1A top-middle): an RNN that learns to explicitly represent position, which generalizes across tasks, and a memory system that binds observation to position representations and stores this memory in synaptic weights. These memory weights change in every problem because different problems have different associations of observation to latent state/position. In its simplest form, the RNN has a single neuron active for each position (Figure 1A middle; full theory in STAR Methods). Memories are encoded in synaptic weights between each RNN neuron (corresponding to a position) and an observation representation, with memories added via Hebbian learning.[34] The RNN tracks the underlying

CellPress
OPEN ACCESS

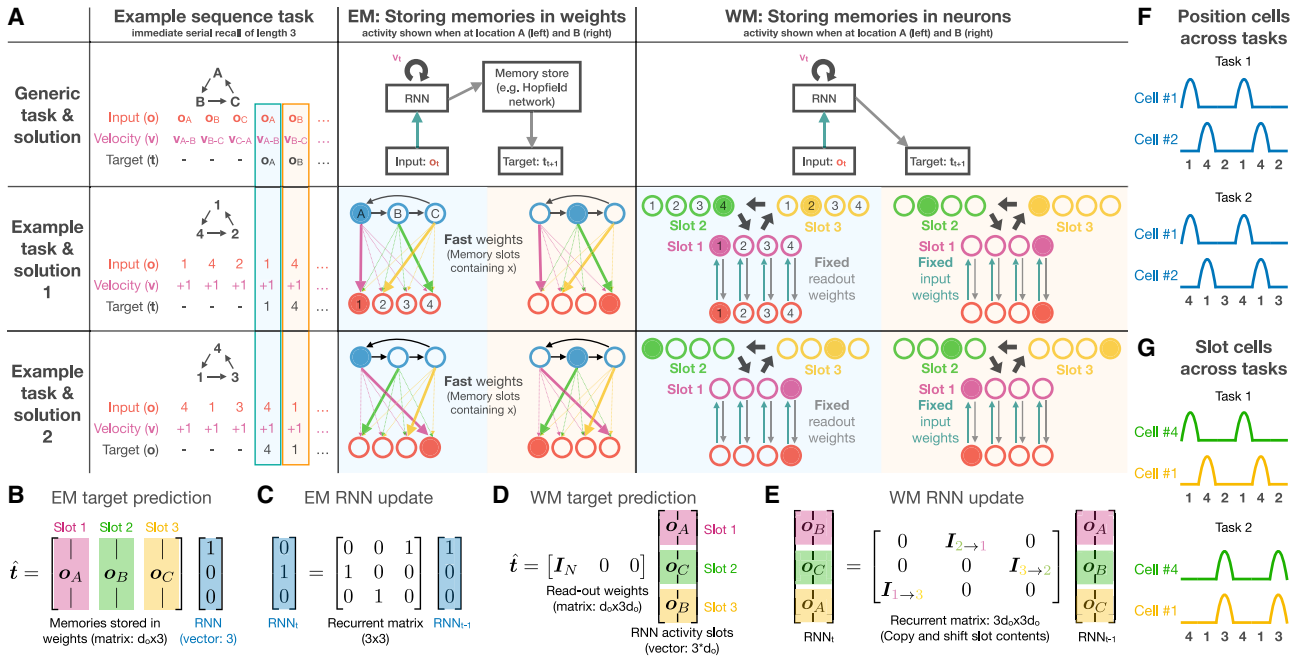

Figure 1. Relationship between episodic memory and working memory models

(A) Schematic of how EM and WM solutions solve a task. Solid/empty circles are active/inactive neurons. Black/teal/gray arrows are recurrent/input/output connections. Left/center/right: task/EM solution/WM solution. Top/middle/bottom: general example/specific example 1/specific example 2. Left: immediate serial recall (ISR) task of length 3. The input is a sequence of three random observations, $\boldsymbol{o}$, that repeat, and a constant "forward" velocity, $\boldsymbol{v}$. The target, $\boldsymbol{t}$, to predict is the upcoming observation. Blue/orange segments highlight the timesteps shown in EM/WM solution schematic (center/right). Center/top: EM models have an RNN that indexes memories stored in synaptic weights. Middle/bottom: simplified schematic of EM solution (full model and derivation of simplification in STAR Methods). Past observations are stored in synaptic memory slots, and recurrent position representations index memory slots to retrieve a memory. Red neurons one-hot encode observations (each neuron represents one observation: 1, 2, 3, or 4). Blue neurons one-hot encode position (each neuron represents one position: A, B, or C). Right/top: WM models are RNNs. Middle/bottom: schematic of WM solution. Past observations are stored in activity slots within the RNN, with slot contents copied and shifted by the RNN (at every timestep) to predict targets with fixed readout weights. This example RNN has three activity slots (green/yellow/purple) each consisting of four neurons that one-hot encode four possible observations (1, 2, 3, and 4).

(B–E) Mathematics of the EM and WM solutions. (B) The EM solution uses an RNN position code (blue vector) to index memories stored in synaptic weight slots (matrix columns). (C) The RNN is updated by recurrent connections. (D) The WM solution uses fixed readout weights that attend to a single RNN activity slot (slot 1). (E) Slot contents are copied and shifted by a recurrent matrix.

(F) Two example position neurons (cells #1 and #2 are the left/middle blue neurons in [A] middle). Cell #1 / #2 is active at position A/B. The two neurons have similar activity patterns in both tasks (top/bottom), up to a global shift in their firing fields—their relative phase relationship across tasks is preserved.

(G) Two example activity slot neurons (cell #4 is the rightmost green neuron, and cell #1 is the leftmost yellow neuron in [A] right). Cell #4 always fires two steps after observation 4. Cell #1 always fires one step after observation 1. These cells do not keep their phase relationships between tasks because they code for specific distance/lag from specific observations.

latent variable, $\boldsymbol{z}$, so that the correct position neuron is activated at the right time to retrieve the right memory (Figure 1A middle). Tracking position means the RNN integrates velocity signals, $\boldsymbol{v}$, to update its representation from $h(z-v)$ to $\boldsymbol{h}(\boldsymbol{z})$ (i.e., from position $z-v$ to position $\boldsymbol{z}$). This can be done with velocity-dependent matrices, $\boldsymbol{W_v}$, that follow the transition rules of $\boldsymbol{z}$ (e.g., $W_v W_{-v} = I$; going north then south returns you to the same position). Mathematically, the RNN update is $h(z) = W_v h(z-v)$ (Figure 1C). Generating target predictions is succinctly described in the following equation (assuming all observations, $\boldsymbol{o}$, have already been added as memories):

$$\hat{\boldsymbol{t}}_{t+1} = \underbrace{\left[\boldsymbol{o}_{\boldsymbol{z}_0} \quad \cdots \quad \boldsymbol{o}_{\boldsymbol{z}_{n_p}}\right]}_{\text{EM slots} \in \mathbb{R}^{d_o \times n_p}} \underbrace{\prod_{\tau=0}^{t} \boldsymbol{W}_{\boldsymbol{v}_\tau}}_{\text{Recurrent matrix} \in \mathbb{R}^{n_p \times n_p}} \underbrace{\left[1 \quad 0 \quad \cdots \quad 0\right]^T}_{\text{Initial RNN}, \boldsymbol{h}_0 \in \mathbb{R}^{n_p}} \quad \text{RNN}, \boldsymbol{h}_{t+1} \in \mathbb{R}^{n_p}$$

(Equation 1)

Here, an initial RNN representation, $\boldsymbol{h}_0$, is successively updated by velocity-dependent matrices, $\boldsymbol{W_v}$, to represent position at timestep $t+1$: $\boldsymbol{h}_{t+1}$. To make predictions, $\boldsymbol{h}_{t+1}$ selects (similar to an attention vector) one of the n_p observations stored in memory slots in synaptic weights (Figure 1B; synaptic memory slots are non-overlapping sets of synaptic connections that can be rapidly updated to store arbitrary memories, and each slot corresponds to a position $\boldsymbol{z}$ and stores the observation at position $\boldsymbol{z}$; $\boldsymbol{o_z}$). For clarity, we presented a basis where the RNN has one neuron active at any time, and the matrices $\boldsymbol{W_v}$ have columns containing a single 1 (Figure 1C), although the above solutions work in any basis, i.e., $\boldsymbol{h} \leftarrow \boldsymbol{O}^{-1}\boldsymbol{h}$, $\boldsymbol{W_v} \leftarrow \boldsymbol{O}^{-1}\boldsymbol{W_v}\boldsymbol{O}$. Under simple constraints, the optimal solution is a grid cell basis.[4,35]

Solving sequence memory tasks with WM

The PFC is thought to store memories in the dynamics of neural activity as opposed to synaptic connections[26,28] (1A top-right; although see Stokes[36]). Here, we show that reshaping and

rearranging Equation 1 produces an alternative but equivalent solution where memories are stored in RNN activity rather than synaptic connections (Figure 1A right):

$$\hat{t}_{t+1} = \underbrace{\begin{bmatrix} I_{d_o} & 0 & \cdots & 0 \end{bmatrix}}_{\text{Readout weights} \in \mathbb{R}^{n_o \times n_p n_o}} \underbrace{\prod_{\tau=0}^{t} \underbrace{W^*_{v_\tau}}_{\text{Recurrent matrix} \in \mathbb{R}^{n_p n_o \times n_p n_o}} \underbrace{\begin{bmatrix} o_{z_0} \\ \vdots \\ o_{z_{n_p}} \end{bmatrix}}_{\text{Initial WM slots } h_0 \in \mathbb{R}^{n_p n_o}}}_{\text{RNN}, h_{t+1} \in \mathbb{R}^{n_p n_o}} \quad \text{(Equation 2)}$$

Equation 2 performs the exact same computation as the Equation 1, although its terms, despite being similar, are different (Table S1). What was a matrix of memories stored in synaptic weights, $\begin{bmatrix} \boldsymbol{o}_{z_0} & \cdots & \boldsymbol{o}_{z_{n_p}} \end{bmatrix}$, becomes a vector of memories stored in RNN activity, $\begin{bmatrix} \boldsymbol{o}_{z_0} \\ \vdots \\ \boldsymbol{o}_{z_{n_p}} \end{bmatrix}$. Thus, neural activity is decomposable into non-overlapping subspaces termed activity slots, with each slot able to store a memory of an arbitrary observation (Figure 1A right). Readout weights are now fixed and attend to a single slot (Figure 1D; here, slot 1 is the readout slot, but in general, it depends on task structure, e.g., Figure S1). To readout the right memory at the right time, the contents of each slot must be copied and shifted to other slots via recurrent weights $\boldsymbol{W}^*_{\boldsymbol{v}} \in \mathbb{R}^{n_p n_o \times n_p n_o}$. These WM velocity-dependent matrices are expanded (and transposed) versions of the EM velocity-dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$ (example in Figure 1E). Intuitively, $\boldsymbol{W}^*_{\boldsymbol{v}}$ is bigger than $\boldsymbol{W}_{\boldsymbol{v}}$ (by a factor of n_o) because the WM RNN tracks relative position to each observation, rather than just position. Adding a memory to a slot is simple. Feedforward weights direct observations to the input slot (e.g., Figure 1A right). The memory is then passed from slot to slot, controlled by velocity, eventually influencing behavior from readout slot. Thus, once learned, this solution requires no synaptic plasticity to solve novel problems. Lastly, similar to the EM solution, the WM solution works in any basis.

Memory slots unify episodic and WM and yield a rich neural basis for WM cognitive maps

The two solutions use memory slots in different ways. In the WM solution, memory slots are structured neural subspaces with the contents (memory) of each subspace copied and shifted to neighboring subspaces. In the EM solution, memory slots are in synaptic connections, with each memory remaining fixed in its slot (a subspace of the weights). The memory slots are also accessed differently. The EM solution uses flexible attention (via an RNN) to index fixed memory slots (the weights do not move), whereas the WM solution uses fixed attention (the readout weights) but flexible memory slots (slots contents get copied and shifted by the RNN).

Interestingly, these solutions have a striking representational difference. The EM RNN represents latent position (relative to an initial position), independent of observations. However, the WM RNN conjunctively encodes all counterfactual actions and observations simultaneously. In particular, each slot is identified by an action/velocity sequence the agent could take, with its contents answering the counterfactual: what would I see if the slot's corresponding action were taken? In the case of Figure 1A, third column, slot 1, aligned with the input/readout weights, contains the correct answer if no action was taken. And slots 2 and 3 contain the correct counterfactual predictions if the agent were to go back one or two positions, respectively. Equivalently, because action sequences correspond to relative positions, each slot represents the relative position to the observation it contains. Every time the agent takes an actual action, the correct counterfactual prediction associated with every slot changes; therefore, the WM RNN must update the contents of every slot by appropriately copying and shifting their contents through the recurrent weights. This ensures that all slots' contents dynamically change to correctly encode the counterfactual observation associated with the slots' fixed action label/relative position.

Moreover, because each neuron in each slot is selective for a single observation (in an idealized basis, e.g., Figure 1A third column), the WM neural code consists of neurons that fire for particular observations at particular relative positions to those observations. Any single neuron's relative position selectivity derives from its slot identity, whereas its object selectivity derives from its functional role within its slot. This implies that EM and WM single neurons behave very differently across problems. In particular, any two EM position neurons that fire next to each other in one problem will fire next to each other in another problem: they maintain their phase relationship (Figure 1F; similar to grid cells). On the other hand, two WM activity slot neurons with a particular phase relationship in one problem (Figure 1G, top) may not have the same phase relationship in another problem (Figure 1G, bottom). This is because WM slots code for relative position to, or action sequence required to arrive at, a particular observation, and the pairing between positions and observations is shuffled between different problem realizations of a common task, meaning that the relative phase between WM neurons will also appear shuffled.

Model architectures and task specifics

To test our theory predictions that EM systems use position representations, whereas WM systems use activity slots, we use a variety of tasks and model architectures (details in STAR Methods).

Model architectures

We build both EM and WM RNNs (Figures 2A, 2B, S2A, and S2B). The key difference between them is that EM RNNs make

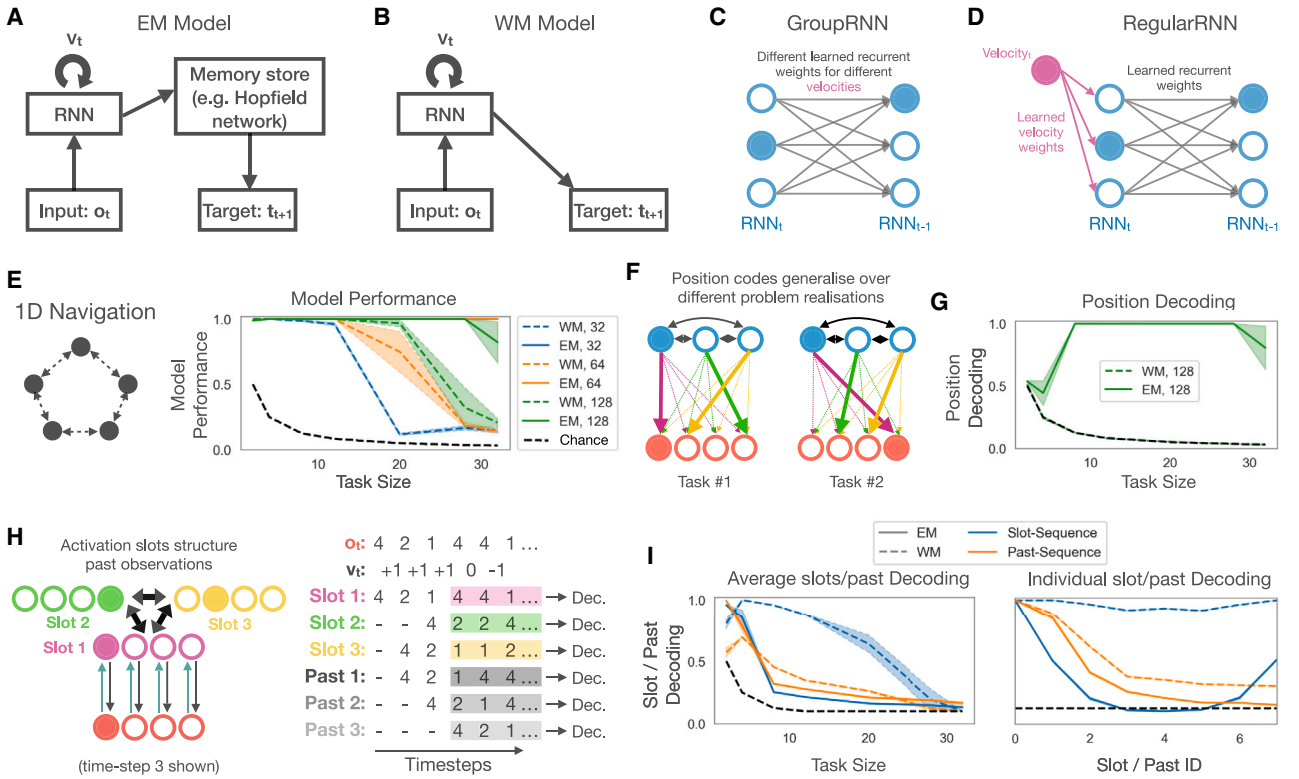

Figure 2. EM models learn position representations, whereas WM models learn activity slots

(A) EM models consist of an RNN and an external memory system. At each timestep, the RNN receives an observation, $\boldsymbol{o}_t$ and a velocity signal $\boldsymbol{v}_t$ and must predict target $\boldsymbol{t}_{t+1}$.

(B) WM models are just an RNN. We use two types of RNN (for both EM and WM models): (C) GroupRNN has learnable velocity-dependent matrices similar to our theory.

(D) RegularRNN is a conventional RNN with velocity as input.

(E) Performance of trained EM (solid) and WM (dashed) models on 1D navigation tasks (using GroupRNN). Black dashed line is chance performance. EM models perform better than WM models as task size increases. Larger RNNs perform better for both EM and WM models. For each combination of task, task size, model type, and RNN size (32, 64, and 128), we train five randomly initialized models. Solid/dashed lines are the mean, and the shaded area is the standard error. Subsequent panels are for RNNs of size 128.

(F) Our theory says EM models learn position representations that generalize over problem realizations (different state-observation pairings in common task structure).

(G) EM models but not WM models learn position representations.

(H) Left: the activity slot theory says that WM RNNs organize past observations into slots (structured according to underlying task structure). Right: thus a particular sequence of observations should be decodable from each slot *i* (the "slot sequence" for each slot *i*). The "slot sequence" differs from the sequence of observations *i* timesteps ago ("past sequence" for each time lag *i*).

(I) Left: average decoding of slot sequences and past sequences. WM models have high slot sequence decoding but not past sequence decoding: they have learned activity slots. EM models do not learn activity slots, apart from those trained on small task sizes. Right: slot and past sequence decoding for individual slots or time lags for 1D navigation on an 8-loop. Slot sequence decoding is higher than past sequence decoding for WM models but not EM models for all slots/time lags. Similar results shown for all tasks with GroupRNN and RegularRNN in Figures S3 and S4.

predictions by retrieving memories from an external memory system, whereas WM RNNs make predictions with a learned readout matrix. For the EM external memory, we use a modern Hopfield network,[37] which our above theoretical results are derived from (STAR Methods). To be general, we use two RNN variants (Figures 2C and 2D): one inspired by our theory with velocity-dependent recurrent matrices (GroupRNN; related to selective state-space models[38]), and the other is a conventional RNN with velocity signals as input (RegularRNN). All model parameters are initialized randomly and trained using backpropagation.

Tasks

We consider four main tasks (Figure S1A, further neuroscience tasks introduced later): ISR, N-back, 1D navigation, and 2D navigation. These tasks range in complexity from simple sequence repetition requiring no velocity integration to tasks where velocity must be integrated correctly using a cognitive map. Observations, $\boldsymbol{o}$, and velocity signals, $\boldsymbol{v}$ are provided at each timestep and the models are trained to predict a target, $\boldsymbol{t}$, at timestep $t+1$. An example problem sequence for the 1D navigation task (a random walk on a loop, e.g., a 3-loop with observations 1, 3, and 2 at the three positions) is: $\boldsymbol{o} = \{1, 3, 2, 2, 3, 1, 2, \ldots\}$,

$\boldsymbol{v} = \{+1, +1, 0, -1, -1, -1, \ldots\}$, and $\boldsymbol{t} = \{-, -, -, 2, 3, 1, 2, \ldots\}$. Here, "−" denotes a target is not possible to predict because it has not been observed before. We do not train on these targets. The ISR task is similar to 1D navigation but with constant forward velocity, the 2D navigation task is a 2D version of the 1D navigation task, and the N-back task is a random observation sequence with the target being the observation N timesteps ago. For each task, we train on multiple problem realizations corresponding to multiple sequences where position-observation pairings are randomized, but the underlying task structure (e.g., 1D navigation) is fixed. See STAR Methods for details.

RESULTS

Differences in EM and WM algorithm performance and representation

EM scales more favorably than WM

Our theory demonstrates that EM and WM systems implement the same computation (solve the same tasks) but use different neural algorithms/mechanisms. These algorithms have trade-offs. Most notably, the WM solution requires more RNN neurons than the EM solution. Thus we predict that, for a fixed RNN neuron budget, the WM model performance will degrade faster than the EM model performance as the number of memories needed to be stored, which we refer to as the task size, increases. (While we match RNN neurons, we note that EM models have additional neurons/parameters in their memory network, which we do account for.) This is observed for the 1D navigation task (Figure 2E), as well as all other tasks (Figure S3A/S4A for GroupRNN/RegularRNN). Expectedly, performance improves with more RNN neurons. Importantly, performance is measured on new problems (with novel state-observation associations), and all network weights are fixed and do not change (apart from the "fast weights" in the memory module of the EM). Thus, a performance of 100%, achieved for both EM and WM networks on smaller task sizes, means that they learned a cognitive map to generalize task structure.

Although the capacity of these small WM networks is larger than human WM capacity, our networks are noiseless and trained on a single task type (e.g., loop of length 10). This is unlike noisy brain neurons that have to solve multiple different task types. Although EM networks are superior in our prediction tasks, the WM model may have additional benefits. This is because it represents the whole problem in neural activity simultaneously (in slots), allowing operations over states to be done in parallel, e.g., planning and decision-making, unlike EM models, which need to sequentially sample states to plan.

EM uses position representations, whereas WM uses activity slots

Our theory predicts that trained EM models use position-like representations to index memories stored in synaptic weight slots (Equation 1), whereas trained WM models store and manipulate memories in RNN activity slots (Equation 2). To test these predictions, we train linear decoders to decode positions and observations from the neural activity of trained models' RNNs. Crucially, we avoid task overfitting by training each decoder on many example tasks, demonstrating that the activity slots, or position representations, are generalizable across tasks.

First, our theory says abstract position should be decodable from EM models but not WM models because EM models use position to index memories (Figure 2F). We confirm this prediction in all tasks and RNN types (Figures 2G, S3B, and S4B). This is consistent with existing EM models[4,14] that learn position codes to index hippocampal memories, as well as entorhinal neurons such as grid cells. In our models, position is often not decodable for large or small task sizes. For large task sizes, the models did not learn the task and so learned no position representation (Figures 2E, S3A, and S4A). For small task sizes, the EM RNN learned another memory indexing representation: the WM solution. We discuss this in a later paragraph.

Second, our theory predicts that previous observations should be decodable from the RNN in WM models but not the EM models because the RNN in WM models store memories of previous observations in neural activity slots, whereas the RNN in EM models only tracks position. More precisely, our theory prescribes exactly which past observation is stored in which neural activity slot at any given time (Figures S1B and S1C). Thus, for each neural activity slot, our theory predicts that it encodes a particular sequence of observations, and, importantly, this sequence, for tasks with varying velocity signals, is not simply a temporally shifted copy of the input observations (Figure 2H). We call this the slot sequence for each predicted activity slot in a given problem. We find that slot sequences can be decoded from RNN activity in all tasks for WM models (average slot sequence decoding across task sizes in Figures 2I left, S3C, and S4C; individual slot sequence decoding for a particular task in Figures 2I right, S3D, and S4D), whereas temporally shifted copies of the input observations (past sequences) cannot be decoded (apart from the ISR and N-back task where past sequences and slot sequences are identical). Slot sequence decoding performance degrades for larger task sizes because the RNN fails to learn the task. These results demonstrate that WM networks learn activity slots and use velocity signals to control activity slots.

Interestingly, although EM networks have, as predicted, poor slot sequence decoding and high abstract position decoding in most situations, the converse can be true for small task sizes (Figures S3G and S3I). This is because activity slots are also an effective memory indexing representation because they uniquely represent each position in each problem. We posit that the WM solution is easier to learn (when not limited by number of neurons) because slot representations are a direct function of input data, whereas position representations are abstract.

Although EM and WM networks, once trained, have the same generalization behavior (due to the formal equivalence of the EM and WM solutions), their learning dynamics and sample complexity differ (Figure S2G) with EM networks learning faster (except on N-back task). This is because the inductive bias of EM networks (fast memory binding) is suited to our tasks with novel state-observation pairs. By contrast, WM networks have no explicit bias and therefore must learn how to appropriately store memories. Further, GroupRNN learns faster than RegularRNN due to its inductive bias that facilitates path integration.

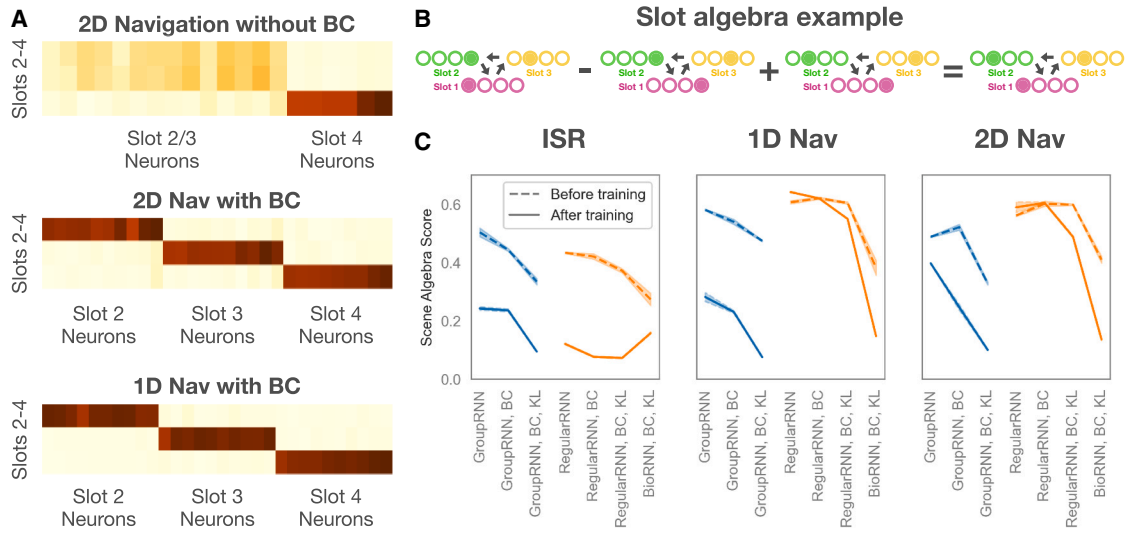

Figure 3. Visualizing WM slots without decoding, WM models afford slot algebra

(A) To visualize WM activity slots, we train models with a technique (BC) that encourages single neuronal tuning to single factors of variation.[39] We show the mutual information matrices between neurons (horizontal) and slots (vertical). We only show neurons with non-negligible activity and only slots 2–4 for clarity because slot 1 tends to have many more active neurons than slots 2–4. Top: without BC, single neurons code for multiple slots. Middle/bottom: with BC single neurons code for single slots.

(B) Slot algebra means representations from three different problems can be combined to give the representation of a fourth problem, e.g., $Rep([1,4,2]) - Rep([4,4,3]) + Rep([4,2,3]) = Rep([1,2,2])$, where $Rep([a,b,c])$ denotes observations a,b,c in slots 1,2,3. The slot algebra score measures this property, with lower values indicating more accurate implementations of this property.

(C) Some WM models display slot algebra, with slot algebra scores low for all GroupRNN models (blue) after training, high for RegularRNN models, and low for the BioRNN variant. For each task and model type, we train three randomly initialized models. Solid/dashed lines are the mean, and the shaded area is the standard error.

Visualizing slots and "slot algebra"

Visualizing slots

Our decoding analyses demonstrate that WM models learn activity slots, as predicted by our theory. We focused on decoding analyses because learned activity slots will in general correspond to neural activity subspaces, not necessarily aligned to single neuron axes similar to our WM schematics (single neurons could exhibit mixed selectivity for multiple slots). To visualize learned activity slots, we leverage a recent technique[39] that encourages single neurons to demix and code for single independent factors (e.g., slots). WM RNNs trained with this technique learn a demixed solution where single neurons participate only in a single slot (Figure 3A) and quantitatively demix on all four tasks and for both RNN types (Figure S5A).

Slot algebra predicts relations between neural representations of different problems from a common task

We test a further prediction of our theory: activity slot representations are compositional and obey a slot algebra with simple linear relationships between WM representations of different problem realizations of the same underlying task. Consider a task with three latent positions and four possible observations (Figure 3B). Let $Rep([a,b,c])$ denote the WM activity slot representation when our theory predicts observations a,b,c are in slots $1,2,3$ respectively. Now consider the neural representations for two problems, $Rep([1,4,2])$ and $Rep([1,2,2])$. These problems differ only in their observation in slot 2, and our activity slot theory predicts the linear relation $Rep([1,2,2]) = Rep([1,4,2]) - Rep([a,4,c]) + Rep([a,2,c])$ (Figure 3B illustrates this relation), where a and c can be any observations.

We examine the WM RNN in situations that satisfy the equality relation above by extracting neural representations from the RNN in four different problems. We sum the three representations according to the right-hand side of the relation and compare the sum with the measured representation on the left-hand side using a squared difference measure ($d_{1-2+3-4}$). We compare this with the squared difference of the first term on the left and the first term on the right (d_{1-2}) via the ratio $SA = \frac{d_{1-2+3-4}}{d_{1-2+3-4}+d_{1-2}}$. Low SA values indicate that the representation obeys slot algebra.

Perfect slot algebra requires pure activity slot representations. However, some velocity-integrating RNNs, both in biology[40] and models,[41] have velocity-modulated neurons. Indeed, it is possible to have activity slot representations that are velocity-modulated (= high SA). Thus, slot algebra is additionally informative of how RNNs integrate velocity. We test slot algebra on the GroupRNN, which modulates synapses with velocity signals (similar to our theory), the RegularRNN, which uses additive velocity signals, and, additionally, the BioRNN, which integrates velocity signals in a separate neural population to the RNN activities themselves. As predicted, the GroupRNN has low SA for all datasets SA (Figure 3C; we do not use the N-back task as SA is not well defined there) because velocity does not directly modulate RNN activity. SA further improves with regularization (KL) that encourages

consistency across timesteps, i.e., limiting velocity modulation (see STAR Methods). Conversely, as predicted, the RegularRNN only achieves low *SA* on the ISR dataset, which requires no velocity signal. Lastly, the BioRNN achieves low *SA* for all datasets. Intriguingly, the BioRNN architecture is related to the fly head direction circuit[42–44] (Figures S2E and S2F; details in STAR Methods).

Velocity-controlled activity slots explain diverse PFC representations

Our theory of controllable WM activity slots explains artificial RNN representations trained on sequence memory tasks. Does this understanding transfer to biological RNNs? And, if so, how are the velocity signals, which are crucial for successful control, represented? Here, we show that activity slots unify PFC-dependent sequential and cued memory tasks, with the distinguishing feature being the velocity control signals.

PFC and RNNs compute and represent velocity signals

In previous tasks, we provided velocity signals to the model. Brains do not have this luxury—they must compute velocity signals from their inputs. In spatial tasks, self-movement vectors are computed from vestibular or other sensory inputs and represented using head direction[45] and speed cells.[40] It is unknown what the corresponding velocity signals are for complex non-spatial tasks. Recent PFC data, however, although not demonstrating velocity signals, have found neurons that track progress to goals, invariant to the actual goal distance.[30,46] Here, a 50% "progress cell" fires midway between two goals, whether they are three steps apart or 20. To update progress, we contend that "progress velocity" signals must be computed and represented. To test this, we train WM models on analogous tasks.

In particular, Basu et al.[46] and El-Gaby et al.[30] trained rodents to repeatedly visit two and four (of nine) spatially located reward ports in sequence, respectively. The animals were trained on many problem realizations with random goal positions. We model these tasks (modulo the spatial component to be general) as 2-ISR and 4-ISR tasks but with delays between each observation (analogous to the delay traveling between spatial goals, Figures 4A and 4E, left). Critically, although delays are random across different problem realizations, delays are the same on each loop of a problem. For example, possible 2-ISR delay sequences for different problem realizations are $[4,*,*,2,*,*,4,*,*,2,\ldots]$ or $[1,*,3,*,1,*,3,\ldots]$ where $*$ is the common delay observation (that also gets predicted), and possible 4-ISR delay sequences are $[4,*,2,*,*,2,*,6,4,*,2,*,*,2,*,6,4,\ldots]$ or $[1,3,*,8,*,2,1,3,*,8,*,2,1,\ldots]$. We provide no explicit velocity signal—the model must compute progress velocity for each delay period from sparse inputs and remember it on returning to that same delay (after a loop of experience).

After training WM models, we can decode progress and progress velocity (Figures 4A and 4E, right) and visualize tuning at the single neuron level (Figures 4B, 4F, and S6). The network is computing progress velocity to overcome the different delays. The question remains: how does this relate to activity slots?

Controlling activity slots with progress velocity

We posit that activity slots are "task-progress" structured (Figures 4C and S1D), with slots coding for, e.g., 25% or 150% since observation (up to 200/400% for the 2/4-ISR delay task) and progress velocity controlling when slot contents move to subsequent slots. (Task-progress slots are continuous rather than discrete, with bumps of activity on continuous attractor manifolds defining an observation within a "slot." For consistency, we still refer to them as if they were discrete.) This allows variable-length delays across different problems to be mapped to a common slot structure for generalization. Indeed, we can decode task-progress activity slots (Figures 4A and 4E, right) and observe task-progress slot neurons (Figures 4D and 4G), which fire at a particular task-progress after a preferred observation (further examples in Figure S6). Notably, as our model predicts, both progress and task-progress slot neurons were recently recorded in rodent mPFC,[30] with the pathway linking the same observation across slots in the task-progress slot analogous to their "structured memory buffers."

Hierarchical slots

In the 4-ISR delay task, to correctly predict observations on the second (and subsequent) repeats within each problem, each of the four delays must have been computed and remembered on first presentation. This induces a hierarchy of activity slots with observations stored in task-progress slots and the four delays stored (after being computed) in separate higher-level slots (Figure 4H, top). Now the right delay (inverse velocity) at the right moment can be retrieved to control the lower-level task-progress slots. Indeed, we can decode slot contents consistent with this hierarchical slots hypothesis (Figure 4H, bottom). We predict that PFC uses hierarchical slots as a neural mechanism for cognitive control.[31]

PFC activity slots in a sequence memory task

Our activity slot theory also accounts for PFC representations in standard sequence memory tasks where only a constant velocity is required. In particular, Xie et al.[28] recorded PFC neurons in a 3-ISR task. Here, monkeys were shown three (of six) positions on a screen in sequence and were trained to, after a delay, saccade to the three positions in order (Figure 5A). Similar to our ISR tasks, the monkeys were trained on many such sequences. Neural activity in the delay period decomposed into three orthogonal subspaces, with the first/second/third saccade position simultaneously encoded in the first/second/third subspace (Figures 5B and 5C). These three subspaces are activity slots (Figure 5D). Indeed, WM networks learn the same subspace decomposition (Figures 5E and 5F).

PFC activity slots in a cued memory retrieval task

We have considered external and internally generated velocity signals; however, observations themselves can serve as velocity signals. Here, we show that sensory cues can be understood as velocity signals controlling PFC activity slots. In particular, Panichello and Buschman[29] recorded PFC neurons in monkeys performing a cue-dependent memory task, where two colors are presented at the top and bottom of a screen. After a delay, a sensory cue dictates whether to report the top or bottom color after another delay (Figure 6A). Importantly, each problem realization is a random combination of two colors and cue. Neural activity in the first delay period decomposed into two orthogonal subspaces, with the first/second subspace representing the top/bottom color (Figure 6B, left). After the cue, the top (when correct) and bottom (when correct) colors are represented in parallel subspaces (Figures 6B, right, and C). This is explained by activity

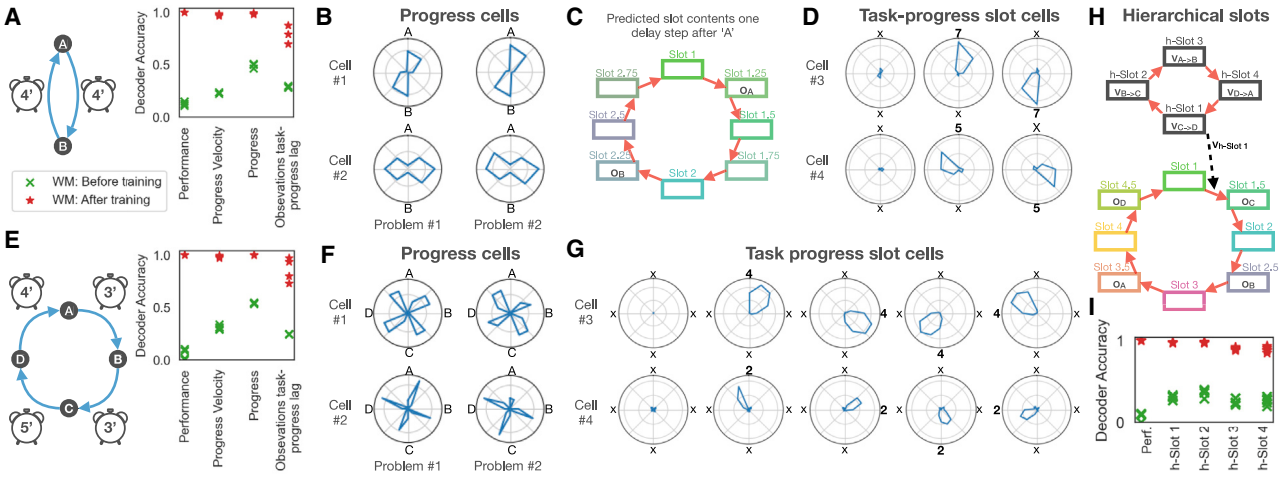

Figure 4. WM RNNs compute velocity to control activity slots

(A) We train a WM RNN on a 2-ISR task but with delays between observations. Delays are sampled from $\{3, 4, 5, 6, 7\}$ for each problem realization but are fixed for the duration of the individual problem. An example delay of four steps shown in the schematic. Inferring the delay for each problem is critical for successful prediction of observations. Decoding analyses reveal that progress velocity (reciprocal of delay duration) and progress (% of delay duration completed) are represented, i.e., the models map varying delay lengths across problems into a common progress representation. We can also decode observations at the predicted task-progress lag (e.g., 3 was 125% progress lengths ago) according to our model of task-progress activity slots (C). Green/red: before/after training.

(B) Example learned progress cells. Angle of circle represents fraction of progress from observation (A) at 0 degrees to observation (B) at 180 degrees, and then back. Radius of blue curve indicates firing rate. These cells fire at a fixed fraction of progress to the next observation, independent of observation identity. The cells transform the variable delay periods for each problem into a common representation.

(C) Schematic of task-progress activity slots. Here, slots correspond to different progresses since an observation, allowing variable-length problems to be mapped to a common slot representation. Here, three intermediate slots between slots 1 to 2 (similarly between slots 2 to 1) corresponding to 25%, 50%, and 75% progress. This model predicts that RNN neurons will be tuned to task-progress since a preferred observation, e.g., a neuron will fire when 3 was 125% progress lengths ago. Note, slots will actually form a continuous manifold because delays can be any duration, but we show discrete slots for schematic clarity.

(D) The RNN learns task-progress slot cells. Top/bottom: RNN neuron that fires at 0%/175% after observation 7/5. Left/middle/right: average neural firing in all problems without a 7/5 observation/problems with observation 7/5 in position A/problems with observation 7/5 in position (B).

(E–G) Same as (A), (B), and (D) but for a 4-ISR task where delays between observations can be different (but still fixed within each problem; example delays of 3, 3, 3, and 5 steps shown in schematic; delays sampled from $\{3, 4, 5\}$).

(H) Hierarchical slots contain velocity signals that control lower-level slots containing observations. In the 4-ISR delay task, delays (velocities) must be computed and remembered on first presentation to correctly predict upcoming observations on subsequent presentations. Top: hierarchical slots store delays (velocities), ensuring that the right velocity controls lower-level slots at the right time (readout from h-slot 1). Bottom: hierarchical slots contents can be decoded from RNNs trained on the 4-ISR task with delays.

slots (Figure 6D): the initial two subspaces are activity slots for the top and bottom color, then the cue behave similar to a velocity signal to route the correct color to the readout slot (hence, why subspaces are parallel after the cue on correct trials; Figure 6D, bottom). Indeed, WM networks learn the same subspace decomposition and dynamics (Figures 6E–6G).

DISCUSSION

We derived a formal relationship between the algorithms and representations of episodic and working sequence memory and, in simulation, validated predictions that EM networks learn abstract position-like representations, whereas WM networks learn activity slot representations, different neural implementations of cognitive maps that generalize to novel state-observation pairings. Further, we demonstrated that our theory of controllable activity slots provides a unified neural mechanism to several PFC studies.[28–30,46] Although these PFC representations have simple (often loop) structures, we anticipate that PFC will represent more complex structures, such as 2D slots, when participants are trained on sequences drawn from 2D tasks.

Activity slots may also explain PFC data from non-explicit WM tasks. For example, human mPFC (fMRI) activity obeys rules of scene algebra (analogous to slot algebra) during scene construction tasks.[47] Notably their task requires compositional understanding (objects are in any scene configuration) similar to our sequential WM tasks. In general, we anticipate that activity slots will be relevant in tasks requiring compositional understanding beyond WM in brains and machines. For example, activity slots could represent sub-goals for planning, which would explain why the same brain region is involved in WM, value-based decision-making, and goal-directed planning.[31,48] Notably, in machine learning, slot models are used in diverse settings from natural language[6] to planning[49] and scene construction.[50,51]

Although we related the EM and WM algorithms to HPC and PFC, their key distinction is whether memories are stored in synaptic connections or neural activity. Because PFC connects to HPC (via entorhinal cortex), the prefrontal RNN can assist in

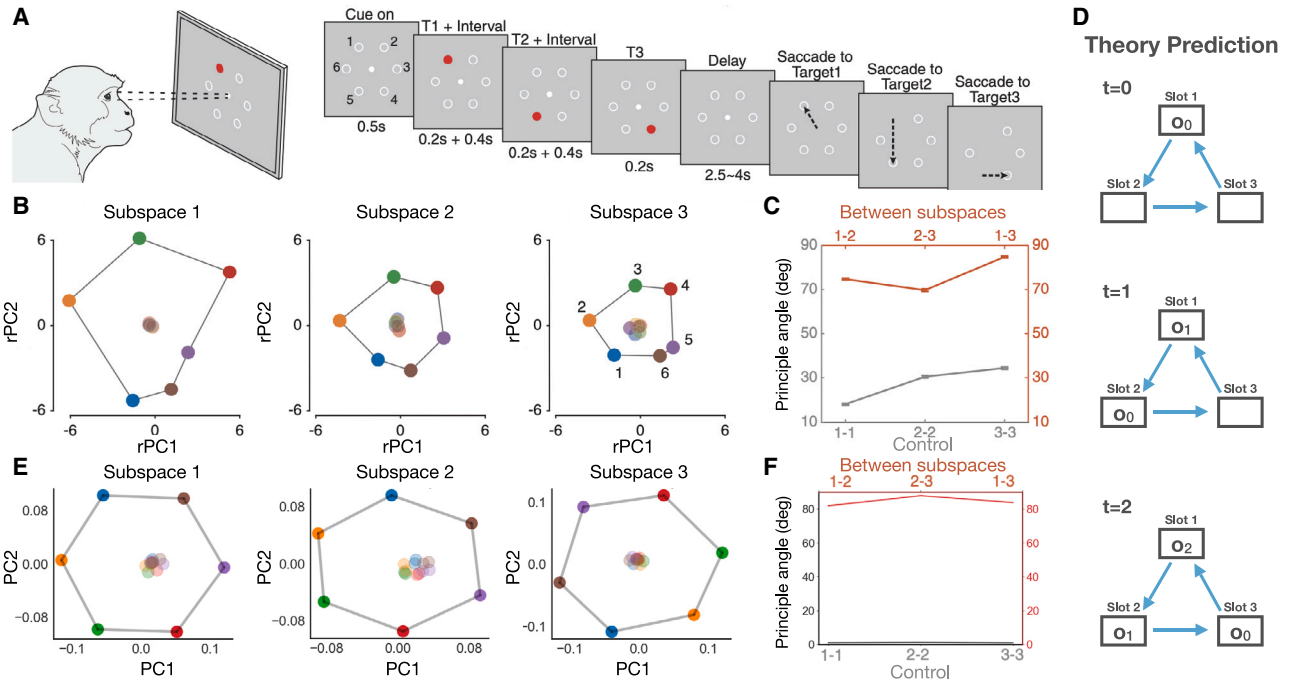

Figure 5. PFC representations in an ISR task are activity slots

(A) Xie et al.[28] recorded PFC neurons while monkeys performed an ISR-like task. Monkeys were presented with, on a screen, three (out of six) spatial positions in sequence, then, after a delay, tasked with repeating the sequence (via saccades).

(B and C) During the delay, PFC recordings revealed observation coding in three neural subspaces, one for each ordered position. Darker dots are projections of the ith observations in the ith subspace, lighter dots are projections of jth observations in the ith subspace ($j \neq i$). Lighter dots have close to zero projection, thus suggesting that the subspaces are orthogonal, which (C) is confirmed to be in analysis.

(D) The activity slots model for this task, with slot contents shown at three different timesteps (final timestep is during delay period).

(E and F) A trained RegularRNN recovers the same three orthogonal subspaces. (A–C) Reproduced from Xie et al.[28]

hippocampal memory retrieval and thus also represent abstract position representations. Indeed, grid-like coding has been observed in mPFC.[52,53] However, in some WM tasks,[30,46] early evidence suggests that position representations are not decodable from rodent mPFC. Furthermore, our theory suggests that any brain region, not just PFC, storing memories in neural activity will utilize activity slots.

It is also plausible that PFC utilizes short-term synaptic plasticity to store memories,[36] with recent modeling suggesting that this may better explain PFC responses.[54] It is not clear, however, how this explains existing PFC data on sequence WM,[28–30] which are consistent with activity slots.

Activity slots for (non-sequence) WM have been considered before,[55–58] where behavioral evidence is against a naive slot implementation because human recall performance degrades gradually with task size rather than a sharp drop-off when all slots are full. Instead, the conceptual "continuous resource models" are preferred in which fixed resources (pool of neurons) flexibly code for differing numbers of memories but with increasing interference for increasing task size leading to gradual performance decline. In fact, activity slots are compatible with continuous resource models, either by chunking memories together when the task size exceeds the number of discrete slots[59] or by using continuous activity slots (similar to Figure 4) and placing memories (bumps of activity) closer together on the slot attractor manifold as task size increases. Both of these methods would have a gradual decline in recall performance due to memory interference. Continuous activity slots may also permit generalization to task sizes beyond the training distribution: analogous to how the grid cell hierarchy allows position coding of environments larger than the largest grid scale,[60] we anticipate that continuous activity slots may have a hierarchical representation facilitating length generalization.

We have not solved the algorithm of RNNs on all sequential WM tasks because naive activity slots are not learned in some tasks (e.g., Figures S7A and S7B). Nevertheless, we suggest that activity slots are a general-purpose solution. Of note to machine learners, our formalism, as well as the GroupRNN, is a selective state-space model[38]; thus, our work offers insights into the neural mechanism of learned state-space models, RNNs, and transformers (and their relationships).

The hippocampal and frontal systems are thought to be essential in mediating higher-order cognition, with patients exhibiting profound deficits in behavioral flexibility if these structures are damaged.[10,11] Although these systems have been the subject of intense study over the last several decades—both experimentally and theoretically—they have been largely treated independently, and their respective cellular representations are hard to reconcile. Our work brings

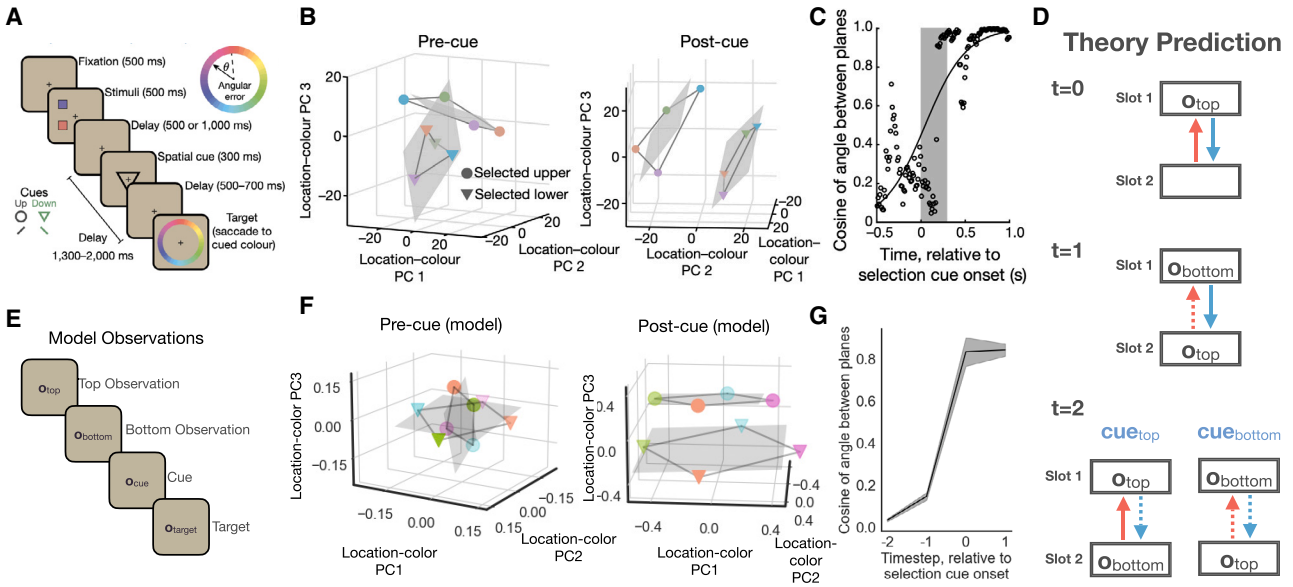

Figure 6. PFC representations in a cue-dependent memory retrieval task as velocity-controlled activity slots

(A) Panichello and Buschman[29] recorded PFC neurons while monkeys were cued to recall one of two colors previously presented at the top and bottom of a screen.

(B and C) Before the cue, PFC neurons represented two orthogonal subspaces—one each for the top and bottom color. After the cue, the colors (when correct) were in parallel subspaces.

(D) The activity slots model places the top and bottom colors in different slots, then uses the cue as a velocity signal to shift the correct color to the readout slot (slot 1).

(E) We modeled the task by sequentially presenting the top color, the bottom color, then the cue, and trained the RNN to predict the target color.

(F and G) RegularRNNs recover the same subspace phenomena consistent with the activity slot model. (A–C) Reproduced from Panichello et al.[29]

together these systems by showing that the algorithms and representations of frontal WM and temporal EM are two sides of the same coin.

RESOURCE AVAILABILITY

Lead contact

Requests for further information and resources should be directed to and will be fulfilled by the lead contact, James Whittington (jcrwhittington@gmail.com).

Materials availability

No materials were generated in this study.

Data and code availability

Python and PyTorch code is available on https://github.com/djcrw/em-wm-slots.

ACKNOWLEDGMENTS

We thank Kris Jensen and Jo Warren for helpful feedback on the manuscript. We thank the following funding sources: Sir Henry Wellcome Postdoctoral Fellowship (222817/Z/21/Z) to J.C.R.W.; the Gatsby Charitable Foundation to W.D.; Wellcome Principal Research Fellowship (219525/Z/19/Z), Wellcome Collaborator award (214314/Z/18/Z), and Jean-François and Marie-Laure de Clermont-Tonnerre Foundation award (JSMF220020372) to T.E.J.B.; the Wellcome Centre for Integrative Neuroimaging is supported by core funding from the Wellcome Trust (203139/Z/16/Z); and the James S. McDonnell, Simons Foundations, NTT Research, and an NSF CAREER Award to S.G.

AUTHOR CONTRIBUTIONS

J.C.R.W. conceptualized the study with input from W.D. and all other authors. J.C.R.W. derived theory and performed simulations. J.C.R.W. wrote the manuscript with input from all other authors.

DECLARATION OF INTERESTS

The authors declare no competing interests.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- METHOD DETAILS
  - Full Theory
  - Model Details
  - Additional Task Details
- QUANTIFICATION AND STATISTICAL ANALYSIS
  - Decoding Analyses
  - Mutual Information Ratio
  - Slot Algebra

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.neuron.2024.10.017.

Received: March 8, 2024
Revised: August 7, 2024
Accepted: October 16, 2024
Published: November 21, 2024

REFERENCES

1. Stachenfeld, K.L., Botvinick, M.M., and Gershman, S.J. (2017). The hippocampus as a predictive map. Nat. Neurosci. *20*, 1643–1653. https://doi.org/10.1038/nn.4650.
2. Buzsáki, G., and Tingley, D. (2018). Space and Time: The Hippocampus as a Sequence Generator. Trends Cogn. Sci. *22*, 853–869. https://doi.org/10.1016/j.tics.2018.07.006https.
3. Dayan, P. (1993). Improving Generalization for Temporal Difference Learning: The Successor Representation. Neural Comput. *5*, 613–624. https://doi.org/10.1162/neco.1993.5.4.613.
4. Whittington, J.C.R., Muller, T.H., Mark, S., Barry, C., Burgess, N., Behrens, T.E.E., Chen, G., Barry, C., Burgess, N., and Behrens, T.E.E. (2020). The Tolman-Eichenbaum Machine: Unifying Space and Relational Memory through Generalization in the Hippocampal Formation. Cell *183*, 1249–1263.e23. https://doi.org/10.1016/j.cell.2020.10.024.
5. Raju, R.V., Guntupalli, J.S., Zhou, G., Lázaro-Gredilla, M., and George, D. (2022). Space is a latent sequence: Structured sequence learning as a unified theory of representation in the hippocampus. Preprint at arXiv.
6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. Adv. Neural Inf. Process. Syst. *30*, 5999–6009.
7. Tolman, E.C. (1948). Cognitive maps in rats and men. Psychol. Rev. *55*, 189–208. https://doi.org/10.1037/h0061626.
8. Hassabis, D., Kumaran, D., Vann, S.D., and Maguire, E.A. (2007). Patients with hippocampal amnesia cannot imagine new experiences. Proc. Natl. Acad. Sci. USA *104*, 1726–1731. https://doi.org/10.1073/pnas.0610561104.
9. Fuster, J.M., and Alexander, G.E. (1971). Neuron activity related to short-term memory. Science *173*, 652–654. https://doi.org/10.1126/science.173.3997.652.
10. Bauer, R.H., and Fuster, J.M. (1976). Delayed-matching and delayed-response deficit from cooling dorsolateral prefrontal cortex in monkeys. J. Comp. Physiol. Psychol. *90*, 293–302. https://doi.org/10.1037/h0087996.
11. Funahashi, S., Chafee, M.V., and Goldman-Rakic, P.S. (1993). Prefrontal neuronal activity in rhesus monkeys performing a delayed anti-saccade task. Nature *365*, 753–756. https://doi.org/10.1038/365753a0.
12. Behrens, T.E.J., Muller, T.H., Whittington, J.C.R., Mark, S., Baram, A.B., Stachenfeld, K.L., and Kurth-Nelson, Z. (2018). What Is a Cognitive Map? Organizing Knowledge for Flexible Behavior. Neuron *100*, 490–509. https://doi.org/10.1016/j.neuron.2018.10.002.
13. Whittington, J.C.R., McCaffary, D., Bakermans, J.J.W., and Behrens, T.E.J. (2022). How to build a cognitive map. Nat. Neurosci. *25*, 1257–1272. https://doi.org/10.1038/s41593-022-01153-y.
14. Uria, B., Ibarz, B., Banino, A., Zambaldi, V., Kumaran, D., Hassabis, D., Barry, C., and Blundell, C. A model of egocentric to allocentric understanding in mammalian brains.Preprint at bioRxiv .
15. Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities (associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices). Biophysics *79*, 2554–2558.
16. O'Keefe, J., and Nadel, L. (1978). The Hippocampus as a Cognitive Map (Oxford University Press).
17. Deshmukh, S.S., and Knierim, J.J. (2013). Influence of local objects on hippocampal representations: Landmark vectors and memory. Hippocampus *23*, 253–267. https://doi.org/10.1002/hipo.22101.
18. Wood, E.R., Dudchenko, P.A., Robitsek, R.J., and Eichenbaum, H. (2000). Hippocampal neurons encode information about different types of memory episodes occurring in the same location. Neuron *27*, 623–633. https://doi.org/10.1016/S0896-6273(00)00071-4.
19. Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E.I. (2005). Microstructure of a spatial map in the entorhinal cortex. Nature *436*, 801–806. https://doi.org/10.1038/nature03721.
20. Høydal, Ø.A., Skytøen, E.R., Andersson, S.O., Moser, M.B., and Moser, E.I. (2019). Object-vector coding in the medial entorhinal cortex. Nature *568*, 400–404. https://doi.org/10.1038/s41586-019-1077-7.
21. Solstad, T., Boccara, C.N., Kropff, E., Moser, M.-B., and Moser, E.I. (2008). Representation of Geometric Borders in the Entorhinal Cortex. Science *322*, 1865–1868. https://doi.org/10.1126/science.1166466.
22. Constantinescu, A.O. (2017). Neural mechanisms underlying advanced cognition in humans. PhD Thesis (Oxford University Press).
23. Bongioanni, A., Folloni, D., Verhagen, L., Sallet, J., Klein-Flügge, M.C., and Rushworth, M.F.S. (2021). Activation and disruption of a neural mechanism for novel choice in monkeys. Nature *591*, 270–274. https://doi.org/10.1038/s41586-020-03115-5.
24. Aronov, D., Nevers, R., and Tank, D.W. (2017). Mapping of a non-spatial dimension by the hippocampal–entorhinal circuit. Nature *543*, 719–722. https://doi.org/10.1038/nature21692.
25. Burgess, N., and Hitch, G.J. (1999). Memory for serial order: A network model of the phonological loop and its timing. Psychol. Rev. *106*, 551–581. https://doi.org/10.1037/0033-295X.106.3.551.
26. Botvinick, M.M., and Plaut, D.C. (2006). Short-term memory for serial order: A recurrent neural network model. Psychol. Rev. *113*, 201–233. https://doi.org/10.1037/0033-295X.113.2.201.
27. Ganguli, S., Huh, D., and Sompolinsky, H. (2008). Memory traces in dynamical systems. Proc. Natl. Acad. Sci. USA *105*, 18970–18975. https://doi.org/10.1073/pnas.0804451105.
28. Xie, Y., Hu, P., Li, J., Chen, J., Song, W., Wang, X.-J., Yang, T., Dehaene, S., Tang, S., Min, B., and Wang, L. (2022). Geometry of sequence working memory in macaque prefrontal cortex. Science *375*, 632–639. https://doi.org/10.1126/science.abm0204.
29. Panichello, M.F., and Buschman, T.J. (2021). Shared mechanisms underlie the control of working memory and attention. Nature *592*, 601–605. https://doi.org/10.1038/s41586-021-03390-w.
30. El-Gaby, M., Harris, A.L., Whittington, J.C.R., Dorrell, W., Bhomick, A., Walton, M.E., Akam, T., and Behrens, T.E.J. (2023). A Cellular Basis for Mapping Behavioural Structure. Preprint at bioRxiv.
31. Miller, E.K., and Cohen, J.D. (2001). An Integrative Theory of Prefrontal Cortex Function. Annu. Rev. Neurosci. *24*, 167–202. https://doi.org/10.1146/annurev.neuro.24.1.167.
32. Baddeley, A.D., and Hitch, G. (1974). Working Memory. In Psychol. Learn. Motiv., *8*, G.H. Bower, ed. (Academic Press), pp. 47–89. https://doi.org/10.1016/S0079-7421(08)60452-1.
33. Postle, B.R. (2006). Working Memory as an Emergent Property of the Mind and Brain. Neuroscience *139*, 23–38. https://doi.org/10.1016/j.neuroscience.2005.06.005.
34. Hebb, D.O. (1949). The Organization of Behavior; A Neuropsychological Theory (Wiley & Sons).
35. Dorrell, W., Latham, P.E., Behrens, T.E.J., and Whittington, J.C.R. (2023). Actionable Neural Representations: Grid Cells from Minimal Constraints. International Conference on Learning Representations. https://openreview.net/forum?id=xfqDe72zh41.
36. Stokes, M.G. (2015). 'Activity-silent' working memory in prefrontal cortex: a dynamic coding framework. Trends Cogn. Sci. *19*, 394–405. https://doi.org/10.1016/j.tics.2015.05.004.
37. Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G.K., Greiff, V., et al. (2020). Hopfield networks is all you need. Preprint at arXiv.
38. Gu, A., and Dao, T. (2023). Mamba: Linear-Time Sequence Modeling with Selective State Spaces. Preprint at arXiv.

39. Whittington, J.C.R., Dorrell, W., Ganguli, S., and Behrens, T.E.J. (2023). Disentanglement with Biological Constraints: A Theory of Functional Cell Types. Preprint at arXiv.

40. Sargolini, F., Fyhn, M., Hafting, T., McNaughton, B.L., Witter, M.P., Moser, M.B., and Moser, E.I. (2006). Conjunctive representation of position, direction, and velocity in entorhinal cortex. Science *312*, 758–762. https://doi.org/10.1126/science.1125572.

41. Khona, M., and Fiete, I.R. (2022). Attractor and integrator networks in the brain. Nat. Rev. Neurosci. *23*, 744–766. https://doi.org/10.1038/s41583-022-00642-0.

42. Kim, S.S., Rouault, H., Druckmann, S., and Jayaraman, V. (2017). Ring attractor dynamics in the Drosophila central brain. Science *356*, 849–853. https://doi.org/10.1126/science.aal4835.

43. Green, J., Adachi, A., Shah, K.K., Hirokawa, J.D., Magani, P.S., and Maimon, G. (2017). A neuronal circuit architecture for angular integration in Drosophila. Nature *546*, 101–106. https://doi.org/10.1038/nature22343.

44. Turner-Evans, D., Wegener, S., Rouault, H., Franconville, R., Wolff, T., Seelig, J.D., Druckmann, S., and Jayaraman, V. (2017). Angular velocity integration in a fly heading circuit. eLife *6*, e23496. https://doi.org/10.7554/eLife.23496.

45. Taube, J.S., Muller, R.U., and Ranck, J.B. (1990). Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. J. Neurosci. *10*, 420–435. https://doi.org/10.1523/JNEUROSCI.10-02-00420.1990.

46. Basu, R., Gebauer, R., Herfurth, T., Kolb, S., Golipour, Z., Tchumatchenko, T., and Ito, H.T. (2021). The orbitofrontal cortex maps future navigational goals. Nature *599*, 449–452. https://doi.org/10.1038/s41586-021-04042-9.

47. Schwartenbeck, P., Baram, A., Liu, Y., Mark, S., Muller, T., Dolan, R., Botvinick, M., Kurth-Nelson, Z., and Behrens, T. (2023). Generative replay underlies compositional inference in the hippocampal-prefrontal circuit. Cell *186*, 4885–4897.e14. https://doi.org/10.1016/j.cell.2023.09.004.

48. Balleine, B.W., and Dickinson, A. (1998). Goal-directed instrumental action: contingency and incentive learning and their cortical substrates. Neuropharmacology *37*, 407–419. https://doi.org/10.1016/S0028-3908(98)00033-1.

49. Watters, N., Matthey, L., Bosnjak, M., Burgess, C.P., and Lerchner, A. (2019). COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration. Preprint at arXiv.

50. Burgess, C.P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. (2019). MONet: Unsupervised Scene Decomposition and Representation. Preprint at arXiv.

51. Whittington, J.C.R., Kabra, R., Matthey, L., Burgess, C.P., and Lerchner, A. (2021a). Constellation: Learning relational abstractions over objects for compositional imagination. Preprint at arXiv.

52. Doeller, C.F., Barry, C., and Burgess, N. (2010). Evidence for grid cells in a human memory network. Nature *463*, 657–661. https://doi.org/10.1038/nature08704.

53. Constantinescu, A.O., O'Reilly, J.X., and Behrens, T.E.J. (2016). Organizing conceptual knowledge in humans with a gridlike code. Science *352*, 1464–1468. https://doi.org/10.1126/science.aaf0941.

54. Kozachkov, L., Tauber, J., Lundqvist, M., Brincat, S.L., Slotine, J.-J., and Miller, E.K. (2022). Robust and brain-like working memory through short-term synaptic plasticity. PLoS Comp. Biol. *18*, e1010776. https://doi.org/10.1371/journal.pcbi.1010776.

55. Ma, W.J., Husain, M., and Bays, P.M. (2014). Changing concepts of working memory. Nat. Neurosci. *17*, 347–356. https://doi.org/10.1038/nn.3655.

56. Brady, T.F., and Alvarez, G.A. (2011). Hierarchical Encoding in Visual Working Memory: Ensemble Statistics Bias Memory for Individual Items. Psychol. Sci. *22*, 384–392. https://doi.org/10.1177/0956797610397956.

57. Sims, C.R., Jacobs, R.A., and Knill, D.C. (2012). An Ideal Observer Analysis of Visual Working Memory. Psychol. Rev. *119*, 807–830. https://doi.org/10.1037/a0029856.

58. Jakob, A.M.V., and Gershman, S.J. (2023). Rate-distortion theory of neural coding and its implications for working memory. eLife *12*, e79450. https://doi.org/10.7554/eLife.79450.

59. Soni, A., and Frank, M.J. (2024). Adaptive chunking improves effective working memory capacity in a prefrontal cortex and basal ganglia circuit. eLife *13*, RP97894. https://doi.org/10.7554/eLife.97894.1.

60. Fiete, I.R., Burak, Y., and Brookings, T. (2008). What Grid Cells Convey about Rat Location. J. Neurosci. *28*, 6858–6871. https://doi.org/10.1523/JNEUROSCI.5684-07.2008.

61. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019), H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, and E.B. Fox, eds. *721*, 8026–8037.

62. Whittington, J.C.R., Warren, J., and Behrens, T.E.J. (2021b). Relating transformers to models and neural representations of the hippocampal formation. International Conference on Learning Representations. https://openreview.net/pdf?id=B8DVo9B1YE0.

63. Pritzel, A., Uria, B., Srinivasan, S., Puigdomènech, A., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural Episodic Control. In International conference on machine learning. https://doi.org/10.1038/nature20101.

64. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Preprint at arXiv.

65. Ba, J.L., Kiros, J.R., and Hinton, G.E. (2016). Layer Normalization. Preprint at arXiv.

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Software and algorithms		
Python 3.11	Python Software Foundation	https://www.python.org/
PyTorch 2.4	Paszke et al.[61]	https://pytorch.org/
Code for this paper	https://doi.org/10.5281/zenodo.13909034	https://github.com/djcrw/EM-WM-slots

METHOD DETAILS

Full Theory

Task reminder

Formally, for each task, we consider a dataset $\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1, \cdots, \mathcal{D}_N\}$, where each $\mathcal{D}_i$ is a sequence consisting of vectors of observations, $\boldsymbol{o}$, of dimension d_o (termed $\boldsymbol{o} \in \mathbb{R}^{d_o}$), targets, $\boldsymbol{t} \in \mathbb{R}^{d_o}$, and (allocentric) velocities, $\boldsymbol{v} \in \mathbb{R}^{d_v}$, i.e., $\mathcal{D}_i = \{(\boldsymbol{o}_0, \boldsymbol{t}_0, \boldsymbol{v}_0), (\boldsymbol{o}_1, \boldsymbol{t}_1, \boldsymbol{v}_1), \cdots, (\boldsymbol{o}_K, \boldsymbol{t}_K, \boldsymbol{v}_K)\}$ if the sequence is of length K. These velocities mimic real actions taken by agents, but in this case we provide them externally. Importantly, the underlying structure of the task prescribes how the velocities add up to cancel each other out (just like North + East + South + West = 0 defines 2D space), and when the velocities cancel each other out, the observation is identical to what it was previously (just like returning to the same position in 2D space). Concretely, there is an underlying latent variable, $\boldsymbol{z} \in \mathbb{R}^{d_z}$, corresponding to 'position', and each position, $\boldsymbol{z}$, is associated with an observation, $\boldsymbol{o}$, and any two neighbouring positions are related by a velocity, $\boldsymbol{v}$ (example for a simple loop structure in Figure 1A left). For each $\mathcal{D}_i$, while the observation-position pairing is random, the underlying structure is preserved (velocities add up and cancel in the same way). The aim of the task is to predict a target, $\boldsymbol{t}$, which is either an upcoming observation (i.e., what you will see after going North), or a past observation (i.e., what you saw 5 steps ago). Importantly, at time t the model predicts the target at time $t+1$ – one-step prediction. While the task formalism (and subsequent model formalism) is general to tasks with discrete and continuous positions, we primarily consider the discrete setting here (with n_p total positions).

Solving these tasks with an EM model

Such tasks can be solved using EM systems that consist of RNNs equipped with an external memory. The external memory can be a Hopfield network,[4] a modern Hopfield network,[37] a transformer neural network,[62] or a differentiable neural dictionary.[63] All these methods are relatable to each other,[62] and crucially can all be viewed as storing memories in synaptic connections. Here, we present the differential neural dictionary / modern Hopfield network version (as they are general external memory devices). Additionally, we consider an RNN that only receives recurrent input and the velocity signal. Predictions are made via

$$\hat{\boldsymbol{t}}_t = \boldsymbol{V}\,\mathrm{softmax}\left(\boldsymbol{K}^T\boldsymbol{q}_t\right) = \boldsymbol{O}_{t-1}\mathrm{softmax}\left(\boldsymbol{G}_{t-1}^T\boldsymbol{g}_t\right) \quad \text{(Equation 3)}$$

Where $\boldsymbol{g}_t \in \mathbb{R}^{d_g}$ is the RNN state representation at time t, and $\boldsymbol{O}_{t-1} = [\boldsymbol{o}_0 \;\; \cdots \;\; \boldsymbol{o}_{t-1}]$ and $\boldsymbol{G}_{t-1} = [\boldsymbol{g}_0 \;\; \cdots \;\; \boldsymbol{g}_{t-1}]$ are stored memories, i.e., the memories bind together each $\boldsymbol{g}$ and $\boldsymbol{o}$ at each timestep. (Normally key, query, and value matrices, are used i.e., $\boldsymbol{W}_v\boldsymbol{O}_{t-1}\mathrm{softmax}(\boldsymbol{G}_{t-1}^T\boldsymbol{W}_q^T\boldsymbol{W}_k\boldsymbol{g}_t)$. We leave them out for simplicity, but the following argument would remain the same if we included them.) We note that the targets are in the form of observations (upcoming/ those that have previously been seen) and so stored memories of observations will facilitate accurate prediction.

The particular choice of RNN does not matter for our purposes here - it just needs to accurately track position, $\boldsymbol{z}$. Nevertheless, the RNN state, $\boldsymbol{g}_t$, will be a function of past velocities, $\boldsymbol{v}$, as it must integrate velocities to track position:

$$\boldsymbol{g}_t = f(\{\boldsymbol{v}_0, \cdots, \boldsymbol{v}_t\}) = \boldsymbol{W}_{\boldsymbol{v}_{t-1}}\boldsymbol{g}_{t-1} \quad \text{or} \quad f(\boldsymbol{W}_r\boldsymbol{g}_{t-1} + \boldsymbol{B}\boldsymbol{v}_t) \quad \text{(Equation 4)}$$

In the bottom line of the above equation, we present two common ways for integrating velocity in RNNs, the first[4] uses velocity dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$, to update RNN state and is inspired from group theory (and can mathematically shown to produce grid cells in 2D space[35]), and the second[14] is a classic RNN where $\boldsymbol{W}_r$ is a recurrent matrix and $\boldsymbol{B}$ is an matrix that maps velocities $\boldsymbol{v}$ to the RNN. Note $\boldsymbol{v}_t$ is the velocity from $\boldsymbol{z}_t \rightarrow \boldsymbol{z}_{t+1}$, and so while the classic RNN takes $\boldsymbol{v}_t$ as input at timestep t, it is really the velocity that was provided at $t-1$ that gets used to update position (via the recurrent weights) at time t (that's why in the velocity-dependent matrix case we use $\boldsymbol{W}_{\boldsymbol{v}_{t-1}}$ to get to $\boldsymbol{g}_t$). This is just how classic RNNs are usually framed.

Regardless of the particular RNN, if $\boldsymbol{g}_t$ learns to have the same structure as $\boldsymbol{z}$ (i.e., it represents position: $\boldsymbol{g}_t = \boldsymbol{g}(\boldsymbol{z}_t)$) then we can rewrite

$$\boldsymbol{O}_{t-1} = [\boldsymbol{o}_0 \;\; \cdots \;\; \boldsymbol{o}_{t-1}] = [\boldsymbol{o}(\boldsymbol{z}_0) \;\; \cdots \;\; \boldsymbol{o}(\boldsymbol{z}_{t-1})] \quad \text{(Equation 5)}$$

and

$$\boldsymbol{G}_{t-1} = [\boldsymbol{g}_0 \quad \cdots \quad \boldsymbol{g}_{t-1}] = [\boldsymbol{g}(\boldsymbol{z}_0) \quad \cdots \quad \boldsymbol{g}(\boldsymbol{z}_{t-1})] \tag{Equation 6}$$

This says that the memories formed at each timestep (when at position $\boldsymbol{z}_t$) are the *true* neural representation of position $\boldsymbol{z}_t$: $\boldsymbol{g}(\boldsymbol{z}_t)$ and the observation at position $\boldsymbol{z}_t$: $\boldsymbol{o}(\boldsymbol{z}_t)$. Thus the attention vector $\boldsymbol{h}_t = \text{softmax}(\boldsymbol{G}_{t-1}^T \boldsymbol{g}_t)$ will attend to the right memory at each timestep (i.e., memories that were paired to $\boldsymbol{g}(\boldsymbol{z})$ in the past) since $\boldsymbol{g}(\boldsymbol{z})$ will be more similar to itself, than to $\boldsymbol{g}(\boldsymbol{z}')$, where $\boldsymbol{z}'$ is a different position.

Simplifying an optimal EM model

We consider a optimal EM model, and consider its representations. We assume that if an underlying 'position', $\boldsymbol{z}$, is visited more than once, the additional memory is not added, thus after all n_p 'positions', $\boldsymbol{z}$, have been visited $\boldsymbol{G}_{t-1} = \boldsymbol{G} \in \mathbb{R}^{n_g \times n_p}$ and $\boldsymbol{O}_{t-1} = \boldsymbol{O} \in \mathbb{R}^{d_o \times n_p}$, where n_p is the number of 'positions' for this task: we only store n_p memories. Thus, since a single memory get retrieved at any one time, the optimal attention vector, $\boldsymbol{h}_t \in \mathbb{R}^{n_p}$ will be one-hot and with a single element active for each position. (If the target were to predict a mixture of past observations, the attention vector would not be one-hot, but the same argument follows just the same.) To make our lives easier, since attention is order invariant, we can relabel, and reorder, individual memories, not by the time, but by their 'position': $\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_{\boldsymbol{z}_0} & \cdots & \boldsymbol{g}_{\boldsymbol{z}_{n_p}} \end{bmatrix}$ and $\boldsymbol{O} = \begin{bmatrix} \boldsymbol{o}_{\boldsymbol{z}_0} & \cdots & \boldsymbol{o}_{\boldsymbol{z}_{n_p}} \end{bmatrix}$. Since the one-hot attention vector, after a velocity $\boldsymbol{v}$, will change to another one-hot attention vector (coding for position $\boldsymbol{z}$ then position $\boldsymbol{z}+\boldsymbol{v}$), the attention vector can be though of as representing a node on a graph. We call the attention vector at timestep t corresponding to position $\boldsymbol{z}_t$ as $\boldsymbol{h}(\boldsymbol{z}_t)$. Thus, the update to the attention vector is *functionally equivalent* to an attention vector being multiplied by an (velocity/action dependent) graph transition matrix, $\boldsymbol{M}_{\boldsymbol{v}_{t-1}}$, i.e., $\boldsymbol{h}(\boldsymbol{z}_t) = \boldsymbol{M}_{\boldsymbol{v}_{t-1}} \boldsymbol{h}(\boldsymbol{z}_t - \boldsymbol{v}_{t-1})$. Thus (optimal) predictions can be rewritten as:

$$\begin{aligned}
\hat{\boldsymbol{t}}_t &= \boldsymbol{O}\text{softmax}\left(\boldsymbol{G}^T \boldsymbol{g}(\boldsymbol{z}_t)\right) \quad \text{Full EM model but with memories reordered according to position.} \\
&= \boldsymbol{O}\boldsymbol{h}(\boldsymbol{z}_t) \quad \text{Functionally equivalent when optimal.} = \boldsymbol{O} \prod_{\tau=0}^{t-1} \boldsymbol{M}_{\boldsymbol{v}_\tau} [1 \quad 0 \quad \cdots \quad 0]^T \quad \text{Using recurrent relation } \boldsymbol{h}(\boldsymbol{z}_t) \\
&= \boldsymbol{M}_{\boldsymbol{v}_{t-1}} \boldsymbol{h}(\boldsymbol{z}_t - \boldsymbol{v}_{t-1}), \text{and assuming initial position } \boldsymbol{z}_0.
\end{aligned} \tag{Equation 7}$$

Expanding this equation out yields:

$$\hat{\boldsymbol{t}}_t = \boldsymbol{O} \prod_{\tau}^{t-1} \boldsymbol{M}_{\boldsymbol{v}_\tau} [1 \quad 0 \quad \cdots \quad 0]^T = \underbrace{\begin{bmatrix} \boldsymbol{o}_{\boldsymbol{z}_0} & \cdots & \boldsymbol{o}_{\boldsymbol{z}_{n_p}} \end{bmatrix}}_{\text{Memory slots stored in weights}} \underbrace{\prod_{\tau}^{t-1} \boldsymbol{M}_{\boldsymbol{v}_\tau} [1 \quad 0 \quad \cdots \quad 0]^T}_{\text{Attention vector controlled by the } \boldsymbol{g} \text{ RNN}} \tag{Equation 8}$$

This equation describes the functional computations of an optimal EM model. We see that it reduces to an attention vector $\boldsymbol{h}$ that gets updates by velocity-dependent graph transition matrices. This attention vector then attends to memories stored in weights. Interestingly, this simplified formulation, can still be thought of as an RNN (the attention vector RNN) and an external memory (the memory slots). But we stress that this is a simplified formulation, as opposed to the actual model implementation (see STAR Methods). Nevertheless it captures the computations of an optimal full EM model implementation.

Rearranging terms gives the WM solution

The above solution stores memories in synaptic connections. Here we show that reshaping and rearranging the above equation produces an alternative, but equivalent, solution where memories are stored in RNN activity rather than synaptic connections:

$$\hat{\boldsymbol{t}}_t = \underbrace{[\boldsymbol{I}_{d_o} \quad 0 \quad \cdots \quad 0]}_{\text{Readout weights} \in \mathbb{R}^{n_o \times n_p n_o}} \underbrace{\prod_{\tau}^{t-1} \underbrace{\boldsymbol{W}^*_{\boldsymbol{v}_\tau}}_{\text{Recurrent matrix} \in \mathbb{R}^{n_p n_o \times n_p n_o}} \underbrace{\begin{bmatrix} \boldsymbol{o}_{\boldsymbol{z}_0} \\ \vdots \\ \boldsymbol{o}_{\boldsymbol{z}_{n_p}} \end{bmatrix}}_{\text{WM slots} \in \mathbb{R}^{n_p n_o}}}_{\text{RNN activity} \in \mathbb{R}^{n_p n_o}} \tag{Equation 9}$$

This equation performs the exact same computation as the previous equation, and its terms look very similar but there are some important differences (Table S1 for relationships between terms). For example, $\boldsymbol{W}^*_{\boldsymbol{v}} \in \mathbb{R}^{n_p n_o \times n_p n_o}$ is like the previous velocity-dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$, but expanded (and transposed); where there was a 1 or a 0, there is now an identity matrix ($\boldsymbol{I} \in \mathbb{R}^{d_o \times d_o}$) or a matrix of zeros ($0 \in \mathbb{R}^{d_o \times d_o}$). Intriguingly, what was a matrix of memories stored in weights, $\begin{bmatrix} \boldsymbol{o}_{\boldsymbol{z}_0} & \cdots & \boldsymbol{o}_{\boldsymbol{z}_{n_p}} \end{bmatrix}$, is now a vector of memories stored in RNN activity, $\begin{bmatrix} \boldsymbol{o}_{\boldsymbol{z}_0} \\ \vdots \\ \boldsymbol{o}_{\boldsymbol{z}_{n_p}} \end{bmatrix}$. Thus the neurons in the RNN can be though of as **activity slots** that store arbitrary memories in neural activity (Figure 1A right). Importantly, since the readout weights are now fixed, the contents of each slot must be copied and shifted to other slots via $\boldsymbol{W}^*_{\boldsymbol{v}}$, depending on the desired observation to readout. Adding a memory to a slot is simple and only requires feed-forward weights from observation to input slot (e.g., Figure 1A right). Thus, once learned, this model requires no synaptic plasticity. Lastly, similar to the EM solution, this WM solution works in any basis (not just the one presented above).

Separate attractor networks for computing velocities, and controlling slots

We now show that an RNN can consist of two components: a component for computing velocities, and a component for using velocities to control activity slots. A generic RNN updates is as follows:

$$\boldsymbol{h}_t = f(\boldsymbol{W}_{rec}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{i}_t) \quad \text{(Equation 10)}$$

If we split $\boldsymbol{h}$ into two parts, $\boldsymbol{h} = [\boldsymbol{h}^s; \boldsymbol{h}^v]$, where $\boldsymbol{h}^s$ are the activity slots and $\boldsymbol{h}^v$ are the neurons involved in computing velocity signals (we're choosing a favourable basis where the two components are in separate neurons). Then we get the following:

$$\begin{aligned}\boldsymbol{h}_t = \begin{bmatrix}\boldsymbol{h}^s_t \\ \boldsymbol{h}^v_t\end{bmatrix} &= \begin{bmatrix} f(\boldsymbol{W}^{ss}_{rec}\boldsymbol{h}^s_{t-1} + \boldsymbol{W}^{sv}_{rec}\boldsymbol{h}^v_{t-1} + \boldsymbol{W}^s_{in}\boldsymbol{i}_t) \\ f(\boldsymbol{W}^{vv}_{rec}\boldsymbol{h}^v_{t-1} + \boldsymbol{W}^{vs}_{rec}\boldsymbol{h}^s_{t-1} + \boldsymbol{W}^v_{in}\boldsymbol{i}_t)\end{bmatrix} \\ &= \begin{bmatrix} f(\boldsymbol{W}^{ss}_{rec}\boldsymbol{h}^v_{t-1} + \boldsymbol{W}^{sv}_{rec}\boldsymbol{h}^v_{t-1} + \boldsymbol{W}^s_{in}\boldsymbol{i}_t) \\ f(\boldsymbol{W}^{vv}_{rec}\boldsymbol{h}^v_{t-1} + \boldsymbol{W}^v_{in}\boldsymbol{i}_t)\end{bmatrix}\end{aligned} \quad \text{(Equation 11)}$$

Where $\boldsymbol{W}^{sv}_{rec}$ is the block of $\boldsymbol{W}_{rec}$ that goes from $v \to s$ neurons etc, and in the last line we have assumed that information from slot neurons are not useful to velocity neurons ($\boldsymbol{W}^{vs}_{rec} = 0$). Thus this RNN consists of a velocity computing RNN, $\boldsymbol{h}^v$, that computes velocities based on inputs and sends these velocities to the slot RNN, $\boldsymbol{h}^s$, vis $\boldsymbol{W}^{sv}_{rec}\boldsymbol{h}^v_{t-1}$.

Model Details

We train models which are all variants on recurrent neural networks (RNNs), which differ in two key components (Figure S2). The two varying components are: 1) we use different RNN variants (varying from least to most biologically plausible), and 2) we use different methods of predictions from RNN activities for EM models (via fast Hebbian weights that change every timestep) and WM models (via slow weights learned by backpropagation).

RNN transitions

We use a gated RNN similar to a GRU,[64] where the external and recurrent inputs are gated: $\boldsymbol{h}_t = f(\boldsymbol{g} \odot \boldsymbol{r}_t + (1 - \boldsymbol{g}) \odot \boldsymbol{W}_{in}\boldsymbol{i}_t)$, where $\boldsymbol{i}_t$ is the external input at timestep t, $\boldsymbol{r}_t$ is the recurrent input, $f(\ldots)$ is an activation function, and $\boldsymbol{g} \in \mathbb{R}^{n_h}$ is an element-wise gate $\boldsymbol{g} = \sigma(\boldsymbol{W}_{\alpha,1}\boldsymbol{r}_t + \boldsymbol{W}_{\alpha,2}\boldsymbol{W}_{in}\boldsymbol{i}_t)$ with the Sigmoid activation, σ. We compute the recurrent and external inputs in two different ways. First, like our theory, we use (learned) velocity-dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$: $\boldsymbol{r}_t = \boldsymbol{W}_{\boldsymbol{v}_{t-1}}\boldsymbol{h}_{t-1}$, and the external input, $\boldsymbol{i}_t$, is just $\boldsymbol{o}_t$. We call this *GroupRNN* (Figure S2C). Second, we use a conventional gated RNN, with $\boldsymbol{r}_t = \boldsymbol{W}_{rec}\boldsymbol{h}_{t-1}$, and the external input, $\boldsymbol{i}_t$, is $\boldsymbol{o}_t$ and $\boldsymbol{v}_t$ concatenated. We call this *RegularRNN* (Figure S2D). In Figure 3 we make use of another RNN inspired by the fly head direction circuit (Figures S2E and S2F). In *BioRNN*, we have $\boldsymbol{r}_t = \boldsymbol{W}_2 f(\boldsymbol{W}_1\boldsymbol{h}_{t-1} + \boldsymbol{v}_{t-1})$, where $\boldsymbol{W}_1$ projects $\boldsymbol{h}$ into a higher dimensional space. The external input, $\boldsymbol{i}_t$, is just $\boldsymbol{o}_t$.

It is unlikely that velocity dependent weights matrices (i.e., the GroupRNN), as per our theory, are exactly what the brain uses since it requires modulation of synapses depending on velocity. It is more likely that the brain uses additive velocity signals like a RegularRNN/BioRNN. Importantly, however, in our simulations of both GroupRNN (velocity dependent weights matrices) and RegularRNN/BioRNN (additive velocity input) slot representations are learned. This is perhaps not surprising since the standard formulations of continuous attractor neural networks have separate populations of neurons for each direction (e.g., left and right neurons for a ring attractor), which is closely related to velocity dependent matrices as the velocity input essentially gates the left/ right neurons which have left/right projections. Thus we anticipate our theory also applies to path-integrating networks learned with additive velocity input like RegularRNN and CANNs.

Fast or slow readout weights

We use two readout methods to correspond to the two model classes in our theory. First, for the **WM** class: predictions are made by $\widehat{\boldsymbol{t}}^{WM}_t = \boldsymbol{W}^{WM}_{out} f(\boldsymbol{r}_t)$, i.e., standard RNN prediction (Figure S2B). Second for the **EM** class: predictions are made using an external memory system related to a transformer / Hopfield network. In particular, $\widehat{\boldsymbol{t}}^{EM}_t = \boldsymbol{W}^{EM}_{out}\boldsymbol{V}_{t-1}\text{softmax}(\boldsymbol{K}^T_{t-1}\boldsymbol{W}_q f(\boldsymbol{r}_t))$, where the key and value matrix are sequentially updated i.e., $\boldsymbol{K}_{t-1} = [\boldsymbol{k}_0 \quad \cdots \quad \boldsymbol{k}_{t-1}]$ and $\boldsymbol{V}_{t-1} = [\boldsymbol{u}_0 \quad \cdots \quad \boldsymbol{u}_{t-1}]$ (Figure S2A; we use $\boldsymbol{u}$ for 'value' as $\boldsymbol{v}$ is already used for velocity). At the beginning of each sequence, $\boldsymbol{K}_t$ and $\boldsymbol{V}_t$ are reset to be empty.

Adding memories in EM models

Memories are simply added to a list, just like the differentiable neural dictionary[63] or modern Hopfield network.[37] These memories consist of two parts - a key and a value (Figure S2A). The added key, as described above, at each timestep is $\boldsymbol{k}_t = \boldsymbol{W}_k\boldsymbol{h}_t$, while the added value is $\boldsymbol{u}_t = \boldsymbol{W}_v\boldsymbol{h}_t - \widehat{\boldsymbol{m}}_t$, where $\widehat{\boldsymbol{m}}_t$ is the retrieved memory at that timestep ($\widehat{\boldsymbol{m}}_t = \boldsymbol{V}_{t-1}\text{softmax}(\boldsymbol{K}^T_{t-1}\boldsymbol{W}_q f(\boldsymbol{r}_t))$). We do this, so that repeat copies of memories are not added.

Since the list of memories grows in time, we add an additional β scaling term in the softmax: $\widehat{\boldsymbol{t}}^{EM}_t = \boldsymbol{W}^{EM}_{out}\boldsymbol{V}_{t-1}\text{softmax}(\beta\boldsymbol{K}^T_{t-1}\boldsymbol{W}_q f(\boldsymbol{r}_t))$. We do this as the normalisation term in the softmax sums over the number of memories, and so more memories down-weights attention probabilities. We want the attention vector to not be affected by the number of elements in the set. In particular, we weight the softmax by $log(n_{memories})$.

Adding memories in WM models

There is no explicit memory adding process in WM models. There are simply input weights from the input at each timestep, $\boldsymbol{i}_t$, and so the RNN must figure out how to hold these inputs in its recurrent memory.

Normalisation

We use normalisation in two places. 1) In EM models, after retrieving a memory i.e., before applying $\boldsymbol{W}_{out}^{EM}$ we perform a layer normalisation[65]: $\text{LayerNorm}(\boldsymbol{V}_{t-1}\text{softmax}(\boldsymbol{K}_{t-1}^{T}\boldsymbol{W}_q f(\boldsymbol{r}_t)))$. This is so all retrieved memories have the same scaling. 2) In some simulations (detailed in Table S2) we also normalise the output of the RNN for both WM and EM models. We do this for consistency when comparing WM and EM models. For WM models this means $\widehat{\boldsymbol{t}}_t^{WM} = \boldsymbol{W}_{out}^{WM}\text{LayerNorm} f(\boldsymbol{r}_t)$. For EM models this means memories are retrieved as $\text{LayerNorm}(\boldsymbol{V}_{t-1}\text{softmax}(\beta\boldsymbol{K}_{t-1}^{T}\boldsymbol{W}_q\text{LayerNorm}(f(\boldsymbol{r}_t))))$.

Optimisation

All weight matrices are learned by backpropagation in both models. We optimise a prediction loss on the target, L_t, which is a cross-entropy loss when observations and targets are one-hot (all simulations other than Figures 5 and 6), or a squared error loss otherwise (Figures 5 and 6). For all simulations we add weights decay. For the 'additional constraints' in Figure 3 we add a L2 loss on the RNN activity. Also in Figure 3, we use an additional KL loss (where specified) which is a squared error loss between inferred ($\boldsymbol{h}$) and predicted ($f(\boldsymbol{r})$) RNN activity. All loss hyperparameters specified in Table S2.

Additional Task Details

Here we provide further task details. In all tasks we train on many instantiation of the task, i.e., observations randomised across each instantiation. In all tasks observations are randomly sampled at positions with replacement, except for the delay task and the PFC tasks.

Immediate Serial Recall (ISR)

Observations repeat cyclically, and the task is to predict the upcoming observation (Figure 1). There is no velocity signal. e.g., a 4-cycle: $\boldsymbol{o} = \{1,4,3,1,4,3,1\ldots\}$ and $\boldsymbol{t} = \{-,-,-,1,4,3,1\ldots\}$. '$-$' means a target that we do not try to predict since it has not been observed before and therefore is not possible to predict. The length of the each sequence is 7 multiplied by the repeat length.

N-back

Observations are randomly drawn at each step, and the task is to recall the observation n timesteps ago. There is no velocity signal. e.g., for $N = 3$: $\boldsymbol{o} = \{1,2,4,2,3,5,9\ldots\}$ and $\boldsymbol{t} = \{-,-,-,1,2,4,2\ldots\}$. The length of the each sequence is 7 multiplied by N.

1D Navigation

Observations come from random walks on a loop. Velocity signals are provided. e.g., a 3-loop: $\boldsymbol{o} = \{1,3,2,2,3,1,2\ldots\}$, $\boldsymbol{v} = \{+1,+1,0,0,-1,-1,-1\ldots\}$ and $\boldsymbol{t} = \{-,-,-,2,3,1,2\ldots\}$. While velocities are randomly sampled at each step, to encourage exploration of the full loop, we increase the likelihood of repeating velocities (apart from the 0 velocity). The length of the each sequence is 7 multiplied by the loop size.

2D Navigation

Observations come from random walks on a 2D torus. Velocity signals are provided. e.g., a 2x2 torus: $\boldsymbol{o} = \{5,1,5,2,3,1,1\ldots\}$, $\boldsymbol{v} = \{(+1,0),(+1,0),(0,+1),(-1,0),(0,-1),(0,0),\ldots\}$, and $\boldsymbol{t} = \{-,-,5,-,-,1,1\ldots\}$. The length of the each sequence is 7 multiplied by the torus size.

To demonstrate the generality of slots, we show that they are learned even when training on all targets i.e., including those that are impossible to predict correctly (as those states have not been visited yet). All models learn slots (Figures S4E–S4H), but understandably the models are less capable of learning larger task sizes because their training is less stable due to having error gradient signals from the impossible targets to predict.

2-ISR with delays

See also Figure 4. Delay period lengths were randomly sampled from [3,4,5,6,7] for each task. Example observation sequences are $\boldsymbol{o} = \{4,*,*,2,*,*,4,*,*,2,\ldots\}$ or $\boldsymbol{o} = \{1,*,3,*,1,*,3,\ldots\}$ where $*$ is the common delay observation (that also must be predicted). Corresponding target sequences are $\boldsymbol{t} = \{-,-,-,-,-,-,-,*,*,2,\ldots\}$ or $\boldsymbol{t} = \{-,-,-,-,-,*,3,\ldots\}$. Again we do not train on targets not possible to predict (i.e., observations/delays that have never been visited). We also so not train on the \nth1 return to the initial position. This is because it also cannot be predicted as it could have been a delay step. Observations are sampled without replacement. The length of the each sequence is 7 multiplied average full loop length (averaging over delays).

4-ISR with delays

See also Figure 4. Delay period lengths were randomly sampled from [3,4,5] for each task. Example observation sequences are $\boldsymbol{o} = \{4,*,2,*,*,2,*,6,4,*,2,*,*,2,*,6,\ldots\}$ or $\boldsymbol{o} = \{1,3,*,8,*,2,1,3,*,8,*,2,\ldots\}$ where $*$ is the common delay observation (that also must be predicted). Corresponding target sequences are $\boldsymbol{t} = \{-,-,-,-,-,-,-,-,-,*,2,*,*,2,*,6,\ldots\}$ or $\boldsymbol{t} = \{-,-,-,-,-,-,-,3,*,8,*,2,\ldots\}$. Again we do not train on targets not possible to predict (i.e., observations/delays that have never been visited). We also so not train on the \nth1 return to the initial position. This is because it also cannot be predicted as it could have been a delay step. Observations are sampled without replacement. The length of the each sequence is 7 multiplied average full loop length (averaging over delays).

1D Navigation with structured velocity signals

See also Figure S7. We used a 7-loop task (1D navigation), but where the underlying velocity cycles a fixed schedule of $\{+2,+1,0,-1,0,+1,+2,+1,0,\ldots\}$ (velocity cycled on a 6-loop). Velocity signals were not provided. The length of the overall sequence is 7 multiplied by the loop size.

2D Navigation with structured velocity signals

See also Figure S7. Sequences we drawn from a 3-by-3 torus, but where the underlying velocity cycles a fixed schedule of $\{N,W,S,S,E,E,N,N,N,W,S,S,\ldots\}$ (velocity cycled on a 8-loop). Velocity signals were not provided. The length of the overall sequence is 7 multiplied by the torus size.

1D Navigation using macro-actions

See also Figure S7. We used a 8-loop task (1D navigation). We used 9 possible macro actions: $\{[+1,+1,+1,+1],[+1,0,+1,+1],[+1,+1,+1,-1],[-1,+1,0,+1],[+1,+1,-1,-1],[0,-1,+1,-1],[-1,0,0,-1],[-1,-1,-1,0],[-1,-1,-1,-1]\}$

Macro-actions were sampled randomly, with a slight bias to repeating the same macro-action (to encourage full exploration of the loop). The length of the overall sequence is 7 multiplied by the loop size.

2D Navigation using macro-actions

See also Figure S7. Sequences we drawn from a 3-by-3 torus. We used 8 possible macro actions: $\{[N,W,0,E],[S,0,E,N],[N,S,E,0],[0,S,W,E],[S,S,E,N],[W,W,S,E],[N,N,W,S],[E,E,N,W]]\}$, where 0 means no velocity. Macro-actions were sampled randomly. The length of the overall sequence is 7 multiplied by the torus size.

Xie et al.

See also Figure 5. Here observations are one of six 2D positions lying on a ring. This is just like the original paper[28]). Thus the observation vector consists of two real numbers rather than a one-hot vector as before. Likewise for the targets. Observations are sampled without replacement. Otherwise the set-up is the same as a 3-ISR task, but only with a single repeat (i.e., 6 observations in total), e.g., $\{\boldsymbol{p}_5,\boldsymbol{p}_3,\boldsymbol{p}_2,\boldsymbol{p}_5,\boldsymbol{p}_3,\boldsymbol{p}_2\}$ and $\boldsymbol{t}=\{-,-,-,\boldsymbol{p}_5,\boldsymbol{p}_3,\boldsymbol{p}_2\}$ where $\boldsymbol{p}_i$ is a 2D position vector (corresponding to the six positions lying on a ring).

Panichello et al.

See also Figure 6. Here observations are one of four 2D coordinates lying on a ring. The original paper uses randomly sampled colours (from a 2D colour ring), and for visualisation purposes they bin these colours into 4 bins. Thus we just use 4 example colours (we could have used randomly drawn colours - the activity slot predictions are the same). Thus the observation vector consists of two real numbers (corresponding to a colour the colour wheel) rather than a one-hot vector as before. Likewise for the targets. Observations are sampled without replacement. The length of the sequence is 4 (top colour, bottom colour, cue, target), e.g., $\{\boldsymbol{c}_3,\boldsymbol{c}_2,cue=bottom,*\}$ and $\boldsymbol{t}=\{-,-,-,\boldsymbol{c}_2\}$, where $\boldsymbol{c}$ is a 2D vector (corresponding to a colour on the colour wheel).

QUANTIFICATION AND STATISTICAL ANALYSIS

Decoding Analyses

For all decoding analyses, we take the neural representations at each timestep in the sequence, along with the value of the variable we wish to decode at each timestep in the sequence. We collect this data for many timesteps and for multiple (240) random sequences, and collate this data into a training dataset (number of data-points is $240\times s$ where s is sequence length), with inputs being the neural representations, and outputs being the variable we wish to decode. In all decoding analyses, we only decode from timesteps after all positions have been visited at least once (so activity slots can be filled). We use (multi-class) logistic regression (using the sklearn python package) for all decoding analyses (apart from task-progress slot decoding - see later), as we discretise continuous random variables into discrete classes. We train the decoders and then test on a held out dataset of 60 random sequences (all presented results are test accuracy). We now describe how we obtain the particular variables we wish to decode.

Position decoding

We define the first element of each sequence to be at position 0, and then track position from then onwards (i.e., we track relative position to initial position). We give each position a unique identifier, so we can perform multi-class logistic regression.

Slot decoding

For every sequence, we calculate each slot's stored observation according to our theory at all timesteps (Figure 2H; we call this the slot 'slot-sequence' for each slot). We decode each slot-sequence individually using multi-class logistic regression. When taking the average decoding performance over slots, we do not include slot 1, i.e., the input slot, as it will always decode to 1 (since the slot-sequence for slot 1 is just in input sequence which gets provided to the model). While for each task the observations in the slots are permutations of one-another (e.g., Figure 2H) and so it may be thought that if you can decode slot 1 (the input slot!) then you can decode any other slot, the permutation is not consistent over task instantiations. Thus, since we decode multiple sequences from multiple task instantiations at once, only RNNs that have actually learned a slot representation will have high slot-sequence decoding accuracy.

Past decoding

For every sequence, at each timestep we calculate which observation occurred n timesteps ago (Figure 2H; we call this 'past-sequence' for each time lag n). We do this for $n=0$ to $n=n_s-1$ where n_s is the number of slots. We run a separate decoder for each value of n. When taking the average decoding performance over past-sequences, do not include $n=0$ as it will always decode to 1 (since the current observation is being provided to the model).

Progress decoding

To calculate progress, at each timestep, we calculate the number of delay steps taken since an observation and divide by the total number of delay steps (between neighbouring observations). We then discretise progress into 4 chunks, i.e., 0%→25%, 25%→50%, 50%→75%, 75%→100%.

Progress-velocity decoding

We calculate progress-velocity, for each timestep, we compute 1 divided by the total number of delay steps (between neighbouring observations). We then provide a unique identifier to its value. Thus decoding progress-velocity, is the same as decoding the number of delay steps between the current neighbouring observations.

Task-progress slot decoding

We first discrete the task into $T \times N$ task-progress slots where T is positions in task (i.e., 2 for the 2-ISR delay task) and $N = 4$ is the number of progress chunks after discretising progress. At each timestep we calculate which observation our theory says should be in each task-progress slot. Most slots will be empty as there are only T observations but $T \times N$ task-progress slots. Thus rather than having a separate decoder for each task-progress slot, we train a single 2-layer linear neural network (with sigmoid activation function on the output; 2 linear layers as it trains faster even though it is no more expressive) where the inputs are neural representation at each timestep and the outputs are whether an observation is in a particular slot. There output dimension is then $n_o \times T \times N = 80$ for the 2-ISR delay task, where n_o is the number of observations. When observation a is task-progress slot b, then the corresponding output neuron is 1 and 0 otherwise. Thus most entries will be 0 for the 2-ISR delay task: there will only be two 1 entries out of $n_o TN = 80$, and so 0.975 is the baseline accuracy (which we normalise to in Figure 4).

Mutual Information Ratio

To calculate the mutual information ratio[39] (MIR), we first compute the mutual information between all active neurons (defined as neurons with average activity greater than $\frac{1}{20}$ of the average neural activity) and the 'slot-sequence' for each slot (slot-sequences are the predicted stream of observations in each slot; Figure 2H). This gives a matrix of dimensions number of active neurons by number of slots. For each neuron we take its mutual information vector (of dimension number of slots), and calculate the maximum value of the vector divided by the sum of the vector (mutual information is always non-negative). This determines how specialised each neuron is for each slot. The MIR is then the average value over all active neurons. We do not include slot 1, i.e., the input slot, in this analysis.

Slot Algebra

To compute the slot algebra score, we generate sequences from four environments of identical structure, where the environments differ in their arrangement of observations in a special way. In particular the first environment has a random arrangement of observations, the second environment has a random arrangement of observations apart from one position (position p) where it has the same observation as the first environment. The third environment has the same arrangement of observations as the second environment apart from position p where it has a different, random, observation. The fourth environment is the same as the first environment, but in position p it has the same observation as the third environment. We then generate different random sequences for each of these environments. We choose a random position i, and record the neural representation for a random visit to position i on each of the four sequences. This gives us $Rep_i(env_1)$, $Rep_i(env_2)$, $Rep_i(env_3)$, and $Rep_i(env_4)$, where $Rep_i(env_j)$ means the neural representation of environment j when at position i. The slot algebra score $SA = \frac{d_{1-2+3-4}}{d_{1-2+3-4}+d_{1-4}}$, where $(d_{1-2+3-4})$ is $\|Rep_i(env_1) - Rep_i(env_2)+Rep_i(env_3) - Rep_i(env_4)\|^2$ and $d_{1-4} = \|Rep_i(env_1) - Rep_i(env_3)\|^2$ where $\|\cdots\|^2$ means square and sum all vector elements. We repeat this process 200 times for each model and average the individual SA scores to give the overall slot algebra score for that model.

We occasionally use an extra regularisation term (KL) which asks the 'predicted' internal representation, $\boldsymbol{r}_t$, to the same as the 'inferred' representation, $\boldsymbol{h}_t$. In particular, we add an extra term to the loss that is $\lambda_{KL}\|\boldsymbol{h}_t - \boldsymbol{r}_t\|^2$, where λ_{KL} is the regularisation strength (see Table S2 for its value).

To ensure that the poor slot algebra scores of the regular RNN are not simply due to the fact that we provide the velocity of the *upcoming transition* to the network at time-step t (i.e., the recurrent input $\boldsymbol{r}_t = \boldsymbol{W}_{rec}\boldsymbol{h}_{t-1}$, and the external input, $\boldsymbol{i}_t$, is $\boldsymbol{o}_t$ and $\boldsymbol{v}_t$ concatenated; reminder that $\boldsymbol{v}_t$ is the velocity from that transition from 'true underlying' state $\boldsymbol{z}_t$ to $\boldsymbol{z}_{t+1}$), we additionally train networks where the velocity signal of the upcoming transition is only provided to the path integration step, i.e., $\boldsymbol{r}_t = \boldsymbol{W}_{rec}\boldsymbol{h}_{t-1} + \boldsymbol{B}\boldsymbol{v}_t$ ($\boldsymbol{B}$ is a learnable matrix), and the external input, $\boldsymbol{i}_t$, is just $\boldsymbol{o}_t$. This new network, in theory, could have no contamination of its internal representation with velocity, i.e., $\boldsymbol{h}_t$ could be independent of velocity, and therefore have low slot algebra scores. Nevertheless, we find that these networks do not have low slot algebra scores (Figure S5B), suggesting that the $\boldsymbol{h}_t$ is dependent on the velocities signals taken to reach state $\boldsymbol{z}_t$.

PFC Analyses

We followed the exact analyses described in Xie et al.[28] for Figure 5 and the exact analyses described in Panichello et al.[29] for Figure 6.