On prefrontal working memory and hippocampal episodic memory: Unifying memories stored in weights and activity slots

James C.R. Whittington[1,2,*], 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, 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
[*]corresponding author(s): jcrwhittington@gmail.com

ABSTRACT

Remembering events in the past is crucial to intelligent behaviour. Flexible memory retrieval, beyond simple recall, requires a cognitive map, or model of how sensations, actions, and latent environmental or task states are all related to one another. Two key brain systems are implicated in this process: the hippocampal episodic memory (EM) system and the prefrontal working memory (WM) system. While an understanding of the hippocampal system, from computation to algorithm and representation, is emerging, less is understood about how the prefrontal WM system can give rise to flexible computations beyond simple memory retrieval, and even less is understood about how the two systems relate to each other. Here 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. In doing so, we develop a formal theory of the algorithms and representations of prefrontal WM in terms of structured, and controllable, neural subspaces (termed activity slots) that together can represent a dynamic cognitive map without any need for synaptic plasticity. By building models using this formalism, we elucidate the differences, similarities, and trade-offs between the hippocampal and prefrontal algorithms. Lastly, we show that several prefrontal representations in tasks ranging 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 basis for understanding dynamic prefrontal representations of WM.

Introduction

Being able to predict what will happen next in novel structured environments constitutes a fundamental feature of intelligent cognition. However, due to the one dimensional and irreversible nature of time itself, we can only learn about structured environments, and the consequences of our actions in them, through a spatiotemporally local sequence of transient sensorimotor experiences. Thus, to learn to predict the future, both brains[1–3] and machines[4–6] must convert this transient sequence of sensorimotor experience into internal models of the world that remember and exploit structured relationships between actions, sensations, and environmental features. 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 local sequential sensorimotor experience is converted into a rich cognitive map capable of predicting the future counterfactual consequences of diverse potential actions remains a major aim of cognitive neuroscience.

Two key brain regions are proposed to 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 why two systems are employed to solve the same problem nor how these brain systems are related in either underlying algorithms or representations.

For sequential episodic memory in the hippocampal formation, ideas are beginning to emerge on underlying algorithms and representations[4,12–14]. In these models, memories are stored in hippocampus (HPC) by updating synaptic connections via Hebbian learning[15] (like a Hopfield network[16]), 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 are able to explain many cellular recordings in the hippocampal formation for both spatial and non-spatial tasks; hippocampal cells such as place cells[17], landmark cells[18], and splitter cells[19] are explained as memory representations, while entorhinal cells such as grid cells[20], object-vector cells[21], border-vector cells[22], non-spatial grid cells[23,24], non-spatial sound frequency cells[25] are explained as representing position, so that the right memory is retrieved at the right position. 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., where you are in space) to recall the right memory at the right time. Thus even in this case, successful sequence memory goes beyond mere rote memorisation of sensory experience, but rather reflects a more sophisticated cognitive map building process that learns systematic relationships between sensations and actions induced by latent spatial or task structure[4,5,12–14].

For sequence working memory in PFC, our understanding is limited to tasks of repeating sequences exactly as they were presented. Here, findings from both artificial[26–28] and biological[29,30] networks suggest memories of items are organised into neural subspaces according to their ordinal position. Crucially, unlike HPC, storing these memories requires no additional synaptic plasticity since recurrent connections in PFC update and maintain memories stored in neural activity. While this has revealed mechanisms of sequential memory retrieval, PFC is implicated in tasks requiring flexible control of memory retrieval[31–33], i.e., not just repeating sequences exactly as they were presented. While we don't understand how PFC working memory models allow such flexible control, we do for HPC episodic memory models. Thus, if we could relate PFC working memory and HPC episodic memory then we could understand how the PFC working memory system flexibly controls its memories to tackle sequence memory and prediction tasks that go beyond simple repetitive sequences and instead require cognitive map-like knowledge of structured relationships between, sensations, actions, and latent spatial or non-spatial variables.

Unfortunately no relationships currently exist between PFC working memory and HPC episodic memory. Excitingly, though, recent results from the machine learning literature suggest there may be a relationship. In particular, simplified (linear) versions of transformer neural networks[6], which are closely related to Hopfield networks[34–36], have a recurrent reformulation[37,38] which is suggestive of a relationship between episodic and working memory models.

In this work we provide an understanding of sequence working memory in PFC and how it relates, in representation and algorithm, to HPC episodic memory. 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 utilise different neural representations, with EM using abstract positions, while WM uses slots (distinct neural subspaces) in neural activity; 4) demonstrate that the WM slot representation affords scene algebra; 5) recover single neuron slot coding under certain constraints; 6) demonstrate how internally computed velocity signals can control the contents of PFC slots; 7) 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 episodic HPC memory and PFC working memory is a new, unifying theoretical framework for how PFC working memory networks could, in principle, 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 and prediction is a structure learning problem. Sequence memory and prediction tasks involve recalling previous observations, but where correct recall depends on an underlying task structure. For example, in Immediate Serial Recall[39] (ISR), a series of observations are presented in order, and then they must be recalled in order. Here the underlying structure is a ordinal line, but other sequences can be drawn from other underlying structures. For example if a sequence is drawn from navigating in 2D space, then including this structural knowledge in an internal representation will dramatically facilitate recall and prediction, as some transitions are possible in 2D space, but others are not. For example, in a 2D navigation task, it is possible to predict what one will see next when traversing a closed loop for the first time, but this requires not only remembering one's past sequence of observations, but also combining this sequence memory with knowledge of the structure of 2D space, and how actions move us within the space (i.e. if the integral of your recent velocity will become 0 then you will return to a previously experienced location). This facilitation of sequence memory and prediction by exploiting knowledge of task structure is possible in all problems where there is a common structural constraint. Even when each problem consists of sequences with different observations (so one problem's sequence cannot be memorised and used for another problem), knowledge of the common task structure across problems facilitates recall and prediction within each problem. In machine learning terminology, 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 $\mathscr{D} = \{\mathscr{D}_0, \mathscr{D}_1, \cdots, \mathscr{D}_N\}$. Each $\mathscr{D}_i$ is one of N problem instances of the common task, and is simply a sequence consisting of vectors of sensory observations, $\boldsymbol{o}$, of dimension d_o (termed $\boldsymbol{o} \in \mathbb{R}^{d_o}$), sensory targets, $\boldsymbol{t} \in \mathbb{R}^{d_o}$, and (allocentric) velocities, $\boldsymbol{v} \in \mathbb{R}^{d_v}$, i.e., $\mathscr{D}_i = \{(\boldsymbol{o}_0, \boldsymbol{t}_0, \boldsymbol{v}_0), (\boldsymbol{o}_1, \boldsymbol{t}_1, \boldsymbol{v}_1), \cdots, (\boldsymbol{o}_K, \boldsymbol{t}_K, \boldsymbol{v}_K)\}$ if the sequence is of length K. These velocities reflect actions taken by agents, but in this case we provide them externally. Importantly, the underlying structure of the task is determined by how successive actions cumulatively change latent states within the task, which themselves are never *directly* observed by the agent.

Concretely, we assume there is an underlying latent variable, $\boldsymbol{z} \in \mathbb{R}^{d_z}$, which we can think of as corresponding to a latent agent or task state. In spatial contexts we can think of the agent's latent state as its position. Furthermore, we assume each

latent state, or position $\boldsymbol{z}$, is associated with a visible observation $\boldsymbol{o}$, corresponding to what the agent would sense if it had the latent state, or position $\boldsymbol{z}$. Also, we assume that for any ordered pair of neighbouring source and target latent states, there is a unique action, or velocity $\boldsymbol{v}$ that modifies the agent's latent state from the source to the target (see Figure 1A left for a simple loop structure with one action). The task structure is encoded by how actions, or velocities, change the latent state, or position. For example, in the case of navigation in a 2D grid, there are 4 elementary actions, or velocities, corresponding to moving one step `North`, `South`, `West` or `East`, and each step moves the latent 2D grid position $\boldsymbol{z}$ by one step in the corresponding direction. In our formulation, successive actions correspond to the addition of velocities, and so when a sequence of velocities additively cancel (i.e. `North` + `East` + `South` + `West` = 0 for a 2D grid), the then the agent necessarily returns to the same latent position and encounters the same observation at the beginning and end of the action sequence.

Now, returning to the data for a single task $\mathscr{D} = \{\mathscr{D}_0, \mathscr{D}_1, \cdots, \mathscr{D}_N\}$, each individual problem instance $\mathscr{D}_i$ 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, specified by how actions, or velocities $\boldsymbol{v}$ change latent states. The aim of the task is to predict a target, $\boldsymbol{t}$, at each timestep 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, neither latent states, nor how actions affect latent states, are directly observed by the agent. Instead the agent only experiences a spatiotemporally local sequence of transient sensations and actions. To successfully solve the prediction or recall task across all problems, the agent must build a cognitive map of the common task structure that captures how actions affect latent states, and how latent states are bound to observations. Only then, for example can the agent predict what it will see next upon the first traversal of a closed loop in a 2D grid. Finally, we note that 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).

Next we will develop a theory for a very simple network realisation of an EM solution, and from that derive a very simple network realisation of a WM solution. But crucially, we will show below that many of the features of these simple solutions hold in more general scenarios, both in generic artificial networks of machines, and in biological networks of animals, that successfully solve structured sequence memory and prediction tasks.

Solving sequence memory tasks with EM. The hippocampal literature has developed EM models that solve these tasks[4,14]. These models are built from two components (1A top-middle). The first component is an RNN that learns to *explicitly* represent position in a manner that generalises across tasks. The second component is a memory system that binds observation representations to position representations and stores this memory in synaptic weights. Critically these weights change in every problem so different problems from a common task can be solved with a common position code. In its simplest form, the RNN has a single neuron active for each position (Figure 1A middle; full theory in Appendix A.1). Sensory memories for each position are encoded in synaptic weights between each RNN neuron (corresponding to that position) and the observation representation, with memories added (weights updated) via Hebbian learning. The role of the RNN is to track the underlying latent variable, $\boldsymbol{z}$, so that the correct position cell is activated at the right time to retrieve the right memory (Figure 1A middle; the leftmost position cell activates to recall the observation at position A). Tracking position means the RNN integrates velocity signals, $\boldsymbol{v}$, to update its representation from $\boldsymbol{h}(\boldsymbol{z}-\boldsymbol{v})$ to $\boldsymbol{h}(\boldsymbol{z})$ (i.e. from position $\boldsymbol{z}-\boldsymbol{v}$ to position $\boldsymbol{z}$). This can be done with velocity dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$, that follow the transition rules of $\boldsymbol{z}$ (e.g., $\boldsymbol{W}_{\boldsymbol{v}}\boldsymbol{W}_{-\boldsymbol{v}} = \boldsymbol{I}$; going `North` then `South` takes you to the same position). Mathematically, the RNN update is $\boldsymbol{h}(\boldsymbol{z}) = \boldsymbol{W}_{\boldsymbol{v}}\boldsymbol{h}(\boldsymbol{z}-\boldsymbol{v})$ (Figure 1C). The overall process of generating target predictions (assuming all observations have already been seen and added as memories - the $\boldsymbol{o}$ terms in the below equation) is succinctly described in the following equation:

$$\hat{\boldsymbol{t}}_{t+1} = \underbrace{\begin{bmatrix}\boldsymbol{o}_{z_0} & \cdots & \boldsymbol{o}_{z_{n_p}}\end{bmatrix}}_{\text{EM slots} \in \mathbb{R}^{d_o \times n_p}} \underbrace{\prod_{\tau=0}^{t} \underbrace{\boldsymbol{W}_{\boldsymbol{v}_\tau}}_{\text{Recurrent matrix} \in \mathbb{R}^{n_p \times n_p}} \underbrace{\begin{bmatrix}1 & 0 & \cdots & 0\end{bmatrix}^T}_{\text{Initial RNN, } \boldsymbol{h}_0 \in \mathbb{R}^{n_p}}}_{\text{RNN, } \boldsymbol{h}_{t+1} \in \mathbb{R}^{n_p}} \tag{1}$$

We see that, an initial RNN representation, $\boldsymbol{h}_0$, is successively updated by velocity dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$, to code for position at timestep $t+1$: $\boldsymbol{h}_{t+1}$. To make predictions, $\boldsymbol{h}_{t+1}$ selects (like 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; each slot corresponds to a position $\boldsymbol{z}$ and stores the observation at position $\boldsymbol{z}$; $\boldsymbol{o}_{\boldsymbol{z}}$). For clarity of presentation, we chose a basis in which the RNN vector is one-hot (has one cell active at any time) and the matrices $\boldsymbol{W}_{\boldsymbol{v}}$ have columns that are all zeros except for a single 1 (e.g., Figure 1C), though the above solutions works in any basis, i.e., $\boldsymbol{h} \leftarrow \boldsymbol{O}^T\boldsymbol{h}$, $\boldsymbol{W}_{\boldsymbol{v}} \leftarrow \boldsymbol{O}^T\boldsymbol{W}_{\boldsymbol{v}}\boldsymbol{O}$, etc. Indeed, under simple constraints, the optimal solution to this basis is a grid cell basis[4,40].

Solving sequence memory tasks with WM. The above solution stores memories in synaptic connections, and is thus relatable to the hippocampal formation. However, the PFC is thought to store sequence memories in the dynamics of neural activity as opposed to synaptic connections[27,30] (1A top-right; although see[41]). 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 (Figure 1A right):

$$\hat{\boldsymbol{t}}_{t+1} = \underbrace{\begin{bmatrix}\boldsymbol{I}_{d_o} & 0 & \cdots & 0\end{bmatrix}}_{\text{Readout weights} \in \mathbb{R}^{n_o \times n_p n_o}} \underbrace{\prod_{\tau=0}^{t} \underbrace{\boldsymbol{W}^*_{\boldsymbol{v}_\tau}}_{\text{Recurrent matrix} \in \mathbb{R}^{n_p n_o \times n_p n_o}} \underbrace{\begin{bmatrix}\boldsymbol{o}_{z_0} \\ \vdots \\ \boldsymbol{o}_{z_{n_p}}\end{bmatrix}}_{\text{Initial WM slots } \boldsymbol{h}_0 \in \mathbb{R}^{n_p n_o}}}_{\text{RNN, } \boldsymbol{h}_{t+1} \in \mathbb{R}^{n_p n_o}} \tag{2}$$

This equation performs the exact same computation as the previous equation, though its terms, while similar, have important differences (see Table 1 for relationships between terms). In particular, what was a matrix of memories stored in synaptic weights, $\begin{bmatrix}\boldsymbol{o}_{z_0} & \cdots & \boldsymbol{o}_{z_{n_p}}\end{bmatrix}$, is now a vector of memories stored in RNN activity, $\begin{bmatrix}\boldsymbol{o}_{z_0} \\ \vdots \\ \boldsymbol{o}_{z_{n_p}}\end{bmatrix}$. Thus the total neural activity pattern can be decomposed into a collection of non-overlapping subspaces which we term **activity slots**, and each such slot can store a memory of an arbitrary observation (Figure 1A right). Readout weights are now fixed and attend to a single slot (Figure 1D). Thus, to readout a memory, the contents of each slot must be copied and shifted to other slots via recurrent weights $\boldsymbol{W}^*_{\boldsymbol{v}}$, so that the correct observation can be readout (in the above equation the first slot is the readout slot, but in general it could be any slot - this is set by the structure of the task, e.g., Figure 8). To copy and shift the contents of activity slots, the WM velocity dependent matrices, $\boldsymbol{W}^*_{\boldsymbol{v}} \in \mathbb{R}^{n_p n_o \times n_p n_o}$, are expanded (and transposed) versions of the EM velocity-dependent matrices, $\boldsymbol{W}_{\boldsymbol{v}}$: 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 ($\boldsymbol{0} \in \mathbb{R}^{d_o \times d_o}$) (See Figure 1E for a schematic of the structure of $\boldsymbol{W}_{\boldsymbol{v}}$). Intuitively, the matrix is bigger (by a factor of n_o in both dimensions) as we are now simultaneously tracking relative position to each observation, rather than abstract task position. Adding a memory to a slot is simple. Feed-forward weights can direct observations directly to the input slot (e.g., Figure 1A right). The memory is then passed from slot to slot, controlled by velocity. When memories reach the output slot, it can influence immediate behaviour. Thus, once learned, this solution requires no synaptic plasticity. Lastly, similar to the EM solution, the WM solution works in any basis (not just the one presented above).

Memory slots unify episodic and working memory and yield a rich neural basis for WM cognitive maps. While both solutions use memory slots, they do so in conceptually and functionally different ways. In the WM solution, memory slots are structured neural subspaces, with memories for different observations stored in each neural subspace, and the contents of each subspace can be copied and shifted to other subspaces. In the EM solution, memory slots are in synaptic connections (each memory is stored in a subspace of the weights), with each memory statically remaining in its slot (subspace). The memory slots are also accessed differently. The EM solution uses flexible attention (via an RNN) to index 'fixed' memory slots (the weights don't move), while the WM solution uses fixed attention (the readout weights) but has flexible memory slots (the contents can be copied and shifted by the RNN)[i].

Interestingly, there is a striking representational difference between the WM and EM solutions. This difference arises because the RNN in the EM solution represents latent position alone (relative to some initial position), independent of sensory observations attached to any position in any problem or task. However, RNN neurons in the WM solution represent relative position *from* individual observations - a type of conjunctive representation of relative position *and* observations for every problem. In particular, the slot identity corresponds to a particular relative position from the agent's current position (or more generally a particular sequence of action/velocities one could take in a structured environment), and the contents of that slot is the observation at that relative position. In other words, the contents of each slot indicates the correct counterfactual answer to the task *if* the slot's corresponding action *were* taken. For example, in the simple case of Figure 1A, third column, Slot 1, aligned with the input/readout weights, contains the correct answer to the prediction task if no action were taken. And Slot 2 and 3 contain the correct counterfactual predictions if the agent *were* to go back one or two positions respectively. Every time the agent takes an actual action, the correct counterfactual prediction associated with each slot necessarily changes, and so the WM RNN must update the contents of very slot by appropriately copying and shifting their contents through the recurrent weights. Because of this dynamic update, at any given time, the entire WM representation acts like a cognitive map that can *simultaneously* answer 3 counterfactual questions (what observation would I predict if went back 0, 1, or 2 positions) using decoders that attend to slots 1, 2, and 3, respectively. As the agent moves forward in position, the slot activity is shifted backwards correspondingly so that *each* slot always contains the correct answer its *own* action specific counterfactual question.

Moreover, in the simple case of Figure 1A, third column, since each neuron in each slot is selective for a single observation, from a single neuron perspective, the neural code will consist of a collection of neurons that fire selectively for particular

[i]We note that our proposed WM activity slot solution may not the universal solution to all WM tasks. However we contend that for tasks involving non-trivial velocity signals, it is the solution neural networks learn, as evidenced by our subsequent simulations. See Discussion for further consideration.

observations, at particular times since the observations were most recently encountered (or more generally for particular observations and particular relative positions / action sequences taken since the observation). Any single neuron's relative position / action selectivity elegantly derives from the *identity* of the slot it is a part of, while that same neuron's object selectivity derives from its functional role *within* its slot.

This fundamental difference between EM and WM solutions implies that single neurons behave very differently across tasks. In particular, any two EM position neurons that fire next to each other in one task, will fire next to each other in another task: they maintain their phase relationship (Figure 1F; just like grid cells do). On the other hand, two activity slot neurons with a particular phase relationship in one task (Figure 1G, top), may not have the same phase relationship in another task (Figure 1G, bottom). This is because WM cells code for relative position (as defined by the slot's intrinsic relative position to the input/readout slot) to a particular observation (as defined by the identity of the neuron within a slot). Since the pairing between positions and observations are shuffled between different problem realisations of a given common task, the relative phase between WM cells will also appear shuffled. Interestingly, as we will see below, this slot derived conjunctive coding of observations and actions, reflecting a cognitive map that can simultaneously answer many counterfactual questions contingent on different actions, can elegantly and succinctly account for a wide range of otherwise perplexing properties of rich prefrontal cortical representational dynamics, both *in silico* and *in vivo*.

Model architectures and task specifics

To experimentally test the predictions of the theory, namely that the EM system uses position representations whereas WM systems use activity slots, we use a variety of tasks and model architectures (full details in Appendix A.2).

Model architectures. We build both EM and WM neural network models (Figure 2A,B). The key difference between the two models is that the EM model makes predictions by retrieving memories from an external memory system, whereas the WM model makes predictions with a learned readout matrix. For the EM external memory, we use a modern Hopfield network[35], which our above theoretical results are derived from (Appendix A.1). Both models use an RNN to control memories. To be general, we use two RNN variants (Figure 2C,D); one directly inspired by our theory that uses velocity dependent transition matrices (GroupRNN; related to selective state-space models[42]), and the other is a conventional RNN which receives velocity signals as input (RegularRNN). All model parameters are initialised randomly, and trained using backpropagation.

Tasks. We consider four main tasks (Figure 8A) that are all related to the neuroscience and cognitive science literature (further neuroscience tasks are introduced later): Immediate Serial Recall (ISR), N-Back, 1D Navigation, and 2D Navigation. These tasks range in complexity from simple sequence repetition that requires no velocity integration to tasks where correct recall relies upon integration of progressively more complex velocity signals. In all tasks, 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 each timestep. After observing the observation and velocity at timestep t, the model is asked to predict the target 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: $\boldsymbol{o} = \{1,3,2,2,3,1,2,\dots\}$, $\boldsymbol{v} = \{+1,+1,0,-1,-1,-1,\dots\}$ and $\boldsymbol{t} = \{-,-,-,2,3,1,2,\dots\}$. Here, '$-$' means a target that we do not train on since it has not been observed before and therefore is not possible to predict. The ISR task is like 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 constant velocity signal) and the targets are observations N timesteps ago. For each task we train on multiple problem realisations corresponding to multiple sequences where position-observation pairings are randomised, but the underlying task structure of how actions affect positions (e.g., 1D Navigation) is fixed. See Appendix A.3 for details on all tasks. It is important to note that the 1D and 2D navigation tasks (as well as some tasks later in this paper) involve more than just repeating the sequence exactly as it was presented and so cannot be solved without learning a cognitive map and correctly integrating velocity signals.

Results

Differences in EM and WM algorithm performance and representation

EM scales more favourably than WM. Our theory demonstrates that EM and WM systems implement the same computation (solve the same task) but use different underlying algorithms (their neural mechanism is different). These algorithms have different trade-offs. Most notably, the WM solution requires many more neurons than the EM solution (a factor of n_o more since each observation must be stored in neural activity). Thus we predict that, for a fixed neuron budget, the WM model performance will degrade when the number of memories needed to be stored, which we refer to as the task size, increases, while the EM model will continue to perform well. We demonstrate this phenomenon for the 1D navigation task (Figure 2E) as well as the other tasks (GroupRNN: Figure 11A, RegularRNN: Figure 12A). Additionally, we observe performance improves when the WM RNN has more neurons, as predicted. Also, in both EM and WM models, we observe reduced performance in

tasks with more action dimensions (e.g. 2D navigation versus ISR; Figures 11A, 12A), likely because more weights need to be precisely tuned with more action dimensions.

EM uses position representations while WM uses activity slots. Our simple theory above predicts that even more potentially complex trained EM models should use position-like representations to index memories stored in synaptic weight slots (equation 1), whereas trained WM models should 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 over-fitting by training each decoder on many example tasks, rather than just one, demonstrating that the activity slots, or position representations, are generalisable across tasks.

First, our theory says abstract position (independent of object identity) should be decodeable from EM models, but not WM models, since EM models use position to index memories (Figure 2F). We confirm this prediction in all tasks and for all RNN types (Figures 2G, 11B, 12B). This is consistent with existing EM models[4,14] that learn position codes to index hippocampal memories, as well as entorhinal data such as grid cells that code for position. We note, however, that position is often not decodable for large or small task sizes. For large sizes, it is easy to understand: the models were unable to learn the task and thus learned no sensible representation (Figure 2E, 11A, 12A). For small task sizes, the RNN in the EM model has learned to use another memory indexing representation, namely the WM solution. We discuss this case further in a later paragraph.

Second, our theory predicts that previous observations should be decodeable from the RNN in WM models, but not the EM models, since the RNN in WM models store memories of previous observations in neural activity (Figure 2H) while RNN activity in EM models only tracks position, with memories of previous observations stored in synaptic connections. More precisely, our theory prescribes exactly which past observation is stored in which neural activity slot at any given time since it details how slot contents are copied and shifted (or more generally rotated) by the trained RNN connectivity in a manner that reflects the underlying structure of the task (i.e. how velocity signals change position) (Figure 8B,C). Thus, over time, the neural activity within each activity slot will encode a particular sequence of observations. Moreover, for tasks with varying velocity signals, this sequence will not be a simple temporally shifted copy of the input observations (Figure 2H). We refer to this sequence, defined purely by the task structure, velocity signals and observations, as the slot-sequence for each predicted activity slot. We find that these specific slot-sequences can indeed be successfully decoded from RNN activity in all tasks for WM models (Figures 2I left, 11C, 12C), whereas other highly natural past-sequences (e.g., the observation N steps ago) cannot be decoded (apart from the ISR and N-Back task where past-sequences and slot-sequences are identical; Figures 11C, 12C). Slot-sequence decoding performance degrades for larger task sizes, as the RNN fails to learn the task. Furthermore, for an example task size on each of our four tasks, we show high individual slot-sequence decoding for WM models (Figures 2I right, 11D, 12D). These results demonstrate that WM models learn activity slots, and they use velocity signals to copy and shift previous observations between slots: velocity signals control activity slots.

Interestingly, while the EM models have, as predicted, poor slot-sequence decoding and high abstract position decoding in most situations, for small task sizes the converse can be true (Figure 11G,I). This is because an activity slot representation can also be effective at indexing memories; like a position representation, activity slots uniquely code for each position in each task (however it does not code position the same way across tasks, hence we cannot decode *abstract* position from activity slots). We posit the WM solution is easier to learn and hence preferred when possible (i.e., when not limited by number of neurons), which intuitively makes sense as slot representations are a direct function of input data, while position representations are abstract.

Visualising slots & slot algebra

Visualising slots without decoding. We have so far provided evidence, through decoding analyses, 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 different neural activity subspaces, that are not necessarily aligned to single neuron axes; i.e. single neurons could exhibit mixed selectivity for multiple slots. To more directly visualise learned activity slots in an even simpler setting, we leverage a recent regularization technique[43] that encourages demixed representations in which single neurons code for single independent factors (e.g., in our case slots). In particular, we train models while enforcing RNN activity to be nonnegative (i.e., a ReLU activation function rather than linear or tanh) and energy efficient (i.e., minimum norm weights and activities).

With these constraints, single neurons do indeed participate only in a single slot (Figure 3A). We quantify the degree to which single neurons code for individual slots, using the metric 'mutual information ratio' (MIR), a well grounded metric[43,44] where high values indicate demixed single neuron slot coding. We demonstrate MIR improves with the additional constraints across all four tasks and for both RNNs types (Figure 13; GroupRNN and RegularRNN; blue and orange colours).

Slot algebra predicts relations between neural representations of different problems from a common task. To further demonstrate that RNNs learn activity slots, as well as show that activity slots permit compositional computations, we test another prediction of our theory: activity slot representations should obey a slot algebra, which predicts remarkable and simple linear relationships between WM representations corresponding to *different* problem realisations of the same underlying task. As an example, consider a task with 3 latent positions and 4 possible observations (Figure 3B). Let $Rep_i([a,b,c])$

denote the entire WM RNN representation while the agent is at position i in a problem realisation with observations a,b,c paired at each of the 3 positions respectively[ii]. Now consider the neural representations for two problems, $Rep_i([1,4,2])$ and $Rep_i([1,2,2])$. These two problems differ only in that the observation 4 at position 2 is replaced with the observation 2. The replacement of one observation with another induces an action on neural representations yielding the slot algebra relation $Rep_i([1,2,2]) = Rep_i([1,4,2]) - Rep_i([a,4,c]) + Rep_i([a,2,c])$ (Figure 3B illustrates this relation when the agent is at position $i = 0$). Here a and c in the middle two terms can be any observations; the key is that the algebraic relation corresponds to taking away observation 4 at position 2 via subtraction of neural representation $Rep_i([a,4,c]$, for any a and c, and replacing it with observation 2 via the addition of another representation $Rep_i([a,2,c])$ with the *same* a and c. Critically, this equality should be true independent of the particular sequence/trajectory taken to position i.

To test whether our model representations obey slot algebra, we examine the WM RNN in situations that satisfy the equality relation above. Each relation involves neural representations from 4 separate runs of the RNN in 4 *different* problems. We sum the 3 representations according to the right hand side of the relation, and compare the sum to the measured representation on the left hand side using a squared difference measure ($d_{1-2+3-4}$). We then compare this measure to 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 the representation obeys slot algebra.

The correctness of slot-algebra requires trajectory independent representations, that depend only on the current agent position independent of how the agent arrived at the position. However, not all methods of integrating velocity in neural networks are trajectory independent, with some attractor networks, both in biology[45] and models[46], having velocity modulated neurons. Indeed it is possible to have a slot-based representation where slots are conjunctive with velocity. Thus, understanding whether neural representations obey slot-algebra, not only assesses compositionality, but is also informative of how the RNN integrates velocity. Here we test which RNNs obey slot-algebra[iii]. We test the GroupRNN which modulates synapses with velocity signals (and is the model analogue of our theory), the RegularRNN which adds velocity signals to its RNN activities, and, additionally, the BioRNN which integrates velocity signals in a separate neural population, but not the RNN activities themselves. As predicted, the GroupRNN has low SA for all datasets (Figure 3C) since velocity does not directly modulate RNN activity. SA further improves with additional regularisation (KL) that further encourages representations to be consistent across timesteps, i.e., trajectory independent (see Appendix for details). Conversely, as predicted, the RegularRNN only achieves low SA on the ISR dataset which requires no velocity signal (see Figure 10 for another RegularRNN variant which exhibits the same phenomena). Lastly, the BioRNN achieves low SA for all datasets. Intriguingly the architecture of the BioRNN is closely related to the neural circuit discovered in the fly head direction system[47–49] (Figure 7E,F; details in Appendix A.2).

Activity slots controlled by velocity signals provide a unifying framework for diverse PFC representational dynamics

We have shown that our theory of controllable WM activity slots explains representations that emerge in artificial RNNs trained on structured sequence memory and prediction tasks. The implication of this is that any task where latent positions are linked through action induced transitions can be optimally navigated using WM controllable activity slots. This, however, relies on the appropriate integration of velocity signals. A key question that remains is what are these velocity signals, and how are they represented in neurons? In this section, we demonstrate that PFC-dependent sequential memory tasks and cued memory tasks can be unified through activity slots, but where the distinguishing feature is the velocity signal that controls activity slots.

PFC and RNNs compute and represent velocity signals. In previous tasks, we directly provided velocity signals to the model (thus specifying its representation). However, actual brains do not have this luxury; they must compute velocity signals from lower level sensory inputs and choose how to represent them. In spatial tasks, self-movement vectors are thought to be computed from vestibular or other sensory inputs and represented using head direction cells[50] and speed cells[45]. For complex PFC tasks, where behaviour may be non-spatial, it is unknown what the velocity signals are, or how they are represented.

Recent neural data however, while not demonstrating velocity signals, have found neurons that track progress to goals, invariant to the actual distance to the goal[30,51]. For example a 50% 'progress cell' fires at the midpoint between two goals whether they are 3 steps apart or 20. In order to update progress, we contend 'progress-velocity' signals must be computed and represented. To test for this, we train WM models on analogous tasks, but to be general and illustrate the principles of working memory, we remove the spatial component to these tasks.

In particular, Basu et al.[51] trained rodents to run back and forth between two (of 9) reward ports, and El-Gaby et al.[30] trained rodents to repeatedly visit 4 (of 9) spatially located reward ports in sequence. In both studies, the animals are trained on many problem realisations with random goal positions. We model these tasks (modulo the spatial component), as a 2-ISR and a 4-ISR task, but with delays between each observation (Figure 4A,E left; just like the 'delay' of travelling between two spatial goals). Critically, while the delays are random across different problem realisations of each task, the delays are the *same* on each loop on every problem. In a spatial context, this is analogous to how distances between goals are the same on a particular

[ii] It is important to consider each agent position i individually, since activity slot representations change from agent position to position.

[iii] We do not use the n-back dataset as there is no repetition of veridical position.

task, but change for different goal positions on new problem realisations of the task. For example, possible 2-ISR delay sequences for different problem realisations are $[4,*,*,2,*,*,4,*,*,2,\dots]$ or $[1,*,3,*,1,*,3,\dots]$ where $*$ is the common delay observation (that also must be predicted), and possible 4-ISR delay sequences are $[4,*,2,*,*,2,*,6,4,*,2,*,*,2,*,6,4,\dots]$ or $[1,3,*,8,*,2,1,3,*,8,*,2,1,\dots]$. We provide no explicit velocity signal, and so the model will have to utilise the sparse inputs to compute an appropriate progress-velocity for each delay period, and remember it on returning to that same delay (i.e., after one full loop of experience).

Indeed, after training WM models on these tasks, we are able to decode progress, as well as progress-velocity (Figures 4A,E right). Furthermore, we observe progress tuning at the single neuron level (Figure 4B,F), as well as progress-velocity tuned neurons (Figures 15A, 16A). This means the network internally computed progress-velocity, on the basis of sparse inputs, 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 get structured by both task and progress (examples for the 2-ISR delay task shown in Figures 4C, 9), in that the slots are organised by task-progress on a loop. This means a slot will now code for 25% since observation, or 150% since observation (up to 200/400% since for the 2/4-ISR delay task). We refer to this as 'task-progress' structured (Figure 4C). The contents of these slots are copied and shifted to subsequent slots by the progress-velocity signal[iv]. Indeed, in trained WM models, we are able to decode task-progress activity slots (Figure 4A,E right), and we observe 'task-progress slot' neurons in Figure 4D,G; these neurons always fire at the same task progress after a preferred observation (e.g., 375% after observation 5; Figure 4G bottom). Further cellular examples are shown in Figures 15, 16. It is of particular note that both 'progress' and 'task-progress slot' neurons were recently recorded in rodent mPFC[30], exactly as our model predicts.

While representing progress and progress-velocity seems an esoteric choice of representation, it is not clear how activity slot representations could solve this task without representing progress-velocity. For example, it is not possible to have an activity slot for each step of the delay period as this cannot be placed in a consistent loop structure different problem realisations of a task. In general, when behaviour (e.g., navigating to goals or waiting for delays) is structured in different coordinates to the underlying space where actions are taken (e.g., physical space or a loop), a coordinate transformation is required to match the two spaces. This is exactly what the progress, and progress-velocity, neurons are doing: transforming from spatial coordinates to task coordinates.

PFC activity slots in a task with constant velocity. Our activity slot theory also accounts for PFC representations in standard sequence memory tasks where only a constant velocity is required. For example, Xie et al.[29] recorded PFC neurons in an ISR task. Here, monkeys were shown a sequence of three spatial positions on a screen (out of 6 positions), and, were trained to saccade to the same objects in order (Figure 5A). Importantly, the monkeys were trained on many such sequences (i.e., just like our ISR task). Neural activity in the delay period decomposed into three successive orthogonal subspaces. Each of the 6 possible spatial saccade locations could be represented in each of the 3 subspaces, with the first/second/third saccade location encoded in the first/second/third subspace (Figure 5B-C). These 3 sequential subspaces therefore directly correspond to our activity slots, with a different slot for the first, second and third ordinal points in the sequence (Figure 5D bottom is after delay period). Indeed, a WM RegularRNN on trained the task, exhibits exactly the same subspace decomposition (Figure 5E-F).

Controlling PFC activity slots in a cue-dependent memory retrieval task. So far we have considered either external or internally generated velocity signals. But sensory observations themselves can also serve as velocity signals. Here we show that sensory cues can be understood as velocity signals controlling PFC activity slots, in a particular study of cue-dependent memory retrieval[33]. In this study, Panichello & Buschman[33] recorded PFC neurons in monkeys performing a working memory task where two colours are presented on a screen, one at the top and one at the bottom. Then, after a delay with a blank screen, a cue informs the monkey whether to report the colour of the top or bottom colour (Figure 6A). Importantly, each problem realisation is a random combination of two colours and cue. Neural activity in the delay period decomposed into two orthogonal subspaces, one for the top colour and one for the bottom (Figure 6B, left). Thus, each colour can be represented in one of two subspaces, depending on whether it was presented at the top or bottom. After the cue, the subspaces rotate so that top (when correct) and bottom (when correct) subspaces are parallel to one another (Figure 6B, right; Figure 6C). This is easily understood with our theory (Figure 6D); the initial two subspaces are two different activity slots, one for the top colour and one for the bottom colour. Since only one slot can be readout from, the cue can be understood as a velocity signal that controls how slot contents are shifted to the readout slot. Because either the top or bottom colour is moved (controlled by the cue (e.g. velocity) signal) to the same readout slot on correct trials, the subspaces become parallel to one another after the cue on correct trials (Figure 6D, bottom). Indeed, a WM RegularRNN trained on the task, exhibits exactly the same subspace decomposition and dynamics (Figure 6E-G).

[iv] While these task-progress slots will now be continuous rather than discrete, with bumps of activity on continuous attractor manifolds now defining an observation within a 'slot', for consistency we still speak about them as if they were discrete.

Discussion

In this work, we derived a formal relationship between the algorithms and representations of episodic and working sequence memory, and empirically validated theoretical predictions that EM utilises more abstract position-like representations while WM uses activity slot representations with a particular and systematic conjunctive coding of observations and (counterfactual) actions. Furthermore, we show that, like the HPC EM system, the PFC WM system can be updated and controlled by velocity signals, which can be flexibly computed from sparse input data. We used our theory of controllable activity slots to show that recordings from several different PFC studies - from serial recall to cue dependent recall and spatial working memory - are all explained in a unified framework of activity slots controlled by velocity signals.

Intriguingly, our activity slot theory may also explain PFC data from non-explicit working memory tasks. For example, human mPFC (fMRI) activity obeys rules of 'scene algebra' when humans perform scene construction tasks involving one object above another[52], i.e., $Rep(\frac{o_a}{o_b}) - Rep(\frac{o_a}{o_x}) + Rep(\frac{o_c}{o_x}) = Rep(\frac{o_c}{o_b})$, where $Rep(\frac{o_a}{o_b})$ denotes the representation of object a above object b. This suggests that the objects are stored in different neural subspaces depending on their spatial location, analogous to our theory of activity slots which organise objects depending on their (e.g. spatial) relationships. This is perhaps no surprise since both our and their task require a compositional understanding (objects can be in any sequence or spatial configuration respectively). We anticipate a slot based understanding will offer further insights into representations in frontal cortex, beyond just working memory. For example, activity slots could represent sub-goals for sequential goal directed planning[30]. Each slot would hold a sub-goal, with the sub-goals cycled forward when the previous sub-goal is achieved. This could offer an explanation as to why the same brain region involved in working memory, is also implicated in value based decision making and goal directed planning[31,53], and why there are goal predictive representations in PFC (recall neurons in activity slots conjunctively encode different degrees of distance/progress to various observations/goals). In this vein, it is of note that slot based models are used in machine learning in diverse settings from natural language[6], to planning[54] and scene construction[55,56].

We related the two algorithms to HPC and PFC, however the key distinction between the algorithms is whether memories are stored in synaptic strengths or neural activity patterns. It is plausible that PFC could utilise short term plasticity to store memories in synapses as well as neurons. Indeed, recent modelling work suggests short term plasticity may better explain PFC activity as compared to recurrent dynamics[57]. Conversely, because of the intimate connectivity between entorhinal cortex and hippocampus (compared to the more distant connectivity between PFC and hippocampus), it makes sense for entorhinal cortex to control hippocampal synaptic memories using position-like representations as opposed to activity slots.

We do not claim to have solved the representations of RNNs on all working memory tasks. Indeed there are WM tasks in which activity slots are not learned (e.g., Figure 14A-B). However, we suggest that activity slots are a general solution to sequence working memory tasks, but in some circumstances there may be other bespoke RNN solutions that get learned instead. Lastly, we note that our formalism, as well as the GroupRNN, is a state-space model (these models have gained considerable attention recently[42]), thus, on the machine learning side, our work offers a relationship between the learned representations of state-space models, RNNs, and transformer neural networks.

The hippocampal and frontal systems are thought to be essential in mediating higher order cognition, with patients exhibiting profound deficits in behavioural flexibility if these structures are damaged[10,11]. While these systems have been the subject of intense study over the last several decades - both experimentally and theoretically - the systems have been largely treated independently and their respective cellular representations are hard to reconcile. Our work brings together these systems by showing that the algorithms and representations of frontal working memory and temporal episodic memory are two sides of the same coin.

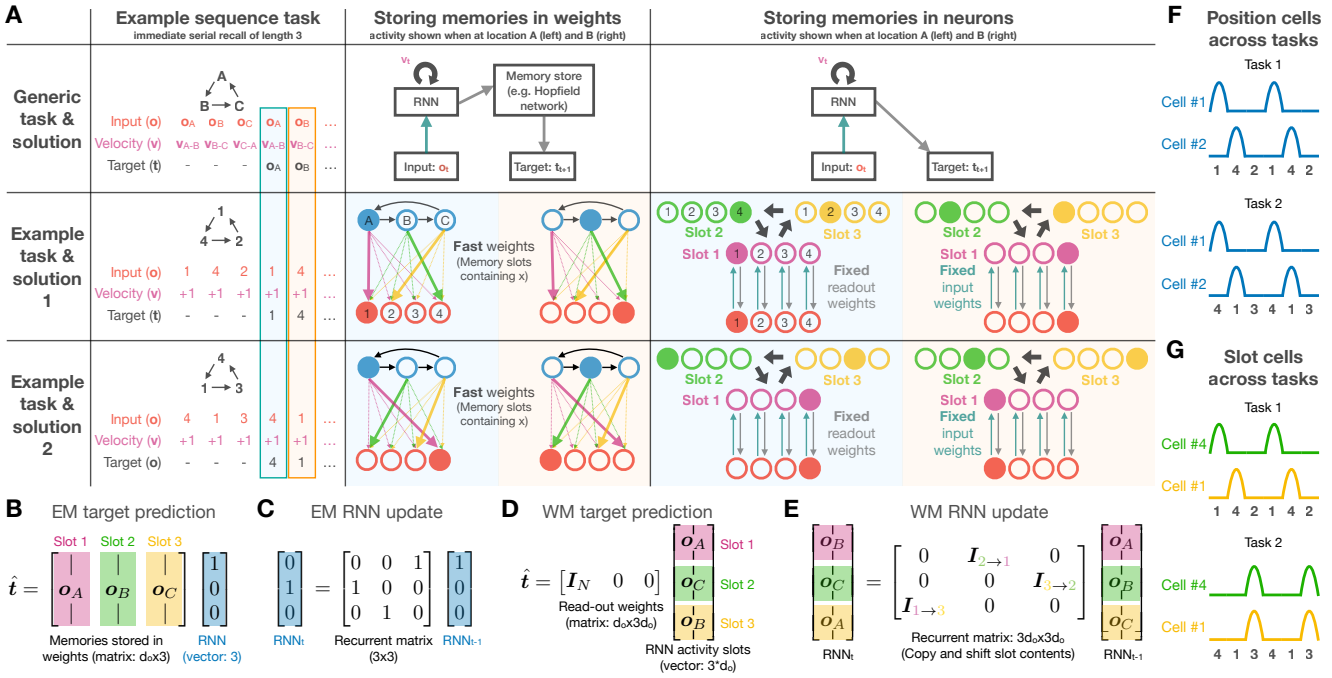

Figure 1. Relationship between episodic memory and working memory models. A) Schematic of how a task can be solved by episodic and working memory solutions. Solid/empty circles are active/inactive neurons. Black/teal/grey arrows are recurrent/input/output connections. Left/Center/Right: Task/EM solution/ WM solution. Top/Middle/Bottom: General example/ Specific example 1/ Specific example 2. **Left:** Example task of Immediate Serial Recall (ISR) of length 3, with specific instances below. The task input is a sequence of 3 random observations, $\boldsymbol{o}$, that repeat, and a constant 'forward' velocity, $\boldsymbol{v}$. The task is to predict the upcoming target, $\boldsymbol{t}$. Blue/orange segments highlight the timesteps described in EM/WM solution schematic (center/right). For the purposes of the schematic, we assume each observation has a one-hot encoding. **Center:** Top: EM models involve an RNN that indexes memories stored in synaptic weights. Middle/Bottom: Schematic of EM solution. This is a simplified version of the top figure (e.g., the RNN is one-hot, and memories are stored in synapses between the RNN and observation neurons; full model and derivation of simplification in Appendix A.1), but it maintains all the key elements. Past observations are stored in synaptic memory slots (here we show synaptic memory slots as non-overlapping sets of neural weights), and recurrent position representations index memory slots to retrieve a memory for predicting the upcoming target. 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). Blue/orange segment highlights present the network at two different timesteps of the task. **Right:** Top: WM models involve just an RNN. Middle/Bottom: Schematic of WM solution. Past observations are stored in activity slots within the RNN (we show activity slots as non-overlapping sets of neurons, but they could be non-overlapping neural subspaces more generally), with the content of each slot copied and shifted by the RNN (at every timestep) to successfully predict targets using fixed readout weights. This example RNN has three activity slots (Green/yellow/purple) each consisting of 4 neurons that can one-hot encode 4 possible observations (1,2,3,4). Blue/orange segment highlights present the network at two different timesteps of the task. **B-E)** Mathematics of the EM and WM solutions. **B)** The EM solution makes predictions via an RNN position code (blue vector) that indexes memories stored in synaptic weight slots (matrix columns). **C)** The RNN position code is updated by recurrent connections. **D)** The WM solution makes predictions using fixed readout weights that attend to a single RNN activity slot (slot 1). **E)** Contents of RNN activity slots are copied and shifted by a recurrent matrix. Note the recurrent matrices will be different depending on the task structure. **F)** The activity of two example position neurons (cells #1 and #2 are the left/middle blue neurons in panel A middle). Cell #1 / #2 is active at position A/B. Thus the two neurons have similar activity patterns in both tasks (top/bottom), up to a possible global shift in their firing fields across tasks, as absolute position could be unknown. However, the neurons preserve their *relative* phase relationship across tasks; i.e. the phase lag between the activity peaks of cell #1 and #2 is preserved across all tasks. **G)** The activity of two example activity slot neurons (cell #4 is rightmost green neuron, and cell #1 is leftmost yellow neuron in panel A right). Cell #4 always fires 2 steps after observation 4. Cell #1 always fires one step after observation 1. In task 1, these neurons have the same activity as the above position neurons, but do not in task 2: they do not keep their phase relationships between tasks as they each code for specific distance/lag from specific observations.

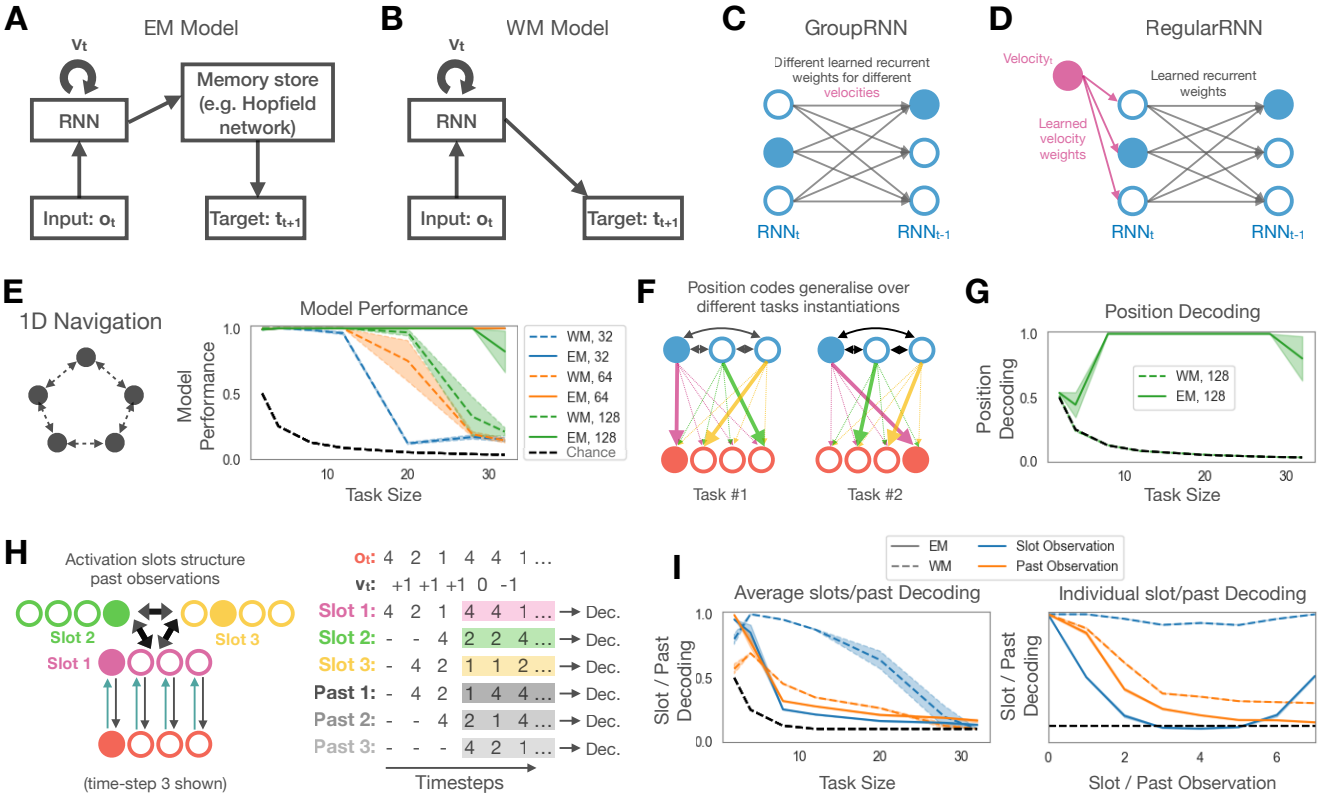

Figure 2. EM models scale better and learn position representations, while WM models learn activity slots. A) We train EM models consisting of an RNN that can store and retrieve memories. At each timestep, the RNN receives an observation, $\boldsymbol{o}_t$, and velocity signal $\boldsymbol{v}_t$, and must predict target $\boldsymbol{t}_{t+1}$. **B)** We train WM models that are just an RNN. We use two types of RNN (for both EM and WM models): **C)** Like our theory we use an RNN with learnable velocity dependent matrices (GroupRNN); **D)** We also use a conventional RNN which receives velocity as input (RegularRNN). **E)** Performance of the EM (solid) and WM (dashed) models on 1D navigation tasks (using GroupRNN). Black dashed line denote chance performance. EM models perform better that WM as task size increases. Larger sized RNN perform better for both EM and WM models. For each combination of task, task size, model type, and RNN size (32, 64, 128), we train five randomly initialised models. Results are mean $\pm$ standard error. All subsequent panels are for RNNs of size 128. **F)** Our theory says EM models should learn position representations that generalise over task instantiations (i.e., in sequences with different sensory observations but same underlying structure). **G)** EM models, but not WM models, learn position like representations. **H)** Left: The WM activity slot theory says the RNN organises past observations into slots (structured according to underlying task structure). Right: Thus a particular sequence of observations should be decodeable for each slot (we call this the 'slot-sequence' for each slot). This sequence is different to the sequence of time-lagged past observations (we call this the 'past-sequence' for each time lag). **I)** Left: Average decoding of 'slot-sequences' (what observation should be in each slot according to our theory) and 'past-sequences' (what observation occurred i timesteps ago, up to the total number of slot). WM models have high slot-sequence decoding, but not past-sequence decoding, indicating they have learned activity slots. The EM models have not learned 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 consistently 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 in Figure 11, and for all tasks with RegularRNN in Figure 12.

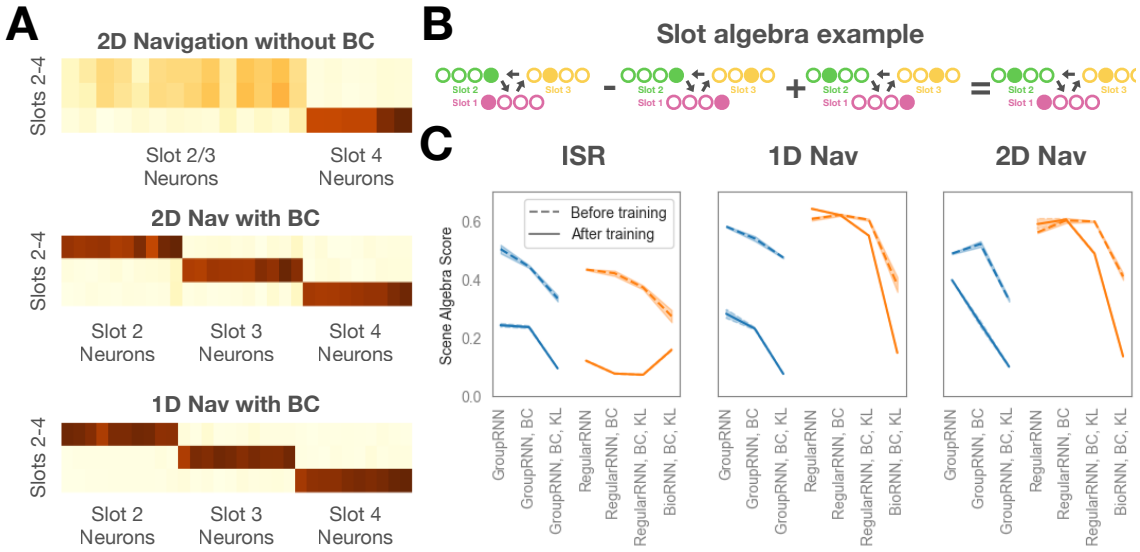

Figure 3. Visualising WM slots without decoding, WM models afford slot algebra. A) To visualise WM activity slots we train models while enforcing non-negative RNN activity (using a ReLU) as well as regularising RNN activity (BC: a technique shown to encourage single neurons to code for single factors of variation[43]). We show the mutual information matrices between neurons (horizontal) and slots (vertical). We only show neurons with non-negligible activity, and only present slots 2-4 for clarity since slot 1 tends to have many more neurons than slots 2-4. Top: Without the technique, single neurons code for multiple slots. Middle/Bottom: With the technique single neurons code for single slots. **B)** Slot algebra means representations from three different tasks can be combined to give the representation of a fourth task, e.g., $Rep_i([1,4,2]) - Rep_i([4,4,3]) + Rep_i([4,2,3]) = Rep_i([1,2,2])$, where $Rep_i([a,b,c])$ denotes the RNN representation at position i in a task with observations a,b,c at each position. The slot algebra score is a measure of this property, with lower values indicating more accurate implementations of this property. **C)** WM models display 'slot algebra' with slot algebra scores low for all GroupRNN models (blue) after training, and are low for RegularRNN models when using the BioRNN variant. For each task and model type, we train three randomly initialised models. Results are mean $\pm$ standard error.

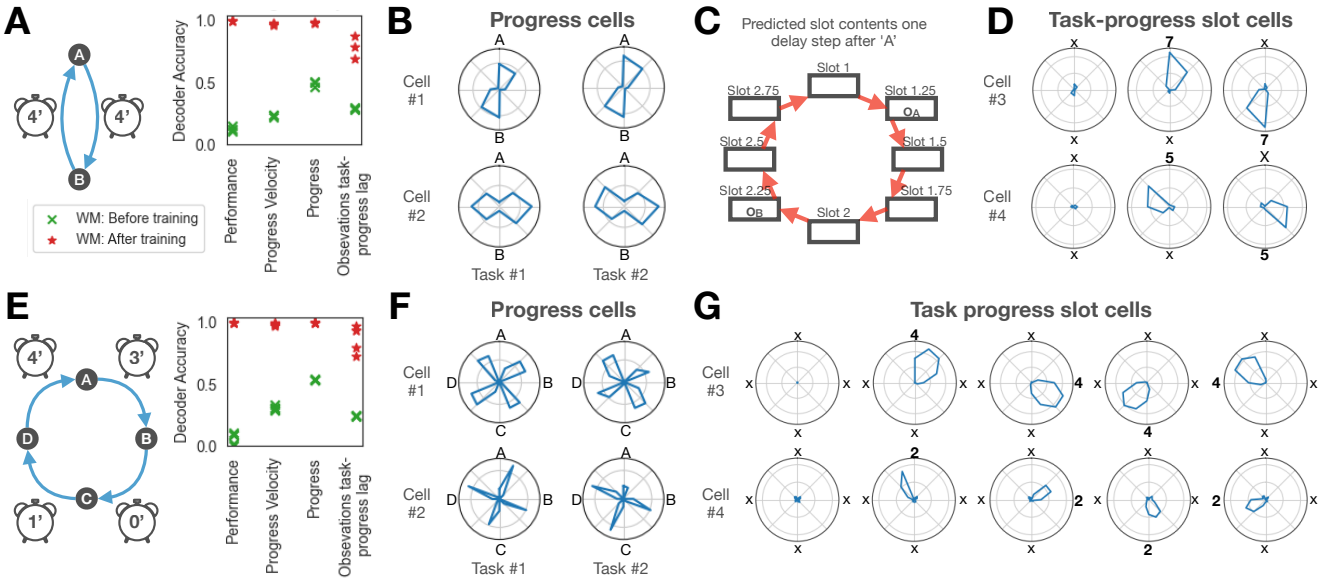

Figure 4. WM-RNNs computes velocity to control activity slots. A) We train an RNN on a 2-ISR task, but with delays between observations. Delays are sampled randomly from $\{3,4,5,6,7\}$ for each problem realisation of each task, and are fixed for the duration of the individual problem. An example delay of 5 steps shown in the schematic. An RNN is trained on multiple problems simultaneously, and so, after training, must be able to learn about and exploit the random delay in each new problem directly through experience, without any plasticity. Decoding analyses reveal that progress-velocity (reciprocal of delay duration) and progress (% of delay length completed) are represented, i.e., the models maps 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. Red/green points correspond to before/after training. **B)** The RNN learn progress cells, i.e., cells that fire at a consistent progress post observation (regardless of observation type). Angle of circle represents fraction of progress from goal A at 0 degrees top to goal B at 180 degrees bottom, and then back. Radius of blue curve indicates firing rate of cell as a function of fractional progress. The bimodal firing rate indicates the cell fires at a fixed fraction of progress to the next goal, independent of goal identity A or B. These cells transform the variable delay periods for each problem into a common representation. **C)** Schematic of task-progress activity slots. Rather than two slots for this 2-ISR task with delays, there are multiple intermediate slots corresponding to progress through the task. This allows the variable length problems to be mapped to a common slot representation. Here 3 additional slots are shown between slot 1 to 2 (as well between slot 2 to 1) corresponding to 25%, 50%, and 75% progress. This models says 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. **D)** The RNN learns task-progress slot cells, i.e., cells that fire at a consistent task-progress post an observation, as predicted by our model (panel C). Top: RNN neuron that fires at 0% after observation '7'. Left/Middle/Right: Average neural firing in all problems without a '7' observation / problems with observation '7' in 'A' position / problems with observation '7' in 'B' position. Bottom: RNN neuron that fires at 175% after observation 5. Left/Middle/Right: Average neural firing in all problems without a '5' observation / problems with observation '5' in 'A' position / problems with observation '7' in '5' position. **E-G)** Same as A,B,D but for a 4-ISR task where delays between observations can be different for different problems (but still fixed within each problem; example delays of 3, 0, 1, and 4 steps shown in schematic; delays sampled from $\{3,4,5\}$).

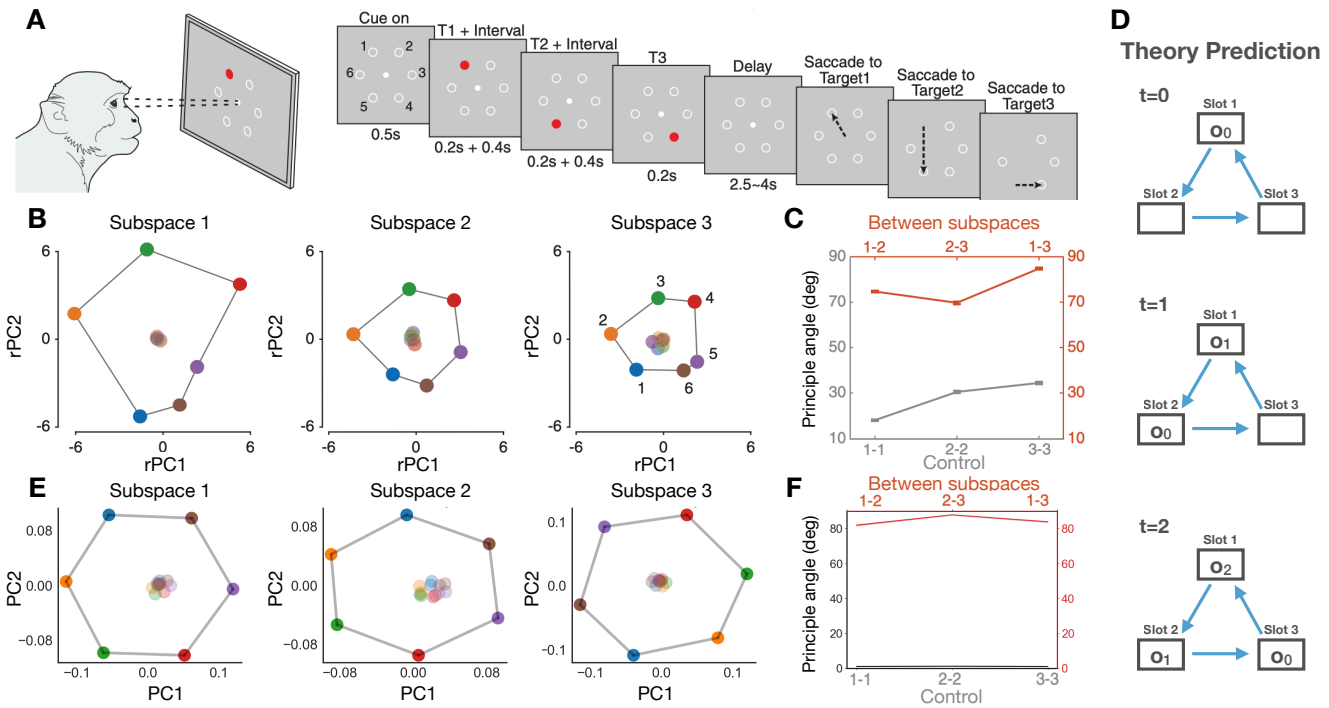

Figure 5. Explaining PFC representations in an ISR task as activity slots. A) Xie et al.[29], recorded PFC neurons while monkeys performed an ISR like task. Monkeys were presented, on a screen, three (out of six) spatial positions in sequence, then, after a delay, tasked with repeating the sequence (via saccades). **B)** 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 the subspaces are orthogonal, which **C)** is confirmed in analysis. **D)** For this ISR task, the activity slots model adds the current observation memory to slot 1, and rotates the contents of the slots unidirectionally, so that after observing all observations, slot i contains the observation $i-1$ timesteps ago. In this schematic, the readout slot is also slot 1. **E-F)** Training a RegularRNN on the same task recovered the same three orthogonal subspaces, and is consistent with an activity slot model. A-C) reproduced from Xie et al.[29].

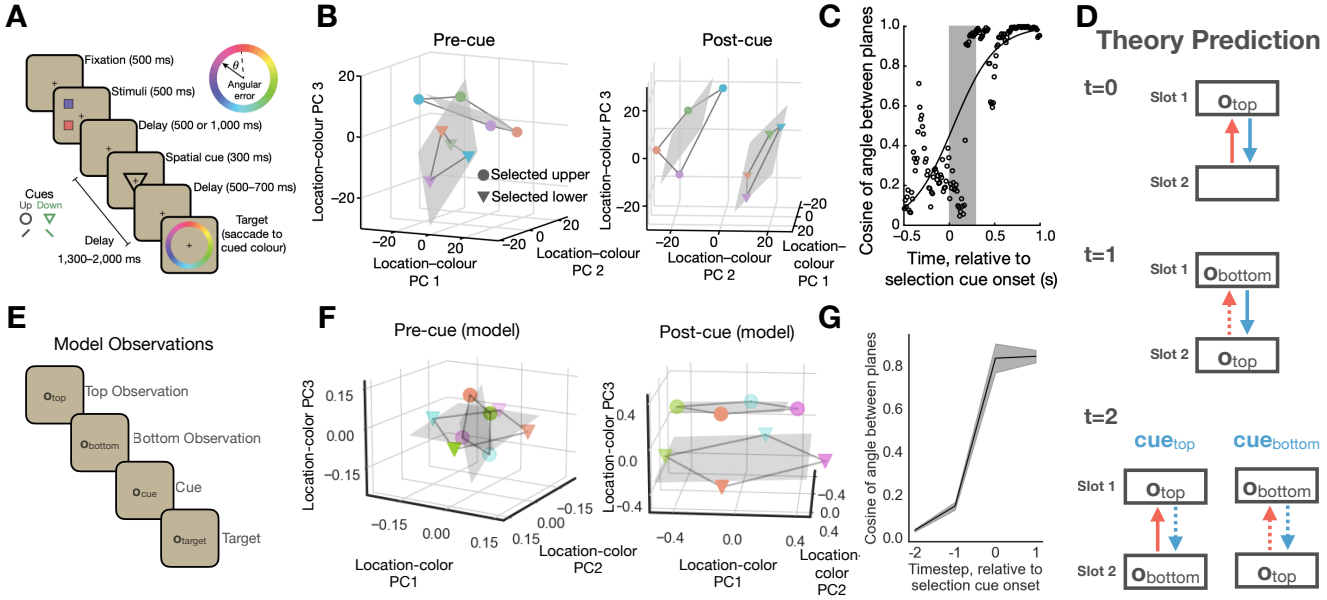

Figure 6. Explaining PFC representations in a cue-dependent memory retrieval task as velocity controlled activity slots. **A)** Panichello & Buschman[33] recorded PFC neurons while monkeys were cued to recall one of two colours previously presented at the top and bottom of a screen. **B-C)** Before the cue, PFC neurons represented two orthogonal subspaces - one for the top colour and one for the bottom colour. After the cue the subspaces became parallel. **D)** For this task, the activity slots model adds observation memories to slot 1, and copies and shifts the contents of slots according to velocity control signals (direction of shift denoted by solid as opposed to dashed arrow). In this schematic, the readout slot is also slot 1. **E)** We modelled this task by sequentially presenting the top colour, the bottom colour, then the cue, and trained the RNN to predict the target colour. **F-G)** A trained RegularRNN recovered the same orthogonal subspaces, which became parallel after the cue. This is consistent with the activity slot model. A-C) reproduced from Panichello et al.[33]

References

1. Stachenfeld, K. L., Botvinick, M. M. & Gershman, S. J. The hippocampus as a predictive map. *Nat. Neurosci.* **20**, 1643–1653, 10.1038/nn.4650 (2017). ISBN: 9788578110796 _eprint: arXiv:1011.1669v3.
2. Buzsáki, G. & Tingley, D. Space and Time: The Hippocampus as a Sequence Generator. *Trends Cogn. Sci.* **22**, 853–869, 10.1016/j.tics.2018.07.006 (2018). ISBN: 0471140864 Publisher: Elsevier Ltd.
3. Dayan, P. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Comput.* **5**, 613–624, 10.1162/neco.1993.5.4.613 (1993).
4. Whittington, J. C. R. *et al.* The Tolman-Eichenbaum Machine: Unifying Space and Relational Memory through Generalization in the Hippocampal Formation. *Cell* **183**, 1249–1263.e23, 10.1016/j.cell.2020.10.024 (2020). Publisher: Elsevier Inc.
5. Raju, R. V., Guntupalli, J. S., Zhou, G., Lázaro-Gredilla, M. & George, D. Space is a latent sequence: Structured sequence learning as a unified theory of representation in the hippocampus (2022).
6. Vaswani, A. *et al.* Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017-Decem**, 5999–6009 (2017). _eprint: 1706.03762.
7. Tolman, E. C. Cognitive maps in rats and men. *Psychol. Rev.* **55**, 189–208, 10.1037/h0061626 (1948). _eprint: h0054651.
8. Hassabis, D., Kumaran, D., Vann, S. D. & Maguire, E. A. Patients with hippocampal amnesia cannot imagine new experiences. *Proc. Natl. Acad. Sci.* **104**, 1726–1731, 10.1073/pnas.0610561104 (2007). _eprint: arXiv:1011.1669v3.
9. Fuster, J. M. & Alexander, G. E. Neuron activity related to short-term memory. *Sci. (New York, N.Y.)* **173**, 652–654, 10.1126/science.173.3997.652 (1971).
10. Bauer, R. H. & Fuster, J. M. Delayed-matching and delayed-response deficit from cooling dorsolateral prefrontal cortex in monkeys. *J. Comp. Physiol. Psychol.* **90**, 293–302, 10.1037/h0087996 (1976).
11. Funahashi, S., Chafee, M. V. & Goldman-Rakic, P. S. Prefrontal neuronal activity in rhesus monkeys performing a delayed anti-saccade task. *Nature* **365**, 753–756, 10.1038/365753a0 (1993).
12. Behrens, T. E. J. *et al.* What Is a Cognitive Map? Organizing Knowledge for Flexible Behavior. *Neuron* **100**, 490–509, 10.1016/j.neuron.2018.10.002 (2018). Publisher: Elsevier Inc.
13. Whittington, J. C. R., McCaffary, D., Bakermans, J. J. W. & Behrens, T. E. J. How to build a cognitive map. *Nat. Neurosci.* 1–16, 10.1038/s41593-022-01153-y (2022). Publisher: Nature Publishing Group.
14. Uria, B. *et al.* A model of egocentric to allocentric understanding in mammalian brains. Tech. Rep., bioRxiv (2022). 10.1101/2020.11.11.378141. Section: New Results Type: article.
15. Hebb, D. O. *The Organization of Behavior; A Neuropsychological Theory* (Wiley and Sons, New York, 1949).
16. Hopfield, J. J. 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 (1982).
17. O'Keefe, J. & Nadel, L. *The Hippocampus as a Cognitive Map* (Oxford University Press, 1978). ISSN: 0946-2716 Publication Title: Oxford University Press.
18. Deshmukh, S. S. & Knierim, J. J. Influence of local objects on hippocampal representations: Landmark vectors and memory. *Hippocampus* **23**, 253–67, 10.1002/hipo.22101 (2013).
19. Wood, E. R., Dudchenko, P. A., Robitsek, R. J. & Eichenbaum, H. Hippocampal neurons encode information about different types of memory episodes occurring in the same location. *Neuron* **27**, 623–633, 10.1016/S0896-6273(00)00071-4 (2000).
20. Hafting, T., Fyhn, M., Molden, S., Moser, M.-b. B. & Moser, E. I. Microstructure of a spatial map in the entorhinal cortex. *Nature* **436**, 801–806, 10.1038/nature03721 (2005). _eprint: /dx.doi.org/10.1038/nature01964.
21. Høydal, A., Skytøen, E. R., Andersson, S. O., Moser, M.-B. & Moser, E. I. Object-vector coding in the medial entorhinal cortex. *Nature* **568**, 400–404, 10.1038/s41586-019-1077-7 (2019).
22. Solstad, T., Boccara, C. N., Kropff, E., Moser, M.-B. & Moser, E. I. Representation of Geometric Borders in the Entorhinal Cortex. *Science* **322**, 1865–1868, 10.1126/science.1166466 (2008). _eprint: NIHMS150003.
23. Constantinescu, A. O. *Neural mechanisms underlying advanced cognition in humans*. PhD Thesis, Oxford University (2017).

24. Bongioanni, A. *et al.* Activation and disruption of a neural mechanism for novel choice in monkeys. *Nature* **591**, 270–274, 10.1038/s41586-020-03115-5 (2021). Publisher: Springer US.

25. Aronov, D., Nevers, R. & Tank, D. W. Mapping of a non-spatial dimension by the hippocampal–entorhinal circuit. *Nature* **543**, 719–722, 10.1038/nature21692 (2017). Publisher: Nature Publishing Group.

26. Burgess, N. & Hitch, G. J. Memory for serial order: A network model of the phonological loop and its timing. *Psychol. Rev.* **106**, 551–581, 10.1037/0033-295X.106.3.551 (1999).

27. Botvinick, M. M. & Plaut, D. C. Short-term memory for serial order: A recurrent neural network model. *Psychol. Rev.* **113**, 201–233, 10.1037/0033-295X.113.2.201 (2006).

28. Ganguli, S., Huh, D. & Sompolinsky, H. Memory traces in dynamical systems. *Proc. Natl. Acad. Sci.* **105**, 18970–18975, 10.1073/pnas.0804451105 (2008). Publisher: Proceedings of the National Academy of Sciences.

29. Xie, Y. *et al.* Geometry of sequence working memory in macaque prefrontal cortex. *Science* **375**, 632–639, 10.1126/science.abm0204 (2022).

30. El-Gaby, M. *et al.* A Cellular Basis for Mapping Behavioural Structure, 10.1101/2023.11.04.565609 (2023). Pages: 2023.11.04.565609 Section: New Results.

31. Miller, E. K. & Cohen, J. D. An Integrative Theory of Prefrontal Cortex Function. *Annu. Rev. Neurosci.* **24**, 167–202, 10.1146/annurev.neuro.24.1.167 (2001).

32. Postle, B. R. Working Memory as an Emergent Property of the Mind and Brain. *Neuroscience* **139**, 23–38, 10.1016/j.neuroscience.2005.06.005 (2006).

33. Panichello, M. F. & Buschman, T. J. Shared mechanisms underlie the control of working memory and attention. *Nature* **4**, 10.1038/s41586-021-03390-w (2021). Publisher: Springer US.

34. Krotov, D. & Hopfield, J. Large Associative Memory Problem in Neurobiology and Machine Learning. *arXiv preprint* 1–12 (2020). _eprint: 2008.06996.

35. Ramsauer, H. *et al.* Hopfield networks is all you need. *arXiv preprint* (2020). _eprint: 2008.02217.

36. Whittington, J. C. R., Warren, J. & Behrens, T. E. J. Relating transformers to models and neural representations of the hippocampal formation. *Int. Conf. on Learn. Represent.* (2021).

37. Katharopoulos, A., Vyas, A., Pappas, N. & Fleuret, F. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention (2020). ArXiv:2006.16236 [cs, stat].

38. Sun, Y. *et al.* Retentive Network: A Successor to Transformer for Large Language Models (2023). ArXiv:2307.08621 [cs].

39. Baddeley, A. D. & Hitch, G. Working Memory. In Bower, G. H. (ed.) *Psychology of Learning and Motivation*, vol. 8, 47–89, 10.1016/S0079-7421(08)60452-1 (Academic Press, 1974).

40. Dorrell, W., Latham, P. E., Behrens, T. E. J. & Whittington, J. C. R. Actionable Neural Representations: Grid Cells from Minimal Constraints. *Int. Conf. on Learn. Represent.* (2023).

41. Stokes, M. G. 'Activity-silent' working memory in prefrontal cortex: a dynamic coding framework. *Trends Cogn. Sci.* **19**, 394–405, 10.1016/j.tics.2015.05.004 (2015). Publisher: Elsevier.

42. Gu, A. & Dao, T. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 10.48550/arXiv.2312.00752 (2023). ArXiv:2312.00752 [cs].

43. Whittington, J. C. R., Dorrell, W., Ganguli, S. & Behrens, T. E. J. Disentanglement with Biological Constraints: A Theory of Functional Cell Types. *Int. Conf. on Learn. Represent.* 10.48550/arXiv.2210.01768 (2023). ArXiv:2210.01768 [cs, q-bio].

44. Hsu, K., Dorrell, W., Whittington, J. C. R., Wu, J. & Finn, C. Disentanglement via Latent Quantization. *Adv. In Neural Inf. Process. Syst.* (2023). ArXiv:2305.18378 [cs, stat].

45. Sargolini, F. *et al.* Conjunctive representation of position, direction, and velocity in entorhinal cortex. *Science* **312**, 758–762, 10.1126/science.1125572 (2006).

46. Khona, M. & Fiete, I. R. Attractor and integrator networks in the brain. *Nat. Rev. Neurosci.* 1–23, 10.1038/s41583-022-00642-0 (2022). Publisher: Nature Publishing Group.

47. Kim, S. S., Rouault, H., Druckmann, S. & Jayaraman, V. Ring attractor dynamics in the Drosophila central brain. *Science* **356**, 849–853, 10.1126/science.aal4835 (2017).

48. Green, J. *et al.* A neuronal circuit architecture for angular integration in Drosophila. *Nature* **546**, 101–106, 10.1038/nature22343 (2017).

49. Turner-Evans, D. *et al.* Angular velocity integration in a fly heading circuit. *eLife* **6**, e23496, 10.7554/eLife.23496 (2017). Publisher: eLife Sciences Publications, Ltd.

50. Taube, J., Muller, R. & Ranck, J. Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *The J. Neurosci.* **10**, 420–435, 10.1523/JNEUROSCI.10-02-00420.1990 (1990). ISBN: 0270-6474 (Print).

51. Basu, R. *et al.* The orbitofrontal cortex maps future navigational goals. *Nature* **599**, 449–452, 10.1038/s41586-021-04042-9 (2021). Number: 7885 Publisher: Nature Publishing Group.

52. Schwartenbeck, P. *et al.* Generative replay underlies compositional inference in the hippocampal-prefrontal circuit. *Cell* **186**, 4885–4897.e14, 10.1016/j.cell.2023.09.004 (2023).

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

54. Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P. & Lerchner, A. COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration (2019). ArXiv:1905.09275 [cs].

55. Burgess, C. P. *et al.* MONet: Unsupervised Scene Decomposition and Representation. *arXiv preprint* 1–22 (2019). _eprint: 1901.11390.

56. Whittington, J. C. R., Kabra, R., Matthey, L., Burgess, C. P. & Lerchner, A. Constellation: Learning relational abstractions over objects for compositional imagination. *arXiv preprint* (2021). _eprint: 2107.11153.

57. Kozachkov, L. *et al.* Robust and brain-like working memory through short-term synaptic plasticity. *PLOS Comput. Biol.* **18**, e1010776, 10.1371/journal.pcbi.1010776 (2022). Publisher: Public Library of Science.

58. Pritzel, A. *et al.* Neural Episodic Control. *arXiv preprint* 10.1038/nature20101 (2017). _eprint: 1703.01988.

59. Cho, K. *et al.* Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proc. 2014 Conf. on Empir. Methods Nat. Lang. Process. (EMNLP)* 1724–1734, 10.3115/v1/D14-1179 (2014). ISBN: 9781937284961 _eprint: 1406.1078.

60. Ba, J. L., Kiros, J. R. & Hinton, G. E. Layer Normalization. *arXiv preprint* 10.1038/nature14236 (2016). ISBN: 978-3-642-04273-7 _eprint: 1607.06450.

61. Hulse, B. K. *et al.* A connectome of the Drosophila central complex reveals network motifs suitable for flexible navigation and context-dependent action selection. *eLife* **10**, e66039, 10.7554/eLife.66039 (2021). Publisher: eLife Sciences Publications, Ltd.

A Appendix

A.1 Full Theory

Task reminder

Formally, for each task, we consider a dataset $\mathscr{D} = \{\mathscr{D}_0, \mathscr{D}_1, \cdots, \mathscr{D}_N\}$, where each $\mathscr{D}_i$ is a sequence consisting of vectors of sensory observations, $\boldsymbol{o}$, of dimension d_o (termed $\boldsymbol{o} \in \mathbb{R}^{d_o}$), sensory targets, $\boldsymbol{t} \in \mathbb{R}^{d_o}$, and (allocentric) velocities, $\boldsymbol{v} \in \mathbb{R}^{d_v}$, i.e., $\mathscr{D}_i = \{(\boldsymbol{o}_0, \boldsymbol{t}_0, \boldsymbol{v}_0), (\boldsymbol{o}_1, \boldsymbol{t}_1, \boldsymbol{v}_1), \cdots, (\boldsymbol{o}_K, \boldsymbol{t}_K, \boldsymbol{v}_K)\}$ 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 $\mathscr{D}_i$, 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[35], a transformer neural network[36], or a differentiable neural dictionary[58]. All these methods are relatable to each other[36], 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

$$\begin{aligned}\hat{\boldsymbol{t}}_t &= \boldsymbol{V}\text{softmax}(\boldsymbol{K}^T\boldsymbol{q}_t) \\ &= \boldsymbol{O}_{t-1}\text{softmax}(\boldsymbol{G}_{t-1}^T\boldsymbol{g}_t)\end{aligned} \tag{3}$$

Where $\boldsymbol{g}_t \in \mathbb{R}^{d_g}$ is the RNN state representation at time t, and $\boldsymbol{O}_{t-1} = \begin{bmatrix}\boldsymbol{o}_0 & \cdots & \boldsymbol{o}_{t-1}\end{bmatrix}$ and $\boldsymbol{G}_{t-1} = \begin{bmatrix}\boldsymbol{g}_0 & \cdots & \boldsymbol{g}_{t-1}\end{bmatrix}$ are stored memories, i.e., the memories bind together each $\boldsymbol{g}$ and $\boldsymbol{o}$ at each timestep[v]. 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:

$$\begin{aligned}\boldsymbol{g}_t &= f(\{\boldsymbol{v}_0, \cdots, \boldsymbol{v}_t\}) \\ &= \boldsymbol{W}_{\boldsymbol{v}_{t-1}}\boldsymbol{g}_{t-1} \qquad \text{or} \qquad f(\boldsymbol{W}_r\boldsymbol{g}_{t-1} + \boldsymbol{B}\boldsymbol{v}_t)\end{aligned} \tag{4}$$

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[40]), 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 $\boldsymbol{z}_t \to \boldsymbol{z}_{t+1}$, 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

$$\begin{aligned}\boldsymbol{O}_{t-1} &= \begin{bmatrix}\boldsymbol{o}_0 & \cdots & \boldsymbol{o}_{t-1}\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{o}(\boldsymbol{z}_0) & \cdots & \boldsymbol{o}(\boldsymbol{z}_{t-1})\end{bmatrix}\end{aligned} \tag{5}$$

and

[v] Normally key, query, and value matrices, are used i.e. $\boldsymbol{W}_v\boldsymbol{O}_{t-1}\text{softmax}(\boldsymbol{G}_{t-1}^T\boldsymbol{W}_q^T\boldsymbol{W}_k\boldsymbol{g}_t)$. We leave them out for simplicity, but the following argument would remain the same if we included them.

$$
\begin{aligned}
\boldsymbol{G}_{t-1} &= \begin{bmatrix}\boldsymbol{g}_0 & \cdots & \boldsymbol{g}_{t-1}\end{bmatrix} \\
&= \begin{bmatrix}\boldsymbol{g}(\boldsymbol{z}_0) & \cdots & \boldsymbol{g}(\boldsymbol{z}_{t-1})\end{bmatrix}
\end{aligned} \tag{6}
$$

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[vi]. 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:

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

Expanding this equation out yields:

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

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 Appendix A.2). 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:

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

[vi] 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.

Term in EM solution	Dimension in EM	Equivalent term in WM solution	Dimension in WM
RNN attending to memories	vector: n_p	RNN Readout Matrix	matrix: $n_o \times n_p n_o$
Memory slots in weights	matrix: $n_o \times n_p$	RNN memories in activity slots	vector: $n_p n_o$
RNN transition matrix	matrix: $n_p \times n_p$	RNN transition matrix	matrix: $n_p n_o \times n_p n_o$

Table 1. Equivalent terms in the EM and WM formulations, along with their dimensions. We see that vectors (activity) turn into matrices (synaptic weights), and vice versa. n_p is the number of positions in the task, n_o is the dimension of the observation vector ($\boldsymbol{o}$).

This equation performs the exact same computation as the previous equation, and its terms look very similar but there are some important differences (Table 1 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 ($\boldsymbol{0} \in \mathbb{R}^{d_o \times d_o}$). 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:

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

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:

$$\begin{aligned} \boldsymbol{h}_t = \begin{bmatrix} \boldsymbol{h}_t^s \\ \boldsymbol{h}_t^v \end{bmatrix} &= \begin{bmatrix} f(\boldsymbol{W}_{rec}^{ss}\boldsymbol{h}_{t-1}^s + \boldsymbol{W}_{rec}^{sv}\boldsymbol{h}_{t-1}^v + \boldsymbol{W}_{in}^s\boldsymbol{i}_t) \\ f(\boldsymbol{W}_{rec}^{vv}\boldsymbol{h}_{t-1}^v + \boldsymbol{W}_{rec}^{vs}\boldsymbol{h}_{t-1}^s + \boldsymbol{W}_{in}^v\boldsymbol{i}_t) \end{bmatrix} \\ &= \begin{bmatrix} f(\boldsymbol{W}_{rec}^{ss}\boldsymbol{h}_{t-1}^v + \boldsymbol{W}_{rec}^{sv}\boldsymbol{h}_{t-1}^v + \boldsymbol{W}_{in}^s\boldsymbol{i}_t) \\ f(\boldsymbol{W}_{rec}^{vv}\boldsymbol{h}_{t-1}^v + \boldsymbol{W}_{in}^v\boldsymbol{i}_t) \end{bmatrix} \end{aligned} \tag{11}$$

Where $\boldsymbol{W}_{rec}^{sv}$ 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 ($\boldsymbol{W}_{rec}^{vs} = 0$). 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 $\boldsymbol{W}_{rec}^{sv}\boldsymbol{h}_{t-1}^v$.

A.2 Model Details

We train models which are all variants on recurrent neural networks (RNNs), which differ in two key components (Figure 7). 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[59], 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, $f(\dots)$ 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 7C). 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 7D). In Figure 3 we make use of another RNN inspired by the fly head-direction circuit (Figure 7E,F). 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 $\boldsymbol{o}_t$.

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 $\hat{\boldsymbol{t}}_t^{WM} = \boldsymbol{W}_{out}^{WM} f(\boldsymbol{r}_t)$, i.e. standard RNN prediction (Figure 7B). Second for the **EM** class: predictions are made using an external memory system related to a transformer / Hopfield network. In particular, $\hat{\boldsymbol{t}}_t^{EM} = \boldsymbol{W}_{out}^{EM}\boldsymbol{V}_{t-1}\text{softmax}(\boldsymbol{K}_{t-1}^T\boldsymbol{W}_q f(\boldsymbol{r}_t))$, where the key and value matrix are sequentially updated i.e., $\boldsymbol{K}_{t-1} = \begin{bmatrix}\boldsymbol{k}_0 & \cdots & \boldsymbol{k}_{t-1}\end{bmatrix}$ and $\boldsymbol{V}_{t-1} = \begin{bmatrix}\boldsymbol{u}_0 & \cdots & \boldsymbol{u}_{t-1}\end{bmatrix}$ (Figure 7A; 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[58] or modern Hopfield network[35]. These memories consist of two parts - a key and a value (Figure 7A). The added key, as described above, at each timestep is $\boldsymbol{k}_t = \boldsymbol{W}_k\boldsymbol{h}_t$, while the added value is $\boldsymbol{u}_t = \boldsymbol{W}_v\boldsymbol{h}_t - \hat{\boldsymbol{m}}_t$, where $\hat{\boldsymbol{m}}_t$ is the retrieved memory at that timestep ($\hat{\boldsymbol{m}}_t = \boldsymbol{V}_{t-1}\text{softmax}(\boldsymbol{K}_{t-1}^T\boldsymbol{W}_q f(\boldsymbol{r}_t))$). 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: $\hat{\boldsymbol{t}}_t^{EM} = \boldsymbol{W}_{out}^{EM}\boldsymbol{V}_{t-1}\text{softmax}(\beta\boldsymbol{K}_{t-1}^T\boldsymbol{W}_q f(\boldsymbol{r}_t))$. 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[60]: $\text{LayerNorm}(\boldsymbol{V}_{t-1}\text{softmax}(\boldsymbol{K}_{t-1}^T\boldsymbol{W}_q f(\boldsymbol{r}_t)))$. This is so all retrieved memories have the same scaling. 2) In some simulations (detailed in Table 2) 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 $\hat{\boldsymbol{t}}_t^{WM} = \boldsymbol{W}_{out}^{WM}\text{LayerNorm} f(\boldsymbol{r}_t)$. 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 5,6), or a squared error loss otherwise (Figures 5,6). 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 2.

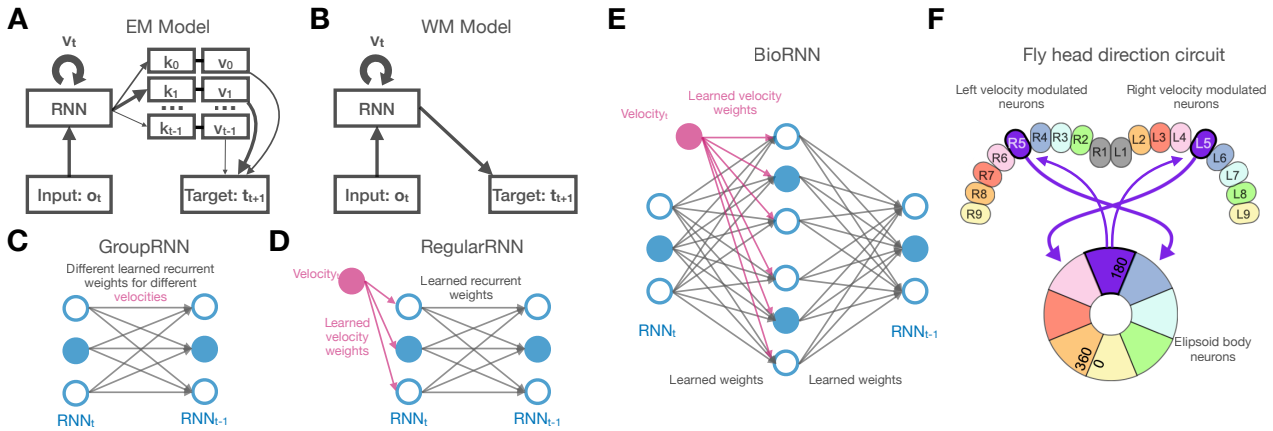

Figure 7. Model variants. A) The structure of the EM model. An RNN stores past memories using keys and values, like a transformer model which is related to Hopfield networks. Predictions are made by the RNN indexing memories. **B)** The structure of the WM model is a standard RNN setup. **C)** Like our theory we use an RNN with learnable velocity dependent matrices (GroupRNN); **D)** We also use a conventional RNN which receives velocity as input (RegularRNN). **E)** The BioRNN variant integrates velocity in an intermediate step between transition, much like the **F)** Head direction circuit discovered in the fly brain[47]. (E) is adapted from[61].

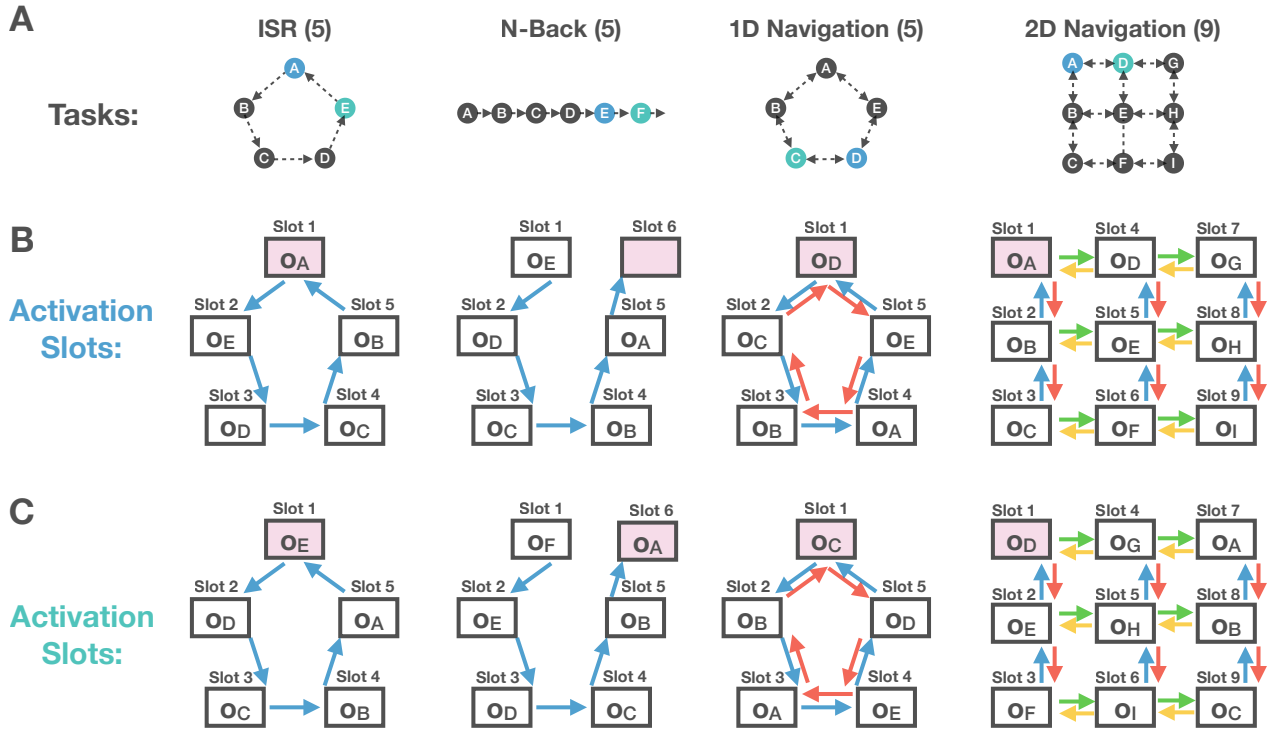

Figure 8. Tasks, and model predictions of activity slots. A) The structure of the four tasks. ISR is the same as unidirectional walks on a loop (5-loop shown). N-Back is unidirectional walks on a line (5-Back task shown). 1D navigation is a random walk on a loop (5-loop shown. 2D navigation is a random walk on a 2D torus (3-by-3 torus shown). **B)** The activity slot structure mimics the underlying structure of the task. Purple slots are the readout slots. Observations shown in slots, for the blue position above. The N-Back task nearly has a loop structure as observations more than N steps ago are forgotten. **C)** Observations now shown in slots, for the green position above. Note how the observations are copied and shifted between activity slots.

Hyperparameter	Figure 2	Figure 3	Figure 4	Figure 5	Figure 6
# RNN neurons	32,64,128	256	512	128	128
Activation Function	None	None,tanh,ReLU	ReLU	ReLU	ReLU
RNN type	gRNN	gRNN, rRNN, bRNN	rRNN	rRNN	rRNN
LayerNorm	True	False	True	False	False
Key/query dim	64	64	64	64	64
RNN regularisation	0	0, 0.01	0	3e-4	3e-4
KL regularisation	0	0, 0.01	0	0	0
Weight regularisation	5e-8	1e-6	5e-8	1e-6	1e-6
Batch Size	128	128	128	128	128
Optimizer	Amsgrad	Adam	Amsgrad	Adam	Adam
Learning Rate	8e-4 $\rightarrow$ 2e-4	8e-4 $\rightarrow$ 2e-4	8e-4 $\rightarrow$ 2e-4	8e-4 $\rightarrow$ 2e-4	8e-4 $\rightarrow$ 2e-4
Gradient steps	100000	500000	100000	100000	100000
# observations	10	10	10	6	4
Sample with replacement	True	True	False	False	False

Table 2. Hyperparameters for Figures in main text. Where there are commas, this indicates that multiple values were used for this hyperparameter, which will be explained in the individual figure. gRNN/rRNN/bRNN: GroupRNN/RegularRNN/BioRNN. X$\rightarrow$Y means this hyperparameter was slowly changed from X to Y during training.

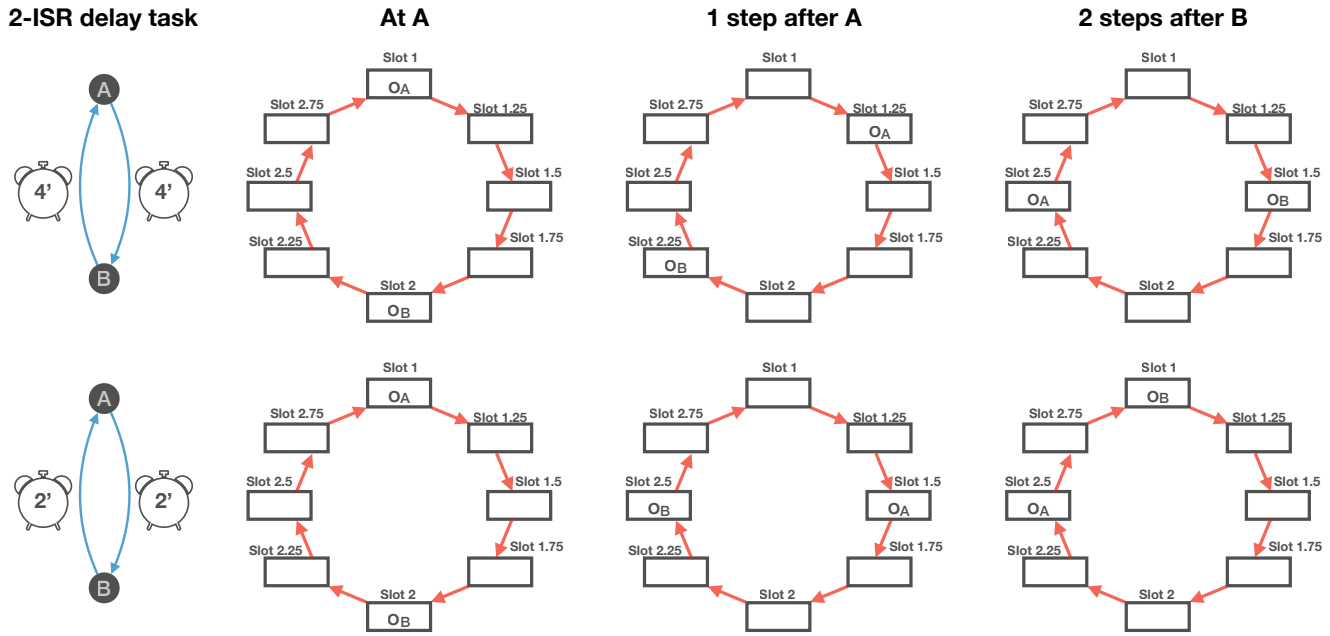

Figure 9. Task-progress slot prediction for 2-ISR delay task. We show two example 2-ISR delay tasks (top/bottom). The top task has delays of 4 steps, while the bottom has delays of 2 steps. For each task, we show task-progress slot predictions at three different time instances. We see differential activity between the two tasks, since the delay periods differ and so observations move round the task-progress slots at different rates.

A.3 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.

A.3.1 Figures 2, 3, 11, 12, 13

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: $\boldsymbol{o} = \{1,4,3,1,4,3,1\dots\}$ and $\boldsymbol{t} = \{-,-,-,1,4,3,1\dots\}$. '$-$' 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 $N = 3$: $\boldsymbol{o} = \{1,2,4,2,3,5,9\dots\}$ and $\boldsymbol{t} = \{-,-,-,1,2,4,2\dots\}$. 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: $\boldsymbol{o} = \{1,3,2,2,3,1,2\dots\}$, , $\boldsymbol{v} = \{+1,+1,0,0,-1,-1,-1\dots\}$ and $\boldsymbol{t} = \{-,-,-,2,3,1,2\dots\}$. 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: $\boldsymbol{o} = \{5,1,5,2,3,1,1\dots\}$, $\boldsymbol{v} = \{(+1,0),(+1,0),(0,+1),(-1,0),(0,-1),(0,0),\dots\}$, and $\boldsymbol{t} = \{-,-,5,-,-,1,1\dots\}$. The length of the each sequence is 7 multiplied by the torus size.

A.3.2 Figure 4

2-ISR with delays. Delay period lengths were randomly sampled from [3,4,5,6,7] for each task. Example observation sequences are $\boldsymbol{o} = \{4,*,*,2,*,*,4,*,*,2,\dots\}$ or $\boldsymbol{o} = \{1,*,3,*,1,*,3,\dots\}$ where $*$ is the common delay observation (that also must be predicted). Corresponding target sequences are $\boldsymbol{t} = \{-,-,-,-,-,-,-,*,*,2,\dots\}$ or $\boldsymbol{t} = \{-,-,-,-,-,*,3,\dots\}$. 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 1st 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. Delay period lengths were randomly sampled from [3,4,5] for each task. Example observation sequences are $\boldsymbol{o} = \{4,*,2,*,*,2,*,6,4,*,2,*,*,2,*,6,\dots\}$ or $\boldsymbol{o} = \{1,3,*,8,*,2,1,3,*,8,*,2,\dots\}$ where $*$ is the common delay observation (that also must be predicted). Corresponding target sequences are $\boldsymbol{t} = \{-,-,-,-,-,-,-,-,-,*,2,*,*,2,*,6,\dots\}$ or $\boldsymbol{t} = \{-,-,-,-,-,-,-,3,*,8,*,2,\dots\}$. 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 1st 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).

A.3.3 Figure 14

1D Navigation with structured velocity signals. We used a 7-loop task (1D navigation), but where the underlying velocity cycles a fixed schedule of $\{+2,+1,0,-1,0,+1,+2,+1,0,\dots\}$ (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. Sequences we drawn from a 3-by-3 torus, but where the underlying velocity cycles a fixed schedule of $\{N,W,S,S,E,E,N,N,N,W,S,S,\dots\}$ (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. 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. 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.

A.3.4 Figure 5.

Xie et al. Here observations are one of six 2D positions lying on a ring. This is just like the original paper[29]). 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).

A.3.5 Figure 6

Panichello et al. 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).

A.4 Analysis Details

A.4.1 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. $0\% \to 25\%$, $25\% \to 50\%$, $50\% \to 75\%$, $75\% \to 100\%$.

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).

A.4.2 Mutual Information Ratio.

To calculate the mutual information ratio[43] (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.

A.4.3 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 2 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 10), suggesting that the $\boldsymbol{h}_t$ is dependent on the velocities signals taken to reach state $\boldsymbol{z}_t$.

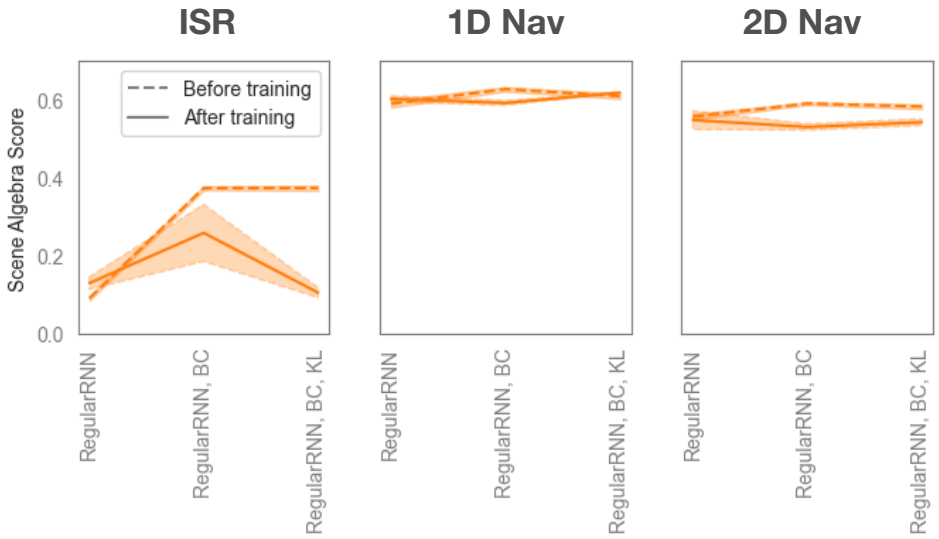

Figure 10. Slot algebra for RNN receiving velocity input during 'path integration': The same slot algebra analysis as Figure 3C, but with a RNN where the velocity signal of the upcoming transition is only provided to the path integration step, i.e. $\boldsymbol{r}_t = \boldsymbol{W}_{rec}\boldsymbol{h}_{t-1} + \boldsymbol{B}\boldsymbol{v}_t$ ($\boldsymbol{B}$ is a learnable matrix), and the external input, $\boldsymbol{i}_t$, is just $\boldsymbol{o}_t$. Slot algebra scores are high for tasks that involve variable velocity signals (1D Nav and 2D Nav) suggesting that the learned internal representation is not invariant to trajectory.

A.4.4 PFC Analyses.

We followed the exact analyses described in Xie et al., 2022[29] for Figure 5 and the exact analyses described in Panichello et al., 2021[33] for Figure 6.

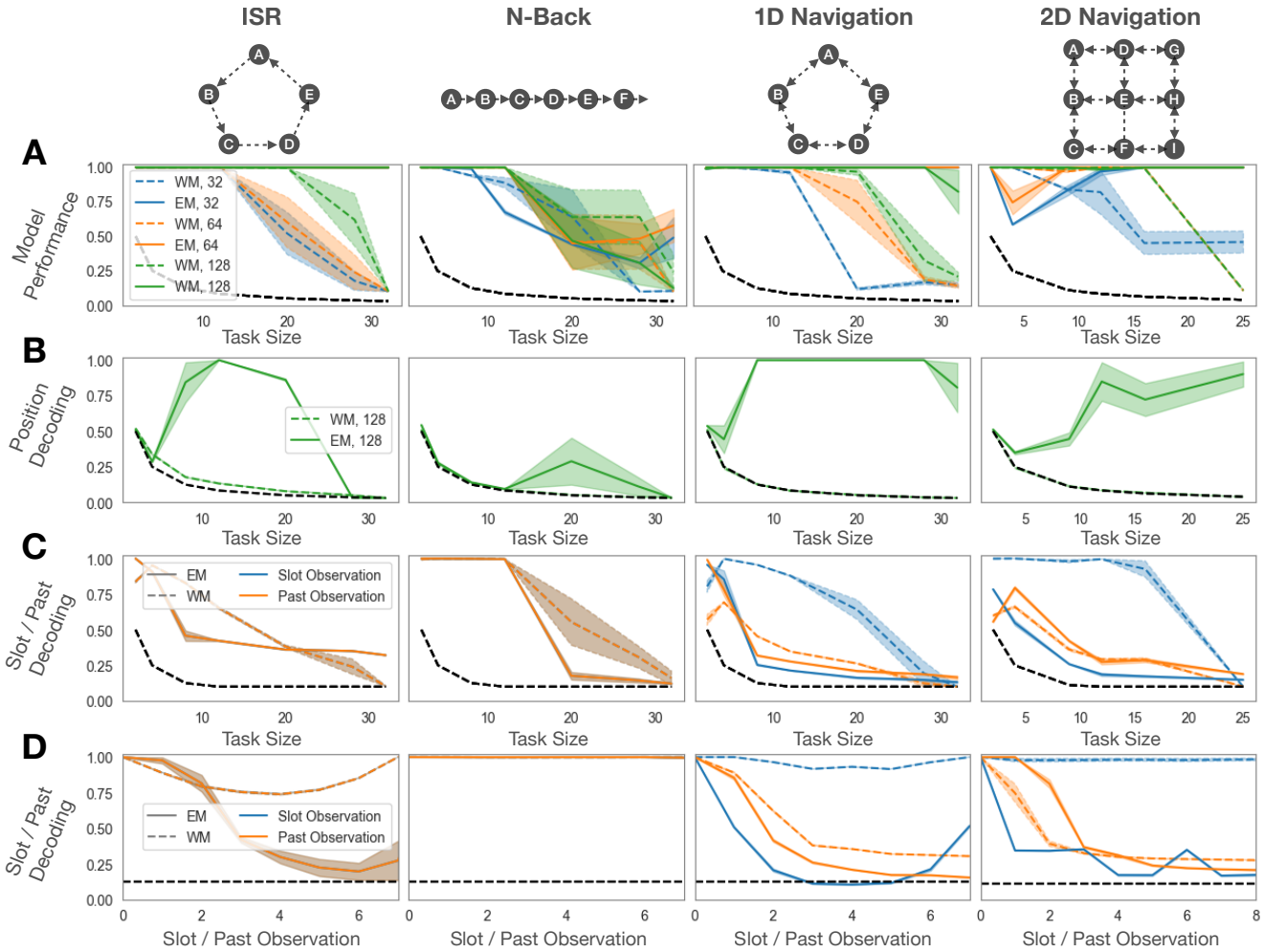

Figure 11. GroupRNN: EM models scale better and learn position representations, while WM models learn activity slots. For each combination of task, task size, model type, and RNN size (32, 64, 128), we train five randomly initialised models. Results are mean ± standard error. **A)** Performance of the EM (solid) and WM (dashed) models on the four tasks. EM models consistently perform better that WM as task size increases. Larger sized RNN consistently perform better for both EM and WM models. All subsequent panels are for RNNs with size 128. **B)** Position decoding. EM models, but not WM models, learn position like representations. **C)** Average decoding of 'slot-sequences' (which observations should be in each slot at each timestep according to our theory) and 'past-sequences' (what observation occurred i steps ago, up to the total number of slot). For ISR and N-Back tasks, slot- and past-sequence decoding is identical since velocity is constant (+1) thus the ith past observation will be held in the ith slot. Otherwise, slot decoding is consistently higher than past-sequence decoding for WM models, but not EM models. **D)** Slot- and past-sequence decoding for individual slots or past timesteps (on task with size 8 for first 3 tasks, and 9 for 2D Navigation). Slot decoding is consistently higher than past-sequence decoding for WM models, but not EM models. We note that on the ISR task, performance is high but position decoding is low for large tasks sizes. We posit this is due to an additional solution only available in this task: a Laplace transform. We further note that for the N-back task, the position (or slot) we decode is not the veridical position (or slot) since positions (or slots) never get repeated. Instead, we decode position (or slot) as if it were cycling on a N-loop. Thus it is the structure of the input-output map that gets learned, rather than the structure of the input.

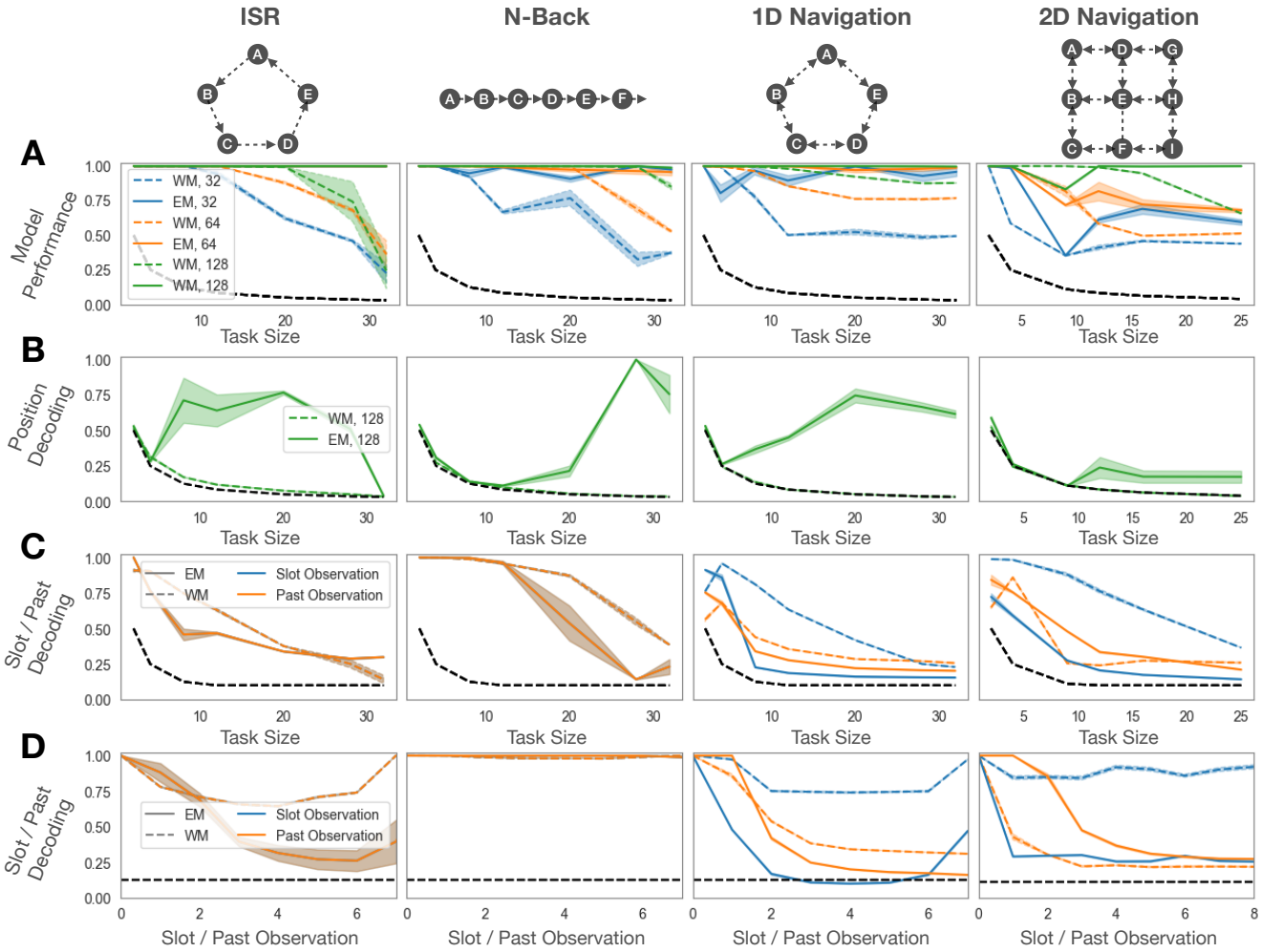

Figure 12. RegularRNN: EM models scale better and learn position representations, while WM models learn activity slots. Same as Figure 11 but for the RegularRNN

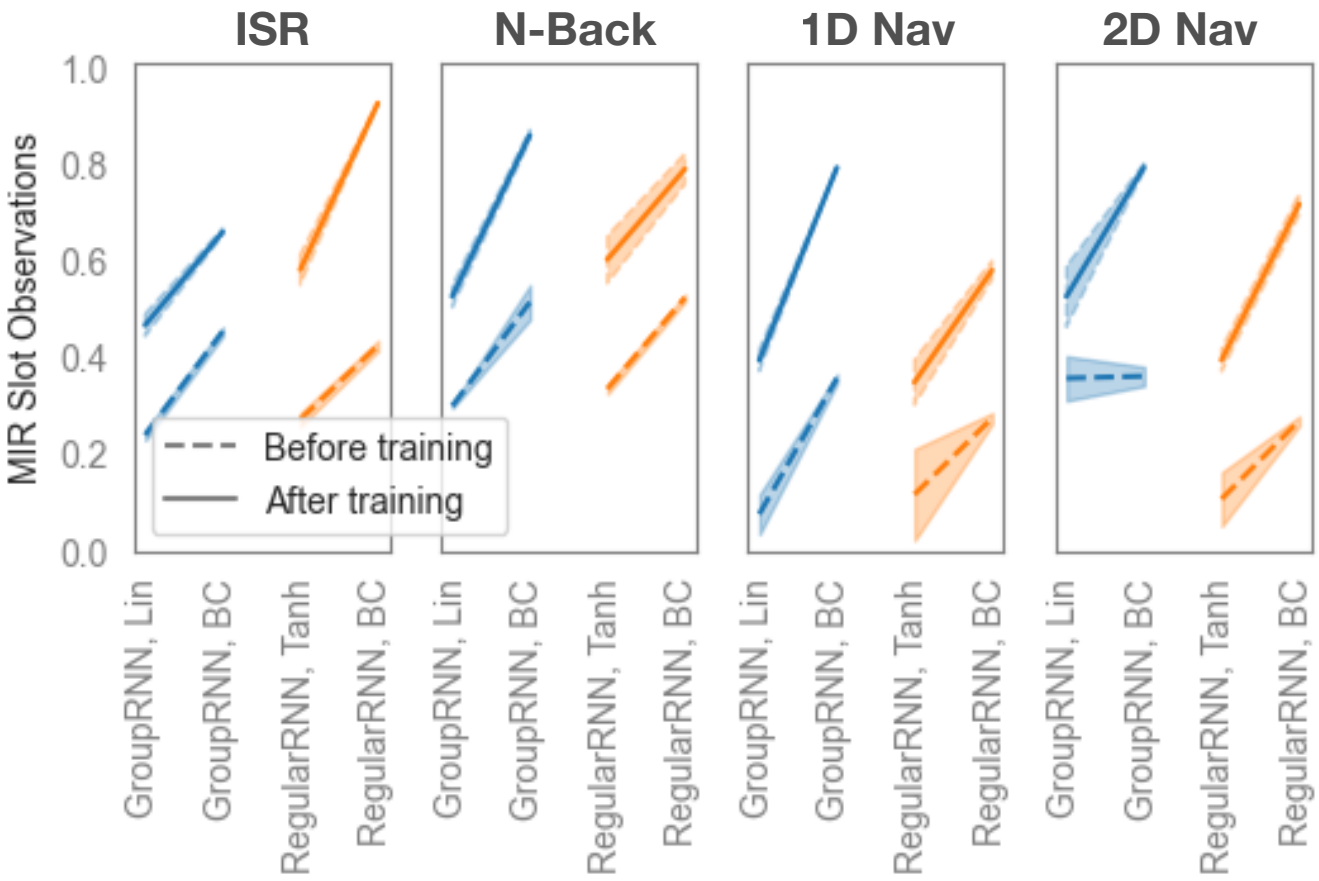

Figure 13. WM models display single neuron coding of slots when additional constraints (BC) are used. The Mutual Information Ratio (MIR) measures the extent of single neuron coding of slots. MIR rises from the beginning of training (dashed) to the end (solid), particularly for models with additional constraints.

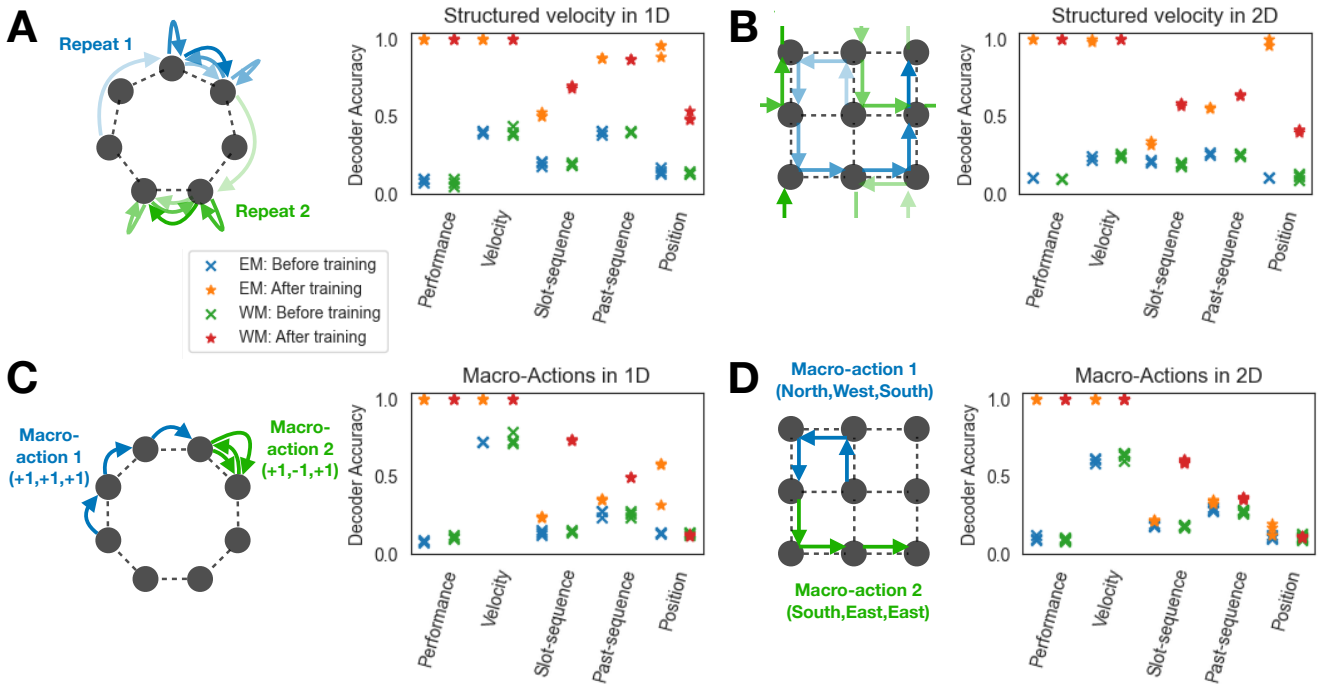

Figure 14. Activity slots are not learned when velocities are non-random sequences. A) Left: On a 7-loop task (1D navigation), but where velocity cycles a fixed schedule of $\{+2,+1,0,-1,0,+1,+2,+1,0,\ldots\}$. Schematic shows 1st / 2nd cycle of fixed velocity schedule shown in blue / green. Right: Each point corresponds to a randomly initialised model before (cross) and after (star) training; three random initialised models shown. Velocity can be decoded in both EM and WM models. WMmodels, however, do **not** learn activity slot representations: past-sequence decoding is higher to slot-sequence decoding. We note that EM models still learn a position representation. **B)** Same as A) but for a fixed velocity schedule ($\{$`North`,`West`,`South`,`South`,`East`,`East`,`North`,`North`$\}$) on a 3-by-3 torus. **C)** Left: Taking macro-actions on a 8-loop task (1D navigation). The input 'velocity' signal is each macro action, which can be up to 4 sub-actions long. Schematic shows 2 macro actions in blue / green. Right: Each point corresponds to a randomly initialised model before (cross) and after (star) training; three random initialised models shown. The model learns to represent the sub-actions in both EM and WM models (decoder accuracy shown for final sub-action). WMmodels **do** learn an activity slot representation, while EM models learn a position representation. **D)** Same as B) but in a 2D navigation task on a 3-by-3 torus. These results suggest that activity slots are not learned when velocity signals are fully structured (like the prescribed velocity sequences in A and B), but when randomness is introduced (like different random macro-actions, macro-actions are in-between structured sequences and fully random actions) activity slots are learned.

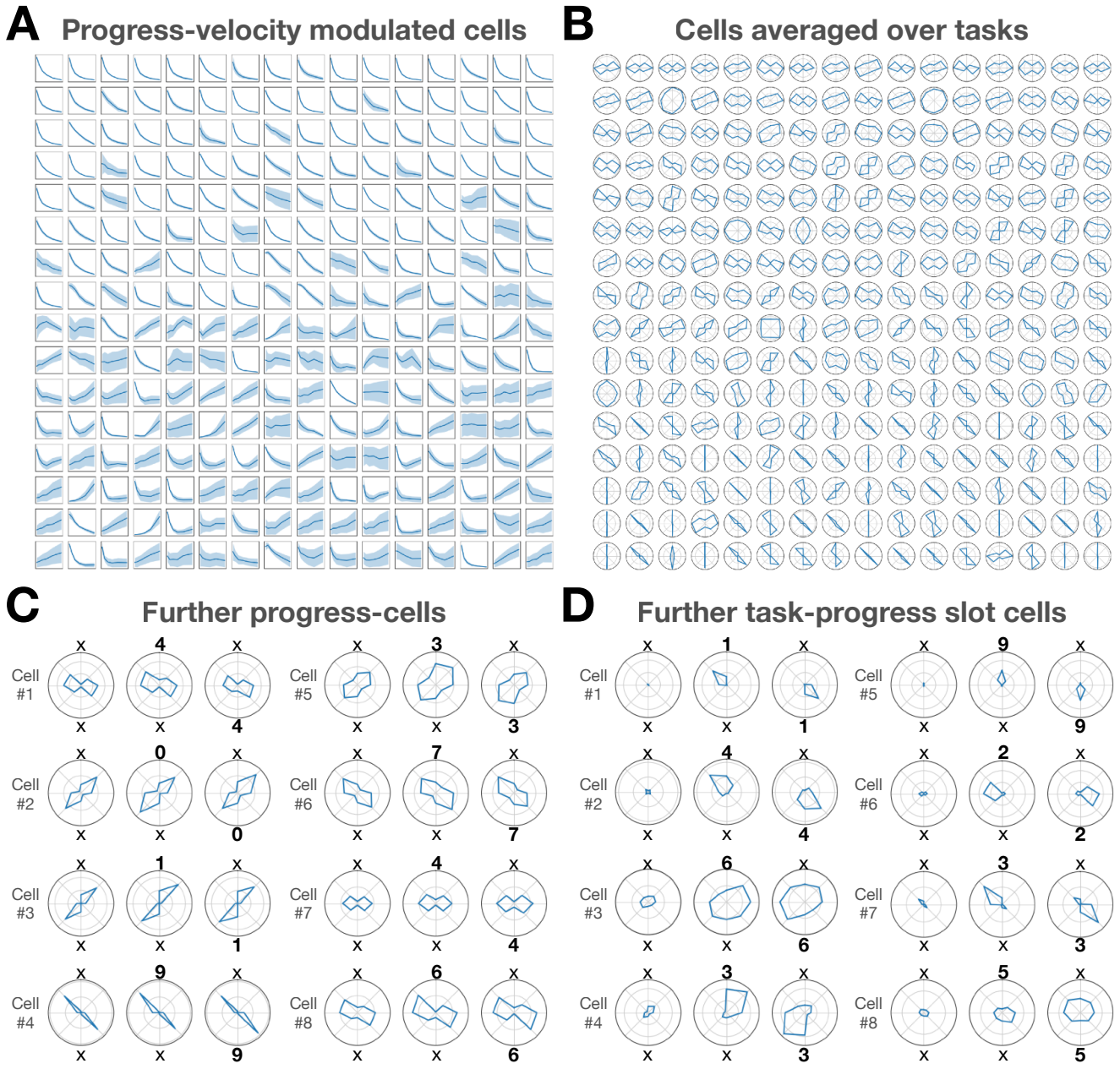

Figure 15. Further RNN cells representations for the 2-ISR delay task in Figure 4 A) RNN cells are modulated by progress-velocity (showing 256 of 512 cells with highest activity). Each subplot is a cell. The x-axis is the 'true' progress-velocity (i.e., 1 divided by the current delay length). The y-axis is the average firing of that cell for that progress-velocity. **B)** The average firing of RNN cells across many tasks (we rescale activity around the 2-ISR delay loop into a common length, i.e., task-progress; we show 256 of 512 cells with highest activity). The average firing across tasks for most cells is bimodal. This is consistent with both progress (bi-modal) and task-progress slot cells (uni-modal but rotated across tasks). **C-D)** In these plots, we have extracted the 'preferred' observation for that particular cell (the observation that, when present in a task, leads to the cell having the highest activity), and plotted the cell's activity when the preferred observation is at different positions on the 2-loop. Left: Observation not present. Middle/Right: Observation in A/B position. **C)** Example progress cells. Progress cells respond the same, even for tasks without the preferred observation (left of each subplot; because progress cells do not have a preferred observation). Progress cells are bimodal as they track progress within the delay. **D)** Example task-progress slot cells. These cells do have a preferred observation; they are inactive when that observation is not present in the task (left of each subplot). They fire at a particular lag after their preferred observation, and so these cells are not bimodal.

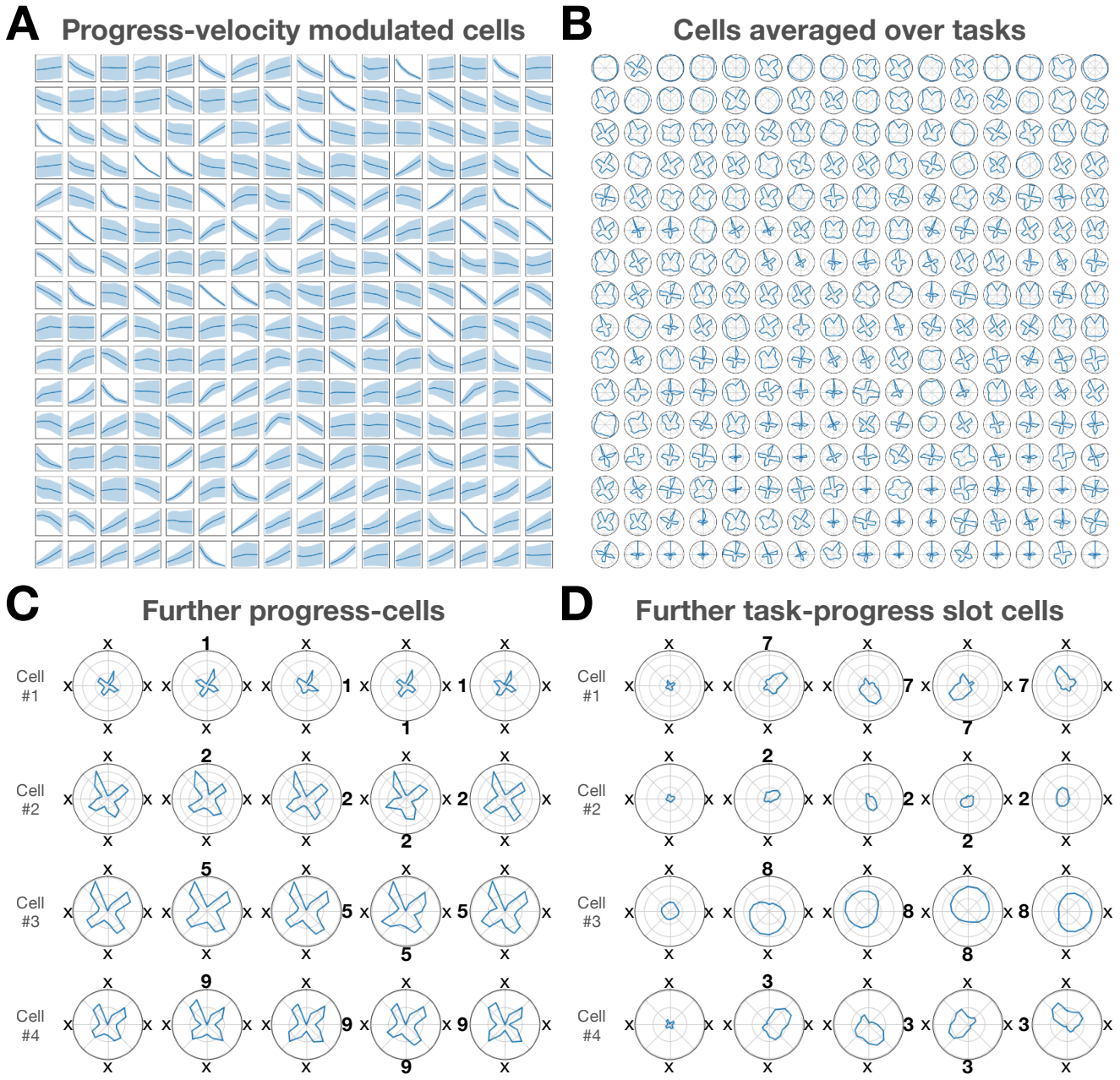

Figure 16. Further RNN neural representations for the 4-ISR delay task in Figure 4 A-D) Same as Figure 15, but for the 4-ISR delay task.

Acknowledgements

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 JCRW.; the Gatsby Charitable Foundation to WD.; 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 TEJB.; the Wellcome Centre for Integrative Neuroimaging is supported by core funding from the Wellcome Trust (203139/Z/16/Z); the James S. McDonnell, Simons Foundations, NTT Research, and an NSF CAREER Award to SG.

Author Contributions

JCRW conceptualised the study in conversations with WD, TEJB, SG, ME. JCRW developed theory and performed simulations. JCRW wrote the manuscript with input from WD, TEJB, SG, ME.

Competing interests

The authors declare no competing interests.

Materials & Correspondence

Correspondence to James CR Whittington (jcrwhittington@gmail.com).

Data availability

No data was generated in this paper..

Code availability

Python and Pytorch code will be made available on publication.