BRAIN-LIKE SLOT REPRESENTATION FOR SEQUENCE WORKING MEMORY IN RECURRENT NEURAL NETWORKS

Mingye Wang & Stefano Fusi
Center for Theoretical Neuroscience, Columbia University
New York, NY, USA
mw3735@cumc.columbia.edu, sf2237@columbia.edu

Kimberly Stachenfeld
Google DeepMind
Center for Theoretical Neuroscience, Columbia University
New York, NY, USA
stachenfeld@deepmind.com

ABSTRACT

Sequence working memory (SWM) is the ability to remember a sequence of items and recall them in order. Recent neural recordings in the prefrontal cortex during SWM found "slot"-like representations, where each item in the sequence is stored in a separate low-dimensional subspace of population activity. However, it is unclear what circuit structure could give rise to such representations, and if such slot-like representation naturally arises from the constraints of the SWM task. Here, we trained recurrent neural networks (RNNs) on a SWM task using standard architectures and training procedures. Two types of networks emerged, relying on two types of representations: (1) is the brain-like fixed slot representation, (2) stores items in time-varying subspaces. They lead to different length generalization behaviors: (1) generalizes learned slots to novel sequence lengths, while (2) makes sequence order errors on novel sequence lengths. For (1), we delve into the network weights and identify a simple, interpretable circuit with modular components that implements slots. Our work bridges biological and artificial networks by demonstrating 1) more brain-like representations in artificial networks (which can emerge from standard architecture and training) may lead to improved generalization, and 2) how one can extract interpretable circuit structures from these brain-like artificial networks to serve as hypotheses for the brain.

1 INTRODUCTION

Sequences are everywhere, from the letters that make up a word to the actions that form your morning routine. A simple but fundamental sequential task is sequence working memory (SWM), i.e. the ability to remember a sequence of items for a brief amount of time. SWM is highly flexible, allowing recall of arbitrary sequences, but it comes at the cost of capacity, which is only about 4-7 items in humans (Miller, 1956; Cowan, 2001).

Many theories have been proposed on how SWM is represented in the brain, including the slot model, which posits that each item in the sequence is stored in a separate 'slot' (O'Reilly & Soto, 2001; O'Reilly & Frank, 2006; Whittington et al., 2024). For example, to remember "apple banana cherry", we can put "apple" into the first slot, "banana" in the second slot, and "cherry" in the third. These slots serve two functions: 1) storage, maintaining each item over time; 2) providing positional information — by putting "apple" in the first slot, we have bound "apple" to the first position. If we later need to retrieve the first item in the sequence, we know which slot to look into. On the neural level, each slot can be implemented as a *subspace* of neural activity (Xie et al., 2022). Putting "apple" into the first slot equates to neurons changing their firing activities so that the *projection*

of the total population activity into the first-item subspace corresponds to the value "apple". By maintaining this pattern of activity in the subspace, we can remember the first item in the sequence.

Recent animal experiments found evidence for such slot representations in the brain (Xie et al., 2022; Chen et al., 2024): monkeys were trained to view a sequence of locations light up on the screen and report it in the correct order after a few seconds of delay (Fig. 1a). Neural recordings in the frontal cortex, a region crucial for working memory and higher-level cognition, reveal separate, near-orthogonal subspaces in the population activity that encode each item in the sequence. The encoding in each subspace is stable across time, starting shortly after the item is shown on the screen and persisting to the delay period (Fig. 1b). We will call this representation *fixed slots*, emphasizing the information in each slot is *fixed* over time.

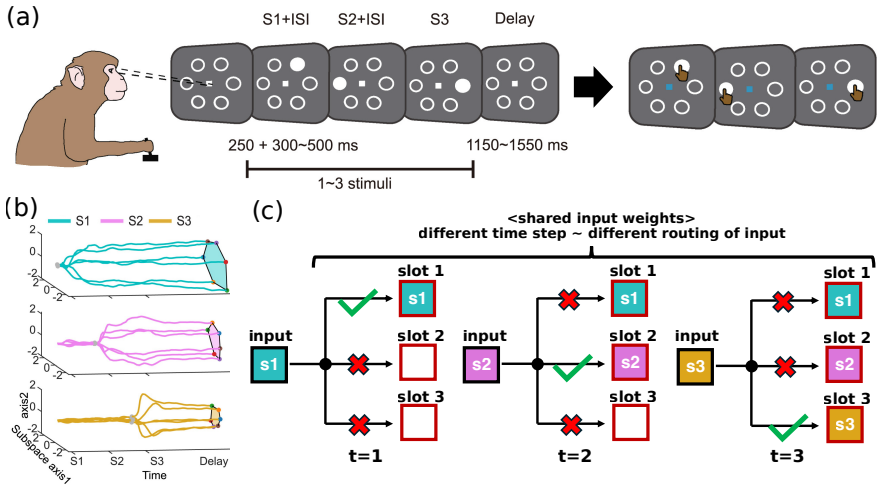


Figure 1: **SWM task and its slot-based neural representation**. **a**-**b** Adapted from Chen et al. (2024). **a**, Task structure. **b**, Trajectory of neural activity projected into identified subspaces ("slots"). **c**, Schematic of input pathway routing for the slot model.

However, we do not understand how slot representations are implemented at the neural circuit level. The sequential nature of SWM introduces a routing or control problem (Fig. 1c), as different items in the sequence share the same input pathway — they are only differentiated by their position in the sequence (i.e. the time they come in). For a network with fixed connection weights during runtime, each input goes through the same input weights, and yet they need to end up in orthogonal subspaces. A similar problem occurs during readout, where the same readout weights must access orthogonal subspaces on different time steps.

To better understand how slots might be implemented in neural circuits, and see if slots naturally emerge under SWM, here we train recurrent neural networks on a SWM task. We find two types of networks with different length generalization behaviors. Networks that learn the brain-like slot representations exhibit better generalization behavior in some ways. From these networks, we identify a simple, interpretable circuit that implements slots and solves the routing problem. We also find an alternative representation, *dynamic chain*, that generalizes less well but naturally arises in some networks.

2 Background

Neural network models of general working memory Various neural network models have been proposed for general working memory (WM). The WM content can be stored in neural activity or synapses. For activity-based memory, a common model is **attractor networks**, which use tuned recurrent connectivity to create fixed points of activity that sustain beyond a single neuron's time constant and thus maintain the memory even after the input has disappeared (Amit, 1989; Amit & Brunel, 1997; Compte et al., 2000). An alternative is to use a **feedforward chain**, in which the network activity continuously changes over time, progressing along a chain of states (Goldman,

2009; Ganguli et al., 2008; Murphy & Miller, 2009; Daie et al., 2023). The memory can be read out by summing across all states along the chain. Alternatively, the input can be stored in **short-term synaptic changes** that occur within a trial, without the need for continuous activity (Mongillo et al., 2008). Since the recent neural recordings during SWM (Xie et al., 2022; Chen et al., 2024) observe stable activity over delays (within the slot-like subspaces), and it is difficult to measure short-term synaptic changes *in vivo*, here we use networks with fixed weights during runtime and test how SWM may be encoded if only activity can be used for memory.

To store a sequence of stimuli, one solution is to take a model structure from above and make multiple copies, each dedicated to a single position in the sequence. The *fixed slots* solution we find in some RNNs is similar to this with multiple attractor networks. Alternatively, the same structure could be reused across stimuli — for example, a single ring attractor with multiple bumps of activity, each corresponding to a stimulus in the sequence. However, additional mechanisms are needed to distinguish the order of the stimuli and handle repeated stimuli. For the feedforward chain, multiple stimuli can traverse the same chain, with later stimuli in the sequence entering the chain later and thus occupying earlier states in the chain. The *dynamic chain* solution we find in some RNNs is in this form, and similar mechanisms have been reported in Karuvally et al. (2024); Keller et al. (2024). However, the readout process becomes more complex, as one cannot simply sum across all states along the chain to handle variable delays.

Past RNN models of SWM Several studies have used RNNs to model SWM. Botvinick & Plaut (2006) trained RNNs to recall a sequence of 1-9 inputs. The network did not learn fixed slots — rather, each item is represented by a vector that conjunctively encodes its identity and position in the sequence. Nearby positions correspond to more similar vectors, in contrast to the near-orthogonal subspaces observed in Xie et al. (2022). Cueva et al. (2021) trained RNNs to recall a sequence of two orientations, and found orthogonal subspaces that encode each. However, the network's encoding of the first input changes after the second input appears, differing from the fixed slot representation observed in Chen et al. (2024). Whittington et al. (2024) proposes (and finds in their trained RNNs) slots that shift their contents when new stimuli are presented, again differing from Chen et al. (2024). Lei et al. (2024) trained RNNs on the N-back task, another task that involves SWM, and found the network's memory representation changes over time, different from fixed slots. Our study differs from these past studies in: 1) we use a variety of sequence lengths and test whether networks can generalize to sequence lengths not seen during training, 2) we analyze how the trained network works on the weights level, 3) we explicitly compare with and fully replicate the fixed slot representation found in recent experiments (Xie et al., 2022; Chen et al., 2024).

Length generalization Length generalization has been studied in various tasks and models. Lake & Baroni (2018) trained RNNs on a compositional sequence translation task and found that networks struggle with compositional generalization, especially length generalization. Many recent studies tested transformers and found that while they show some length generalization, it tends to be limited and non-robust, and dependent on details such as the positional encoding and prompting techniques (Anil et al., 2022; Zhou et al., 2024; Kazemnejad et al., 2023; Zhou et al., 2023). In contrast, humans are capable of robust, systematic length generalization, motivating investigations into biological representations of sequences, which might better support length generalization.

3 Training RNNs on SWM

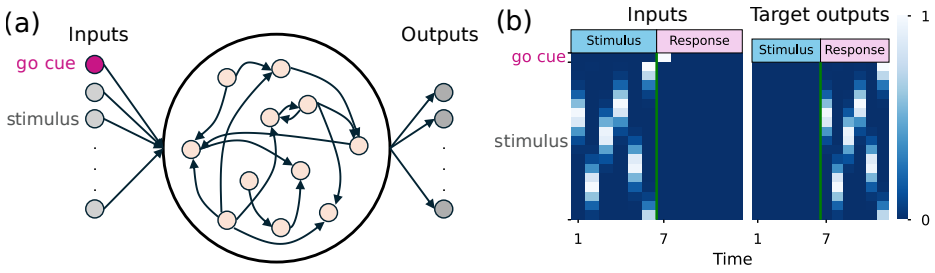


Figure 2: **Network and task setup**. **a**, RNN architecture. **b**, Example trial inputs and target outputs.

3.1 SWM TASK SETUP

During each trial, the network is presented with a sequence of stimuli that unfolds over time (Fig. 2b). The stimulus consists of a circular variable (e.g. an orientation), with 16 possible values equally spaced between $-\pi \sim \pi$. The stimulus is encoded across 17 input units, each with a preferred stimulus value equally spaced between $-\pi \sim \pi$ (Appendix C.1.1). The network is required to output zero until the go cue appears, at which point the network should output the stimuli in the sequence one-by-one in order. For simplicity, there is no delay period between the sequence presentation and the response period. However, because the network is trained on a mix of sequence lengths, and the network does not know the length of the sequence until the go cue appears, the network effectively needs to remember each stimulus for a variable amount of time after it appears, thus partially mimicking a variable delay period in standard working memory tasks. We train on sequences of lengths 1-4 and 6-9, and use lengths 5 and 10 to test length generalization. In addition, for each training length, we randomly sample a small set of sequences to be held out from training (Appendix C.1.1).

3.2 RNN ARCHITECTURE AND TRAINING

We use RNNs with a single recurrent layer (Fig. 2a, Appendix C.1.2). We explore two losses: 1) mean squared error (MSE) loss and 2) a weighted version of MSE (Eq. (4)), where the error of each output unit is weighted by the circular distance between its tuning center and the true stimulus, thus incorporating knowledge of the output units' circular structure. The recurrent weights are initialized as random orthogonal matrices (Saxe et al., 2014) (standard Kaiming initialization yields similar results, see Appendix C.1.4), with no other structure built in. We vary the activation function (ReLU or tanh), output nonlinearity (linear or a scaled tanh), and the network size (100-400 neurons). Overall, we train $N = 72$ networks, with 3 networks (different random seeds) for each hyperparameter combination.

4 LENGTH GENERALIZATION BEHAVIOR: TWO TYPES OF NETWORKS

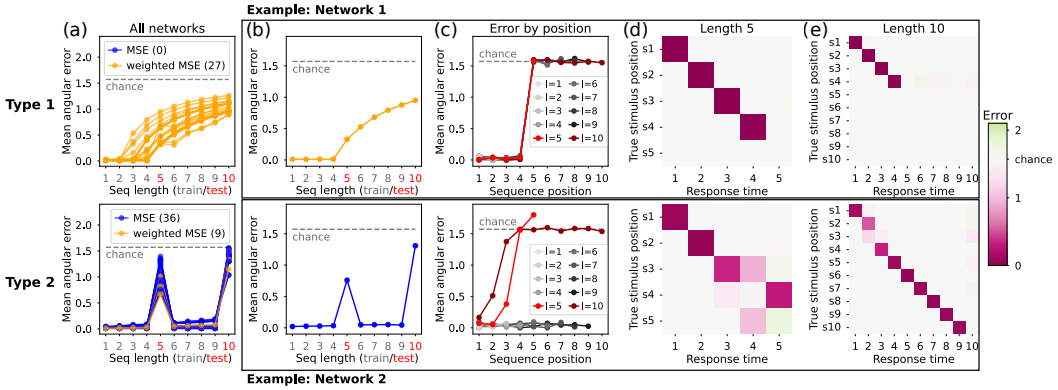


Figure 3: **Behavior of two types of networks**. **a**, Error on test sequences of varying lengths. Error is averaged across all positions within a sequence. Each line denotes a single network (72 in total). Color indicates the loss function, number in parentheses is the number of networks — note no type 1 networks use MSE loss (blue). **b**-**e**, Two example networks. **b**, Same as **a**. **c**, Error for each position in the sequence, for each sequence length (color). **d**-**e**, Error on length-5 (**d**) and length-10 (**e**) sequences, when comparing the output at each response time point to stimuli at all positions in the input sequence. Diagonal values correspond to actual task errors as shown in **c**.

Fig. 3a shows the performance of all trained RNNs as a function of the sequence length (see Appendix C.2.1 for alternative metrics). For each sequence length, we average the error across all positions in the sequence. We find two types of networks with distinct length generalization behaviors — **type 1** networks show a smooth degradation of error across sequence length, irrespective of whether the sequence length is present in training or not. These networks appear to *underfit*, with

sizable errors on training lengths above 2-4, but their errors on the heldout lengths (5 and 10) do not greatly differ from similar training lengths. In contrast, **type 2** networks appear to *overfit* on the training lengths, achieving very low errors on all training lengths but producing large spikes in error for the two heldout lengths. This is surprising as these networks can remember long sequences of up to 9 stimuli and yet appear to struggle for a sequence of 5. However, these networks are not simply memorizing *specific* training sequences, as these networks successfully generalize to sequences held out from training as long as their lengths were seen during training (Fig. S3).

Interestingly, the network type is strongly correlated with the loss function — all type 1 networks are trained with the weighted MSE loss, while all networks trained with the standard MSE loss are type 2 (Fig. 3a). Additional networks reveal correlations with the learning rate as well (Fig. S4).

To further characterize network behaviors, we break down the error by sequence position (Fig. 3c). We call the first stimulus in the sequence s1, the second s2 etc. For simplicity, we pick a single example network from each type and focus on them for the rest of the paper — we call the type 1 example **Network 1**, and the type 2 example **Network 2** (Fig. 3b) (see more networks in Fig. S5).

Network 1 successfully recalls the first four items (s1-s4) across all sequence lengths, and is at chance for all later items (Fig. 3c, top) – in fact, the network no longer outputs anything beyond the fourth item. The network can generalize this ability of recalling s1-s4 to the test lengths of 5 and 10. This is consistent with a model that has learned four "slots" for the input sequence.

Network 2, on the other hand, remembers all positions in all training lengths perfectly, but can only recall the first 2-3 items for the test lengths (Fig. 3c, bottom). Further analysis shows that Network 2 can actually remember almost all items in the test-length sequences, it just outputs them in the wrong order — e.g. on length-10 sequences, the network skips s3 and directly outputs s4 when it should be recalling the third item, and it continues to be one position early for the rest of the response period (Fig. 3d-e, bottom). This suggests the control process of selecting which item to output is distorted, so items become recalled at the wrong time.

5 Two types of representations

5.1 Both networks replicate orthogonal, low-dimensional subspaces

Neural recordings have shown that each item in the sequence is encoded in a separate low-dimensional subspace of population activity (Fig. 4a, top) (Xie et al., 2022; Chen et al., 2024). The experiments used circular stimuli (six locations on a circular ring; see Fig. 1a), and within each subspace, the representation also forms a corresponding ring. These subspaces are near-orthogonal to each other (Fig. 4b, top), thus avoiding interference between different items in the sequence.

Similarly, for both Network 1 and Network 2, we find low-dimensional subspaces that encode the value of each stimulus (see Appendix C.3.1 for methods), and within each subspace, the circular structure of the stimulus is roughly preserved (Fig. 4a, middle and bottom). These subspaces are near-orthogonal (Fig. 4b, middle and bottom). At this point, it appears both networks can replicate brain-like representations in the SWM task, even though their behaviors are drastically different.

5.2 Different dynamics: only Network 1 fully replicates fixed slots

Next, we analyze the dynamics of the representation over time. Fig. 4c shows the temporal trajectory of the representation in each of the stimulus-encoding subspaces. In the experimental data from Chen et al. (2024) (Fig. 4c, top), shortly after each stimulus is presented, the stimulus becomes encoded in its corresponding subspace, and this encoding remains **fixed** throughout the rest of the stimulus presentation and the delay period, consistent with the fixed slots model.

Repeating the same analysis in our networks, we now find a difference between Network 1 and Network 2. Network 1 replicates the trajectories observed in experimental data (Fig. 4c, middle). Network 2 has similar stable trajectories for s1-s2, but the trajectories are less stable for s3-s4 (Fig. 4c, bottom) – the stimulus encoding does not appear in the identified subspace until towards the end of the sequence presentation (these subspaces are identified to encode the stimulus at the end the sequence presentation). This suggests the stimulus is only *transiently* encoded in each subspace.

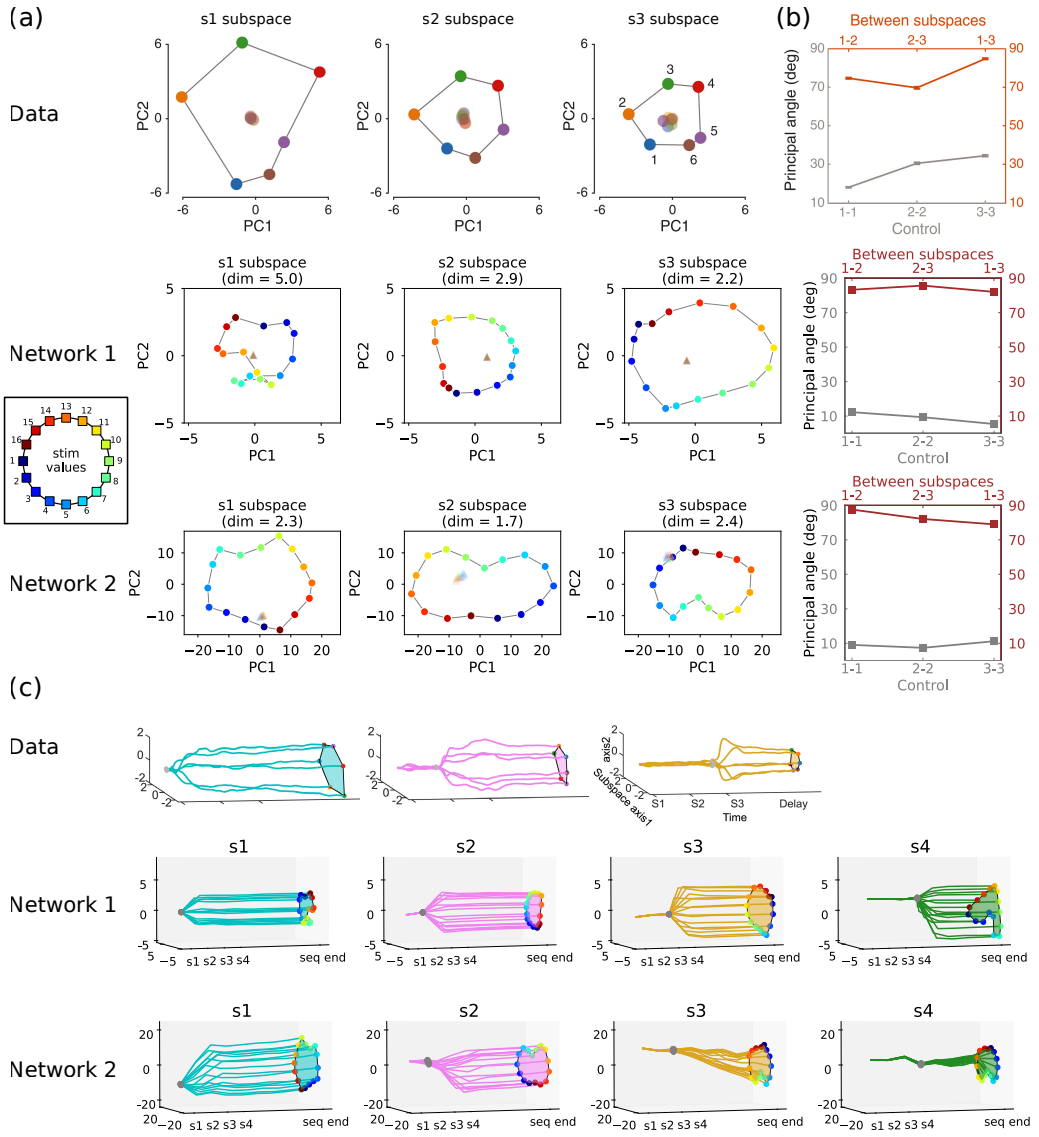


Figure 4: **Learned representation of the two example networks, compared with neural data**. For each sub-figure, top row=neural data (from Xie et al. (2022) (**a**-**b**) and Chen et al. (2024) (**c**), see original captions); bottom two rows=example networks (the following only describes these), using length-9 sequences. **a**, Circles show activity at the last time step before the response period, projected into the top 2 PCs that encode the target stimulus at this time step (Appendix C.3.1). Each dot is the average projection across sequences where the target stimulus (e.g. s1) equals a specific value (color). **Inset** is how the 16 stimuli values are mapped to color. Semi-transparent triangles show the activity when the target stimulus is presented. Dimensionality ("dim") is the participation ratio. **b**, Principal angle between the identified subspaces (red). Grey is control (Appendix C.3.2). **c**, Trajectory of network activity in the identified subspaces over time. Each line traces the projection of activity over time for a particular stimulus value, averaged across sequences. Colored circles=the last time step before response ("seq end"), grey=when the target stimulus is presented.

5.3 Alternative to fixed slots: dynamic chain

In contrast to fixed slots, an alternative representation is to have a *dynamic chain* of subspaces (Fig. 5a, right). On each time step, the stimulus encoded in a subspace will be passed on to the next subspace in the chain. In this way, each subspace encodes different stimuli on different time steps — specifically, each subspace encodes the i^{th} *last* stimulus in the sequence. This can produce the

phenomenon in Network 2 where a particular stimulus is only transiently encoded in each subspace. Although this may seem more complicated than the "fixed slots" scheme, this might be a more natural solution for RNNs (Appendix B).

To further test the fixed slots and dynamic chain hypotheses for the trained RNNs, we compute the cross-decoding error across time and position (Appendix C.3.3). With fixed slots, the same subspace encodes a fixed stimulus over time, so e.g. for the decoder trained to decode s1 using activity at $t = 9$, we should get low decoding error when using it to decode s1 on all earlier time steps (producing a vertical "column", Fig. 5b left). In contrast, under the dynamic chain regime, the same subspace encodes the i^{th} last stimulus over time, so e.g. the decoder trained on s9 at $t = 9$ should have low decoding error when used on s_k at $t = k$ (producing a "diagonal", Fig. 5b right).

Fig. 5c and Fig. S6 show the results for Network 1 and Network 2. As we expect, Network 1 shows the vertical column pattern for s1-s4, providing further evidence that Network 1 uses the fixed slot representation for the first four stimuli. In contrast, in Network 2, s1-s2 show the vertical column pattern consistent with fixed slots, but the column largely disappears for s3, and later stimuli show the diagonal pattern consistent with the dynamic chain regime. This is consistent with the temporal trajectory patterns in Fig. 4c, where there appears to be a qualitative change from s1-s2 to s3-s9, with the former using a fixed-slots strategy, and the latter using a dynamic-chain strategy. This switch in representation is consistent with the length generalization behavior of Network 2, where it can successfully recall the first 2-3 stimuli on test-length sequences but fail on the rest. In fact, across type 2 networks, we find that they use a mix of fixed-slots and dynamic-chain strategies for different stimuli, and when the switch occurs in the *representation* is correlated with when length generalization *behavior* begins to fail (Fig. S7-8).

To summarize, at the representation level, we find that both types of networks learn near-orthogonal low-dimensional subspaces that encode each stimulus in the sequence. However, the underlying dynamics of the representation forms two types. The brain-like fixed-slots dynamics corresponds to successful length generalization (as observed in type 1 networks and initial stimuli for some type 2 networks), while dynamic-chain representations correspond to failed length generalization (as observed for later stimuli in type 2 networks).

6 Circuit mechanism for slot representation

6.1 Circuit reconstruction from tracing learned weights

We have shown Network 1 learns the brain-like fixed slots representation. Next, we reconstruct how its learned weights give rise to such representations and solve the task. Since the network is relatively simple (a single hidden layer of 400 units), we directly trace from the learned weights and look for neurons with large weight magnitudes to our target (Fig. 6a and Appendix C.4.1). We start from the output and identify neurons that have large output weights (we call them "readout neurons"). This identifies distinct groups of neurons, each only active for a single readout time step, in contrast to shared readout neurons active across all readout time steps. Each readout time step maps to a distinct group of neurons, not just a particular direction in the population activity space (i.e. a distributed representation that involves many neurons). This might be due to the ReLU activation function, which can encourage modularity and sparse representations (Driscoll et al., 2024; Whittington et al., 2023; Johnston & Fusi, 2024). Similarly, we can trace back from readout neurons to neurons that provide input to them, which leads us to *slot neurons*, and tracing back from them leads to *sensory neurons*. After identifying these groups of neurons, we examine their representations and look at the connection weights between them to reconstruct the simplified circuit.

6.2 Specialized populations form motifs to control information flow

The reconstructed circuit (Fig. 6b) consists of three classes of neurons: **sensory**, **slot**, **readout**. Within each class, we find four distinct groups that correspond to s1-s4 — each group only encodes a single stimulus, thus creating orthogonal representations.

Overall flow: sensory → slot → readout. When a stimulus in the sequence is presented, it will first activate the corresponding sensory neurons, which are transiently active and pass on the information to the corresponding slot neurons. Slot neurons have self-recurrence that produces persistent

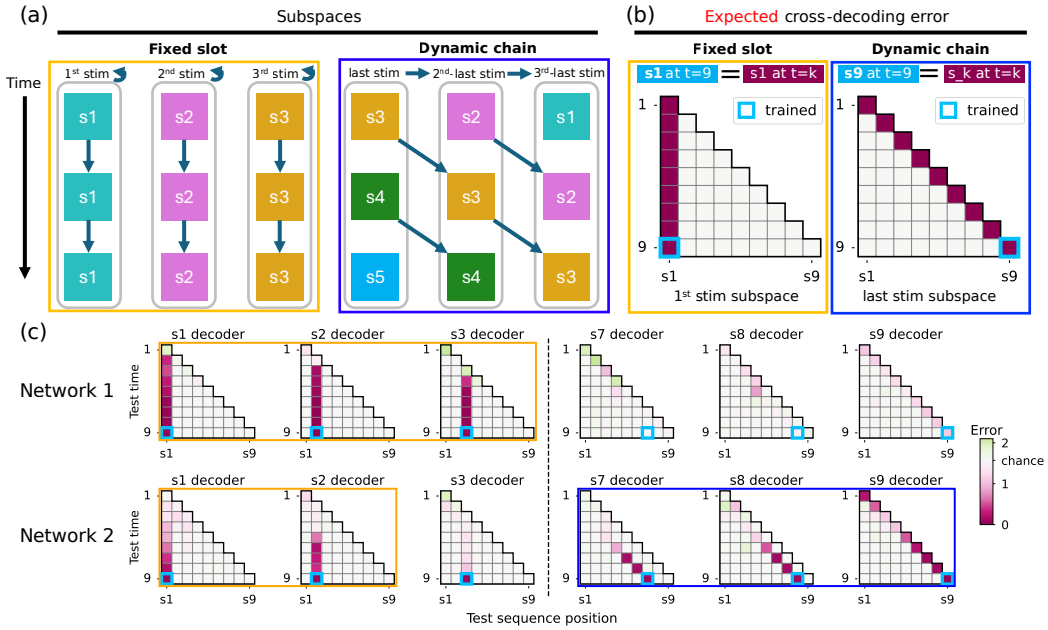


Figure 5: **Two types of representational dynamics**. **a**, Schematic of two types of dynamics. Each column (grey box) is a single subspace. Color and symbol indicate what is encoded in the subspace at a given time step. Arrows indicate which subspace the stimulus will be encoded in on the next time step. **b**-**c**, Cross-decoding analysis (Appendix C.3.3). Light blue box indicates the decoder's error on the time step that it is trained on, against the stimulus it is trained for. **b**, Expected under the two types of dynamics (left: s1 decoder; right: s9 decoder); **c**, observed in the example networks. Orange box: putative fixed slot; dark blue box: putative dynamic chain.

activity to remember the stimulus value. Since the stimuli are circular, we posit each slot uses Mexican hat-like recurrent connectivity to form a continuous ring attractor that stores the stimulus value (Ben-Yishai et al., 1995; Skaggs et al., 1994; Zhang, 1996). When it is time to output this stimulus, slot neurons pass the information to the corresponding readout neurons, which produce the network output, and slot neurons are then silenced to clear out information about the already-readout stimulus. Fig. 6c illustrates the information encoded in each population over time in this process.

As we explained before, a huge challenge in SWM is to control the flow of information. In this circuit, the problem is solved by carefully placed connections between different groups of neurons. We explain the full set of connections in Appendix A, but here we focus on a few examples.

Input pathway motif 1: slot inhibits its source sensory neurons. For the input pathway, all sensory neurons have large input weights, so on their own these neurons will be active on every time step and just encode the currently presented stimulus. But in the full circuit, *sensory 1* neurons are only transiently active when s1 is presented, and remain silent for all future inputs (Fig. 6c). This can be achieved by **inhibition from *slot 1* neurons to *sensory 1* neurons**: *slot 1* becomes active after s1 is presented, so on future time points, the persistent activity in *slot 1* will suppress activity in *sensory 1* (Fig. 6d, top left). This forms a simple gating motif so that once a slot is "filled", it will shut off its input source so that future inputs will not overwrite the content in the slot.

Input pathway motif 2: earlier sensory neurons inhibit slot. Motif 1 solves the problem of preventing future stimuli from entering the slot (though see Appendix A), but it does not address why earlier stimuli do not enter the slot. This can be solved by a second motif, where earlier sensory neurons inhibit the current slot. For example, ***sensory 1* neurons inhibit *slot 2*** (Fig. 6d, bottom left) – therefore, when s1 is presented, even though *sensory 2* neurons will also receive s1 and provide that as input to *slot 2*, the inhibition from *sensory 1* neurons will cancel out this input, leaving *slot 2* unfilled. However, when s2 is presented next, *sensory 1* neurons are silent because of motif 1, so now *sensory 2* neurons have no impedance in passing s2 information to *slot 2*.

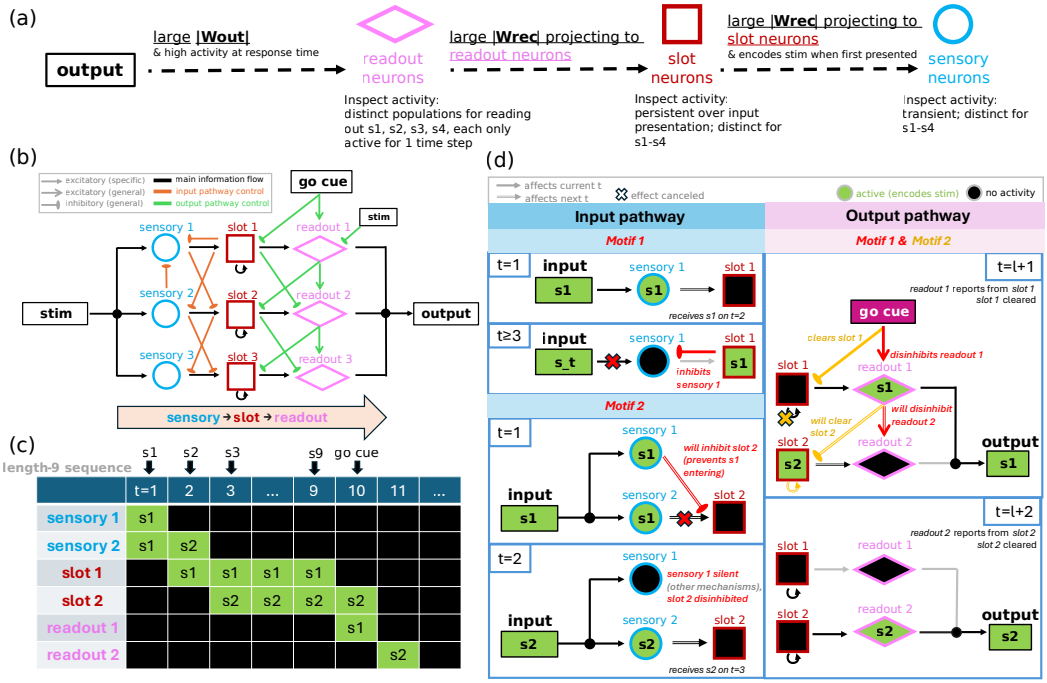


Figure 6: **Hypothesized circuit mechanism based on Network 1**. **a**, Process for identifying distinct neural populations (see Appendix C.4.1). **b**, Schematic of reconstructed neural circuit for s1-s3 (see Appendix A). Each colored shape is a specific group of neurons. **c**, What each neural population encodes at each time point for a length-9 sequence. Black indicates no activity. For simplicity, only neurons for s1-s2 are shown (same for **d**). **d**, Schematic of example motifs. Because the network is discrete-time, recurrent connections act on target neurons at time $t+1$ based on source neuron activities at time t — we plot them either at $t+1$ (solid arrow) or t (double-line arrow). Red cross indicates the effect of a connection is nulled due to inhibitory inputs from other connections.

Although these two input pathway motifs do not constitute the entire complexity of the trained RNN, they are sufficient to correctly control the flow of input information for the task and, in fact, will work if a variable delay period is added (Appendix A).

Output pathway motif: chain excitation in readout neurons. The output pathway control uses a chain of excitatory connections from the go cue to successive readout neurons (Fig. 6d right, motif 1). When the go cue appears, excitatory connections from the go cue to *readout 1* activate them, allowing *readout 1* neurons to receive inputs from *slot 1* and report them as the output. Next, *readout 1* neurons excite *readout 2* neurons, effectively serving as the go cue for *readout 2*, so that on the next time step *readout 2* neurons will be active and report the content of *slot 2*.

The circuit also has a secondary motif (Fig. 6d right, motif 2), where the go cue inhibits *slot 1*, so that on the next time step, *slot 1* will be cleared out, thus preventing *readout 1* from reporting s1 on future time steps. Similarly, *readout 1* inhibits *slot 2* to clear it after s2 is read out. We note that this slot clearing motif is not necessary for the circuit to work (Appendix A).

6.3 Evidence of circuit mechanism in trained RNN

To check if our reconstructed circuit faithfully describes the trained network, we verify that the representation of each subpopulation is what we expect (Fig. 6c). Fig. 7a shows that each group of neurons is only active on the time steps that we expect them to, and Fig. 7b shows that on these active time steps, we can only decode the stimulus that we expect (also see Fig. S9).

Next, we perturb these groups of neurons to see if they are important for the network (Appendix C.4.3). Although each group is only about 5% of the total network, perturbing them on a single time

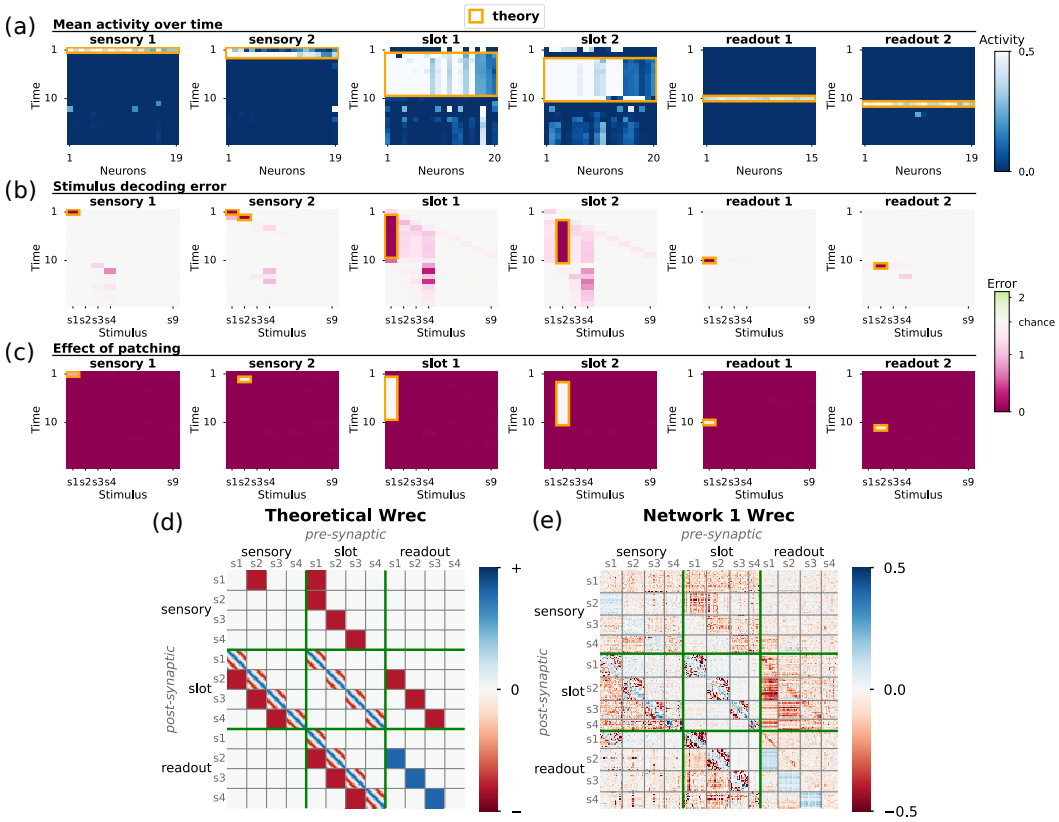


Figure 7: **Evidence of hypothesized circuit in Network 1**. **a**, Mean activity of individual neurons over time on length-9 sequences. Orange=time steps where neurons are active in the hypothesized circuit. **b**, Stimulus decoding from each group of neurons. A separate decoder is trained for each time step and each sequence position. Orange=(time, position) decodable in the hypothesized circuit. **c**, Effect of perturbing each group on task performance (Appendix C.4.3). Each matrix cell is perturbing at a specific time step and evaluating the change in the recall performance of a particular stimulus. **d**-**e**, Recurrent weight matrix between specific neural populations, theoretical circuit (**d**) and Network 1 (**e**). Each row (post-synaptic) and column (pre-synaptic) is a single neuron. Neurons are ordered by their group, and within each group by their preferred stimulus value.

step greatly impairs task performance (Fig. 7c). The results fit our expectations — the effect on the final network output is position-selective and only occurs when perturbed on specific time steps.

Finally, we compare the weights between the reconstructed circuit and the trained RNN (Fig. 7d-e and Fig. S10). We find a qualitative match, especially 1) the Mexican-hat like recurrence within each slot, and the connections from the corresponding sensory to slot, and slot to readout (these connections also pass on circular stimuli values), 2) the excitatory chain between successive readout neurons. For inhibitory connections, overall, the ones we expect to be inhibitory are indeed negative in the trained RNN, but there are more inhibitory connections in the trained RNN – these are likely other mechanisms to implement the appropriate control and the redundancy may ensure robustness.

7 Conclusion

We find that brain-like slot representations can naturally emerge in RNNs trained on SWM, and these representations enable better generalization to novel sequence lengths in some ways. We identify a simple circuit mechanism that implements slots and their proper control, though it is rigid and task-specific. Future work could explore more flexible control mechanisms, incorporate diverse input types, and use short-term synaptic plasticity to enhance flexibility.

8 ACKNOWLEDGMENTS

This research was supported in part by the Kavli Foundation and by the Gatsby Charitable Foundation (GAT3708).

REFERENCES

Larry F. Abbott, Kanaka Rajan, and Haim Sompolinsky. Interactions between intrinsic and stimulus-evoked activity in recurrent neural networks. In *The Dynamic Brain: An Exploration of Neuronal Variability and Its Functional Significance*. Oxford University Press, 2011. ISBN 9780195393798. URL https://doi.org/10.1093/acprof:oso/9780195393798.003.0004.

Daniel J Amit. *Modeling brain function: The world of attractor neural networks*. Cambridge university press, 1989.

Daniel J Amit and Nicolas Brunel. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cerebral cortex (New York, NY: 1991)*, 7(3):237–252, 1997.

Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=zSkYVeX7bC4.

R Ben-Yishai, R L Bar-Or, and H Sompolinsky. Theory of orientation tuning in visual cortex. *Proceedings of the National Academy of Sciences*, 92(9):3844–3848, 1995. doi: 10.1073/pnas.92.9.3844. URL https://www.pnas.org/doi/abs/10.1073/pnas.92.9.3844.

Matthew M. Botvinick and David C. Plaut. Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, 113(2):201–233, 2006. ISSN 1939-1471, 0033-295X. doi: 10.1037/0033-295X.113.2.201. URL https://doi.apa.org/doi/10.1037/0033-295X.113.2.201.

Jingwen Chen, Cong Zhang, Peiyao Hu, Bin Min, and Liping Wang. Flexible control of sequence working memory in the macaque frontal cortex. *Neuron*, 112(20):3502–3514.e6, October 2024. ISSN 0896-6273. doi: 10.1016/j.neuron.2024.07.024. URL https://doi.org/10.1016/j.neuron.2024.07.024. Publisher: Elsevier.

Albert Compte, Nicolas Brunel, Patricia S. Goldman-Rakic, and Xiao-Jing Wang. Synaptic mechanisms and network dynamics underlying spatial working memory in a cortical network model. *Cerebral Cortex*, 10(9):910–923, 09 2000. ISSN 1047-3211. doi: 10.1093/cercor/10.9.910. URL https://doi.org/10.1093/cercor/10.9.910.

Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114, 2001. doi: 10.1017/S0140525X01003922.

Christopher J. Cueva, Adel Ardalan, Misha Tsodyks, and Ning Qian. Recurrent neural network models for working memory of continuous variables: activity manifolds, connectivity patterns, and dynamic codes, 2021. URL https://arxiv.org/abs/2111.01275.

Kayvon Daie, Lorenzo Fontolan, Shaul Druckmann, and Karel Svoboda. Feedforward amplification in recurrent networks underlies paradoxical neural coding. *bioRxiv*, 2023. doi: 10.1101/2023.08.04.552026. URL https://www.biorxiv.org/content/early/2023/08/07/2023.08.04.552026.

Laura N. Driscoll, Krishna Shenoy, and David Sussillo. Flexible multitask computation in recurrent networks utilizes shared dynamical motifs. *Nature Neuroscience*, 27(7):1349–1363, July 2024. ISSN 1546-1726. doi: 10.1038/s41593-024-01668-6. URL https://doi.org/10.1038/s41593-024-01668-6.

Surya Ganguli, Dongsung Huh, and Haim Sompolinsky. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, 105(48):18970–18975, 2008. doi: 10.1073/pnas.0804451105. URL https://www.pnas.org/doi/abs/10.1073/pnas.0804451105.

Mark S. Goldman. Memory without Feedback in a Neural Network. *Neuron*, 61(4):621–634, February 2009. ISSN 08966273. doi: 10.1016/j.neuron.2008.12.012. URL https://linkinghub.elsevier.com/retrieve/pii/S0896627308010830.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/abs/1502.01852.

W. Jeffrey Johnston and Stefano Fusi. Modular representations emerge in neural networks trained to perform context-dependent tasks. *bioRxiv*, 2024. doi: 10.1101/2024.09.30.615925. URL https://www.biorxiv.org/content/early/2024/10/11/2024.09.30.615925.

Arjun Karuvally, Terrence J. Sejnowski, and Hava T. Siegelmann. Hidden traveling waves bind working memory variables in recurrent neural networks. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 24892–24928. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/4e85362c02172c0c6567ce593122d31c-Paper-Conference.pdf.

T. Anderson Keller, Lyle Muller, Terrence Sejnowski, and Max Welling. Traveling waves encode the recent past and enhance sequence learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=p4S5Z6Sah4.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2873–2882. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/lake18a.html.

Xiaoxuan Lei, Takuya Ito, and Pouya Bashivan. Geometry of naturalistic object representations in recurrent neural network models of working memory. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 100604–100629. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/b63dba40303ae7adb3e2b16e5dd5fd0b-Paper-Conference.pdf.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023. URL https://arxiv.org/abs/2202.05262.

George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956. ISSN 1939-1471(Electronic),0033-295X(Print). doi: 10.1037/h0043158. Place: US Publisher: American Psychological Association.

Gianluigi Mongillo, Omri Barak, and Misha Tsodyks. Synaptic theory of working memory. *Science*, 319(5869):1543–1546, 2008. doi: 10.1126/science.1150769. URL https://www.science.org/doi/abs/10.1126/science.1150769.

Brendan K. Murphy and Kenneth D. Miller. Balanced amplification: A new mechanism of selective amplification of neural activity patterns. *Neuron*, 61(4):635–648, 2009. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2009.02.005. URL https://www.sciencedirect.com/science/article/pii/S0896627309001287.

Randall O'Reilly and R. Soto. A model of the phonological loop: Generalization and binding. In T. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/312351bff07989769097660a56395065-Paper.pdf.

Randall C. O'Reilly and Michael J. Frank. Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, 18(2):283–328, 02 2006. ISSN 0899-7667. doi: 10.1162/089976606775093909. URL https://doi.org/10.1162/089976606775093909.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014. URL https://arxiv.org/abs/1312.6120.

William Skaggs, James Knierim, Hemant Kudrimoti, and Bruce McNaughton. A model of the neural basis of the rat's sense of direction. In G. Tesauro, D. Touretzky, and T. Leen (eds.), *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994. URL https://proceedings.neurips.cc/paper_files/paper/1994/file/024d7f84fff11dd7e8d9c510137a2381-Paper.pdf.

Zhenghe Tian, Jingwen Chen, Cong Zhang, Bin Min, Bo Xu, and Liping Wang. Mental programming of spatial sequences in working memory in the macaque frontal cortex. *Science*, 385(6716): eadp6091, 2024. doi: 10.1126/science.adp6091. URL https://www.science.org/doi/abs/10.1126/science.adp6091.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 12388–12401. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/92650b2e92217715fe312e6fa7b90d82-Paper.pdf.

James C. R. Whittington, Will Dorrell, Surya Ganguli, and Timothy E. J. Behrens. Disentanglement with biological constraints: A theory of functional cell types, 2023. URL https://arxiv.org/abs/2210.01768.

James C.R. Whittington, William Dorrell, Timothy E.J. Behrens, Surya Ganguli, and Mohamady El-Gaby. A tale of two algorithms: Structured slots explain prefrontal sequence memory and are unified with hippocampal cognitive maps. *Neuron*, pp. S0896627324007657, November 2024. ISSN 08966273. doi: 10.1016/j.neuron.2024.10.017. URL https://linkinghub.elsevier.com/retrieve/pii/S0896627324007657.

Yang Xie, Peiyao Hu, Junru Li, Jingwen Chen, Weibin Song, Xiao-Jing Wang, Tianming Yang, Stanislas Dehaene, Shiming Tang, Bin Min, and Liping Wang. Geometry of sequence working memory in macaque prefrontal cortex. *Science*, 375(6581):632–639, 2022. doi: 10.1126/science.abm0204. URL https://www.science.org/doi/abs/10.1126/science.abm0204.

K Zhang. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *Journal of Neuroscience*, 16(6):2112–2126, 1996. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.16-06-02112.1996. URL https://www.jneurosci.org/content/16/6/2112.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization, 2023. URL https://arxiv.org/abs/2310.16028.

Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly, 2024. URL https://arxiv.org/abs/2402.09371.

A Detailed discussion of the reconstructed circuit

A.1 All connections in the reconstructed circuit

Here we describe the full reconstructed circuit in Fig. 6b (we omit s4 for simplicity — all connections involving s4 are similar to s3). See the main text for the description of each neuron group and the general flow of information from the input stimulus to *sensory* to *slot* to *readout* and to the final output.

First we note that each connection could be *general* or *specific* (as indicated by the arrow type in Fig. 6b): a *specific* connection means the connections between the pre-synaptic neurons and the post-synaptic neurons depend on the individual neuron identity — in particular, these connections pass on stimulus information, so, for example, a *sensory 1* neuron that is tuned for the stimulus value $-\pi$ should preferentially be connected to *slot 1* neurons that are also tuned for $-\pi$. In fact, we hypothesize these connections to be Mexican hat-like, so that the connection weight between two neurons is negatively correlated with the magnitude of the circular distance in their preferred stimulus values (this forms the diagonal stripes in Fig. 7d). In contrast, *general* connections are across all neurons involved and do not depend on the individual neuron identity — for example, we would expect all or most pairwise connections from *readout 1* neurons to *readout 2* neurons to be excitatory.

The two input pathway motifs we describe in the main text are present in the full reconstructed circuit. However, because the network runs in discrete time, these two motifs alone are insufficient for stopping the wrong stimulus from entering the wrong neurons (in particular, the problem is due to motif 1) — for example, *sensory 1* is inhibited by *slot 1*, which should stop *sensory 1* from encoding future stimuli (and passing them into *slot 1*). However, because of the discrete-time nature, slot 1 only becomes active on $t = 2$, so its outgoing connections can only take effect starting on $t = 3$, which means *sensory 1* has no source of inhibition on $t = 2$ and would thus encode s2 on that time step. This is why in the reconstructed circuit, we include an additional inhibitory connection from *sensory 2* to *sensory 1* — this connection does not interfere with *sensory 1*'s encoding of s1 on $t = 1$ again because of the discrete-time nature (although *sensory 2* is also active on $t = 1$, its inhibition on *sensory 1* doesn't take effect until $t = 2$). Similarly, applying motif 1 for s2-s4 (e.g. inhibition of *sensory 2* by *slot 2*) would not fully work, so instead in the reconstructed circuit they are *replaced* by inhibition from the earlier sensory neurons (so *sensory* $i + 1$ is inhibited by *slot* i for $i \geq 1$). We note that if the network is not discrete-time, or if there is at least one time step in between each stimulus presentation, then the two input pathway motifs are perfectly sufficient.

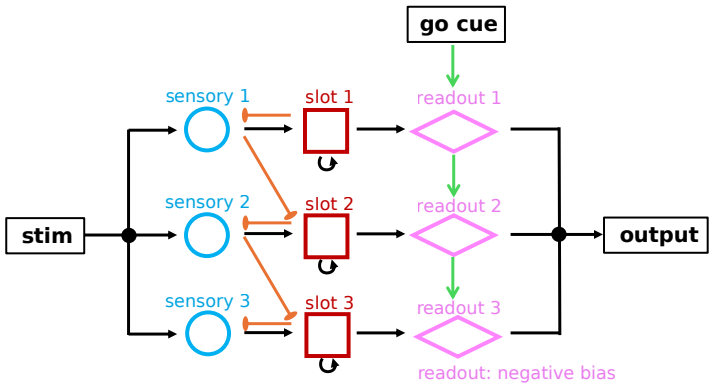
For the output pathway, in the main text, we omit how the readout neurons are kept silent until the appropriate readout time step. Since they get direct inputs from slot neurons, which are active for various time steps during stimulus presentation, they would need a source of inhibition to stay silent until the correct readout time step. The simplest solution is to have a large negative bias (i.e. high activation threshold) for all the readout neurons, so that only strong activation (provided by the chain excitation along the readout neurons) will activate them. This also makes the slot-clearing motif we discuss in the main text (output pathway motif 2) unnecessary — because the go cue is transient, each group of readout neurons only gets transient excitation from the chain of excitation, so even on future time steps when the slot is active and sends information to readout neurons, the negative bias will clamp them from reporting information from the slot.

However, we do not observe strong negative biases for readout neurons in Network 1 — instead, additional network connections actively suppress the readout neurons. Therefore, in the reconstructed circuit, we include also inhibition from the stimulus-encoding input units to *readout 1* (this keeps *readout 1* silent throughout the stimulus presentation period), as well as inhibition from *slot* i to *readout* $i+1$ (since *slot* i is filled earlier than *slot* $i+1$, this ensures that when *slot* $i+1$ is providing input to *readout* $i+1$ on earlier time steps before the actual readout, *slot* i will provide the inhibition to prevent activity in *readout* $i + 1$).

A.2 Simplified circuit

The additional connections we explain above help the network solve the task in the discrete-time version that it is trained on, but we note that a **simplified circuit** (Fig. S1), which only consists of three motifs (all the example motifs we discuss in the main text except for output pathway motif

2, the slot-clearing motif), can perfectly solve the task if there is at least one time step in between each stimulus presentation, and in fact will work if there are variable delays added at the end of the sequence presentation, or even in between individual stimuli presentations. This simplified circuit would be our current best hypothesis for how the brain might implement slot dynamics.



Supplementary Figure 1: **Simplified circuit**. Only includes input pathway motif 1 and 2, and output pathway motif 1, plus large negative bias to readout neurons.

A.3 Further discussion

Although the reconstructed circuit is in terms of distinct groups of neurons (as we found in Network 1), they may be implemented as specific directions or subspaces in population activity space. In fact, from existing experimental data, we already know that about 30% of the recorded neurons participate in each slot subspace (Chen et al., 2024), in contrast to about 5% in Network 1 (though this is partly a consequence of the artificial thresholds we chose to identify these groups of neurons). This is a large discrepancy between Network 1 and the real brain, but we think the general motifs we identify here can potentially transfer to more distributed representations. A challenge for distributed representations is to form generic excitation or inhibition between different subspaces — it is more straightforward for a group of neurons to e.g. inhibit another group, since the activation function nonlinearity is applied on the neuron level.

Chen et al. (2024) reports finding a group of sensory neurons that are shared across stimuli, always encoding the currently presented stimulus (e.g. it transiently encodes s1 after s1 appears, and transiently encodes s2 after s2 appears). This may appear at odds with our circuit, but in fact these neurons could just be e.g. the input units to our networks. Our prediction would be that these shared sensory neurons do not directly provide synaptic inputs to the slot neurons (though see the next point).

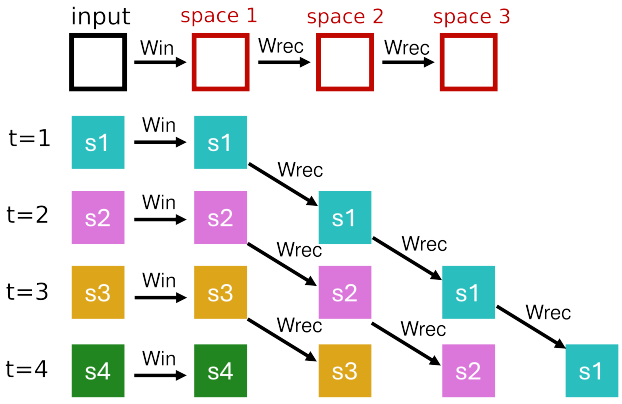
So far we assume that synaptic weights are fixed after training, but the real brain has many short-term plasticity mechanisms. Therefore, some of the motifs we identify may be implemented directly on the synaptic level, bypassing the need of intermediate neurons — for example, for input pathway motif 1, the key idea is for each slot to shut off its input path once it is filled. In our circuit, this is implemented by having *sensory 1*, an intermediate group of neurons that relay the input, be inhibited by *slot 1*. Alternatively, sensory input could directly project to *slot 1* neurons, but these synapses may depress if *slot 1* is active. We note that in order for this to work, local synaptic depression may be insufficient, as we want any generic activity in a slot to inhibit all future inputs to the slot (for example, if the sequence is *A B*, the first item *A* will only excite *slot 1* neurons that prefer *A* — if only those synapses are depressed, then when the second item *B* comes in, *slot 1* neurons that prefer *B* may still be activated). It is harder to imagine how such synaptic mechanisms could work if we have distributed subspaces instead of distinct neuron groups, as synaptic plasticity also applies on the neuron level.

More fundamentally, our reconstructed circuit suffers from inflexibility — the connectivity motifs between the different groups of neurons are specifically designed for our SWM task, which simply asks the network to report the input sequence one-by-one in the same order as it is presented. This

may suffice for the input pathway, because a sequence of inputs by definition always starts with the first stimulus, followed by the second and so on. But there are many possible task outputs — for example, Tian et al. (2024) trained monkeys to report the sequence in either the forward or backward order, depending on a cue in the middle of the delay period. Our networks will trivially fail on this. Future work, e.g. training RNNs on this flexible output order task, with potential short-term plasticity mechanisms, may reveal alternative circuit structures that enable more flexible control.

B Dynamic chain as a natural solution for RNNs

Here we discuss further our speculations on the emergence of the dynamic chain representation in some of the trained RNNs. We note that similar ideas have been discussed in Karuvally et al. (2024); Keller et al. (2024). Because the recurrent weight matrix is initialized as a random orthogonal matrix, if we ignore the nonlinearity of the hidden units, the input will naturally be represented in different and roughly orthogonal subspaces at different time steps. As an example, if we ignore the nonlinearity, the bias, and the added noise, for a sequence of inputs $\boldsymbol{u_1}$, ..., $\boldsymbol{u_l}$, the hidden activity on the first time step is $\boldsymbol{h_1} = \boldsymbol{W}_{\text{in}}\boldsymbol{u_1}$, the second time step $\boldsymbol{h_2} = \boldsymbol{W}_{\text{rec}}\boldsymbol{W}_{\text{in}}\boldsymbol{u_1} + \boldsymbol{W}_{\text{in}}\boldsymbol{u_2}$, the third time step $\boldsymbol{h_3} = \boldsymbol{W}_{\text{rec}}^2\boldsymbol{W}_{\text{in}}\boldsymbol{u_1} + \boldsymbol{W}_{\text{rec}}\boldsymbol{W}_{\text{in}}\boldsymbol{u_2} + \boldsymbol{W}_{\text{in}}\boldsymbol{u_3}$ and so on. The i^{th} stimulus, at time t, will have been transformed by $\boldsymbol{W}_{\text{rec}}^{t-i}\boldsymbol{W}_{\text{in}}$. Because $\boldsymbol{W}_{\text{rec}}$ and $\boldsymbol{W}_{\text{in}}$ are randomly initialized, in high-dimensional space $\boldsymbol{W}_{\text{rec}}^{t-i}\boldsymbol{W}_{\text{in}}$ will lead to roughly orthogonal subspaces for different values of $t-i$. This means that 1) for a single stimulus, its representation will go through $\boldsymbol{W}_{\text{rec}}$ after each time step, effectively going through a chain of subspaces (Fig. S2), 2) at any given time point, different stimuli are encoded in different subspaces (for example, at $t = 3$, the first stimulus is transformed by $\boldsymbol{W}_{\text{rec}}^2\boldsymbol{W}_{\text{in}}$, while the second stimulus is transformed by $\boldsymbol{W}_{\text{rec}}\boldsymbol{W}_{\text{in}}$), 3) the same subspace gets reused by different stimuli on different time steps — for example, the $\boldsymbol{W}_{\text{rec}}\boldsymbol{W}_{\text{in}}$ "subspace" contains s1 at $t = 2$, s2 at $t = 3$ and so on (technically $\boldsymbol{W}_{\text{rec}}\boldsymbol{W}_{\text{in}}$ doesn't define a particular subspace, it is in conjunction with the low-dimensional encoding of the stimuli in the input units that produces the low-dimensional subspace). This is essentially the same as the dynamic chain representation we describe in the paper. The challenge, of course, for this randomly initialized network, is to read out the correct stimulus at the right time. Note if the sequence length l is fixed, at the i^{th} readout time step $t = l + i$, the i^{th} stimulus (which should be reported) is transformed by $\boldsymbol{W}_{\text{rec}}^{\boldsymbol{l}}\boldsymbol{W}_{\text{in}}$ — this is invariant over i, so a simple fixed linear readout from this "subspace" can be sufficient. However, because we mix the sequence lengths, $\boldsymbol{l}$ is not constant, so the stimulus that should be read out will be in different subspaces depending on the sequence length (this problem will also occur if a variable delay period is used). We do not fully understand how our trained networks solve this problem, but we suspect that they don't solve this challenge in a general way and this is why they fail on novel sequence lengths in the way they do. Given that many type 2 models use a mix of fixed slots and dynamic chain for different stimuli, it is possible that an interaction between these two representations may be used to produce the correct readout.



Supplementary Figure 2: **Schematic of the dynamic chain representation in a linear RNN**. Each column is a single subspace, each row is a time step.

C METHODS

C.1 TASK SETUP AND RNN TRAINING

The task, the network setup, and the training process are adapted from Piwek et al., (2023). We follow their notations when possible.

C.1.1 TASK

There are 17 input units that encode the circular stimulus. Each unit has a preferred value ϕ, with the values equally spaced between $-\pi \sim \pi$. There are 16 possible stimulus values α, again equally spaced between $-\pi \sim \pi$. For an input unit that prefers ϕ, its response u_i to stimulus α is the value of the von Mises probability density function:

$$u_i = c \cdot \frac{e^{\kappa \cos(\alpha - \phi)}}{2\pi I_0(\kappa)} \tag{1}$$

where $I_0(\kappa)$ is the modified Bessel function of order 0, $\kappa = 5$ controls the tuning width, and c is a scaling factor so that u_i is in the range [0, 1]. A binary input unit (0 or 1) indicates the go cue (which is active only on the first time step after the last input stimulus in the sequence), for a total of 18 input units. There are 17 output units, which have the same structure as the stimulus-encoding input units.

On each trial, each stimulus in the sequence is presented one by one from $t = 1$ to $t = l$ for a length-l sequence. On $t = l + 1$, the go cue is on, and the response period is from $t = l + 1$ to $t = 2l$. The desired output is $\mathbf{0}$ on $t = 1 \sim l$, and $\boldsymbol{u_{t-l}}$ on $t = l + 1 \sim 2l$.

During training, the sequence length is uniformly sampled from 1-4 and 6-9, and training sequences are randomly sampled. Repeats are allowed in a sequence. For each training length, we randomly sample a small set of sequences to be held out from training: $n = 1$ for length $l = 1$, 10 for $l = 2$, and 100 for $l \geq 3$ (note we do not exclude longer sequences that contain the heldout sequences, so for example a heldout length-1 sequence could appear as a part of a length-2 sequence that may be sampled during training).

C.1.2 RNN

The hidden activity of the RNN at time t is:

$$\boldsymbol{h_t} = f(\boldsymbol{W_{rec}}\boldsymbol{h_{t-1}} + \boldsymbol{W_{in}}\boldsymbol{u_t} + \boldsymbol{b_{rec}} + \boldsymbol{\eta}) \tag{2}$$

where $\boldsymbol{W_{rec}}$ is the recurrent weight matrix, $\boldsymbol{W_{in}}$ is the input weight matrix, $\boldsymbol{b_{rec}}$ is the bias, $\boldsymbol{\eta}$ is i.i.d. Gaussian noise from $\mathcal{N}(0; \sigma^2)$ (we use $\sigma = 0.1$), and f is the activation function. The hidden activity is initialized to $\mathbf{0}$ at the start of each trial.

The output at time t is:

$$\boldsymbol{\hat{y}_t} = g(\boldsymbol{W_{out}}\boldsymbol{h_t} + \boldsymbol{b_{out}}) \tag{3}$$

where $\boldsymbol{W_{out}}$ is the output weight matrix, $\boldsymbol{b_{out}}$ is the output bias, and g is the output nonlinearity.

All weights and biases use the standard PyTorch initialization ($\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in\_features}}$), except for $\boldsymbol{W}_{\text{rec}}$, which is initialized as a random orthogonal matrix (Saxe et al., 2014).

The total loss is the sum of two components: 1) the loss during the sequence presentation (which is always the standard MSE loss), and 2) the loss during the response period (which is either the standard MSE loss or the weighted MSE loss). For each component, the loss is averaged across output units and time steps. The weighted MSE loss on time t, given the network output $\boldsymbol{\hat{y}_t}$ and the ground truth output $\boldsymbol{y_t}$, is:

$$\mathcal{L}_{\text{response}} = \langle [(\boldsymbol{y_t} - \boldsymbol{\hat{y}_t})(\boldsymbol{\phi} - \alpha)_{\text{circ}}]^2 \rangle \tag{4}$$

where α is the ground truth stimulus value, $\boldsymbol{\phi}$ is the vector of the output units' preferred stimulus values (which mirror the input units), and $(\boldsymbol{\phi} - \alpha)_{\text{circ}}$ is the circular (angular) difference between the two. In this way, large activity at output units that are distant from the true stimulus (i.e. ϕ is far from α) will be punished more than activity at nearby output units. Fig. S11 illustrates the difference between the standard MSE loss and the weighted MSE loss on two example scenarios: the weighted MSE loss is very large when the network outputs the wrong stimulus, but is smaller if the network simply outputs zero.

We vary:

- Activation function f: tanh or ReLU (2 options)
- Output nonlinearity g: identity or scaled tanh ($g(x) = \tanh(x)/2 + 0.5$ so that $g(x)$ is in (0, 1)) (2)
- Number of hidden units: 100, 200, or 400 (3)
- Loss function (during the response period): MSE or weighted MSE (2)
- Random initialization (weight initializations, heldout sequences, and sample sequences throughout training) (3)

for a total of 72 networks. Each network is trained using the standard Adam optimizer with a batch size of 1. All networks are trained for 2000 epochs, with 1000 random sequences sampled per epoch, using a fixed learning rate of 10^{-4}. For Fig. S4, we train additional batches of networks with learning rates 10^{-3}, 10^{-5}, and 10^{-6} (other settings are exactly the same, except only 2 random initializations were used for each hyperparameter combination).

C.1.3 Training dynamics

Fig. S13 shows the loss over training: the loss decreased significantly over training for all networks, though not all networks have completely converged. Interestingly, the lack of convergence tends to occur for type 2 networks, which already achieve excellent performance across all training lengths (Fig. 3a, Fig. S16), while type 1 networks, which still have significant errors on long training lengths, have largely plateaued loss.

Delving into the training dynamics of Network 1, we inspect its performance on different sequence positions over training. Fig. S14a shows the mean angular error on s1-s4 of length-9 heldout sequences over training: the stimuli positions are learned in the forward order, with s1 being learned first, followed by s2, s3, and then s4. Each stimulus position appears to be learned rather suddenly. Fig. S14b shows the number of epochs at which each stimulus position is learned (we use the minimum number of epochs at which the mean angular error of the target stimulus position is below 0.1 for heldout sequences of all train lengths) — this is well fit by an exponential function, indicating that it may take many more epochs to learn additional stimuli positions. This may be due to the fact that the number of possible sequences scales exponentially with the sequence length.

C.1.4 Recurrent weight matrix initialization

To test whether the orthogonal initialization of the RNN recurrent weight matrix is crucial for our main results, we train a separate batch of networks ($N = 72$) where we initialize the recurrent weight matrix randomly using the standard Kaiming uniform initialization (He et al., 2015). Fig. S15 shows that we observe qualitatively similar results, replicating the two types of length generalization behaviors and the two types of representational dynamics.

C.2 Evaluating network behavior

We use the mean angular error as the main metric for evaluating network performance. For each time step during the response period, we convert the network's distributed outputs into a single angular value by computing the circular mean of the output units' preferred stimulus values weighted by their activations, and use the absolute value of the circular difference between this value and the ground truth stimulus value. The chance error is $\frac{\pi}{2}$, assuming the output angular value is uniformly random on $-\pi \sim \pi$. If all output units' activities are zero on a trial, we use this chance value as the angular error (this did not occur for any of the 72 networks we report in the main text, it only

occurred in 5 networks we trained with alternative learning rates in Fig. S4). We test the networks on 10,000 random sequences for each sequence length.

Based on performance on the different sequence lengths, we classify networks into two types: specifically, if a model's mean angular error on length-5 sequences is at least 0.3 larger than on length-4 and at least 0.3 larger than on length-6, then it is classified as a type 2 model; otherwise it is classified as type 1. For each type, we pick an example network: Network 1 uses the ReLU activation function, identity output nonlinearity (i.e. linear output), 400 hidden units, and the weighted MSE loss; Network 2 uses the ReLU activation function, the scaled tanh output nonlinearity, 100 hidden units, and the standard MSE loss.

C.2.1 Different metrics of network behavior

We mainly use mean angular error to evaluate network performance in the main text. Fig. S16 shows network performance evaluated in terms of the standard MSE loss and the weighted MSE loss — the results are broadly similar to the mean angular error (Fig. 3a), with type 1 and type 2 networks exhibiting different length generalization behaviors irrespective of the metric used. Note type 1 networks, which are trained on the weighted MSE loss, also achieve relatively low standard MSE loss on length-5 and length-10 sequences. The only discrepancy occurs for two type 1 networks, which show large errors on length-5 and length-10 sequences when evaluated on the weighted MSE loss, but not when evaluated on the standard MSE loss or the mean angular error. Further inspection reveals that these networks are still able to generalize the ability of remembering several items to these test lengths (leading to low errors when evaluated on MSE loss or mean angular error), but because they report incorrect stimuli for other sequence positions (instead of remaining silent), this leads to very large weighted MSE loss values on these positions (as weighted MSE heavily penalizes wrong stimuli compared to just zero output, Fig. S11), so that averaged across sequence positions, the weighted MSE loss is very high for length-5 and length-10 sequences. Interestingly, these networks remain silent and do not report incorrect stimuli for these positions on train lengths (leading to low weighted MSE loss on training lengths).

C.3 Characterizing network representation

C.3.1 Identifying stimulus-encoding subspaces

To identify the subspaces that encode each stimulus in the sequence, we compute the activities of each network's hidden units on 1,000 random length-9 sequences. Because our task does not have any explicit delay, we take the last time step of the stimulus presentation period ($t = 9$) as a proxy for the delay period (at this point, e.g. s1 is presented 8 time steps in the past, so the model has to memorize s1 for 8 time steps, effectively a "delay period"). For each stimulus position (e.g. s1), we average the activity at $t = 9$ across all sequences with the same stimulus value at the target position (e.g. all sequences where s1 is $-\pi$), thus obtaining 16 vectors (one for each possible stimulus value) with size equal to the number of hidden units. We compute the dimensionality as the participation ratio of this set of vectors (Abbott et al., 2011). Then we apply PCA and use the top two PCs as the subspace that encodes the target stimulus.

C.3.2 Principal angle between subspaces

To evaluate the relationship between two subspaces, we compute the principal angle between them following Xie et al. (2022): given the two PCs that span each of the subspaces ($\hat{V}_a$ and $\hat{V}_b$), we compute the singular value decomposition of the matrix $\hat{V}_a{}^T \hat{V}_b$ (size 2×2), and the first singular value is equal to the cosine of the first principal angle, which we use for the paper.

Following Xie et al. (2022), we compute the control by randomly splitting the sequences in half and estimating the subspaces independently from each half of the data, and then computing the principal angle between the two estimated subspaces for the same stimulus position. We repeat the process for a total of 100 splits.

C.3.3 CROSS-DECODING ANALYSIS

To probe the representational dynamics of the trained networks, we perform the following cross-decoding analysis: we compute the hidden representation on a random sample of 10,000 length-9 sequences, and for each stimulus position, we train a separate linear decoder using the hidden activity at $t = 9$ (just before the go cue). Then we directly use this decoder on the hidden activity on all other time steps, and use it to decode all possible stimuli in the sequence. We test the decoder on an independent sample of 10,000 length-9 sequences. We use linear regression to decode the raw distributed values of the 17 input units. We add a small amount of zero-mean Gaussian noise ($\sigma = 0.01$) to the hidden activity during both training and testing for implicit regularization.

C.3.4 CORRELATION BETWEEN SLOT REPRESENTATION AND SUCCESSFUL LENGTH GENERALIZATION BEHAVIOR

We hypothesize that **slot representations** lead to successful **length generalization behavior**. For example, Network 2 exhibits putative slot representations for s1-s2 and on length-10 sequences successfully recalls s1-s2, but it skips s3 and reports the remaining sequence one step too early. To test this hypothesis more broadly, we use the following heuristics for each type 2 network:

1. **Identifying slot representations:** We determine whether a stimulus s_i has a slot representation based on the cross-decoding analysis. If the mean cross-decoding error for the s_i decoder (trained on $t = 9$) is at most 1 when tested on s_i from $t = i + 1$ to $t = 9$ (the error is averaged across all these time points), we classify s_i as having a slot representation.
2. **Determining behavioral failure points in length-10 sequences:** We analyze when the network begins skipping ahead. For each stimulus s_i, we examine the network's response at $t = 10 + i$ (when s_i should be reported). If the mean error when comparing this response to s_i is at least 0.5 larger than when comparing this response to s_{i+1}, this suggests the network is one time step ahead, reporting s_{i+1} instead of s_i. In this case, we classify s_i as a stimulus where the network fails.

We then identify:

1. The smallest i where the representation ceases to be a slot representation (x-axis in Fig. S7).
2. The smallest i where the network fails on length-10 sequences (y-axis in Fig. S7).

Fig. S7 shows that there is a positive correlation between these two values ($r = 0.75$), providing evidence for a link between slot representation and length generalization behavior. We identify all networks where these two values are equal and select an example network for each value from 1 to 5. This corresponds to networks with 0 to 4 slots that start failing on s1 to s5 in length-10 sequences. These networks are shown in Fig. S8.

C.4 RECONSTRUCTING THE CIRCUIT MECHANISM

C.4.1 IDENTIFYING SPECIALIZED NEURAL POPULATIONS FROM TRACING WEIGHTS

We compute Network 1's neural activity on 10,000 random length-9 sequences. We identify readout neurons as follows: for each stimulus s_i, a neuron is classified as a *readout* i neuron if its **mean activity on the readout time step** $t = 9 + i$ (averaged across sequences) is larger than 0.25, and its **readout weight** (we use the absolute value of $\boldsymbol{W}_{\text{out}}$ and take the maximum across each neuron's output weights to all 17 output units) is larger than 0.05. We manually exclude 3 neurons that overlap between different stimuli.

We identify slot neurons as follows: for each stimulus s_i, first we filter out neurons with low variance over s_i — here we use z-scored activities (the z-scoring is applied across both time steps and sequences), and we average the activity across time steps from $t = i + 1$ (one time step after s_i is presented) to $t = i + 8$ (one time step before s_i should be read out). We compute the mean activity across all sequences with the same value for s_i (e.g. all sequences where $s_i = -\pi$). This gives us 16 values (one for each possible s_i value), and we compute the variance over them. If this variance is less than 0.5, we discard the neuron. This step removes neurons that might have large activities but don't encode the value of the stimulus. After this filtering step, we use each neuron's **mean**

recurrent weight to *readout* i neurons (we use the absolute value of $\boldsymbol{W_{rec}}$ and average across its weights to all *readout* i neurons) and multiply it with its raw **activity** (not z-scored) averaged across $t = i + 1$ to $t = i + 8$ and across all sequences. This gives us an aggregate value per neuron, and we select the top 20 neurons with the largest values as the slot neurons (if fewer than 20 neurons are left after the filtering step, we simply use all of them). This aggregate value, a product of activity and weights, provides a rough estimate of how much the neuron contributes to the activity of the readout neurons.

We identify sensory neurons using a similar method to slot neurons: for each stimulus s_i, first we filter out neurons with low variance over s_i, this time using their activities on $t = i$ (when s_i is presented). Next, we use each neuron's **mean recurrent weight to *slot* i neurons** and multiply it with its **mean activity** on $t = i$, and select the top 20 neurons. We manually remove 3 neurons that overlap between different stimuli.

In total, we identify 12 groups of non-overlapping neurons for a total of 203 neurons: 19 *sensory 1* neurons, 19 *sensory 2* neurons, 17 *sensory 3* neurons, 16 *sensory 4* neurons, 20 *slot 1* neurons, 20 *slot 2* neurons, 16 *slot 3* neurons, 10 *slot 4* neurons, 15 *readout 1* neurons, 19 *readout 2* neurons, 18 *readout 3* neurons, and 14 *readout 4* neurons. Within each group, we sort them by their preferred stimulus values (this is used in Fig. 7e): again we use their activities on random length-9 sequences, we average their activities based on the value of their target stimulus (e.g. for a *sensory 1* neuron, we average its activity across all sequences where s1 equals a particular value), and we compute the circular mean across all the possible stimuli values weighted by the averaged activities.

We note that these are the final selection criteria — we picked them based on our hypothesis and added additional criteria to have more refined, "clean" groups of neurons. In our first attempt, before we had any idea of the structure of the network, we used simpler criteria (Fig. 6a). We started with readout neurons, where we used the same criteria as the final version: the idea is simple, a neuron would have large contributions to the readout if it has large output weights and also large activity on the readout time steps. We used the activity on a single readout time step and found surprisingly that the neurons selected are *only* active on that time step, not other readout time steps, and not even on any time step during the stimulus presentation (Fig. 7a). This is in contrast to a shared readout plane, where the same group of neurons (or the same direction of activity) is active across all readout time steps. This was our first tip that the network has developed a modular structure, where separate neurons are dedicated to each stimulus. We visualized a few of these neurons to verify that their activities on their corresponding readout time steps do indeed encode the value of the appropriate stimulus.

These readout neurons beg the question: where do they get their activities from? Since they are silent for all time steps before the target readout time step, they must be getting the stimulus information from other neurons. Therefore, we used the recurrent weights to the readout neurons to identify candidate slot neurons (on our first pass, the weights are the only criteria we used — we did not add any additional filtering by variance, or multiplying the mean weight with the mean activity). By visualizing these neurons' activities over time, we find that 1) their activities do encode the value of the corresponding stimulus, and they largely do not encode any other stimulus, and 2) they tend to have persistent activity throughout stimulus presentation, starting one time step after the target stimulus is presented ($t = i + 1$ for s_i) and until one time step before the target stimulus should be read out ($t = i + 8$) (similar to Fig. 7b). This persistent activity suggests that these neurons might implement slots, maintaining stimuli information after they have been presented. Fig. S12 shows two example slot neurons — we borrow the visualization technique from Cueva et al. (2021) and plot each neuron's activity as a function of two stimuli values (e.g. s1 and s2). A horizontal stripe or vertical stripe indicates tuning for one stimulus but not the other stimulus. A random pattern suggests a lack of systematic tuning for either stimulus. Fig. S12a,c suggest this neuron is only tuned for s1, while Fig. S12b,d suggest this neuron is only tuned for s2.

However, these neurons do not immediately encode their target stimulus on the time step that the stimulus is presented ($t = i$). This suggests the stimulus information first goes through another part of the network before entering these slot neurons. Therefore, we looked for neurons with large recurrent weights to the slot neurons — this alone is insufficient because many slot neurons have large recurrent weights within themselves (in order to create attractor-like circuits that could maintain the persistent activity), so on our first pass we added the criterion that these neurons should have large activity on $t = i$. Many of the identified neurons only have transient activities and become

silent after $t = i$. We observe that many of them, in addition to encoding the target stimulus s_i on $t = i$, also encode earlier stimuli when they are presented.

Visualizing the activity of these neurons led us to Fig. 6c, an abstracted description of what these neurons encode on each time step. Based on this more idealized scheme of what each group of neurons should represent, we went back to our selection criteria and refined them to form the final selection criteria.

C.4.2 RECONSTRUCTING CIRCUIT CONNECTIONS

Because we identified the specialized groups of neurons partly from tracing the weights directly, we already know a part of the circuit —- namely, that the stimulus first goes to sensory neurons, which project to slot neurons, which project to readout neurons, which project to the final output units. However, this is not enough to implement the appropriate control — for example, why are *readout 1* neurons only active on the single time step when s1 should be read out? In order to identify the additional connections that implement the control, we took two approaches: 1) a bottom-up approach where we plot subsections of the recurrent weight matrix between two groups of neurons we have identified, and look for any notable patterns (e.g. if the sub-matrix has mostly positive values); and 2) a top-down approach where we use the specific patterns of representations we have observed (Fig. 6c) to come up with hypotheses on how the circuit might be wired in order to produce such representations.

We start with s1-related neurons (and its interaction with s2-related neurons), and check if any patterns we find exist between s2 and s3 neurons, and s3 and s4 neurons. This is an iterative process, and it is very imperfect — for one, it relies completely on the subpopulations of neurons we have identified. For the top-down approach, we can only find circuit mechanisms that we can imagine. In fact, Fig. 6b is not the full description of what the network does, as we see many discrepancies between the theoretical connections and the observed trained weights (Fig. 7d-e). However, 1) the fact that we can make *some* sense of a trained network (that is completely randomly initialized and has no built-in structure) is a victory to us, and 2) we see the trained RNNs as a source of inspiration, a hypothesis generator for the brain — we are more interested in simple, potentially generalizable mechanisms that could solve the task, rather than identifying the most comprehensive description of what the trained network does.

C.4.3 TESTING THE HYPOTHESIZED CIRCUIT

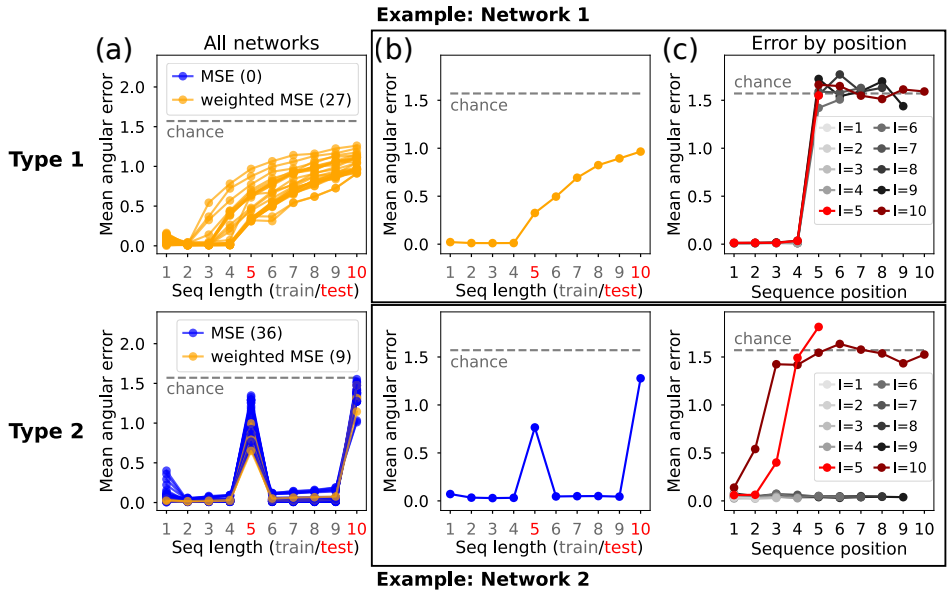
To test if the different groups of neurons in Network 1 behave as we expect, we compute their activities on length-9 sequences and train linear decoders using the same procedure as C.3.3. Each decoder is trained on the activities of a group of neurons. Instead of cross-decoding, we train a separate decoder for each time step and stimulus position combination, and test the decoder on the same time step and stimulus position using an independent sample of sequences.

To perturb each group of neurons, we use the technique of activation patching from mechanistic interpretability (Vig et al., 2020; Meng et al., 2023): for each target group of neurons and each time step, we compute their activities on two independent sets (call them A and B) of random length-9 sequences. Then we test the entire network on set A sequences, but at the target time step, replace the activities of the target neurons with their activities on set B on the same time step. After this manipulation, we compute the final network outputs and compare them with the ground truth set A sequences to compute the mean angular error, and we report the difference between this error and the original mean angular error when the network is tested on set A without perturbations.

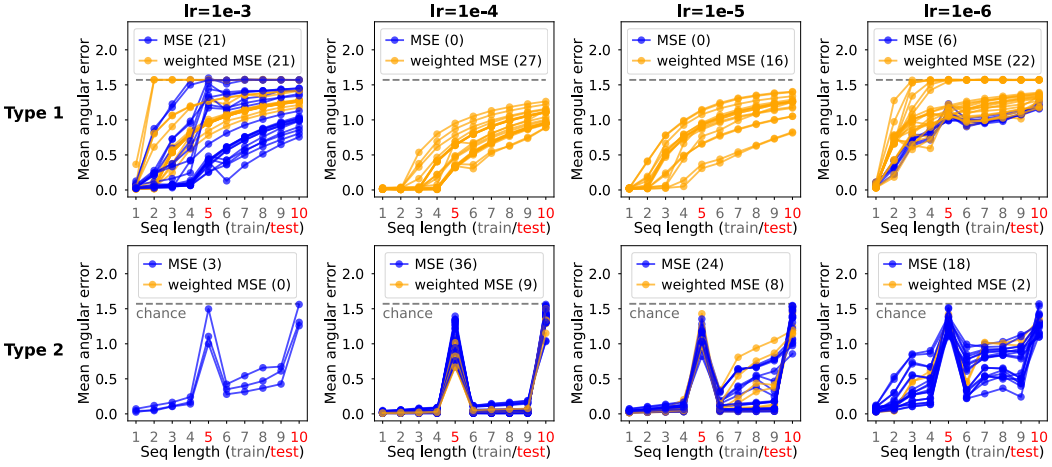
To quantify if the observed network weights fit the reconstructed circuit, for the hypothesized *general* connections, we compute the mean value of Network 1's trained weights between the corresponding neuron groups (Fig. S10a). To compute the p-value for a given connection (which corresponds to a sub-matrix, e.g. the weights from *slot 2* neurons to *sensory 3* neurons, which is a size-17 $\times$ 20 sub-matrix of the full recurrent weight matrix $\boldsymbol{W_{rec}}$), we randomly sample 1,000 sub-matrices of the same size from the full matrix and compute the fraction of times that the mean value of this sub-matrix is smaller or larger than the one observed. For each hypothesized *specific* connection (e.g. from *sensory 1* neurons to *slot 1* neurons), for each pair of pre-synaptic and post-synaptic neurons, we compute the magnitude of the angular difference between the preferred values of their target stimuli (see C.4.1). We compute the Pearson correlation between this angular differ-

ence and the observed weight from the pre-synaptic neuron to the post-synaptic neuron across all pairs (Fig. S10b). If the connection is Mexican hat-like, then neurons with similar preferred values would excite each other, while neurons with distant preferred values inhibit each other, thus leading to a negative correlation. To compute the p-value, we randomly shuffle the angular differences 10,000 times and recompute the correlation.

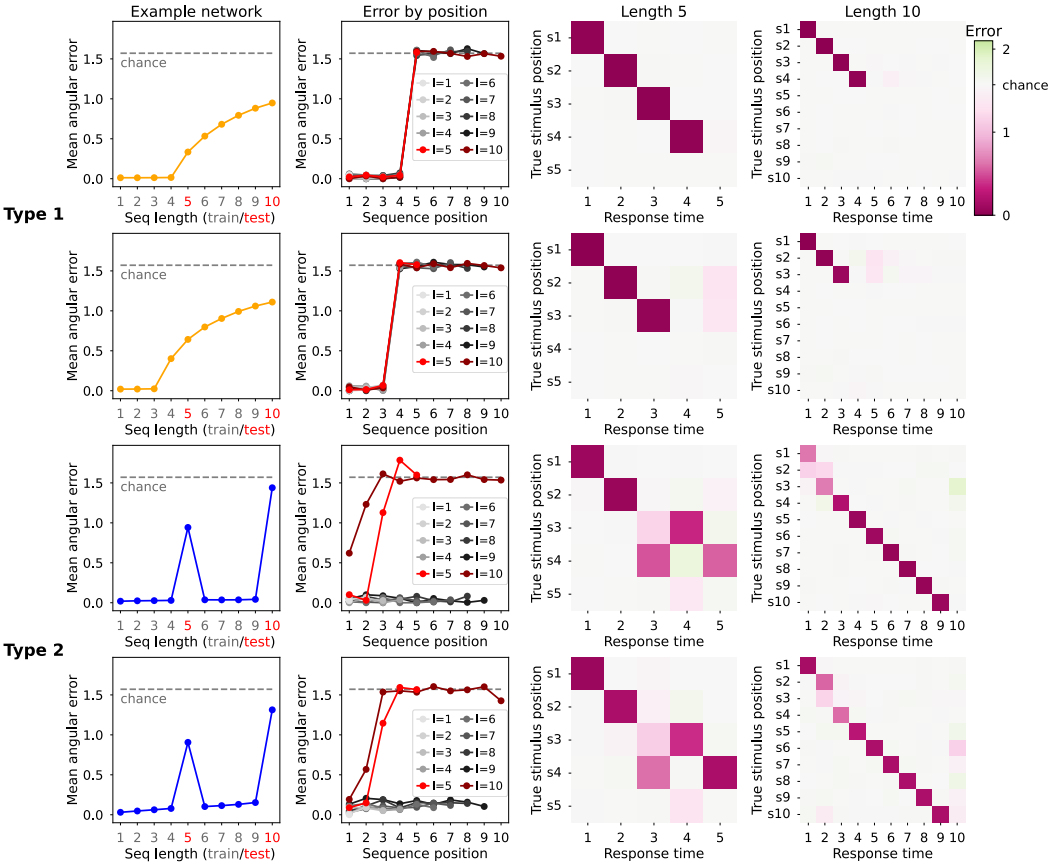
D SUPPLEMENTARY FIGURES



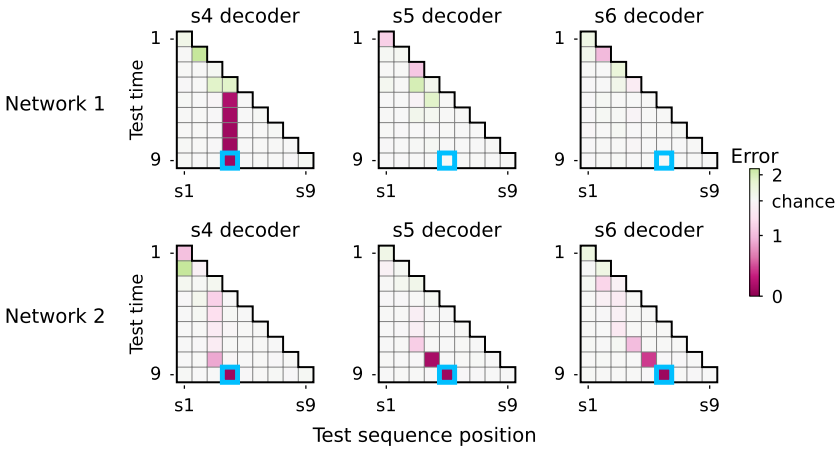
Supplementary Figure 3: **Network performance on sequences held out from training**. See captions for Fig. 3a-c. For lengths 5 and 10 (no sequences of these lengths are seen in training), 100 random sequences are used. Note type 2 models achieve low errors on train lengths even for these held-out sequences.



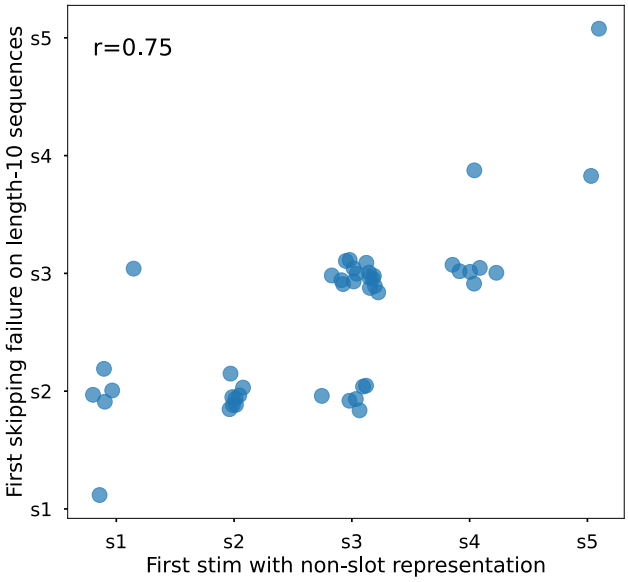
Supplementary Figure 4: **Effect of learning rate (lr) on network behavior**. See captions for Fig. 3a. 10^{-4} is the learning rate used for all other results in the paper. We exclude 3 networks with learning rate 10^{-3} that have large errors (larger than 1.5) for all sequence lengths. Note at 10^{-3}, a large fraction of networks with the standard MSE loss are type 1.



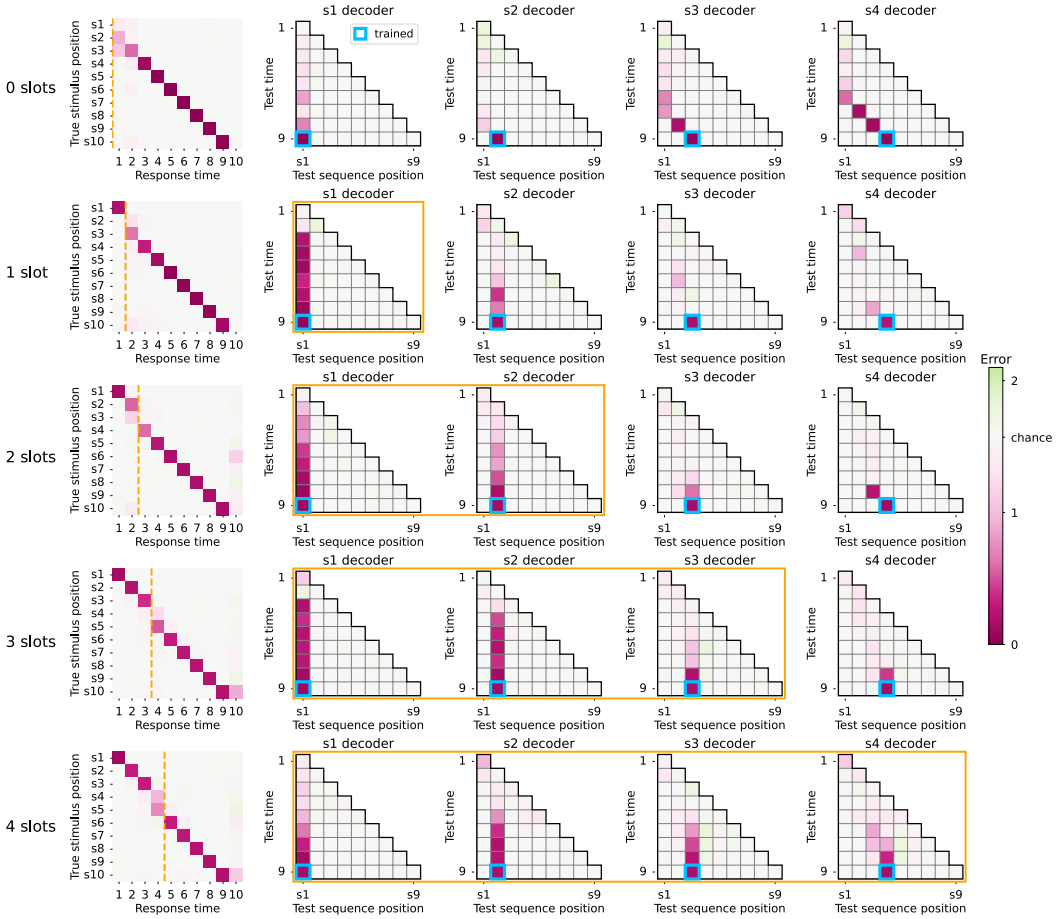
Supplementary Figure 5: **Behavior of additional example networks**. See captions for Fig. 3b-e. Each row is a single network: top two rows are type 1, bottom two are type 2. The first and third row have the same hyperparameter settings as Network 1 and Network 2 and only differ in their random seeds. The second and fourth row are tanh networks: both use the tanh activation function, the scaled tanh output nonlinearity, 100 hidden units; they only differ in the loss function used (second row uses weighted MSE, fourth row uses standard MSE).



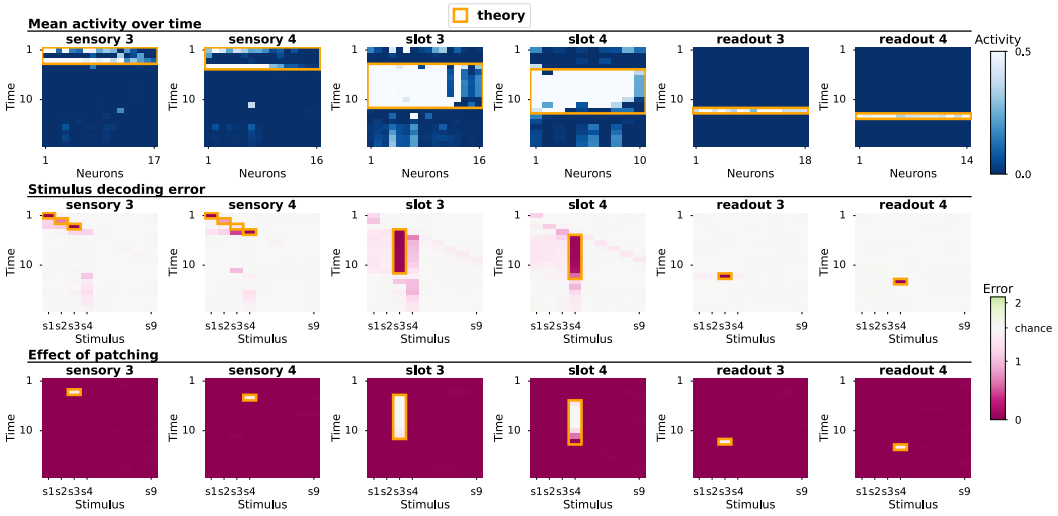
Supplementary Figure 6: **Additional cross-decoding results**. See captions for Fig. 5c. For simplicity, the main figure only reports s1-s3 and s7-s9.



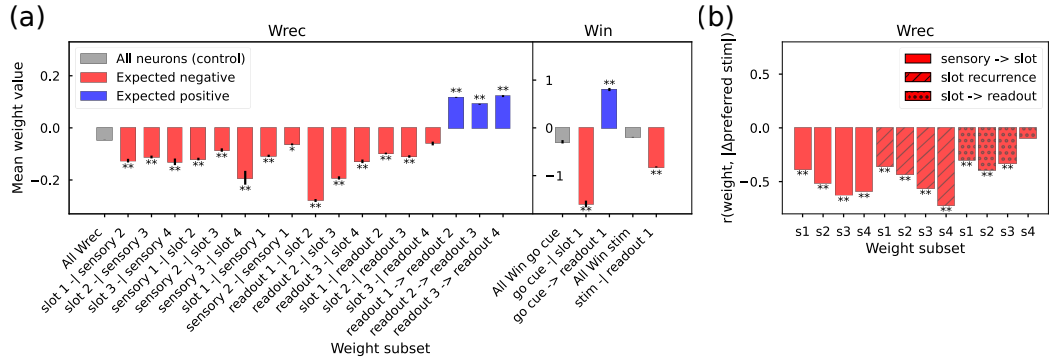
Supplementary Figure 7: **Correlation between slot representation and successful length generalization behavior**. See C.3.4. Each dot is a single type 2 network (45 in total). r is the Pearson correlation coefficient.



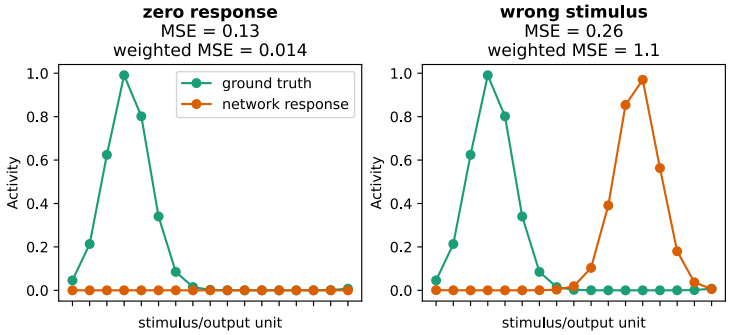
Supplementary Figure 8: **Example networks with a match between slot representation and successful length generalization**. See C.3.4 for identification of slot representation and failure on length generalization. Each row is an example network. Leftmost column is the network's behavior on length-10 sequences (see captions for Fig. 3e): orange dashed line indicates when length generalization begins to fail and the network skips the current stimulus (as inferred by the heuristic). Right four columns are cross-decoding results on the network's hidden activities (see captions for Fig. 5c): large orange box indicates stimuli that have slot representations based on the heuristic.



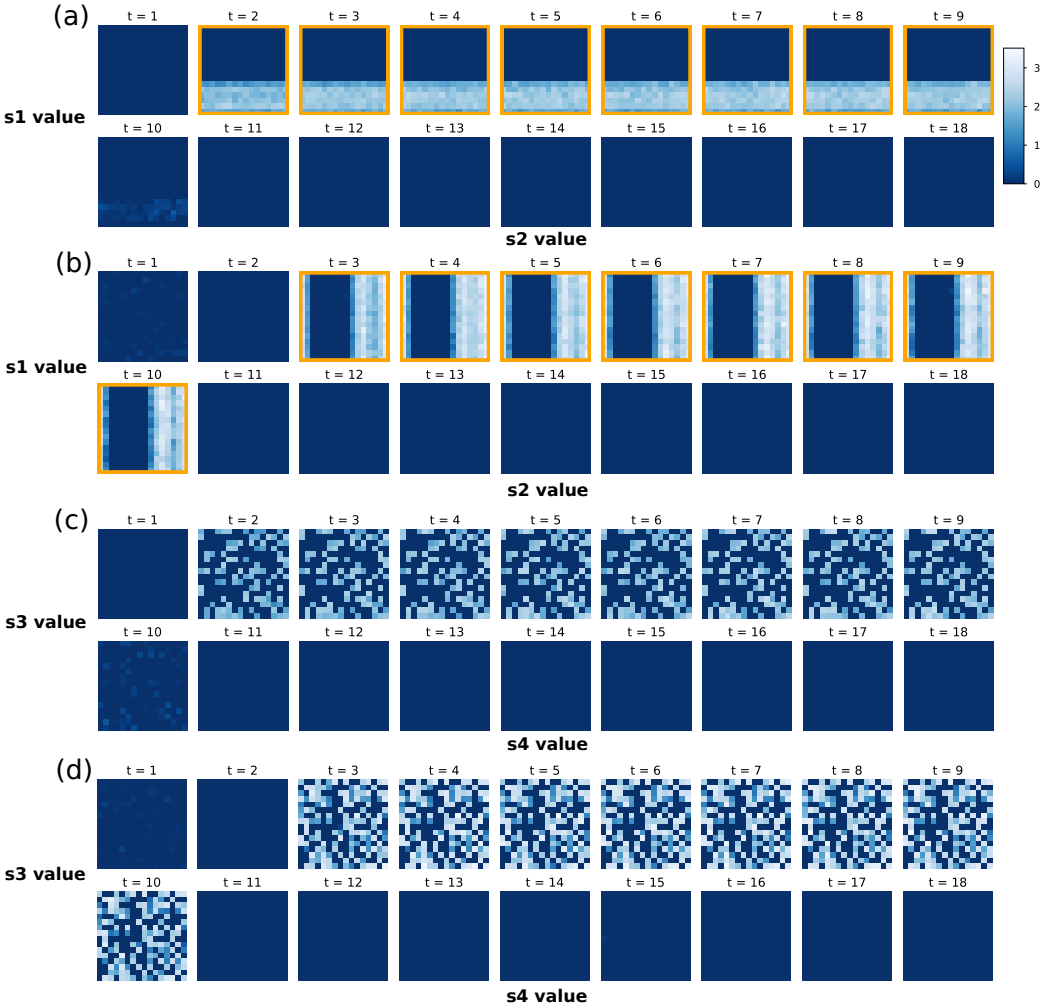
Supplementary Figure 9: **Additional evidence of hypothesized circuit in Network 1**. See captions for Fig. 7a-c. For simplicity, we only show neuron groups for s1 and s2 in the main text, but here we provide the same analysis on s3 and s4 neuron groups.



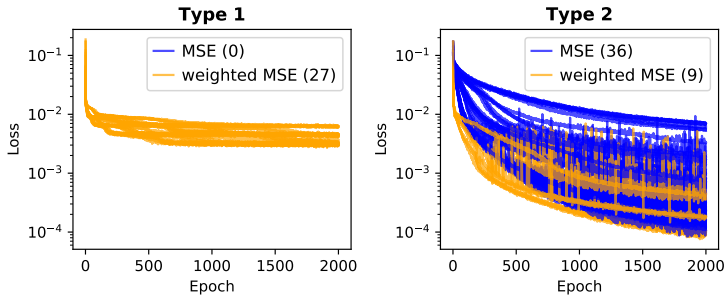
Supplementary Figure 10: **Quantitative comparison of Network 1's trained weights with the reconstructed circuit**. **a**, Mean observed weight value for each hypothesized general connection in the reconstructed circuit. Each hypothesized connection corresponds to a sub-matrix between two groups of neurons — we compute the mean value of this entire sub-matrix. Color indicates if the connection is hypothesized to be excitatory (positive) or inhibitory (negative); grey indicates values across all neurons; x-axis label $-|$ indicates inhibitory, $\rightarrow$ excitatory. Left subplot shows recurrent connections (**Wrec**: recurrent weight matrix), right shows input connections (**Win**: input weight matrix; we separate input weights from the stimulus-encoding input units and the go cue input unit). Error bar is the standard error of the mean (treating each entry in the sub-matrix as independent). ** and * indicate $p < 0.01$ and 0.05 respectively. **b**, Correlation between recurrent weight value and the absolute difference in their preferred stimulus values for each hypothesized specific connection (see C.4.3). All correlations are expected to be negative. Shading indicates which set of connections (e.g. s1 bar with circles indicates the connections from *slot 1* neurons to *readout 1* neurons).



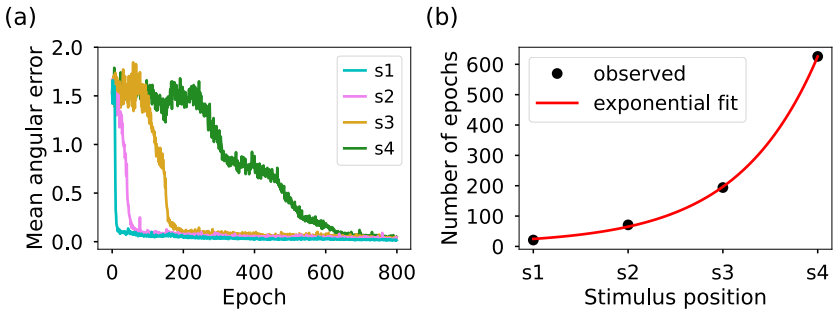
Supplementary Figure 11: **Comparison of standard MSE loss and weighted MSE loss on two examples**. Left: network outputs zero. Right: network outputs a valid but wrong stimulus.



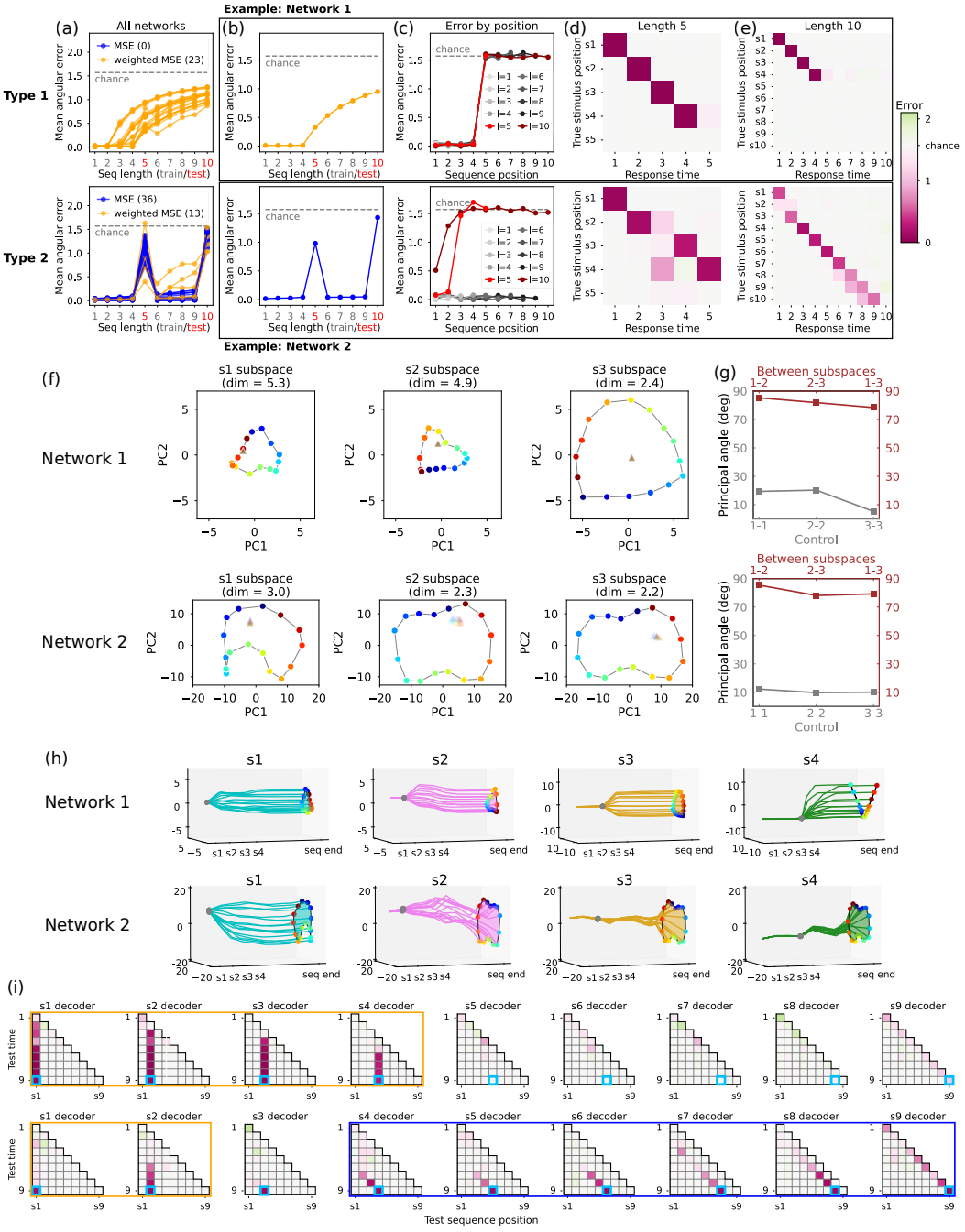
Supplementary Figure 12: **Activity of example slot neurons**. **a**, Example *slot 1* neuron. Each subpanel shows its activity on a single time step for length-9 sequences. Within each subpanel, the activity is shown as a function of s1 value (y-axis) and s2 value (x-axis): for each combination of s1 value and s2 value, we randomly sample a length-9 sequence with these values for s1 and s2, and compute the neuron's activity. Orange box indicates time steps that we expect to see encoding. **b**, Example *slot 2* neuron. **c**-**d** Same as **a**-**b**, but showing activity as a function of s3 and s4 values.



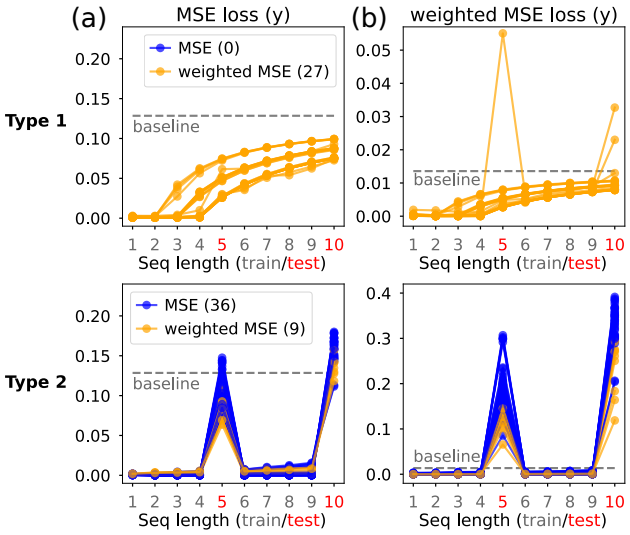
Supplementary Figure 13: **Loss over training**. Each line is a single network. Color indicates the loss function used for training; the y-axis reports the corresponding loss.



Supplementary Figure 14: **Training dynamics of Network 1**. **a**, Error over training for different stimulus positions (first 800 epochs), using length-9 heldout sequences. **b**, Number of epochs at which each stimulus position is learned. Red curve is the best-fit exponential function.



Supplementary Figure 15: **Trained networks using Kaiming initialization of the recurrent weight matrix**. **a**-**e**, Network behavior, see captions for Fig. 3a-e. **f**-**h**, Representation of two example networks, see captions for Fig. 4a-c. **i** Cross-decoding analysis, see captions for Fig. 5c. The example networks (Network 1 and Network 2) have the same hyperparameters as the example networks reported in the main text.



Supplementary Figure 16: **Network behavior evaluated using MSE and weighted MSE loss**. **a**, Standard MSE loss. **b**, Weighted MSE loss. See captions of Fig. 3a. Note the y-axis scale is different across panels. We use the response period loss only. The baseline value is computed as the loss if the network outputs 0 (note this is different than the network outputting a random stimulus). Network type classification is identical to the one used in the main text (which is based on the mean angular error).