

1 **Flexible multitask computation in recurrent networks utilizes shared dynamical motifs**
2
3
4

5 Laura Driscoll¹, Krishna Shenoy^{1,2,7}, David Sussillo^{1,5}
6
7
8
9

10 ¹ Department of Electrical Engineering, Stanford University, Stanford, CA, USA
11 ² Department of Neurosurgery, Stanford University, Stanford, CA, USA
12 ³ Department of Bioengineering, Stanford University, Stanford, CA, USA
13 ⁴ Department of Neurobiology, Stanford University, Stanford, CA, USA
14 ⁵ Wu Tsai Neurosciences Institute, Stanford University, Stanford, CA, USA
15 ⁶ Bio-X Institute, Stanford University, Stanford, CA, USA
16 ⁷ Howard Hughes Medical Institute at Stanford University, Stanford, CA, USA
17
18

19 **Flexible computation is a hallmark of intelligent behavior. Yet, little is known about how neural**
20 **networks contextually reconfigure for different computations. Humans are able to perform a new task**
21 **without extensive training, presumably through the composition of elementary processes that were**
22 **previously learned. Cognitive scientists have long hypothesized the possibility of a compositional neural**
23 **code, where complex neural computations are made up of constituent components; however, the neural**
24 **substrate underlying this structure remains elusive in biological and artificial neural networks. Here we**
25 **identified an algorithmic neural substrate for compositional computation through the study of**
26 **multitasking artificial recurrent neural networks. Dynamical systems analyses of networks revealed**
27 **learned computational strategies that mirrored the modular subtask structure of the task-set used for**
28 **training. Dynamical motifs such as attractors, decision boundaries and rotations were reused across**
29 **different task computations. For example, tasks that required memory of a continuous circular variable**
30 **repurposed the same ring attractor. We show that dynamical motifs are implemented by clusters of units**
31 **and are reused across different contexts, allowing for flexibility and generalization of previously learned**
32 **computation. Lesioning these clusters resulted in modular effects on network performance: a lesion that**
33 **destroyed one dynamical motif only minimally perturbed the structure of other dynamical motifs.**
34 **Finally, modular dynamical motifs could be reconfigured for fast transfer learning. After slow initial**
35 **learning of dynamical motifs, a subsequent faster stage of learning reconfigured motifs to perform novel**
 tasks. This work contributes to a more fundamental understanding of compositional computation
 underlying flexible general intelligence in neural systems. We present a conceptual framework that
 establishes dynamical motifs as a fundamental unit of computation, intermediate between the neuron
 and the network. As more whole brain imaging studies record neural activity from multiple specialized
 systems simultaneously, the framework of dynamical motifs will guide questions about specialization
 and generalization across brain regions.

36 Introduction

37 Cognitive flexibility is a key feature of the human brain. While artificial systems are capable of outperforming
38 humans in specific complex cognitive tasks¹⁻³, they so far lack flexibility for rapid learning and task switching.
39 Therefore, a major open question in the fields of neuroscience and artificial intelligence is how the same circuit
40 flexibly reconfigures to perform multiple tasks⁴.

41 Conceptual models for cognitive flexibility propose a hierarchy of elementary processes that are reused across
42 similar tasks^{5,6}. According to these models, the neural substrate for computation is modular such that
43 combinations of previously learned subtasks may be reconfigured to perform unfamiliar tasks. This
44 combination of subtasks is referred to as compositionality⁶. For example, a saccade task typically involves a
45 cue that indicates in which direction to move the eyes. After learning a saccade task, a person could quickly
46 learn an ‘anti’ version of the same task where the same cue now instructs a saccade in the opposite direction.
47 This new task may be quickly learned by combining a computational building block for the original task with
48 a previously learned ‘anti’ building block. These computational building blocks could be considered in task
49 space or state space. While there is some experimental evidence that neural computation is compositional^{7,8}, a
50 concrete model for the neural implementation of compositional computation would provide major headway
51 toward understanding cognitive flexibility for multiple tasks in the human brain.

52 While the time and effort required to train animals to perform many tasks has proven prohibitive to the
53 exploration of multitask computation in biological networks, artificial neural networks now present an
54 opportunity to explore the topic. The study of cognitive tasks through simulations in artificial networks has led
55 to substantial advances in understanding neural computation in the past decade⁹⁻¹⁸. However, researchers
56 typically trained artificial neural networks to perform single tasks in isolation, with few exceptions¹⁹⁻²²,
57 somewhat limiting the insights into biological neural circuits that perform many tasks. One exception to this
58 trend is Yang et al. 2019²⁰, in which the authors trained a single network to perform twenty related tasks and
59 thereby identified clustered representations in state space that supported task compositionality. Yet, their
60 analysis left open the critical questions concerning **why and how recurrent networks develop a compositional**
61 **organization**. In this work, we identified the computational substrate that allowed for compositional
62 computation and answered the nontrivial question of how this computational substrate is flexibly employed in
63 different contexts.

64 We examined multitask networks through the lens of dynamical systems. This approach allowed us to explore
65 the mechanisms underlying computation in a recurrently connected artificial network²³. We found that tasks
66 that required the same computational elements (e.g. memory, categorization, delayed-response) were
67 implemented by sharing and repurposing dynamical motifs (e.g. attractors, decision boundaries, and rotations).
68 We demonstrate that the dynamical motifs learned in a multitask network support compositional computation.

69 Dynamical motifs provided a useful granularity to study network computation. While individual network units
70 appeared heterogenous and difficult to interpret, dynamical motifs were often interpretable and consistent
71 across networks. Our framework provides a conceptual building block between the single artificial unit and
72 the whole network. With the access and control afforded by the use of this simulated ‘model organism’, we
73 explored how multiple related computations might interact in a single network. This work contributes to a
74 better understanding of functional specialization in neural networks, and more broadly advances techniques
75 for reverse-engineering artificial networks and analysis of high-dimensional dynamical systems.

76 Results

77 Network Structure

78 We implemented a similar input-output structure and learning protocol as in previously examined multitasking
79 recurrent neural networks (RNNs)²⁰. These tasks included reaction-timed, delayed response and memory tasks
80 with contextual integration, categorization, pro and anti response components (see Supp. Fig.1 and Methods
81 Section 1.2 for task definitions). For every task, the network received three noisy inputs: fixation (1-
82 dimensional), stimulus (4-d), and rule (15-d) (Fig.1a). The fixation input directed the network to either output
83 zero or respond, similar to fixation cues which instruct rodents, monkeys and humans to keep the eyes and
84 limbs still. The set of stimuli contained two separate two-dimensional vectors composed of $A\sin\theta$ and $A\cos\theta$,
85 where each vector encoded a different one-dimensional circular variable (θ_1, θ_2) scaled by an amplitude A .
86 Depending on the rule, one stimulus vector may be contextually ignored. The rule input indicates the current
87 task on any given trial and this information is continuously available to the network throughout each trial. Rule
88 input was encoded in a one-hot vector where the index associated with the current task was one and all other
89 indices were zero.

90 The RNN is defined by

$$\tau \frac{d\mathbf{h}}{dt} = -\mathbf{h}(t) + F(W_{rec}\mathbf{h}(t) + W_{in}\mathbf{u}(t) + \mathbf{b}_{in}) \quad (1)$$

$$\mathbf{z}(t) = W_{out}\mathbf{h}(t) + \mathbf{b}_{out} \quad (2)$$

$$\sigma(\mathbf{h}) = \ln(1 + \exp(\mathbf{h})) \quad (3)$$

94 All inputs, $\mathbf{u}(t)$ (20×1), enter the system and induce a specific pattern of activity, $\mathbf{h}(t)$ ($N_{rec} \times 1$), in the units
95 of the RNN (Eq. 1). We refer to this N_{rec} -dimensional vector, $\mathbf{h}(t)$, as the state of the network at time t . The
96 output, $\mathbf{z}(t)$ (3×1), is a linear projection of the state (Eq. 2). The output units indicate whether the network is
97 responding in the first dimension and in which direction on a circle the RNN responds in the next two
98 dimensions (e.g., saccade direction) (Fig.1a right). For consistency, in the majority of the paper we will focus
99 on RNNs as described by Eq. 1, using diagonal initialization of W_{rec} , the softplus nonlinear activation function
100 (Eq. 3) and L2 activity and weight regularization. We identified shared dynamical motifs in all explored
101 network designs, and include comparisons to other parameter choices and common network architectures
102 throughout. All network weights were trained to minimize the squared difference between the network output
103 and a desired (target) using back propagation through time.

104 Our approach to studying the trained RNNs was to uncover the underlying learned dynamical systems in order
105 to mechanistically understand how the networks implement computation. This approach utilizes fixed points
106 of Eq. 1^{23,24} to provide an often interpretable “skeleton” of the complex high-dimensional dynamics²⁵. By
107 studying how fixed points change as a function of the inputs that configure the task period at hand, we may
108 understand if and how these fixed point structures are repurposed for different computations. See Methods for
109 further details on network setup, training, and fixed point analysis.

110 Single Task Networks

111 To lay the groundwork to understand the multitask RNNs, we first trained individual networks to perform each
112 task in isolation. Every task began with a context period where the rule input indicated which task to perform,
113 the fixation signal was on, and there was no stimulus information (Fig.1a). Stimulus onset marked the
114 beginning of the next task period (stimulus) while all other inputs remained constant. In the absence of noise,
115 inputs to the network were piecewise constant, where every change in the inputs marked the beginning of a
116 new task period (Fig.1a vertical lines). Therefore, during each task period with unique inputs (e.g. stimulus,
117 context/memory, response) the network could be treated as a separate, autonomous dynamical system with a
118 distinct set of fixed points from other task periods. Going forward, we use *dynamical motif* to mean the high-
119 dimensional nonlinear dynamics around a fixed point skeleton that implements computation for a specified

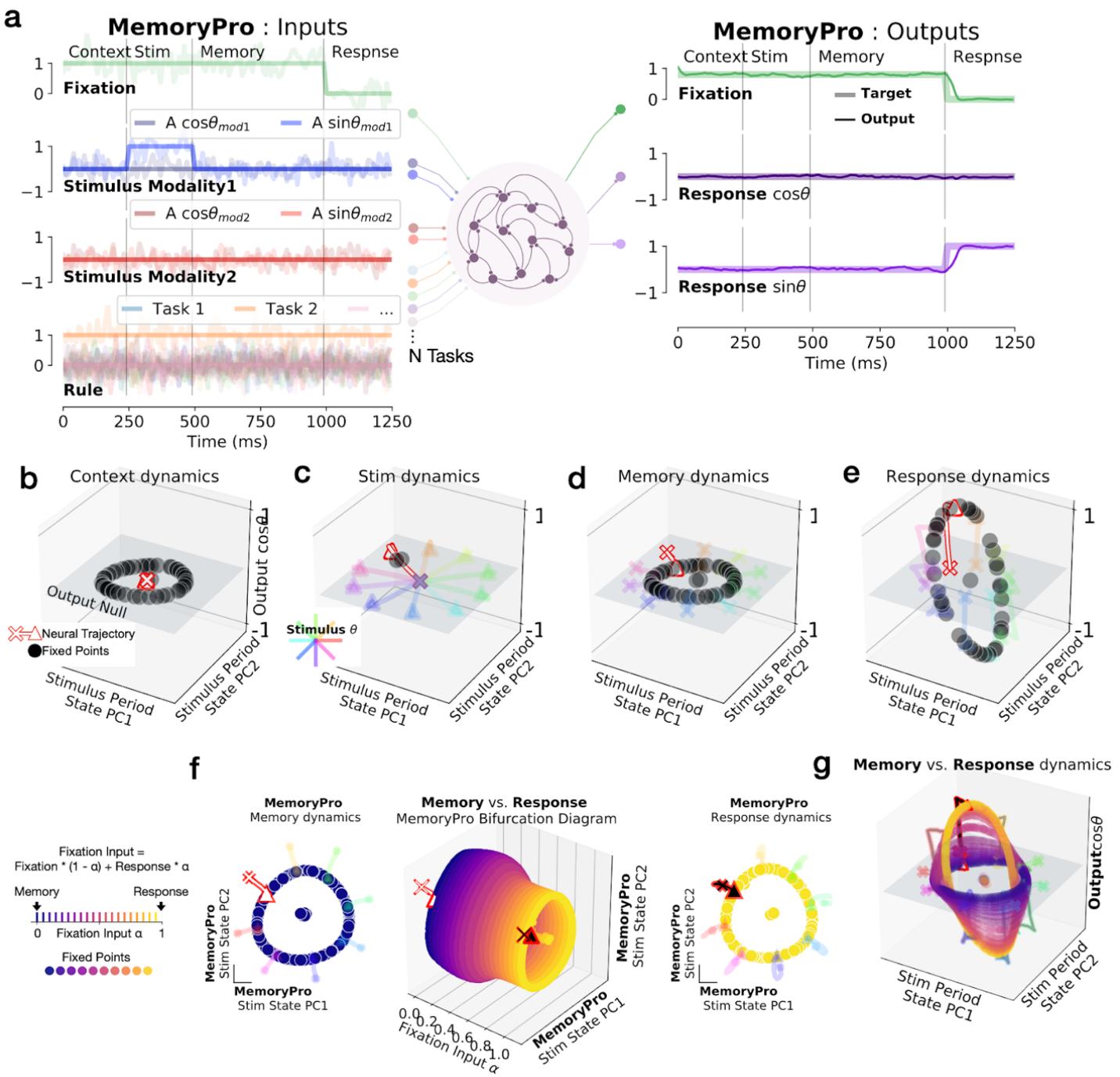
120 input. By examining locally linearized dynamics around the set of fixed points associated with a particular task
121 period, we learn about the dynamical motifs that implement computation.

122 For example, in a MemoryPro task (respond in the same direction as the stimulus after a memory period), there
123 were 4 periods (visually divided by vertical lines Fig.1a). In the first period (context), the rule input indicated
124 which task the network performed for that trial. In a network trained to perform only the MemoryPro task, the
125 context period inputs result in one fixed point at the center of a ring of fixed points (Fig.1b). The central fixed
126 point serves as an initial condition for performing the task computation during the ensuing stimulus
127 period. Notice the context period inputs are identical to the memory period (rule and fixation on, stimulus off)
128 (Fig.1a), so the fixed points are necessarily identical between these task periods. We will show later that the
129 additional ring of fixed points was relevant to the memory computation during the memory task period.

130 In the stimulus period, we examined the fixed point structure for each stimulus input separately. Stimulus
131 period state trajectories for different stimuli diverged from the central initial condition towards stimulus-
132 dependent fixed points, mapping out a stimulus representation that was orthogonal (null) to the response
133 readout dimension (Fig.1c). While fixed points were identical in the context and memory periods, the network
134 state interacted with different fixed points due to different initial conditions. During the memory period, the
135 state evolved toward a ring of fixed points that made up an approximate ring attractor (locally attracting
136 structure in all dimensions except tangent to the ring, which is neither contracting nor expanding) (Fig.1d).
137 Together, these fixed points stored the identity of the stimulus orientation based on the initial conditions of the
138 state at the beginning of the memory period (end of the stimulus period). Finally, in the response period, the
139 fixation input changed to zero and a new ring attractor emerged (Fig.1e). During the response period, the ring
140 was oriented such that it had a non-zero dot product with the output weights (W_{out}) and was therefore output
141 potent²⁶. The new ring caused the network to respond in the appropriate orientation based on the initial
142 conditions of the response period (end of the memory period). Thus, the network responded with the
143 appropriate orientation for this task ($\phi_{response} = \theta_{stimulus}$).

144 What is the relationship between the ring of fixed points in the memory and response periods? To address this
145 question, we traced locations of the fixed points as we interpolated across memory and response period inputs,
146 $(1-\alpha)u_{memory} + \alpha u_{response}$ with α in 0.05 increments between 0 and 1. We identified fixed points for each incremental
147 input setting as a function of α (see Methods Section 1.6). We employ this method frequently and call it input-
148 interpolation fixed point identification (shortened to input interpolation). By interpolating across input
149 conditions for the memory and response periods, we traced how fixed points moved and possibly changed
150 stability as the dynamical system reconfigured for different task periods.

151 We interpolated the fixation input from its memory period value (1) to its response period value (0). For every
152 intermediate input value throughout this interpolation, an approximate ring attractor was present (Fig.1f). The
153 smooth transition of this fixed point structure implies that each intermediate ring attractor was functionally the
154 same ring attractor across input conditions. In this single task network, the dynamical motif that performed
155 memory and response computations was shared across task periods. The ring attractor rotated from output null
156 space into output potent space when the fixation input changed to zero (Fig.1g). The change in inputs across
157 task periods shifted the location of the fixed points and thus changed their collective computational purpose.



158

159 **Figure 1: Single-task network shared fixed points across task periods.** (a) **left:** Noisy fixation, stimulus
160 (modality 1 and 2), and rule input time-series (overlaid without noise for clarity.) Noise was used during
161 training while analyses were performed on running the network without noise. Vertical lines divide task
162 periods: context, stimulus, memory & response. **right:** Targets (thick lines) overlaid with outputs of a trained
163 network (thin lines). (b-e) State space plots for single task network performing MemoryPro task during (b)
164 context (c) stimulus (d) memory (e) response task periods. State trajectories and fixed points projected onto
165 the first two principal components (PCs) defined by state evolution for 100 different stimulus conditions during
166 the stimulus period on the x and y-axes and the output weight vector (from W_{out}) associated with $\cos\theta_{stimulus}$ on the
167 z-axis. We visualized fixed point locations for $\theta_{stimulus}=0$ (black dots) in all subpanels of Figure 1 and additionally
168 plotted state trajectories for other stimulus conditions (see Methods Section 1.5 for further details on fixed
169 point identification). State trajectories (colored lines) colored according to stimulus orientation with $\theta_{stimulus}=0$
170 highlighted in red, starting from 'x' and ending with '▲'. (f-g) Interpolation between inputs for memory ($\alpha=0$)
171 and response ($\alpha=1$) periods (f) **middle:** Fixed points for 20 intermediate α values (x-axis) projected into top

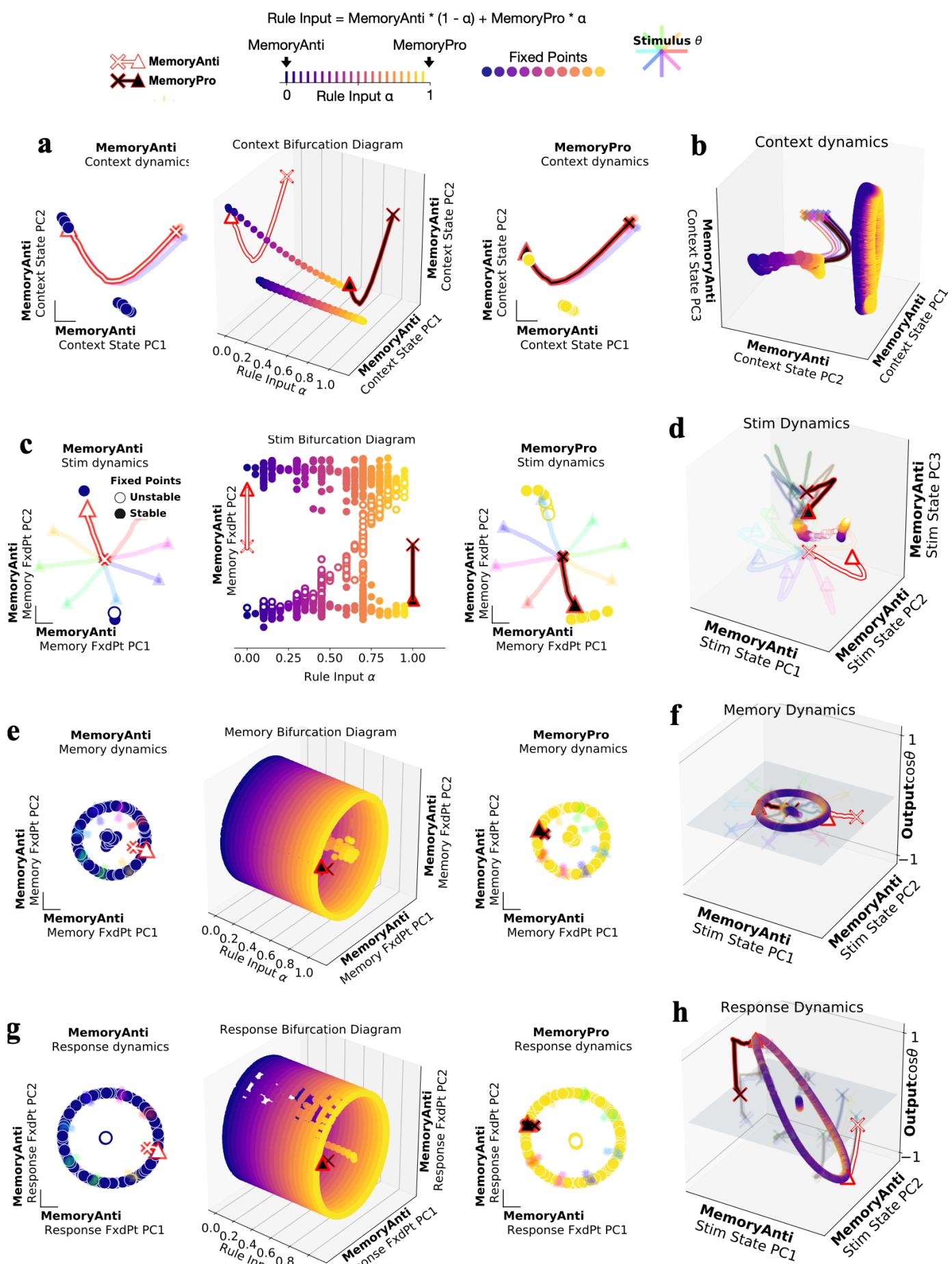
172 two stimulus period PCs (as in b-e) (y and z-axes) with (**left**) memory $\alpha=0$ and (**right**) response $\alpha=1$ fixed
173 points and trajectories. (**g**) Fixed points for input-interpolation between memory (blue) and response inputs
174 (yellow). State trajectories colored according to stimulus orientation. Same axes as (b-e).

175 Two Task Networks

176 We then simultaneously trained networks to perform two tasks on interleaved trials. We studied two-task
177 networks to learn if and how the addition of a second task changed the fixed point structures compared to
178 single task networks. The MemoryPro and MemoryAnti tasks were both memory guided response tasks that
179 received identical stimulus inputs. The target outputs in the pro task were the same as the stimulus inputs
180 ($\varphi_{\text{response}} = \theta_{\text{stimulus}}$) whereas in the anti task, targets were in the opposite direction as the stimulus ($\varphi_{\text{response}} = \theta_{\text{stimulus}} + \pi$) (see
181 Supp. Fig.1 and Methods Section 1.2 for full task definitions).

182 **Input-interpolation across rule inputs** for a network trained on the MemoryPro and MemoryAnti tasks revealed
183 shared fixed points across tasks during the context/memory, stimulus and response periods (Fig.2). Context
184 period fixed points were similar to the single task network throughout rule input interpolation, with one stable
185 fixed point that was relevant to the context period and a ring of fixed points that was relevant to the memory
186 period (Fig.2a-b,e-f). Stimulus period rule input interpolation revealed two separate stable fixed points and an
187 unstable fixed point between the basins of attraction for each intermediate input condition (Fig.2c-d). The
188 network state evolved away from the unstable fixed point, which smoothly moved in state space across
189 interpolated input conditions, resulting in the network state evolving toward a different stable fixed point for
190 each task. From that point onward, the state interacted with a shared ring attractor across both tasks
191 (MemoryPro and MemoryAnti) and task periods (memory and response) according to the response direction
192 (Fig.2e-h). This network flexibly performed two related computations through small changes in fixed point
193 locations. In addition to shared fixed points across different tasks and task periods, we could identify shared
194 fixed points across different stimulus conditions for the same task period (Supp Fig.2a-c). In this work, we will
195 explore how these changes in the location of shared fixed points enable a single network to perform different
196 computations and why this structure results in a compositional organization of tasks in multitask networks.

197 One might expect that networks share fixed points due to the limited computational resources in small
198 networks. We therefore trained networks that were nearly an order of magnitude larger and without noise to
199 determine how abundant computational resources might change this solution. To our surprise, we found even
200 large networks without noise still shared **dynamical motifs** (see Supp Fig.2d-k). We interpret these findings to
201 mean shared dynamical motifs are not a product of limited resources and rather provide a more universal
202 solution for multitask computation.



204 **Figure 2. Two-task networks shared fixed points across related tasks.** Fixed points for interpolation
205 between inputs for MemoryAnti ($\alpha=0$) and MemoryPro ($\alpha=1$) tasks during **(a,b)** context **(c,d)** stimulus **(e,f)**
206 memory and **(g,h)** response periods. **(a middle:** Fixed points for 20 intermediate α values (x-axis) projected
207 into top two PCs defined by state evolution during the context period of the MemoryAnti task (y and z-axes)
208 with **(left)** MemoryAnti $\alpha=0$ and **(right)** MemoryPro $\alpha=1$ fixed points and trajectories. **(b)** Fixed points for
209 rule input-interpolation between tasks, MemoryAnti (blue fixed points, white state trajectory) and MemoryPro
210 (yellow fixed points, black state trajectory) projected into the top three MemoryAnti context period state
211 evolution PCs. **(c)** Same as a for stimulus period, with unstable (open) and stable (closed) fixed points projected
212 into top PCs defined by fixed points during the memory period of the MemoryAnti task (y-axis) **(d)** Same as
213 b for stimulus period, projected into the top three MemoryAnti stimulus period state evolution PCs. **(e)** Same
214 as a for memory period, projected into top two PCs defined by fixed points during the memory period of the
215 MemoryAnti task (y and z-axes) **(f)** Same as b for memory period, projected into the top two MemoryAnti
216 stimulus period state evolution PCs (x and y-axes) and the output weight vector (from W_{out}) associated with
217 $\cos\theta_{stimulus}$ on the z-axis. **(g)** Same as a for response period, projected into top two PCs defined by fixed points
218 during the response period of the MemoryAnti task (y and z-axes) **(h)** Same as f for response period.

219 Identifying Dynamical Motifs in 15 Task Networks

220 To quantify shared structure across a large number of tasks in a single network and also to compare this shared
221 structure across multiple networks, we developed a modified version of the task variance metric described by
222 Yang et al. 2019²⁰. We were motivated to study task periods because changes in the inputs reconfigure the
223 RNN's dynamics across different task periods. For example, when the stimulus input turns off in some tasks,
224 the network goes from processing a stimulus to maintaining a memory of the stimulus. Aside from training
225 noise, the inputs are static within a task period and therefore the RNN's dynamics are fixed. Task periods,
226 therefore, provide the relevant granularity to identify the dynamical motifs that perform distinct computations.

227 We divided tasks into task periods and computed the variance across stimulus conditions for each unit,
228 normalized across all task periods (see Methods Section 1.9). The result of this variance analysis was a matrix
229 of each unit's normalized variance for each task period of every task (Fig.3a), which we refer to as the variance
230 matrix in subsequent analyses. We sorted the rows and columns of this matrix based on similarity to better
231 visualize its inherent structure (see Methods Section 1.10).

232 The variance matrix can be considered as a map of functional specialization in the network, similar to the
233 BOLD signal in an fMRI scan, or widefield imaging data. This analysis could be performed directly on neural
234 data to learn about the computational substructure for multiple tasks in a behaving animal. Sorting the rows
235 and columns of the variance matrix revealed a blockwise structure, where groups of units had large variance
236 for groups of task periods with similar computational requirements (Fig.3a). Similar computations can be seen
237 in the task period color labels (*Stimulus1*, *Stimulus2* - for tasks with two sequential stimulus presentations,
238 *Memory*, *Response*, etc) and in the task names (Category, DecisionMaking, Memory, etc) (Fig.3a, left). For
239 example, task period cluster #2 (Fig.3a, right) corresponds to reaction timed response task periods (see Supp.
240 Fig.1 for definitions of all tasks). These tasks receive new stimulus information during a response period that
241 must be incorporated into the computation immediately. Therefore, the network cannot prepare a response
242 direction before the fixation cue disappears. On the other hand, in task period cluster #9, the network receives
243 no new information during the response period and must instead use the memory of the stimulus to produce
244 the correct output during the response period. These separate blocks in the variance matrix reveal two distinct
245 clusters of units that contribute to response period dynamics: one for tasks with reaction timed responses and
246 another for tasks with memory guided responses. Other unit clusters for stimulus (unit cluster k-o, task period
247 cluster #4-5) and memory (unit cluster a-b, task period cluster #10) computations are apparent in the block-
248 like structure aligned with task period type (Fig.3a task titles and task period color labels to the left of the
249 variance matrix).

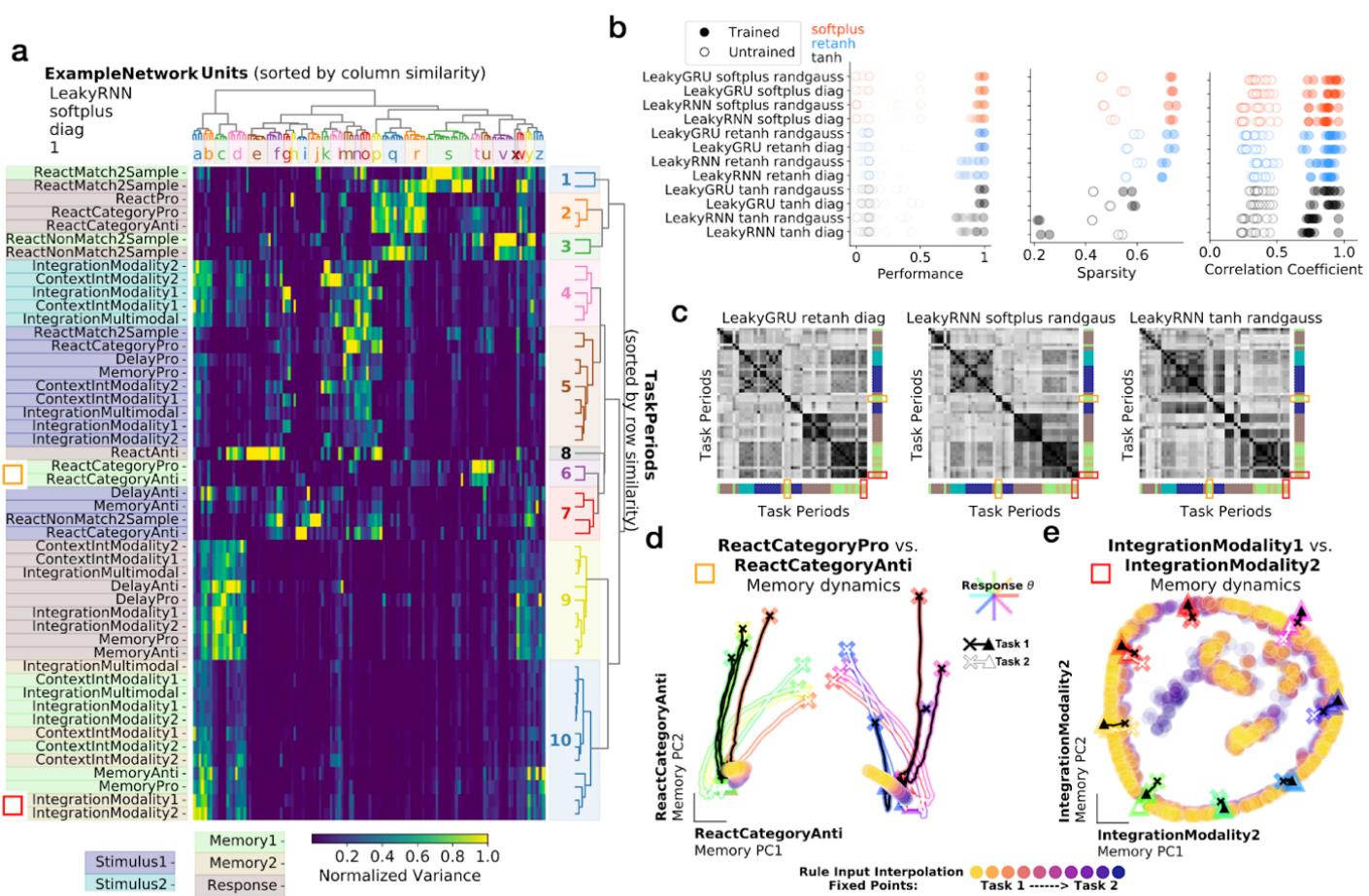
250 Block structure in the variance matrix was robust to different network architectures and hyperparameters
251 (examples in Supp. Fig.3). To quantify task period similarity across networks with different design choices,
252 we calculated the variance matrix for each trained network. We then sorted task periods according to similarity
253 of rows in one reference network and computed the correlation matrix of the sorted rows in the variance matrix
254 for each network (see Methods Section 1.9). The correlation matrix for each trained network revealed the block
255 wise similarity of task periods (Fig.3c). By comparing the correlation matrices across networks (Pearson
256 correlation coefficient between correlation matrices, see Methods Section 1.9), we found that trained networks
257 were more correlated to every other trained network than to untrained networks (Fig.3b right). Higher
258 correlation across trained networks compared to untrained networks confirmed the block structure in the
259 variance matrix emerged from learning the task computations rather than from network design choices or the
260 structure of the inputs.

261 As we will show, the clusters of the variance matrix arise from shared dynamical motifs across task periods.
262 This shared structure takes the form of attractors, decision boundaries and rotations. We highlight two different
263 examples of shared memory dynamical motifs using rule input interpolation (Fig.3d-e) and highlight their
264 positions in the variance matrix (Fig.3a left of task period names, red and yellow squares). A pair of memory
265 category task periods are within the same cluster in the variance matrix, suggesting their computations are
266 performed by a similar set of units (Fig.3a yellow square left: task period cluster #6, unit cluster t). Both
267 category tasks utilized the same two point attractors for memory of the initial stimulus. Rather than store the
268 identity of the initial continuous circular stimulus, the network stored which category it must respond to,
269 regardless of task (Fig.3d). In another example of a shared attractors across tasks, we found a ring attractor that
270 was shared across several tasks (Fig.3e, Fig.3a task period cluster #9-10, unit clusters a-d). All of these tasks
271 required the memory of the initial continuous circular stimulus variable. To show that this ring attractor was
272 shared across tasks, we interpolated across rule inputs for a pair of these tasks (IntegrationModality1 and
273 IntegrationModality2) and found a similar shared ring structure as in the two task networks (Fig.2e). We
274 highlight shared category and continuous memory dynamical motifs in networks with different activation
275 functions in Supp. Fig.4.

276 In addition to clusters of task periods with similar variance, there are also some task periods that do not cluster
277 with other task periods. These can be seen in the variance matrix as isolated row segments with high variance
278 and in the correlation matrices as rows with low correlation across all other task periods. For example, task
279 period cluster #8 is dedicated to the ReactAnti task, cluster #1 is dedicated to the ReactMatch2Sample task and
280 cluster #3 is ReactNonMatch2Sample (Fig.3a). In these cases, the computation performed in the unique task
281 period is so distinct from other computations the network performs, the dynamical motif is not reused across
282 tasks. The set of tasks that employed unique dynamical motifs was similar across hyperparameter settings
283 (Fig.3c, Supp. Fig.3).

284 Motif Alignment to Unit Axes

285 One notable difference across network hyperparameters was sparsity in the variance matrix. We define sparsity
286 to be the fraction of entries in the variance matrix below a threshold of 15% maximum unit variance. Networks
287 with non-negative nonlinear activation functions had sparse task variance matrices whereas networks with the
288 *tanh* activation function, which has a range of (-1, 1) did not (Fig.3b middle). We understand this sparsity to
289 be a function of optimal network performance requiring potentially interfering dynamical motifs to be
290 organized into orthogonal subspaces. In a network where all units can take only positive values, this
291 orthogonalization can only occur along unit axes (Supp. Fig.5), thus resulting in sparse variance matrices for
292 networks with non-negative activation functions. We find clusters to be present in *tanh* networks, simply not
293 aligned to unit axes and therefore non identifiable using methods described in Yang et al. 2019²⁰. By examining
294 the correlation matrix and the correlation coefficient across networks, we see that similar clusters are present
295 in the *tanh* networks (Fig.3b right, c right).



296

297 **Figure 3. Modular organization in 15 task networks was not dependent on activation function or**
 298 **network initialization.** (a) Variance matrix: variance of unit activations across stimulus conditions,
 299 normalized across task periods (columns normalized by the maximum entry in each column). Rows and
 300 columns sorted according to similarity (see Methods Section 1.10). (b) left: Performance across all tasks for 3
 301 networks of each of 12 hyperparameter settings. middle: Sparsity of the task variance measured as the fraction
 302 of entries >15% maximum unit variance. right: Task period correlation matrix (examples shown in c) for
 303 trained and untrained networks are sorted according to rows in a and correlated to trained networks for all other
 304 hyperparameter settings. (c) Correlation matrix of rows in variance matrix (as in a) for 3 different example
 305 networks; rows and columns sorted according to rows in a. Orange and red rectangles highlight discrete
 306 memory (ReactCategoryPro, ReactCategoryAnti) and continuous memory (IntegrationModality1,
 307 IntegrationModality2) tasks respectively. (d) Shared point attractors for two category memory tasks as seen by
 308 input interpolation across tasks during memory period. State trajectories for 8 stimulus conditions (colored by
 309 response direction) starting from ‘x’ in ReactCategoryAnti state evolution PC space for ReactCategoryPro
 310 (black) and ReactCategoryAnti (white) tasks. Rule input interpolation across tasks during memory period with
 311 fixed points for intermediate rule input conditions in filled circles. (e) Same as d for two continuous circular
 312 variable memory tasks, highlighting shared ring attractors. State trajectories starting from ‘x’ in
 313 IntegrationModality2 state evolution PC space for IntegrationModality1 (black) and IntegrationModality2
 314 (white) tasks.

315

Shared Stimulus Period Dynamical Motifs in 15 Task Networks

316 The variance matrix provides a useful overview of which task periods are implemented by similar clusters of
 317 units, but falls short of addressing exactly *how* these subpopulations implement shared dynamical motifs.
 318 Shared motifs are implemented by organizing the state in the appropriate region of state space to evolve on the
 319 relevant shared dynamical landscape. To walk through this explanation in detail, we focus on stimulus period

320 dynamics and highlight two examples, one in which dynamical motifs are shared and another where motifs are
321 not shared.

322 Tasks with similar stimulus computations (noisy integration, pro vs. anti, reaction-timed vs. delayed response)
323 organized their initial conditions for the stimulus period to be nearby in state space and evolved in a similar
324 way after stimulus onset (see schematic Fig.4a). We visualized this organization of stimulus period initial
325 conditions in principal component (PC) space defined by the final state of the context period across all tasks
326 (Fig.4b). To summarize the relationship between initial conditions and the ensuing stimulus dynamics for
327 different tasks, we compared pairs of trials presented with the same stimulus across different tasks. We plotted
328 the Euclidean distance between initial conditions against the angle between the state evolution on the first time
329 step for these pairs of trials (Fig.4c). We observed that pairs of tasks with similar computations had initial
330 conditions that were closer together and had smaller angles between state trajectories on the first time step of
331 the stimulus period compared to pairs of tasks with distinct computations. Similar initial conditions for stimulus
332 onset resulted in shared context dependent stimulus amplification in some networks (Supp Fig.6). In these
333 cases, the state update was scaled in magnitude according to whether the stimulus input was either modality
334 one or two, dependent on the position of that state at stimulus onset.

335 The relationship between context period states and stimulus period trajectory angles across tasks support the
336 idea that nearby initial conditions allowed tasks with similar stimulus computations to reuse the same
337 dynamical landscape and therefore evolve in similar ways. To provide further detail, in (Fig.4d-i) we examine
338 these features in two examples of comparisons between tasks that (1) share the same stimulus period dynamical
339 motif and then tasks that (2) do not share the same dynamical motif.

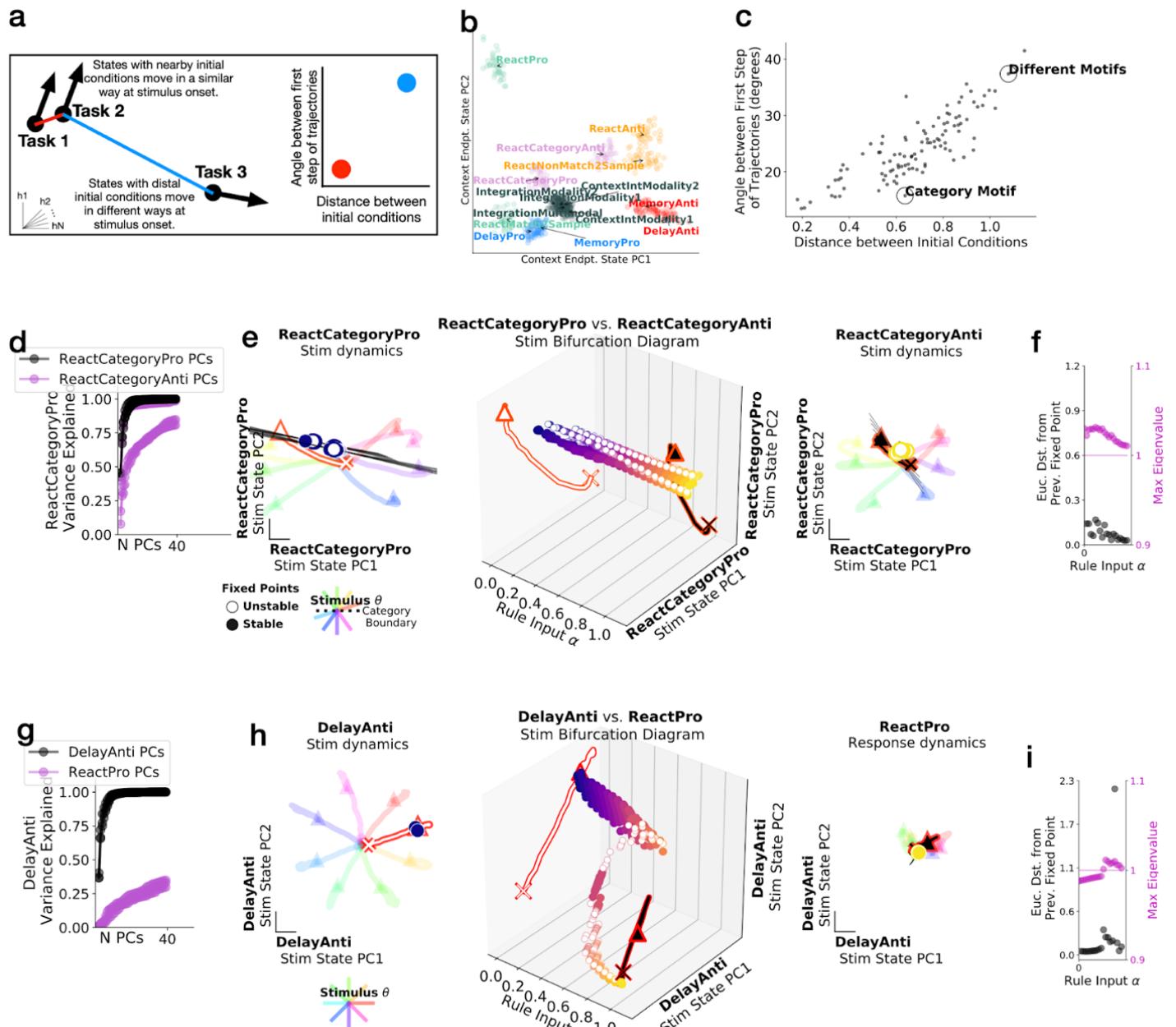
340 In the case of the two categorization tasks, ReactCategoryPro and ReactCategoryAnti, we found a shared
341 stimulus motif (Figure 4d-f). In the ReactCategoryPro task, the network was trained to respond if both
342 sequential stimuli were less than or both greater than π ; whereas, in the ReactCategoryAnti task the network
343 was trained to respond if stimuli were on opposite sides of π . In either task, there was a decision boundary at
344 $\theta_{stimulus} = \pi$. The initial conditions for these tasks were nearby and trajectories during the stimulus period were
345 aligned (Fig.4c ‘Category Motif’). We quantified stimulus response overlap by computing the fraction of
346 variance explained for the evolution of the state trajectory on one task by the other task’s PCs (purple)
347 compared to its own PCs (black) (Fig.4d), revealing that both tasks were performed in an aligned subspace.
348 Aligned stimulus responses for both category tasks were visualized in PC space defined by the stimulus period
349 state trajectories of the ReactCategoryPro task (Fig.4e left, right). These analyses revealed that activity evolved
350 in a qualitatively similar way for trials with the same stimulus conditions on both tasks, suggesting the state
351 evolution could have occurred on a similar dynamical landscape.

352 To better understand the relationship between the dynamical landscapes across task contexts, we interpolated
353 across rule inputs during the stimulus period for both category tasks with the same stimulus input. We
354 visualized the fixed points for each intermediate rule input configuration (Fig.4e middle). We found that similar
355 stimulus responses were governed by shared stable and unstable fixed points, demonstrated by the smooth
356 bridge of fixed points between both tasks. We projected the unstable dimension of each unstable fixed point
357 into PC space and found this dimension was aligned with the direction of the state evolution and roughly
358 orthogonal to the decision boundary. (Fig.4e). We defined the most relevant fixed point to be the closest
359 unstable fixed point to the state at the end of this task period. This simplification of one relevant fixed point
360 was often necessary to tease apart how relevant dynamics are reconfigured across tasks while several to
361 hundreds of other fixed points related to computations during other task periods also moved through state
362 space. A continuous bridge mapped movement of the relevant fixed point, suggesting that rule inputs shifted a
363 relevant shared fixed point that was reused across both tasks (Fig.4e). We quantified the distance between
364 consecutive locations of the relevant fixed point for each interpolated input, highlighting the smooth and small
365 transition of the fixed point location across tasks (Fig.4f). Moreover, the stability of the local linear dynamics
366 around this shared fixed point was consistent across all intermediate input conditions, as shown by the the
367 maximal real part of the eigenvalue of the linearized RNN state update around each interpolated fixed point

368 location (see Methods Section 1.7) (Fig.4f). This real part of the eigenvalue does not pass below the threshold
369 of 1, where the fixed point would become stable. We interpret this result to mean that both category tasks reuse
370 the unstable fixed point that moves the state away from the category boundary.

371 The DelayAnti and ReactPro tasks were an example pair that did not share any dynamical motifs (Fig.4g-i).
372 The DelayAnti task began with the context period, followed by a stimulus presentation that signaled the
373 opposite response direction ($\varphi_{\text{response}} = \theta_{\text{stimulus}} + \pi$), followed by a go cue that signaled when to initiate a delayed
374 response (see Supp. Fig.1 for all task definitions). The ReactPro task began with the context period, followed
375 by a stimulus presentation that signaled the same response direction and required an immediate response.
376 During the context period, the network state evolved toward dissimilar locations for trials of either task and
377 trajectories during the stimulus period were not aligned (Fig.4c ‘Different Motifs’). We defined the subspace
378 for the ReactPro task by performing principal components analysis on the state trajectories during the stimulus
379 period. We projected the DelayAnti task in the same subspace and found that very little variance was captured
380 by the other task PCs (Fig.4g). We also visualized both tasks in a subspace defined by the first two PCs of the
381 DelayAnti task and again found very little overlap, suggesting both tasks evolved in mostly non-overlapping
382 subspaces (Fig.4h right). We interpolated across these two rule inputs during the stimulus period, revealing
383 that there was a bifurcation where the relevant stimulus-dependent fixed point did not form a continuous bridge
384 across interpolated rule inputs (Fig.4h middle). We quantified the distance between consecutive fixed points
385 that were closest to the endpoint of the state trajectory for each interpolated input and identified a large discrete
386 jump in the location of the relevant fixed point (Fig.4i). We visualized the maximal real part of the eigenvalues
387 of the linearized RNN state update around each consecutive fixed point, revealing dissimilar local dynamics
388 around fixed points for interpolated input conditions (Fig.4i).

389 Taken together, these features suggest that shared dynamical motifs are implemented by evolving the state to
390 the appropriate region of state space such that it interacts with a shared fixed point across similar task
391 computations. Category tasks shared both unstable and stable fixed points. On the other hand, stimulus period
392 dynamics for the DelayAnti and ReactPro tasks evolved in separate subspaces and were governed by different
393 stable fixed points. These analyses revealed that shared structure was not merely an artifact of all tasks within
394 the same network. Rather only tasks with similar computations implemented shared dynamical motifs. All
395 subspace and distance analyses in this section could be experimentally tested on neural data of animals
396 performing multiple tasks.



397

398 **Figure 4. Tasks with similar stimulus computations were in nearby parts of state space and shared**
399 **dynamical motifs.** (a) Schematic of analyses in b and c. (b) The state for each trial (colored dot for each of
400 20 trials on each task) at the end of the context period (just before stimulus period) projected onto the top two
401 PCs defined by the state at the end of the context period for all tasks. Trials colored by similar stimulus
402 computations as given by task definitions: delayed response pro (blue), delayed response anti (red), noisy
403 integration (gray), categorization (pink) reaction-timed pro (teal) and reaction-timed anti (orange). (c)
404 Euclidean distance between pairs of trials from different tasks at the end of the context period plotted against
405 cosine angle between same pair after stimulus onset for a particular stimulus input for one timestep, then
406 averaged across stimulus angle inputs. Pairs of tasks in d-i circled and labeled, (d-f) ‘Category Motif’:
407 ReactCategoryPro and ReactCategoryAnti, (g-i) ‘Different Motifs’: DelayAnti and ReactPro (d) Fraction of
408 variance explained for ReactCategoryPro task by the ReactCategoryAnti task PCs (purple) compared to its
409 own PCs (black) for 5 trained networks with different random seeds. (e) Rule input interpolation across
410 category tasks for one stimulus angle. **middle:** Unstable (open) and stable (closed) fixed points for 20
411 intermediate α values (x-axis) projected onto top two PCs defined by state evolution during the stimulus period
412 of the ReactCategoryPro task (y and z-axes) with (**left**) ReactCategoryPro $\alpha=0$ and (**right**) ReactCategoryAnti
413 $\alpha=1$ fixed points and trajectories for 8 different stimulus angles (rainbow colors) Expanding dimensions

414 around unstable fixed points are visualized as black lines. **(f)** Euclidean distance between fixed points (black)
415 and maximum real eigenvalue for the linearization of the state update around each fixed point (purple) for the
416 single unstable fixed point closest to the state at the end of the stimulus period for 20 consecutive α values
417 between 0 and 1. We analyzed only one unstable fixed point that is most proximal to the end of the state
418 trajectory for each input condition (see Methods Section 1.7). **(g-f) ‘Different Motifs’:** DelayAnti and
419 ReactPro. Same as (d-i), but for DelayAnti and ReactPro tasks. **(i)** Euclidean distance between fixed points
420 (black) and maximum real eigenvalue for the linearization of the state update around each fixed point (purple)
421 for the single fixed point closest to the state at the end of the stimulus period for 20 consecutive α values
422 between 0 and 1. We analyzed one fixed point that is most proximal to the end of the state trajectory for each
423 input condition.

424 **Dynamical Motifs Result in Modular Lesion Effects**

425 The authors of Yang et al. 2019²⁰ previously found that network lesions affected sets of tasks that shared
426 computational features. For example, if the output of a particular cluster of units was set to zero, then all tasks
427 involving a particular computation decreased their performance, while other tasks were unaffected. Their work
428 left open the major question of *why* lesion effects were modular. We identified the cause of these modular
429 lesion effects to be related to the underlying modular dynamical motifs that perform computation.

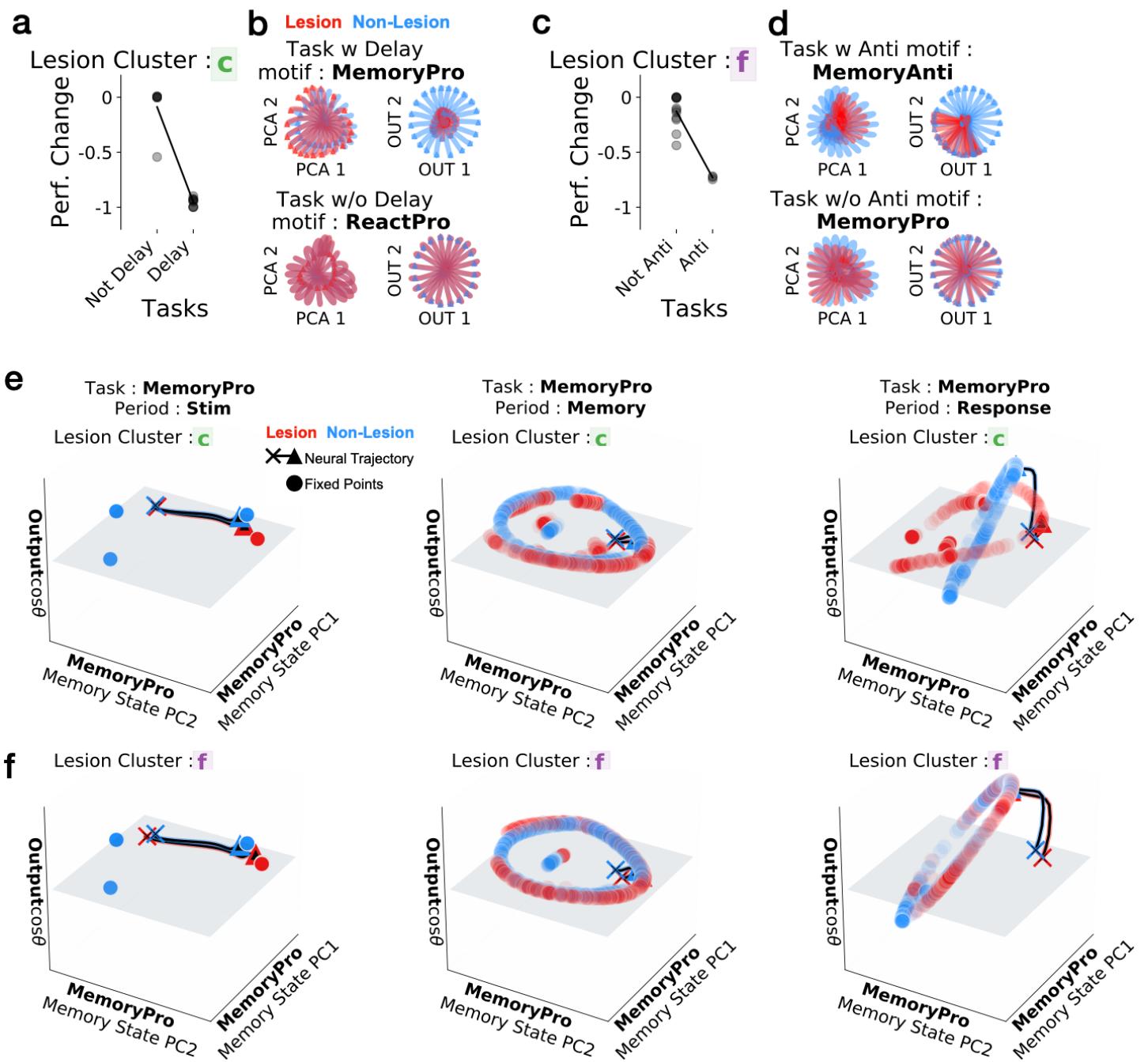
430 We examined the impact of lesioning clusters of units described in the variance matrix in Fig.3a. We lesioned
431 a cluster of units by setting the output of all units within the cluster to be zero throughout a given trial. Clusters
432 were identified by performing hierarchical clustering on the columns and rows of the variance matrix and
433 identifying a distance criterion to maximize the ratio of intercluster to intracluster distances, which resulted in
434 clusters of units a-z (Fig.3a, Methods Section 1.10).

435 Many unit clusters had high variance for a set of task periods with similar computations. For example, unit
436 clusters a and c had high variance for memory and response task period clusters 10 and 9 respectively (Fig.3a).
437 Other unit clusters had high variance for modality 2 stimulus periods (unit cluster k), or anti stimulus periods
438 (unit cluster f), etc.

439 In two example lesions, we demonstrate that each unit cluster lesion impacted only a subset of tasks that shared
440 computational features where units had high variance, reproducing a finding previously reported in Yang et al.
441 2019. Here, we show that lesions either did or did not impact task-relevant computations depending on whether
442 the relevant underlying dynamical motif was impacted (Fig.5). A lesion to one cluster only impacted
443 performance on tasks that included a delay period (Fig.5a,b,e). We visualized state trajectories for a task that
444 did include a delay period (MemoryPro) and a task that did not include a delay period (ReactPro) in lesioned
445 (red) and non-lesioned networks (blue)(Fig.5b). State trajectories were dramatically impacted when the delay-
446 related dynamical motif was lesioned during the performance of the MemoryPro task, but not for the ReactPro
447 task. In a second example, we lesioned a cluster of units with high variance during the performance of anti-
448 response tasks and found there was little to no change in the performance or state trajectories of pro-response
449 tasks (Fig.5c,d,f).

450 When a unit cluster was lesioned, all task periods that shared the relevant dynamical motif were impacted.
451 When tasks did not utilize the dynamical motif impacted by a unit cluster lesion, they were not impacted by
452 the lesion. For example, MemoryPro stimulus period activity was not impacted by a lesion to the unit cluster
453 which implemented the memory and response period dynamical motif of the same task (Fig.5e left). The state
454 evolved toward a stable fixed point in approximately the same location in both the lesioned (red) and full
455 network (blue) after this cluster was lesioned. However, this lesion resulted in discontinuity of the relevant
456 ring attractor in the memory and response periods and minimal rotation into output potent space during the
457 response period (Fig.5e middle, right). Conversely, there was little change in the fixed point structure after a
458 lesion to the anti-stimulus motif (cluster f) for all task periods during the MemoryPro task (Fig.5f). Taken

459 together, results from our lesion studies suggest that modular lesion effects are a result of modular fixed point
 460 structures that implement dynamical motifs.



461

462 **Figure 5. Unit cluster lesions had modular effects on task period clusters that shared the same dynamical**
 463 **motif. (a,c)** Fraction performance change after lesion (by setting unit output to zero) to a cluster of units with
 464 **high variance on (a)** delayed-response (cluster c in Fig.3a) **(c)** anti-response (cluster f in Fig.3a) relevant task
 465 **periods. (b,d)** State evolution in **(left)** PC space defined by the state evolution for the non-lesioned network
 466 **and (right)** output weight space (from W_{out}) for **(top)** task with relevant computational feature (i.e. delay period,
 467 **anti-response**) and **(bottom)** task without relevant computational feature for lesioned (red) and non-lesioned
 468 (blue) trials. **(e)** Fixed points and state trajectories for $\theta_{stimulus}=0$ with a lesion to cluster c (red) and non-lesioned
 469 (blue) networks (same as in panel a,b) during **(left)** stimulus, **(middle)** memory and **(right)** response periods
 470 for MemoryPro task projected onto the first two PCs (x and y-axes) defined by state evolution during the
 471 memory period of the MemoryPro task without lesion and the output weight vector (from W_{out}) associated with

472 $\cos\theta_{stimulus}$ on the z-axis. State trajectories in black emanating from ‘x’ and ending with ‘▲’. (f) Same as e with
473 lesion to cluster f.

474 Fast Learning of Novel Tasks by Reusing Dynamical Motifs

475 Networks were able to rapidly learn new tasks sequentially by reconfiguring previously learned dynamical
476 motifs. We first identified a task where each task period shared a dynamical motif with at least one of the other
477 14 tasks: MemoryAnti. In this task, the network uses the anti stimulus motif (as shown in two task network,
478 Fig.2c) and the delayed-response memory motif (as in one and two task networks Fig.1f, 2e,g). We next trained
479 a network to perform every task except the MemoryAnti task. After learning all input, recurrent and output
480 weights and biases for other tasks, we trained only the one-hot rule input weights to learn the MemoryAnti
481 task. This vector of length N_{rec} maps a single rule input onto the recurrent weights (Fig.6a). By only training the
482 single rule input weights, we did not interfere with any previously learned dynamical motifs that were
483 constructed in the recurrent weight matrix, W_{rec} , enabling learning of new tasks without catastrophic forgetting.
484 The network was able to learn the MemoryAnti task when previously trained on all other tasks (Fig.6b, black).

485 To determine if the anti stimulus motif and delayed response motif are sufficient for this effect, we pre-trained
486 a new network on a key set of tasks that included these motifs: DelayAnti and MemoryPro. These networks
487 could also sequentially learn the MemoryAnti task with single rule input training (Fig.6b, blue). Pre-training
488 on this minimal set of tasks with relevant dynamical motifs learned as fast and to similar proficiency as
489 networks pretrained on all tasks, suggesting the relevant dynamical motifs were sufficient for this effect.
490 Conversely, if we pretrained the network on two pro tasks that did not include the anti motif (orange) or if we
491 started from a network with no pre-training (green), networks required many more training steps or could not
492 learn the MemoryAnti task at all (Fig.6b).

493 Without pre-training on any tasks that included the anti computation, the network had to learn a new dynamical
494 motif by modifying only the N_{rec} -dimensional rule input vector. This resulted in stimulus period state trajectories
495 that were not highly overlapping with previously learned tasks (Fig.6c). The MemoryAnti stimulus period
496 relevant fixed point was distinct from the previously learned DelayPro stimulus period relevant fixed point
497 (Fig.6e,f). However, the network was pre-trained on a task that included a relevant memory period dynamical
498 motif, the MemoryPro task. Despite learning a new stimulus period anti motif, the network was still able to
499 reuse the previously learned memory motif. The state converged on the previously learned ring attractor
500 (Fig.6g-j). This result highlights the modularity of dynamical motifs.

501 Networks that were pre-trained with the relevant dynamical motifs reused the anti-stimulus and memory
502 dynamical motifs for fast learning of the novel MemoryAnti task. We found that MemoryAnti state trajectories
503 were in highly overlapping subspaces with the DelayAnti task during the stimulus period (Fig.6k-l) and with
504 the MemoryPro task during the memory period (Fig.6o,p). Rule input interpolation between both anti tasks
505 during the stimulus period (Fig.6m) and memory tasks during the response period (Fig.6q) provided strong
506 evidence that the fixed point structures were shared.

507 Tasks with unique motifs that were not shared with other tasks could not be learned as well using this pre-
508 training method compared to full network training. We trained networks on all tasks except one, for each task,
509 and then trained the rule input weights for the held out task. We compared the training cost of these networks
510 to single task networks where all weights were trained on only the held out task (Supp Fig.7a,b).

511 We wanted to better understand the relationship between how well the pre-training method works for a given
512 task and the uniqueness of its relevant dynamical motifs. We identified tasks with unique response period
513 motifs using our task period variance matrix (Fig.3a), as rows with low correlations to other rows (task periods
514 with low correlation to other task periods) (See Methods Section 1.12 for details). Three response task periods
515 that were dissimilar from all other response periods could not be learned as well by rule input training alone

(Supp Fig. 7c). In summary, we found that rapid learning was not possible in the context of novel tasks with unique dynamical motifs. These results provide support that rapid learning of novel tasks required reconfiguration of relevant previously learned dynamical motifs.

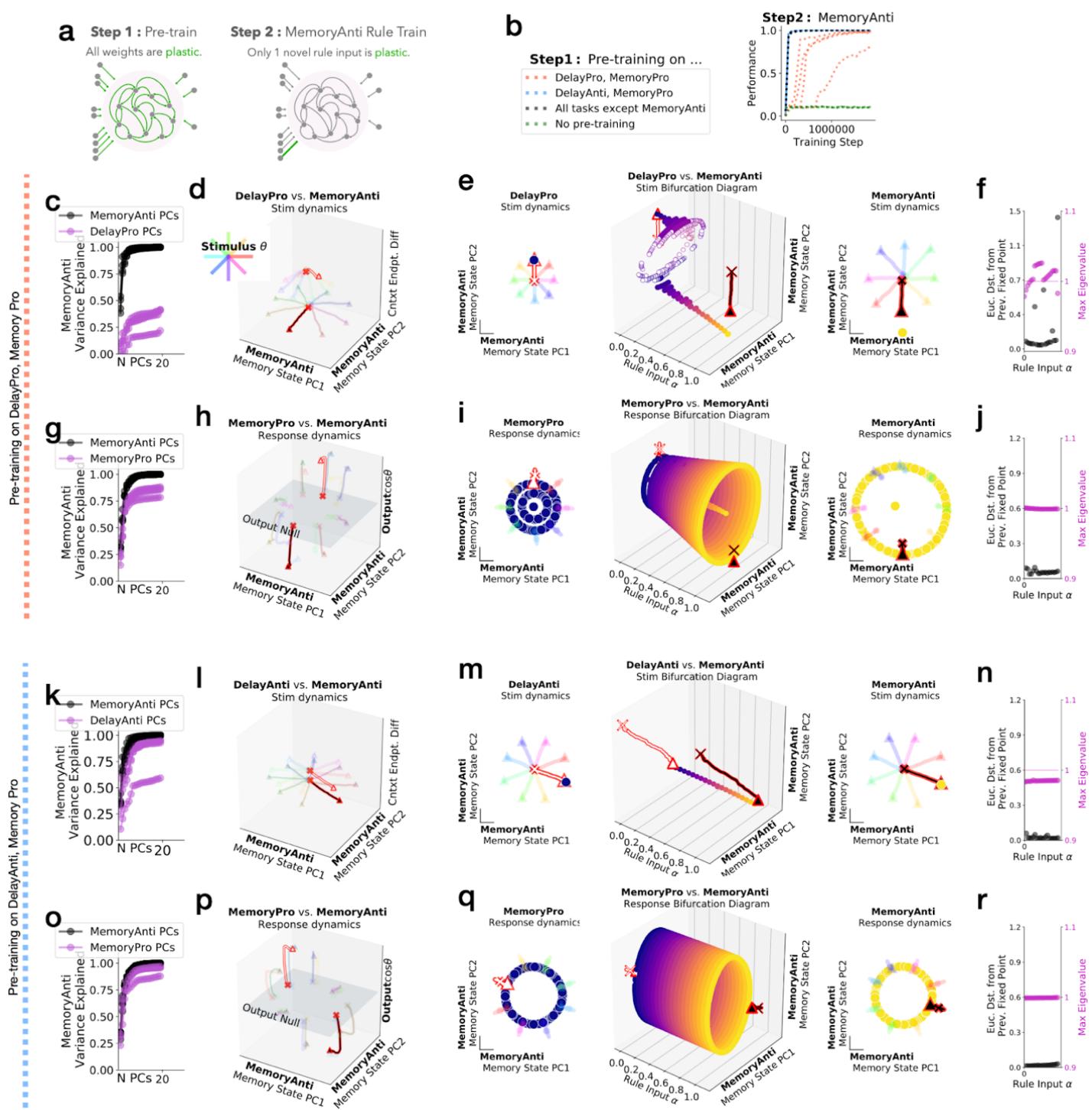


Figure 6. Dynamical motifs were reused for fast learning of novel tasks with familiar computational elements. (a) Schematic of two stage learning. Networks were pre-trained on a set of tasks while all weights were plastic. The same network was then trained on a novel task by only learning a single one-hot rule input. (b) **left:** Networks were pre-trained on two tasks that include pro and memory motifs (orange), anti and memory motifs (blue), all motifs (black), no motifs (green). **right:** Performance during MemoryAnti task rule input training after pre-training on various sets of motifs for five different networks each. (c-j) Network pre-trained on DelayPro and MemoryPro tasks (pro and memory motifs) was then trained to perform

527 MemoryAnti through weight changes to the MemoryAnti rule input (length N_{rec} vector). **(c)** Fraction of
528 MemoryAnti stimulus period variance explained by MemoryAnti stimulus period PCs (black) and DelayPro
529 stimulus period PCs (purple) quantifies the extent to which both pro and anti tasks are in a similar subspace
530 during the stimulus period. **(d)** Stimulus period state trajectories for DelayPro (white) and MemoryAnti
531 (black) tasks for 8 different stimulus angles (rainbow colors) projected into PC space (x and y axes) defined
532 by state trajectories for 100 different stimulus angle inputs during MemoryAnti task and context period state
533 endpoint difference between both tasks (z-axis). **(e)** Rule input interpolation across tasks for one stimulus
534 angle. **middle:** Unstable (open) and stable (closed) fixed points for 20 intermediate α values (x-axis)
535 projected onto top two PCs defined by state evolution during the memory period of the MemoryAnti task (y
536 and z-axes) with **(left)** DelayPro $\alpha=0$ and **(right)** MemoryAnti $\alpha=1$ fixed points and trajectories for 8
537 different stimulus angles (rainbow colors) **(f)** Euclidean distance between fixed points (black) and maximum
538 real eigenvalue for the linearization of the state update around each fixed point (purple) for the single fixed
539 point closest to the state at the end of the stimulus period for 20 consecutive α values between 0 and 1. Only
540 analyzing one fixed point that is most proximal to the end of the state trajectory for each input condition (see
541 Methods Section 1.7). **(g-j)** same as c-f for response period of MemoryPro and MemoryAnti tasks. **(k-r)**
542 Network pre-trained on DelayAnti and MemoryPro tasks (anti and memory motifs) was then trained to
543 perform MemoryAnti task through weight changes to the MemoryAnti rule input (length N_{rec} vector). **(k-n)**
544 same as c-f for stimulus period of DelayAnti and MemoryAnti tasks with pre-training on DelayAnti and
545 MemoryPro tasks. **(o-r)** same as c-f for response period of MemoryPro and MemoryAnti tasks with pre-
546 training on DelayAnti and MemoryPro tasks.

547 Discussion

548 In this work, we addressed the question of *how* recurrently connected artificial networks flexibly repurpose
549 their learned dynamics to perform multiple tasks. Our collection of commonly studied cognitive tasks could
550 be broken down into an underlying set of subtasks (contextual integration, memory, categorization, anti-
551 response, etc). We showed that networks learned this underlying subtask structure, which resulted in
552 specialized computational building blocks that we call dynamical motifs, dedicated to each subtask. Using
553 input interpolation and fixed point analyses, our work showed how dynamical motifs were organized in relation
554 to one another and often shared across tasks or task periods. Inputs reconfigured the dynamical system in each
555 task period, resulting in often smooth changes to the dynamical landscape underlying the performed
556 computation. The motifs necessary to perform each subtask included different types of attractor structures,
557 input amplifications, decision boundaries and rotations. This modular subtask structure in our set of tasks is
558 analogous to the structure of language, algebraic thought and other natural behaviors in everyday life ^{27,28}.

559 Our framework of examining subtask computation through the lens of dynamical motifs made it possible to
560 explain lesions and learning results described previously ^{18,20}. As in Yang et al. 2019, we found that lesioning
561 specific unit clusters resulted in specific deficits in sets of tasks that were related computationally. Units within
562 a cluster had high variance during a set of task periods that shared a dynamical motif. When we lesioned a
563 given unit cluster, the fixed points that made up the associated dynamical motif were greatly impacted in terms
564 of their locations and stability. A unit cluster associated with one dynamical motif could be lesioned with little
565 impact to other computations the network performed. That is, disturbances in the fixed point structure of one
566 motif had little impact on the fixed point structures that implemented other dynamical motifs. This finding was
567 surprising given the all-to-all connectivity possible in our networks, as well as the fact that no regularizations
568 or constraints to induce modularity were employed in the training of the recurrent neural networks. Recent
569 work on subpopulation structure for the implementation of multiple complex tasks provides insight for these
570 findings ^{21,29}.

571 We demonstrated that networks equipped with relevant dynamical motifs were able to repurpose those motifs
572 in a modular fashion for fast learning of novel tasks. The initial phase of learning novel dynamical motifs was
573 a slow process. However, given a rich repertoire of previously learned motifs, a network could quickly

repurpose motifs to perform novel tasks by learning a single input vector. We found that training with modifiable recurrent weights resulted in faster and higher performance learning of novel dynamical motifs than training input weights alone (Fig.6b, orange, green). However, sequential task learning with changes to the recurrent weights leads to catastrophic forgetting^{18,30}. Our findings suggest a useful lifelong learning strategy could include two stages of learning. Early in learning, it may be beneficial for highly plastic recurrent connections throughout the brain to learn novel subtasks (dynamical motifs). Late in learning, reduced plasticity in recurrent connections and new plastic layers that function as contextualizing inputs could repurpose previously learned subtasks. This hypothesis is interesting to consider in the context of critical periods³¹ and re-aiming³². We hypothesize that this two-stage process of slow and fast learning could provide some intuition for off- and on-manifold brain-machine interface learning results in nonhuman primates³³⁻³⁵. Additionally, the ability to repurpose dynamical motifs may inform our thinking about state-of-the-art models that require pre-training³⁶.

Our results are based on examining artificial systems, which are missing many of the constraints and complexities of biological neural circuits. We did not consider diverse cell types or prescribed architectures in our networks, and only applied noise-corrupted static inputs. We expect our results to inform a larger class of computational systems, while future work is required to link our findings directly to biological systems. More broadly, artificial networks have proven to be a valuable new tool for generating hypotheses about possible solutions to computation in biological networks⁹⁻¹⁸. While our learning rules are not biological, we hypothesize that optimized artificial neural networks and the principles we uncover from them are informative about biological neural circuits based on principles of optimality and robustness³⁷. While some constraints changed the way dynamical motifs were shared, our main finding that they *are* shared was robust across all types of networks and hyperparameter choices that we tested, including large networks without noise (Supp Fig.2d-k). These findings suggest shared motifs are not a result of limited computational resources. We hypothesize that the modular, compositional organization of dynamical motifs was a result of the modular subtask structure of our tasks but learning dynamics through gradient descent could play a role. It will be of great interest to further explore the prevalence of dynamical motifs in other artificial³⁸ and biological systems^{5,39}.

Dynamical motifs underlying modular computation were organized into largely distinct subspaces (Fig.3a, 4g, 6c). This organization of dynamical motifs allowed for both modular lesion effects and rapid learning through the composition of distinct motifs without interference. In networks with nonlinearities that constrained activations to be positive, dynamical motifs were aligned to unit axes, highlighting these largely distinct subspaces.

Fixed point structures often moved in different contexts rather than appear, disappear or change stability (Fig.1f, 2, 3d-e). For example, we found that during the stimulus task period, a stable fixed point moved in state space according to the stimulus input orientation (Supp Fig.2a), resulting in the same qualitative fixed point structure for each stimulus orientation. We found that multitasking networks sometimes employed parameter bifurcations across input conditions for dissimilar computations. The relationship between high dimensional parameter bifurcations, composition and computation is an important area of future research; see for example recent work⁴⁰⁻⁴³.

Fixed points often persisted as we varied the inputs, even when the associated motif was not relevant to the computational task at hand (Fig.2c). These irrelevant fixed points did not interfere with network computation because the network state was organized to be more proximal to the task relevant fixed points. This is related to the idea of sloppiness in under-constrained systems⁴⁴⁻⁴⁶.

Shared dynamical motifs can be thought of as both modular and continuous. For our broad set of hyperparameters, dynamical motifs were modular in our networks due to the modularity of subtasks within our defined collection of tasks (contextual integration, memory, categorization, anti-response, etc). If our set of tasks was made up of a continuous spectrum of related tasks (e.g. contextual integration of two input modalities

620 weighted by a continuous parameter), we might expect to find a dynamical motif that smoothly varies as the
621 continuous parameter is varied (e.g. a left eigenvector of a line attractor that rotates in accordance with the
622 weighting). Smoothness of dynamics was a prevalent feature in our networks, where similar computations
623 were implemented in nearby regions of state space, on a similar dynamical landscape (Fig.4a-c). This feature
624 of smoothness is consistent with biological noise-robust networks^{13,16,47}.

625 Our findings resulted in a number of experimentally testable results. The method of studying unit variance
626 across tasks²⁰, which we expanded to task periods to identify units contributing to dynamical motifs, could be
627 readily performed on neural data (Fig.3a). This analysis could be informative about perturbations to biological
628 network activity that would most dramatically impact performance on a computation of interest (Fig.5). Our
629 findings could apply to biological networks on multiple spatial scales. For example, we could consider multiple
630 motifs implemented within a cortical region or we could think of different cortical areas as implementing
631 different motifs. Our approach for training networks sequentially by reusing previously learned dynamical
632 motifs could be used to determine ideal curricula for training animals on complex tasks. For example, given a
633 particular task of interest, one could train an artificial network to perform the task and inspect all relevant
634 dynamical motifs. For example, the ‘anti’ and ‘memory’ motifs were the sufficient set of relevant motifs in
635 Figure 6b. Based on the task relevant motifs, one could systematically design a set of tasks to learn a sufficient
636 set of motifs rather than designing a curriculum through guesswork. Additionally, this work highlights the
637 relevance of reporting training protocols as they likely shape the dynamical motifs that implement
638 computation^{48,49}. Beyond experimental predictions, our work provides some intuition for why we find functional
639 specialization in the brain.

640 In summary, through the lens of dynamical systems, we identified the underlying computational substrate for
641 clustered representations described previously in Yang et al. 2019²⁰ and highlighted a new level of organization
642 between the unit and the network: groups of units that implement dynamical motifs. More broadly, our findings
643 highlight the relevance of dynamical systems as a framework to better understand the response properties of
644 neurons in the brain. As researchers record more whole brain activity, the framework of dynamical motifs will
645 guide questions about specialization and generalization across brain regions.

646 **Main References**

- 647 1. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
- 648 2. Silver, D. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–
649 489 (2016).
- 650 3. Silver, D. *et al.* A general reinforcement learning algorithm that masters chess, shogi, and Go through self-
651 play. *Science* **362**, 1140–1144 (2018).
- 652 4. Yang, G. R., Cole, M. W. & Rajan, K. How to study the neural mechanisms of multiple tasks. *Curr Opin
653 Behav Sci* **29**, 134–143 (2019).
- 654 5. Cole, M. W. *et al.* Multi-task connectivity reveals flexible hubs for adaptive task control. *Nat. Neurosci.*
655 **16**, 1348–1355 (2013).
- 656 6. Frankland, S. M. & Greene, J. D. Concepts and Compositionality: In Search of the Brain’s Language of
657 Thought. *Annu. Rev. Psychol.* **71**, 273–303 (2020).
- 658 7. Reverberi, C., Görgen, K. & Haynes, J.-D. Compositionality of rule representations in human prefrontal
659 cortex. *Cereb. Cortex* **22**, 1237–1246 (2012).

- 660 8. Willett, F. R. *et al.* Hand Knob Area of Premotor Cortex Represents the Whole Body in a Compositional
661 Way. *Cell* **181**, 396–409.e26 (2020).
- 662 9. Barak, O., Sussillo, D., Romo, R., Tsodyks, M. & Abbott, L. F. From fixed points to chaos: three models
663 of delayed discrimination. *Prog. Neurobiol.* **103**, 214–222 (2013).
- 664 10. Mante, V., Sussillo, D., Shenoy, K. V. & Newsome, W. T. Context-dependent computation by recurrent
665 dynamics in prefrontal cortex. *Nature* **503**, 78–84 (2013).
- 666 11. Yamins, D. L. K. *et al.* Performance-optimized hierarchical models predict neural responses in higher
667 visual cortex. *Proc. Natl. Acad. Sci. U. S. A.* **111**, 8619–8624 (2014).
- 668 12. Carnevale, F., de Lafuente, V., Romo, R., Barak, O. & Parga, N. Dynamic Control of Response Criterion
669 in Premotor Cortex during Perceptual Detection under Temporal Uncertainty. *Neuron* **86**, 1067–1077 (2015).
- 670 13. Sussillo, D., Churchland, M. M., Kaufman, M. T. & Shenoy, K. V. A neural network that finds a
671 naturalistic solution for the production of muscle activity. *Nat. Neurosci.* **18**, 1025–1033 (2015).
- 672 14. Remington, E. D., Narain, D., Hosseini, E. A. & Jazayeri, M. Flexible Sensorimotor Computations
673 through Rapid Reconfiguration of Cortical Dynamics. *Neuron* **98**, 1005–1019.e5 (2018).
- 674 15. Mastrogjuseppe, F. & Ostojic, S. Linking Connectivity, Dynamics, and Computations in Low-Rank
675 Recurrent Neural Networks. *Neuron* **99**, 609–623.e29 (2018).
- 676 16. Russo, A. A. *et al.* Motor Cortex Embeds Muscle-like Commands in an Untangled Population Response.
677 *Neuron* **97**, 953–966.e8 (2018).
- 678 17. Pinto, L. *et al.* Task-Dependent Changes in the Large-Scale Dynamics and Necessity of Cortical Regions.
679 *Neuron* **104**, 810–824.e9 (2019).
- 680 18. Duncker, L., Driscoll, L., Shenoy, K. V., Sahani, M. & Sussillo, D. Organizing recurrent network
681 dynamics by task-computation to enable continual learning. *Adv. Neural Inf. Process. Syst.* **33**, 14387–14397
682 (2020).
- 683 19. Masse, N. Y., Grant, G. D. & Freedman, D. J. Alleviating catastrophic forgetting using context-
684 dependent gating and synaptic stabilization. *Proc. Natl. Acad. Sci. U. S. A.* **115**, E10467–E10475 (2018).
- 685 20. Yang, G. R., Joglekar, M. R., Song, H. F., Newsome, W. T. & Wang, X.-J. Task representations in neural
686 networks trained to perform many cognitive tasks. *Nat. Neurosci.* **22**, 297–306 (2019).
- 687 21. Dubreuil, A., Valente, A., Beiran, M., Mastrogjuseppe, F. & Ostojic, S. The role of population structure
688 in computations through neural dynamics. *Nat. Neurosci.* **25**, 783–794 (2022).
- 689 22. Riveland, R. & Pouget, A. A neural model of task compositionality with natural language instructions.
690 *bioRxiv* 2022.02.22.481293 (2022) doi:10.1101/2022.02.22.481293.
- 691 23. Sussillo, D. & Barak, O. Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional
692 Recurrent Neural Networks. *Neural Comput.* **25**, 626–649 (2013).
- 693 24. Golub, M. & Sussillo, D. FixedPointFinder: A Tensorflow toolbox for identifying and characterizing
694 fixed points in recurrent neural networks. *The Journal of Open Source Software* **3**, 1003 (2018).

- 695 25. Strogatz, S. H. & Strogatz, R. *Nonlinear Dynamics and Chaos : With Applications to Physics, Biology,*
696 *Chemistry, and Engineering, Second Edition()* - 2014 Edition. (Westview press, 1994).
- 697 26. Kaufman, M. T., Churchland, M. M., Ryu, S. I. & Shenoy, K. V. Cortical activity in the null space:
698 permitting preparation without movement. *Nat. Neurosci.* **17**, 440 (2014).
- 699 27. Baroni, M. Linguistic generalization and compositionality in modern artificial neural networks. *Philos.*
700 *Trans. R. Soc. Lond. B Biol. Sci.* **375**, 20190307 (2020).
- 701 28. Lake, B. M., Ullman, T. D., Tenenbaum, J. B. & Gershman, S. J. Building machines that learn and think
702 like people. *Behav. Brain Sci.* **40**, e253 (2017).
- 703 29. Beiran, M., Dubreuil, A., Valente, A., Mastrogjuseppe, F. & Ostoic, S. Shaping Dynamics With Multiple
704 Populations in Low-Rank Recurrent Networks. *Neural Comput.* **33**, 1572–1615 (2021).
- 705 30. McCloskey, M. & Cohen, N. J. Catastrophic Interference in Connectionist Networks: The Sequential
706 Learning Problem. in *Psychology of Learning and Motivation* (ed. Bower, G. H.) vol. 24 109–165 (Academic
707 Press, 1989).
- 708 31. Hensch, T. K. Critical period regulation. *Annu. Rev. Neurosci.* **27**, 549–579 (2004).
- 709 32. Morehead, J. R., Qasim, S. E., Crossley, M. J. & Ivry, R. Savings upon Re-Aiming in Visuomotor
710 Adaptation. *J. Neurosci.* **35**, 14386–14396 (2015).
- 711 33. Sadtler, P. T. *et al.* Neural constraints on learning. *Nature* **512**, 423–426 (2014).
- 712 34. Golub, M. D. *et al.* Learning by neural reassociation. *Nat. Neurosci.* **21**, 607–616 (2018).
- 713 35. Oby, E. R. *et al.* New neural activity patterns emerge with long-term learning. *Proc. Natl. Acad. Sci. U.*
714 *S. A.* **116**, 15210–15215 (2019).
- 715 36. Han, X. *et al.* Pre-trained models: Past, present and future. *AI Open* **2**, 225–250 (2021).
- 716 37. Maheswaranathan, N., Williams, A. H., Golub, M. D., Ganguli, S. & Sussillo, D. Universality and
717 individuality in neural dynamics across large populations of recurrent networks. *Adv. Neural Inf. Process.*
718 *Syst.* **2019**, 15629–15641 (2019).
- 719 38. Goudar, V., Peysakhovich, B., Freedman, D. J. & Buffalo, E. A. Elucidating the neural mechanisms of
720 Learning-to-Learn. *bioRxiv* (2021).
- 721 39. Chen, G., Kang, B., Lindsey, J., Druckmann, S. & Li, N. Modularity and robustness of frontal cortical
722 networks. *Cell* **184**, 3717–3730.e24 (2021).
- 723 40. Jaeger, H. FROM CONTINUOUS DYNAMICS TO SYMBOLS. in *Dynamics, Synergetics, Autonomous*
724 *Agents* vol. 8 29–48 (WORLD SCIENTIFIC, 1999).
- 725 41. Jordan, I. D., Sokół, P. A. & Park, I. M. Gated Recurrent Units Viewed Through the Lens of Continuous
726 Time Dynamical Systems. *Front. Comput. Neurosci.* **15**, 678158 (2021).
- 727 42. Beer, R. D. Codimension-2 parameter space structure of continuous-time recurrent neural networks. *Biol.*
728 *Cybern.* (2022) doi:10.1007/s00422-022-00938-5.

- 729 43. Wang, X.-J. Theory of the Multiregional Neocortex: Large-Scale Neural Dynamics and Distributed
730 Cognition. *Annu. Rev. Neurosci.* **45**, 533–560 (2022).
- 731 44. Gutenkunst, R. N. *et al.* Universally sloppy parameter sensitivities in systems biology models. *PLoS*
732 *Comput. Biol.* **3**, 1871–1878 (2007).
- 733 45. Marder, E. & Goaillard, J.-M. Variability, compensation and homeostasis in neuron and network
734 function. *Nat. Rev. Neurosci.* **7**, 563–574 (2006).
- 735 46. O’Leary, T., Sutton, A. C. & Marder, E. Computational models in the age of large datasets. *Curr. Opin.*
736 *Neurobiol.* **32**, 87–94 (2015).
- 737 47. Saxena, S., Russo, A. A., Cunningham, J. & Churchland, M. M. Motor cortex activity across movement
738 speeds is predicted by network-level strategies for generating muscle activity. *Elife* **11**, (2022).
- 739 48. Latimer, K. W. & Freedman, D. J. Low-dimensional encoding of decisions in parietal cortex reflects
740 long-term training history. *bioRxiv* (2021).
- 741 49. Molano-Mazon, M., Duque, D., Yang, G. R. & de la Rocha, J. Pre-training RNNs on ecologically
742 relevant tasks explains sub-optimal behavioral reset. *bioRxiv* 2021.05.15.444287 (2021)
743 doi:10.1101/2021.05.15.444287.

744 **Data Availability and Code Availability**

745 All trained networks and analysis code will be published online before the time of publication.

746 **Acknowledgements**

747 We thank Lea Duncker, Niru Maheswaranathan, Dan O’Shea, Matt Golub, Frank Willett and Larry Abbott
748 for feedback on this manuscript. We also thank our larger scientific community for feedback on this work
749 over the years. The authors acknowledge support by Simons Collaboration on the Global Brain (SCGB)
750 project numbers 543045 and 543049.

751 **Author Contributions**

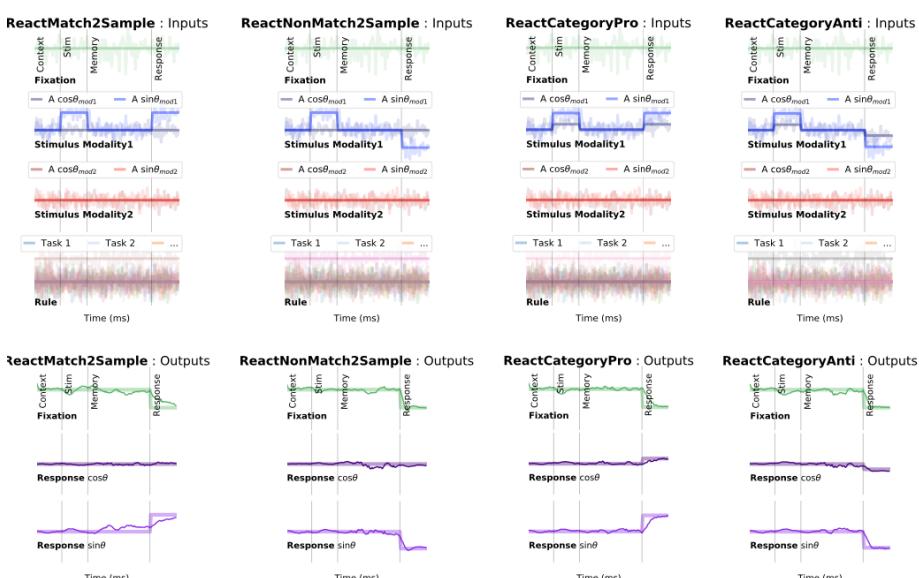
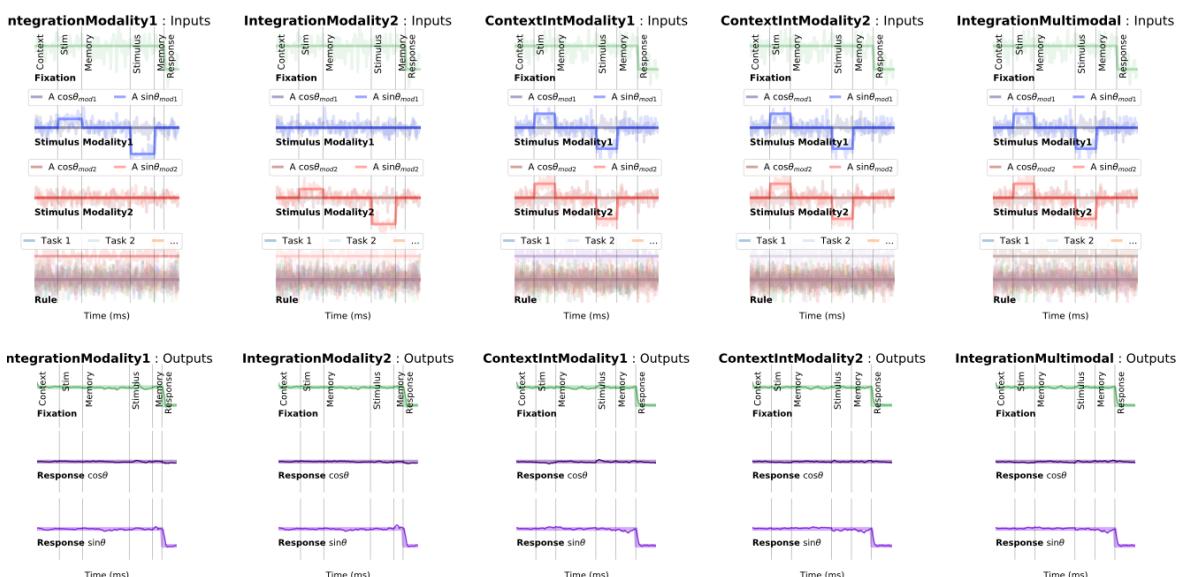
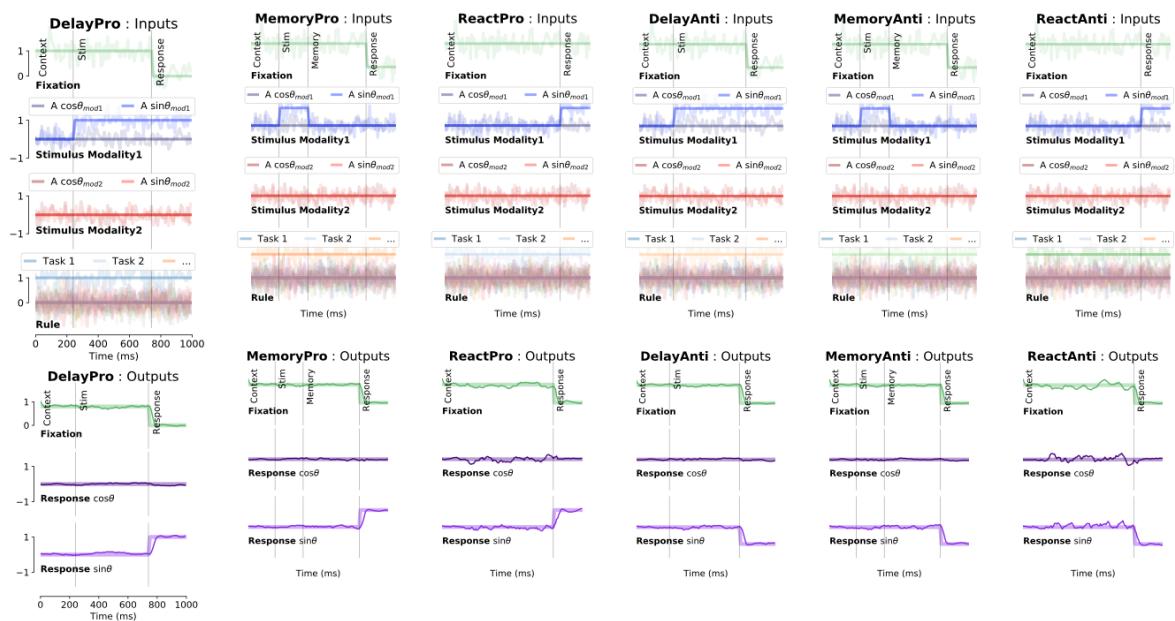
752 LD, KS and DS conceived of this project. LD performed analyses under the supervision of KS and DS. LD,
753 KS and DS wrote the manuscript.

754 **Competing Interests**

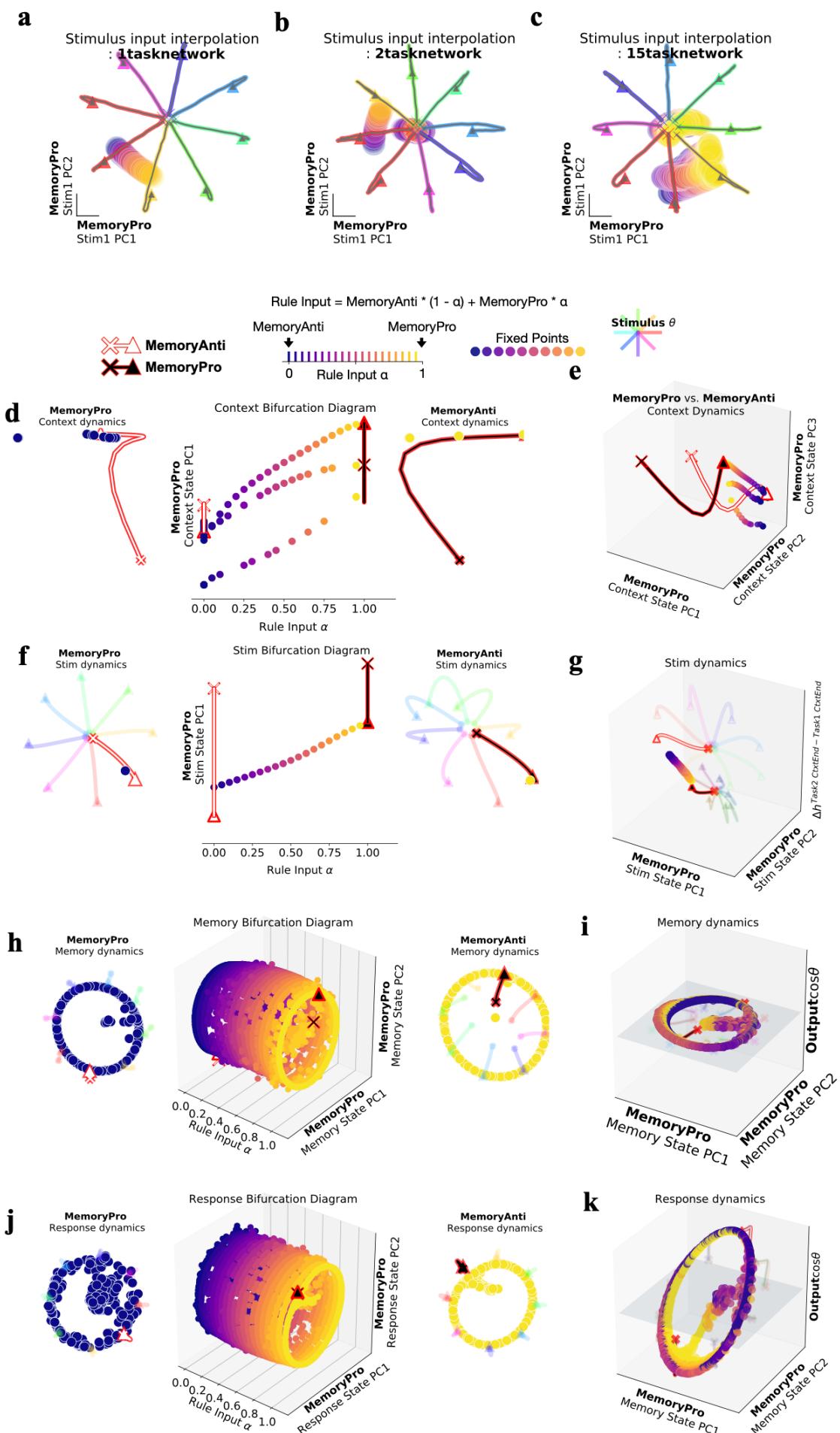
755 KS serves on the Scientific Advisory Boards (SABs) of MIND-X Inc. (acquired by Blackrock Neurotech,
756 Spring 2022), Inscopix Inc. and Heal Inc. He also serves as a consultant / advisor (and was on founding
757 SAB) for CTRL-Labs (acquired by Facebook Reality Labs in Fall 2019, and is now a part of Meta Platform's
758 Reality Labs) and serves as a consultant / advisor (and is a co-founder, 2016) for Neuralink. DS works for
759 Meta Platform's Reality Labs, but the work presented here was done entirely at Stanford. LD has no
760 competing interests.

761 Correspondence and requests for materials should be addressed to lndrisco@stanford.edu.

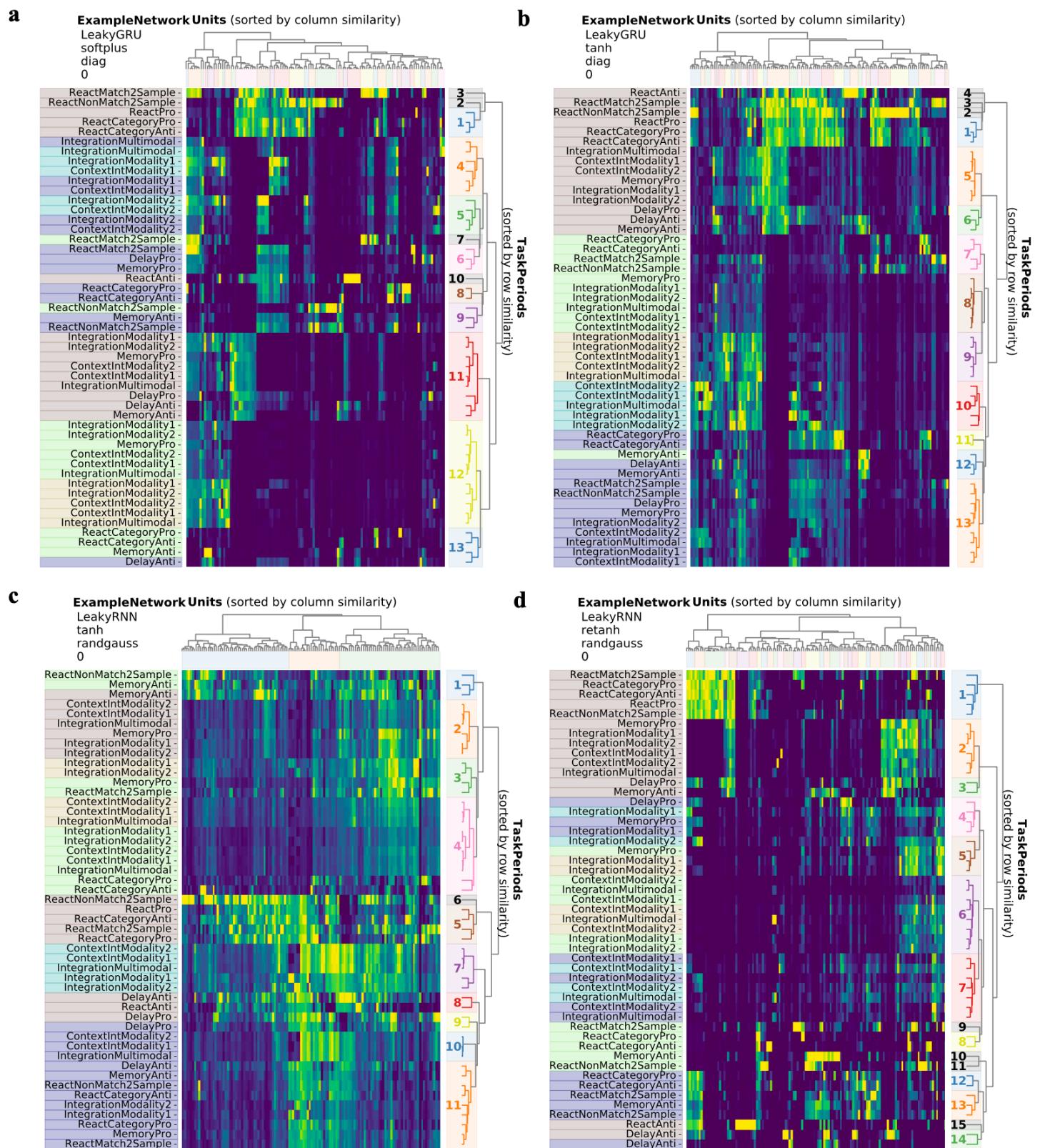
762 **Supplementary Figures**



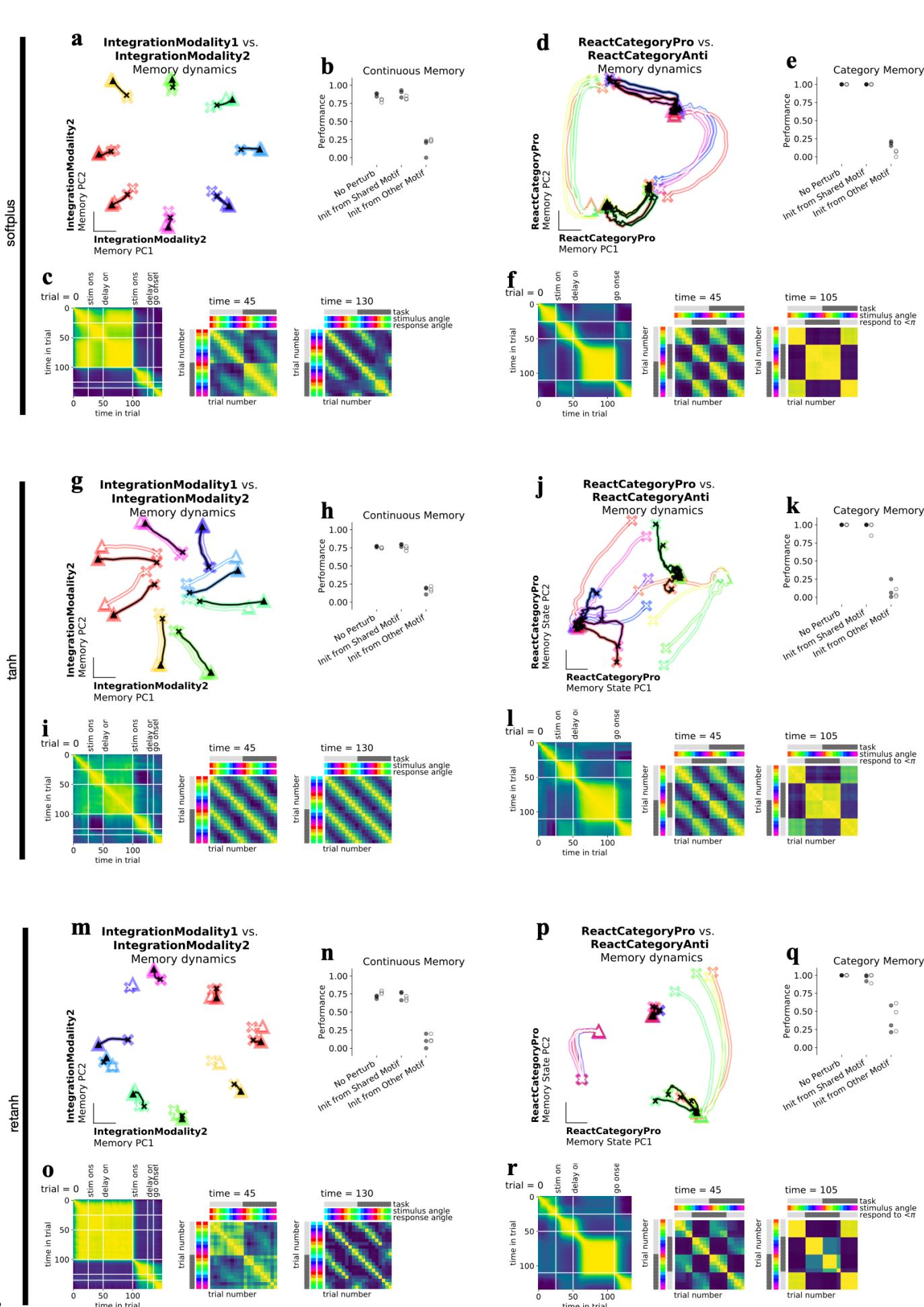
764 **Supplementary Figure 1: Inputs and outputs for each task for one example trial.** One panel for each task.
765 **Top:** Noisy fixation, stimulus (modality 1 and 2), and rule input time-series (overlaid without noise for
766 clarity.) Noise was used during training while analyses were performed on running the network without noise.
767 Vertical lines divide task periods. **Bottom:** Targets (thick lines) overlaid with outputs of a trained network
768 (thin lines).



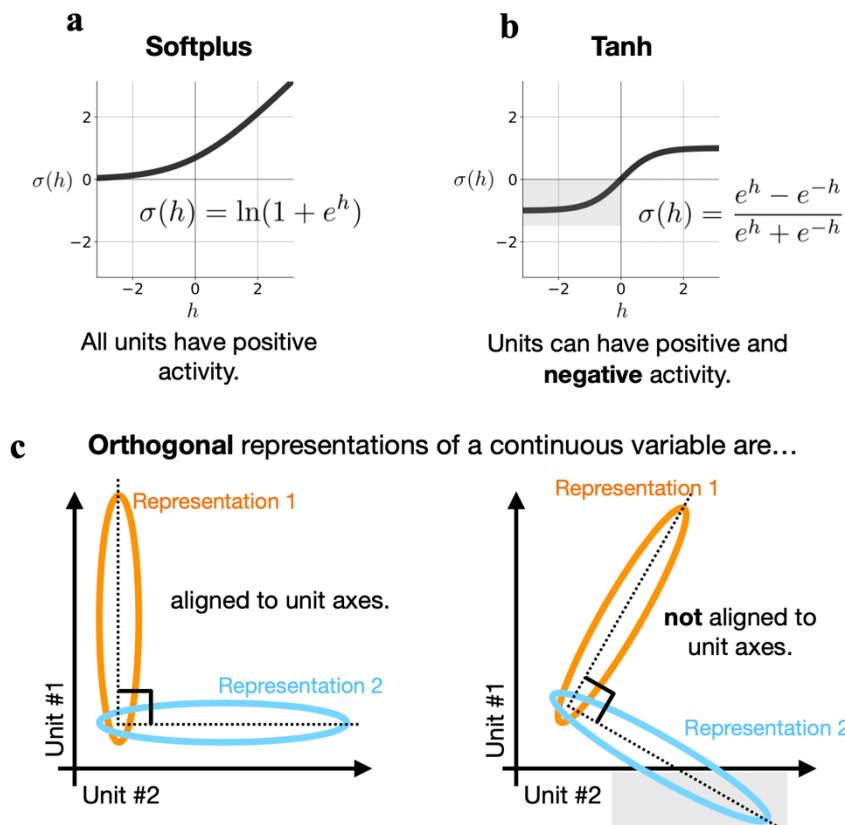
770 **Supplementary Figure 2: Fixed point structure is dependent on other tasks the network is trained to**
771 **perform. (a)** Fixed points for interpolated inputs across stimulus conditions 0° (blue) $\rightarrow 45^\circ$ (yellow), reveals
772 one fixed point that moves dependent on stimulus input conditions. Trajectories colored according to stimulus
773 orientation. Additional fixed points are revealed in **(b)** 2 task networks and **(c)** 15 task networks. **(d-k)** $N_{rec} =$
774 1024 network with softplus activation, diagonal initialization and no input noise or private noise (see equations
775 2 and 7 in Methods for noise definitions). Fixed points for interpolation between inputs for MemoryAnti ($\alpha =$
776 0) and MemoryPro ($\alpha = 1$) tasks during **(d,e)** context **(f,g)** stimulus **(h,i)** memory **(j,k)** and response periods.
777 **(d middle:** Fixed points for 20 intermediate α values (x-axis) projected into top PC defined by state evolution
778 during the context period of the MemoryPro task (y-axis) with **(left)** MemoryPro $\alpha = 0$ and **(right)**
779 MemoryAnti $\alpha = 1$ fixed points and trajectories. **(e)** Fixed points for rule input-interpolation between tasks,
780 MemoryPro (blue fixed points, white state trajectory) and MemoryAnti (yellow fixed points, black state
781 trajectory) projected into the top three MemoryPro context period state evolution PCs. **(f)** Same as **d** for
782 stimulus period, with fixed points projected into top PC defined by the state evolution during the stimulus
783 period of the MemoryPro task (y-axis) **(g)** Same as **e** for stimulus period, projected into the top two MemoryPro
784 stimulus period state evolution PCs (x and y axes) and the dimension separating both tasks at the end of the
785 context period (z axis). **(h)** Same as **d** for memory period, projected into top two PCs defined by the state
786 evolution during the memory period of the MemoryPro task (y and z-axes) **(i)** Same as **e** for memory period,
787 projected into the top two MemoryPro memory period state evolution PCs (x and y-axes) and the output weight
788 vector (from W_{out}) associated with $\cos\theta_{stimulus}$ on the z-axis. **(j)** Same as **d** for response period, projected into top
789 two PCs defined by the state evolution during the response period of the MemoryPro task (y and z-axes) **(k)**
790 Same as **i** for response period.



Supplementary Figure 3: Example variance matrices for four different networks. Variance matrix: variance of unit activations across stimulus conditions normalized across task periods. Rows and columns sorted according to similarity (see Methods Section 1.10). **(a)** LeakyGRU, softplus activation, diagonal initialization, $N_{rec}=128$ **(b)** LeakyGRU, tanh activation, diagonal initialization, $N_{rec}=128$ **(c)** LeakyRNN, tanh activation, random Gaussian initialization, $N_{rec}=128$ **(d)** LeakyRNN, retanh activation, random Gaussian initialization, $N_{rec}=128$.

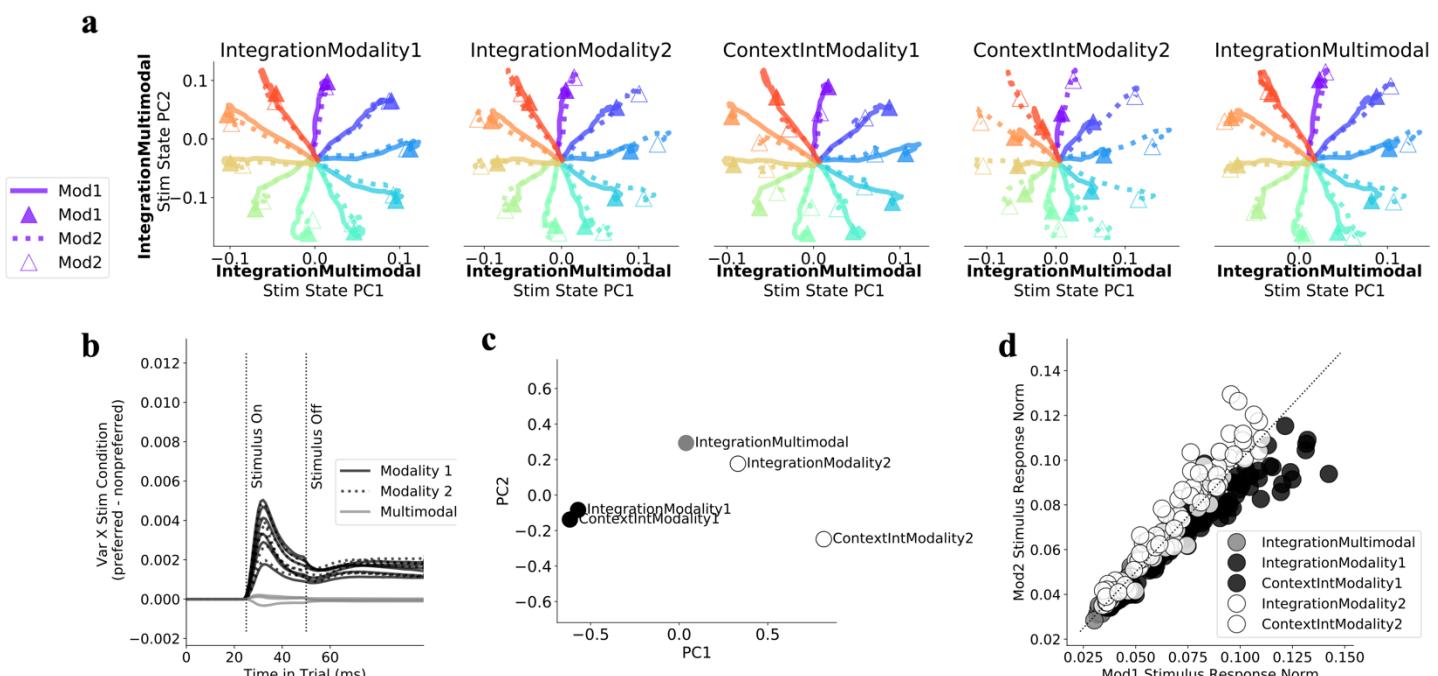


799 **Supplementary Figure 4: Discrete and continuous variable memory computations result in two kinds of**
800 **shared memory attractors.** Networks were trained with softplus activation function for **a-f** **(a)** Neural state
801 trajectories for 8 stimulus conditions (colored by response direction) starting from ‘x’ in IntegrationModality2
802 PC space for IntegrationModality1 (white) and IntegrationModality2 (black) tasks. **(b)** Perturb initial condition
803 for IntegrationModality1 response period state to the memory period final state of either IntegrationModality2
804 task (Init from shared motif), ReactCategoryAnti (Init from other motif) or IntegrationModality1 (No perturb)
805 for trials with the same response direction and run the network forward through the response period. One data
806 point for each of three networks trained with diagonal initialization (filled circles) or with a random gaussian
807 initialization (open circles). Performance was similar across ‘No perturb’ and ‘Init from shared motif’
808 conditions but was dramatically impacted for ‘Init from other motif’ conditions. **(c left:** Correlation across
809 timesteps throughout an IntegrationModality1 trial. Task period transitions highlighted in white lines. **middle:**
810 Correlation across trials for different stimulus conditions for IntegrationModality1 and IntegrationModality2
811 tasks at the end of the first stimulus period. **right:** Correlation across trials for different stimulus conditions for
812 IntegrationModality1 and IntegrationModality2 tasks at the end of the memory period. Trials are highly
813 correlated across tasks according to response direction. **(d)** Same as (a) for two category memory tasks,
814 ReactCategoryPro (white) and ReactCategoryAnti (black). **(e)** Perturb initial condition for ReactCategoryPro
815 response period state to the memory period final state of ReactCategoryAnti task (Init from shared motif),
816 IntegrationModality1 (Init from other motif) or ReactCategoryPro (No perturb) for trials with the same
817 response direction and run the network forward through the response period. **(f left:** Correlation across
818 timesteps throughout an ReactCategoryPro trial. Task period transitions highlighted in white lines. **middle:**
819 Correlation across trials for different stimulus conditions for ReactCategoryPro and ReactCategoryAnti tasks
820 at the end of the stimulus period. **right:** Correlation across trials for different stimulus conditions for
821 ReactCategoryPro and ReactCategoryAnti tasks at the end of the memory period. Trials are highly correlated
822 across tasks according to response direction. **(g-l)** Same as (a-f) for tanh activation function. **(m-r)** Same as (a-
823 f) for retanh activation function.



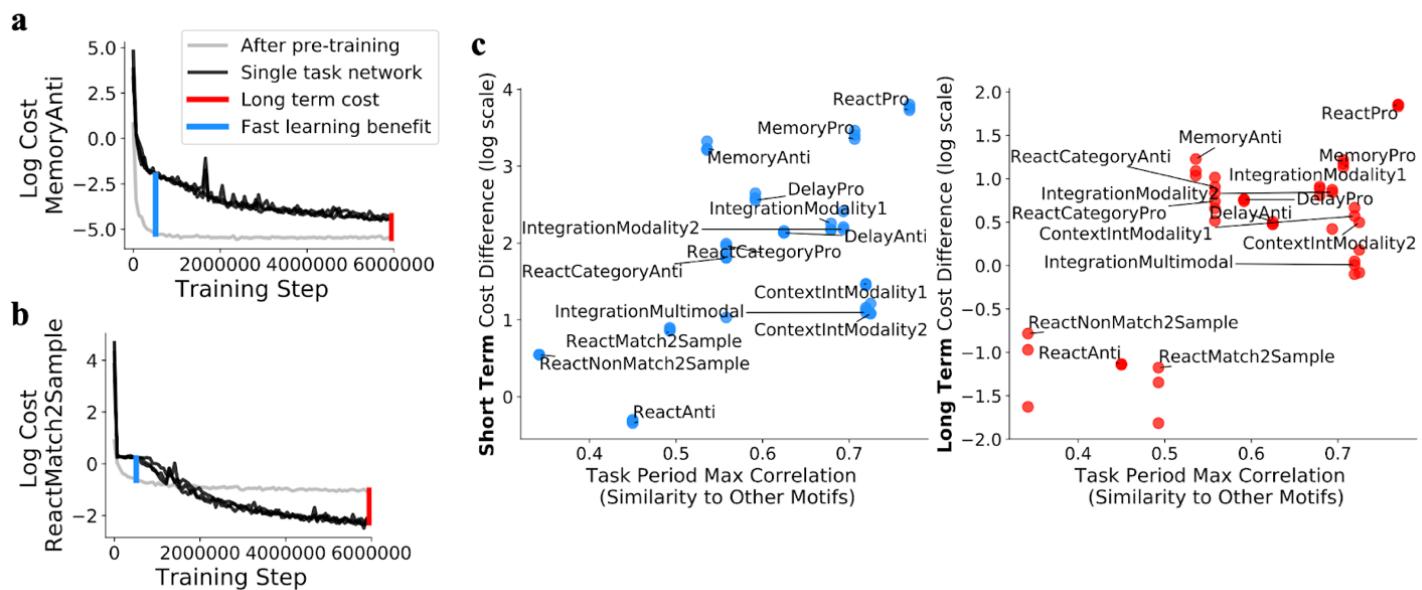
824

825 **Supplementary Figure 5: Positive activation functions align orthogonal motifs to unit axes.** (a) Softplus
826 activation function maps activity to the positive real line. (b) Tanh activation function does not impose
827 positivity constraints. (c) **left:** Two activity patterns that are constrained to lie in the positive orthant can only
828 be orthogonal to one another when aligned to the unit axes. **right:** Activation functions without positivity
829 constraints do not constrain activity patterns to unit axes. *Note: Either system can use additional dimensions*
830 *to orthogonalize representations not aligned to unit axes in an infinitely large network.*



831
832
833
834
835
836
837
838
839
840
841
842

Supplementary Figure 6: Input modulation for context dependent integration (a) Stimulus dependent network state for both modality 1 (solid) and 2 (dotted) for 5 different tasks where relevant modality is specified in title for each subpanel. (b) Difference between variance across stimulus conditions for preferred modality minus variance across stimulus conditions for nonpreferred modality. Preference is defined by task rules. One line for each of five tasks in three different networks. (c) Context period network state for all 5 tasks projected onto the first and second PCs defined by variance of the state for each task. (PCA on $5 \times N_{rec}$ matrix). Organization of network states at the end of the context period shows compositional structure. PC1 goes from modality 1->2 and PC2 separates ContextIntegration vs. Integration decision tasks. (d) Norm of input jacobian for 24 stimulus conditions spanning $[0, 2\pi]$ on each of 5 tasks (see Methods Section 1.8). Trials where the network should prefer modality 1 are black, modality 2 white, multimodal gray. Hyperparameters: LeakyRNN, softplus activation, diagonal initialization, $N_{rec}=128$



843

844 **Supplementary Figure 7: Leave-one-out transfer learning is less effective for tasks with unique**
 845 **dynamical motifs. (a)** Log cost during training single rule input weights for **(a)** MemoryAnti and **(b)**
 846 ReactMatch2Sample tasks after pre-training all other weights on all other tasks (gray) and during training all
 847 weights and biases on a single rule (black, 3 lines for 3 different networks with different random reeds).
 848 Single task networks provide a baseline comparison to measure how well the pre-training method works for
 849 any given task. **(c)** Cost difference between single task networks and transfer learning network (as in a,b)
 850 plotted against task period max correlation (See Methods Section 1.12 for details) early in learning (left) and
 851 late in learning (right) for each task. The fast learning benefit was smaller and the long term cost was
 852 negative (single task networks outperformed transfer learning networks) for tasks with unique dynamical
 853 motifs. Hyperparameters: LeakyRNN, softplus activation, diagonal initialization, $N_{rec}=128$ for transfer
 854 learning and single task networks. Task period max correlation was averaged across networks with all
 855 hyperparameter settings in Fig.3b.

856 1 Methods

857 1.1 Network Structure

858 We examined “vanilla” continuous-time RNNs for the majority of this work, although see methods
859 subsection on varying hyperparameters and architectures [Section 1.4]. Before time discretization,
860 RNN network activity \mathbf{h} , a vector of length N_{rec} , followed the dynamical equation

$$\tau \frac{d\mathbf{h}}{dt} = -\mathbf{h} + \sigma(\mathbf{i}_t), \quad (1)$$

861 with the total neuron input \mathbf{i}_t defined as

$$\mathbf{i}_t = W_{rec}\mathbf{h}_t + W_{in}\mathbf{u}_t + \mathbf{b}_{in} + \xi. \quad (2)$$

862 W_{in} and W_{rec} were the input and recurrent connection matrices of size $N_{rec} \times N_{in}$ and $N_{rec} \times N_{rec}$.
863 These matrices specified the contribution of the inputs and upstream network activity to downstream
864 network activity. The bias vector, \mathbf{b}_{in} , was of length N_{rec} . The private noise variable, ξ , was N_{rec}
865 independent Gaussian white noise processes with zero mean and standard deviation of 0.05.

866 The state of this system evolved over time according to the current state \mathbf{h} and inputs to the system
867 \mathbf{u} . The nonlinear function $\sigma(\cdot)$ was chosen to be softplus, tanh or retanh [see Section 1.4 for details].
868 The time constant, τ , specified the rate of decay of the network state. After using the first-order Euler
869 approximation with a time-discretization step Δt , we had

$$\mathbf{h}_{t+1} = (1 - \gamma)\mathbf{h}_t + \gamma\sigma(\mathbf{i}_t), \quad (3)$$

870 where $\gamma \equiv \Delta t / \tau$, which we set to 0.2. The full update equation was

$$\mathbf{h}_{t+1} = (1 - \gamma)\mathbf{h}_t + \gamma\sigma(W_{rec}\mathbf{h}_t + W_{in}\mathbf{u}_t + \mathbf{b}_{in} + \xi). \quad (4)$$

871 A set of output units \mathbf{z} were read out from the network according to

$$\mathbf{z}_t = W_{out}\mathbf{h}_t + \mathbf{b}_{out}. \quad (5)$$

872 W_{out} was the output connection matrix of size $N_{out} \times N_{rec}$ and \mathbf{b}_{out} was a bias vector of length N_{out} .
873 All W matrices, W_{in} , W_{rec} , W_{out} and bias vectors \mathbf{b}_{in} , \mathbf{b}_{out} were learned over the course of training
874 [see Section 1.3 for details].

875 The network received four types of noisy input.

$$\mathbf{u} = [u_{fix}, \mathbf{u}_{mod1}, \mathbf{u}_{mod2}, \mathbf{u}_{rule}] + \mathbf{u}_{noise} \quad (6)$$

$$\mathbf{u}_{noise}[j] \sim 0.1\sqrt{2/\gamma} \mathcal{N}(0, 1) \quad (7)$$

876 The fixation input u_{fix} was 1 when the network was required to fixate and 0 when the network was
877 required to respond. The stimulus inputs \mathbf{u}_{mod1} and \mathbf{u}_{mod2} each a length-2 vector of ($A \sin \theta$ and
878 $A \cos \theta$) representing a different “modality” and each modality representing a one-dimensional cir-
879 cular variable described by the degree around a circle. The strength of the stimulus inputs varied in
880 amplitude according to a . We greatly reduced the dimensionality of the stimulus inputs from the orig-
881 inal implementation of these tasks in order to simplify visualizations and analysis. This simplification
882 in stimulus inputs required removal of five of the original 20 tasks, because we could no longer present
883 multiple stimuli in the same modality simultaneously. The network also received a set of rule inputs

884 encoded in the vector, \mathbf{u}_{rule} . This vector represented which task the network was supposed to perform
885 on each trial as a one-hot vector. The rule input unit corresponding to the current task was 1, while
886 other rule input units were 0. Therefore, the number of rule input units was equal to the number of
887 tasks trained. The rule unit activation patterns for different rules were orthogonal to each other in this
888 1-hot encoding. Therefore, relationships between tasks were learned by the network rather than baked
889 into the inputs. Finally, each input had Gaussian noise added to it according to equations (6-7). Here
890 the input noise strength was scaled by the factor 0.1. Note this was an order of magnitude greater than
891 in previous work, to prevent over-fitting [1].

892 In total, there were

893 $N_{in} = 1(\text{fixation}) + 2(\text{modalities}) \times 2(A \sin \theta \text{ and } A \cos \theta) + 15(\text{rule}) = 20$ input units.

894 The network projected the state, \mathbf{h}_t , to an output ring, which contained 2 units ($\sin \phi, \cos \phi$) to encode
895 response direction on a circle. In addition, the network projected \mathbf{h} to a fixation output unit, which
896 should be at the high activity value of 1 before the response and at 0 once a response is generated.

897 In total, there were

898 $N_{out} = 1(\text{fixation}) + 1(\text{modality}) \times 2(\sin \phi \text{ and } \cos \phi) = 3$ output units.

899 1.2 Tasks and performances

900 Inputs and outputs for an example trial on each task are shown in Supplementary Figure 1. Fixation
901 input was 1 for the duration of the trial until the response period, when fixation input changed to zero.
902 Reaction timed tasks never received a go cue; therefore, the fixation input was always at 1 and the
903 network was required to break fixation to respond as soon as the relevant stimulus arrived. Target
904 fixation output activity was high ($\hat{z} = 0.8$) before the response period and low ($\hat{z} = 0$) during the
905 response period for all tasks. If the activity of the fixation output prematurely fell below 0.5, the
906 network was considered to have erroneously broken fixation and the trial was incorrect. The response
907 direction of the network was read out in a 2 dimensional vector ($\sin \phi$ and $\cos \phi$). The decoded response
908 direction was considered correct if it was within $\pi/10$ of the target direction.

909 Tasks could be divided into periods, where each task period was a segment of sequential time steps
910 with continuous inputs (without noise). Each set of inputs reconfigured the network into a new dy-
911 namical landscape, with a different fixed point structure. Distinct dynamical landscapes for each input
912 condition is a crucial concept for this work and should be emphasized. For all tasks, in the first pe-
913 riod (context) the rule input provided the network with information about task context. The onset of
914 stimulus information marked a change in the stimulus inputs and the beginning of the next task period
915 (stimulus1). All tasks had at least one stimulus period, but some had two stimulus periods. The period
916 between the stimulus and response or between two stimuli was the memory period (memory1). If there
917 was a second stimulus (stimulus2), sometimes the network was required to respond immediately to the
918 second stimulus and in other tasks there was an additional memory period (memory2) before the re-
919 sponse period (response). The duration of the context, stimulus1, memory1, stimulus2, memory2, and
920 response periods were $T_{context}, T_{stimulus1}, T_{memory1}, T_{stimulus2}, T_{memory2}, T_{response}$, respectively. We
921 adjusted the distribution of task periods to be wider than in previous work and drawn from a uniform
922 distribution to prevent the network from predicting task period transitions. These modifications had a
923 simplifying effect on fixed point structures).

924 $U(t_1, t_2)$ is a uniform distribution between t_1 and t_2 . The unit for time is milliseconds. Stimuli were
925 presented in either modality 1 or 2 at random unless stated otherwise.

926 **Delayed Response:** (2 tasks) DelayedPro: Move in same direction as stimulus ($\phi_{response} = \theta_{stimulus}$)
927 after delay. DelayedAnti: Move in opposite direction as stimulus ($\phi_{response} = \theta_{stimulus} + \pi$) after delay.
928 Stimulus remains on throughout stimulus and response periods.

$$\begin{aligned}T_{context} & U(300, 700) \\T_{stim1} & U(200, 1500) \\T_{response} & U(300, 700)\end{aligned}$$

929 **Memory Response:** (2 tasks) MemoryPro: Move in same direction as stimulus ($\phi_{response} = \theta_{stimulus}$)
930 after memory. MemoryAnti: Move in opposite direction as stimulus ($\phi_{response} = \theta_{stimulus} + \pi$) after
931 memory. Stimulus disappears for memory period and remains off during response.

$$\begin{aligned}T_{context} & U(300, 700) \\T_{stim1} & U(200, 1600) \\T_{memory1} & U(200, 1600) \\T_{response} & U(300, 700)\end{aligned}$$

932 **Reaction Timed:** (2 tasks) ReactPro: Move in same direction as stimulus ($\phi_{response} = \theta_{stimulus}$)
933 immediately. ReactAnti: Move in opposite direction as stimulus ($\phi_{response} = \theta_{stimulus} + \pi$) immedi-
934 ately.

$$\begin{aligned}T_{context} & U(500, 2500) \\T_{response} & U(300, 1700)\end{aligned}$$

935 **Decision Making:** (5 tasks) Move in direction of stimulus with largest amplitude. IntegrationModali-
936 ty1: Only modality 1 is presented. IntegrationModality2: Only modality 2 is presented. ContextInt-
937 Modality1: Both modalities presented, only attend modality 1. ContextIntModality2: Both modalities
938 presented, only attend modality 2. IntegrationMultimodal: Both modalities presented, attend both
939 modalities equally.

$$\begin{aligned}T_{context} & U(200, 600) \\T_{stim1} & U(200, 1600) \\T_{memory1} & U(200, 1600) \\T_{stim2} & U(200, 1600) \\T_{memory2} & U(100, 300) \\T_{response} & U(300, 700)\end{aligned}$$

940 **Delay Match:** (4 tasks) Immediately move in direction of θ_{stim2} if sequentially presented pair match.
941 ReactMatch2Sample: Match if same angle ($\theta_{stim1} = \theta_{stim2}$). ReactNonMatch2Sample: Match if
942 opposite angle ($\theta_{stim1} = \theta_{stim2} + \pi$). ReactCategoryPro: Match if same category ($\theta_{stim1}, \theta_{stim2} < \pi$)
943 or ($\theta_{stim1}, \theta_{stim2} > \pi$). ReactCategoryAnti: Match if opposite category ($\theta_{stim1} < \pi \& \theta_{stim2} > \pi$) or
944 ($\theta_{stim1} > \pi \& \theta_{stim2} < \pi$).

$$\begin{aligned}T_{context} & U(200, 600) \\T_{stim1} & U(200, 1600) \\T_{memory1} & U(200, 1600) \\T_{stim2/response} & U(300, 700)\end{aligned}$$

945 1.3 Training procedure

946 The loss L to be minimized was computed by time-averaging the squared errors between the network
947 output $\mathbf{z}(t)$ and the target output $\hat{\mathbf{z}}(t)$.

$$L = L_{mse} \equiv \langle m_{i,t}(\mathbf{z}_{i,t} - \hat{\mathbf{z}}_{i,t})^2 \rangle_{i,t}$$

948 Here, i was the index of the output units and t was the index for time. We implemented a mask, $m_{i,t}$, for
949 modulating the loss with respect to certain time intervals. For example, in the first 100 ms of the context
950 and response periods there was a grace period with $m_{i,t} = 0$. During the response period, $m_{i,t} = 5$ and
951 for the rest of the trial $m_{i,t} = 1$. For the fixation output unit, $m_{i,t}$ was two times stronger than the mask
952 for the $\phi_{response}$ output units. The training was performed with Adam, a variant of stochastic gradient
953 descent [2]. The learning rate ranged from 10^{-4} (tanh networks) to 10^{-3} (all other networks). The
954 decay rate for the first and second moment estimates were 0.9 and 0.999, respectively. During training,
955 we randomly interleaved all the tasks with equal probabilities, except for the ContextIntModality1
956 and ContextIntModality2 tasks that appeared five times more frequently to prevent the network from
957 integrating both modalities equally. This alternative strategy gave the network an accuracy close to
958 75% if trials from these tasks were not over-represented. During training, we used mini-batches of 64
959 trials, in which all trials were generated from the same task for computational efficiency. Training was
960 terminated when L stopped decreasing, which was generally after 5e7 training steps.

961 The network and training were implemented in TensorFlow.

962 1.4 Alternative hyperparameters and network architectures

963 We trained networks with the following possible hyperparameters and architectures:

964 The network architecture was either the leaky RNN architecture defined previously, or the leaky GRU
965 architecture [3].

966 We explored a number of nonlinear functions $\sigma(\cdot)$,

$$\begin{aligned} \text{Softplus} : \sigma(x) &= \ln(1 + e^x) \\ \text{Retanh} : \sigma(x) &= \max(\tanh(x), 0) \\ \text{Tanh} : \sigma(x) &= \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \end{aligned} \tag{8}$$

967 We initialized each weight matrix from a diagonal matrix,

$$W_{rec0} = g I_{N_{rec}} \tag{9}$$

968 or from a random Gaussian

$$[W_{rec0}]_{ij} \sim \frac{g}{\sqrt{N_{rec}}} \mathcal{N}(0, 1). \tag{10}$$

969 Here, g scaled the values in the initial weights. In networks with the tanh activation function and
970 the leaky RNN architecture $g = 1$ and all other networks $g = 0.8$. We found that networks with
971 the tanh activation function required this higher g value to prevent quenching network activity during
972 training.

973 To avoid overly complex solutions that didn't generalize well, we penalized high activity and strong
974 weights using an L2 regularization on \mathbf{h} and on each weight matrix W_{in} , W_{rec} , W_{out} . The hyperpa-
975 rameter selection criterion was the highest level of regularization that resulted in >80% performance

976 on held out test data. We found the highest level regularization that still resulted in greater than 80%
977 performance on all tasks to be 10^{-6} for both weight and activity regularization.

978 1.5 Fixed Points

979 Our networks were high-dimensional nonlinear systems, rendering them difficult to understand intuitively.
980 Examination of these networks was made easier through analysis of fixed points, which are
981 locations in state space where the motion of the system is approximately zero. Through a Taylor ex-
982 pansion of our dynamical equation, we may see that our nonlinear system can be approximated as a
983 linear one around fixed points, \mathbf{h}^* :

$$\frac{d\mathbf{h}}{dt} = F(\mathbf{h}, \mathbf{u})$$
$$F(\mathbf{h}^* + \delta\mathbf{h}, \mathbf{u}^* + \delta\mathbf{u}) \approx F(\mathbf{h}^*, \mathbf{u}^*) + \frac{\partial F}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*)\delta\mathbf{h} + \frac{\partial F}{\partial \mathbf{u}}(\mathbf{h}^*, \mathbf{u}^*)\delta\mathbf{u} \quad (11)$$

984 The second order terms (not shown) are approximately zero because $\|\delta\mathbf{h}\|^2 \approx 0$. The first term,
985 $F(\mathbf{h}^*, \mathbf{u}^*)$ is zero by definition of the fixed point, where \mathbf{h}^* is the location where the update $F(\mathbf{h}^*, \mathbf{u}^*) =$
986 0. For the majority of this work, with the exception of Supplementary Figure 6, we hold input values
987 to their constant value during a task period. We can therefore ignore the last term where $\delta\mathbf{u} = 0$.
988 Therefore, around fixed points, we can approximate our nonlinear dynamical systems as the linear
989 system, $\frac{d\delta\mathbf{h}}{dt} \approx \frac{\partial F}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*)\delta\mathbf{h}$.

990 Eigendecomposition of the matrix, $\frac{\partial F}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*)$, reveals in which dimensions of state space the dy-
991 namics are contracting, expanding or are marginally stable (ie. neither contracting or expanding).
992 Eigenvectors with an associated real part of the eigenvalue $\lambda < 1$ are contracting dimensions, $\lambda > 1$
993 are associated with expanding dimensions and $\lambda \approx 1$ are marginally stable. At a fixed point that is
994 contracting in every dimension, the state is at a basin of attraction. This is a particularly useful dy-
995 namical system for preparing an optimal initial condition for the next task period. Marginally stable
996 dimensions are useful for integrating noisy pulses of stimulus information and for memory of a contin-
997 uous variable. Saddle points are contracting in some dimensions, while repulsive in other dimensions.
998 Saddle points are useful for decision making along the repulsive dimensions. Repulsive dimensions
999 can additionally be useful for keeping the neural state away from a particular region of state space.
1000 For example, the neural state must remain outside of output potent space (orthogonal to the readout
1001 weights) until the response period.

1003 To identify fixed points, we empirically optimized for a set of $\{\mathbf{h}^{1*}, \mathbf{h}^{2*}, \dots\}$ satisfying $F(\mathbf{h}^*, \mathbf{u}^*) = 0$
1004 while defining \mathbf{u}^* by holding the inputs \mathbf{u} constant for each task period. Each different input condition
1005 reconfigured the RNN into a new dynamical landscape with a different set of fixed points. Therefore,
1006 each set of $\{\mathbf{h}^{1*}, \mathbf{h}^{2*}, \dots\}$ was associated with a particular input \mathbf{u}^* . At the fixed point \mathbf{h}^* with inputs
1007 \mathbf{u}^* , the update to the state at the next time point is zero and therefore, the state doesn't move away
1008 from this location. We used the term fixed point to include approximate fixed points, where the up-
1009 date is small on the timescale of our task. Our fixed points range between $q = 10^{-3}$ to 10^{-15} where
1010 $q = \frac{1}{2}[\mathbf{h}^* - f(\mathbf{h}^*, \mathbf{u}^*)]^T[\mathbf{h}^* - f(\mathbf{h}^*, \mathbf{u}^*)]$. We included a wide range of q values in order to best highlight
1011 relevant dynamics on a case by case basis.

1013
1014 We used the Fixed Point Finder package in Tensorflow [4].

1015 1.6 Input Interpolation

1016 We examined how fixed point structures moved and changed stability as the dynamical system was
1017 reconfigured by different inputs. To do this, we interpolated across pairs of inputs and identified fixed
1018 points for each intermediate input condition. For input vectors \mathbf{u}_1 and \mathbf{u}_2 , we identified fixed points
1019 for $\alpha\mathbf{u}_1 + (1 - \alpha)\mathbf{u}_2$ where α was varied between 0 and 1 in 0.05 increments.

1020 1.7 Analysis of Fixed Points for Interpolated Inputs

1021 After input interpolation [see Section 1.6 for details], we wanted to compare fixed points across input
1022 conditions in order to track their positions and stability in high dimensional space. However, there
1023 were often multiple fixed points, making it difficult to track an individual fixed point across input
1024 conditions. We focused on the fixed point closest to the state at the end of a task period of interest
1025 (except in Figure 4f, where we focused on the closest unstable fixed point because it appeared more
1026 relevant to the nonlinear dynamics - the closest fixed point was stable and was also shared across tasks).
1027 Our reasoning was that if the state evolved toward a particular fixed point, it was likely relevant for
1028 computation. After identifying fixed points during rule input interpolation, we ran the network forward
1029 from the beginning of the context period for each interpolated rule input and identified the fixed point
1030 closest to the network state at the end of the task period of interest (stimulus or response period).
1031 We refer to this closest fixed point as the 'relevant' fixed point for a given interpolated input. We
1032 calculated the Euclidean distance between relevant fixed points associated with adjacent interpolated
1033 input conditions (α_1, α_2) as:

$$d(\alpha_1, \alpha_2) = \|\mathbf{h}_{relevant}^*(\alpha_1) - \mathbf{h}_{relevant}^*(\alpha_2)\|_2. \quad (12)$$

1034 We also tracked the stability of the relevant fixed point for each interpolated input. To do this, we per-
1035 formed eigenvalue decomposition on the Jacobian of the RNN state transition function at the relevant
1036 fixed point,

$$\frac{\partial F}{\partial \mathbf{h}}(\mathbf{h}, \mathbf{u}) \Big|_{\mathbf{h}=\mathbf{h}_{relevant}^*(\alpha_1), \mathbf{u}=\mathbf{u}^*(\alpha_1)} \quad (13)$$

1037 The eigenvalue with the maximum real value is informative about whether the relevant fixed point is
1038 stable. By tracking the stability over input interpolation, we could identify bifurcations in the dynam-
1039 ical landscape.

1040 To examine the relevant dynamical motif for a given task period, we defined a 'relevant fixed point' to
1041 be the fixed point closest to the state at the end of the task period. If the input interpolation between $\alpha =$
1042 0 and $\alpha = 1$ resulted in approximately the same location of the relevant fixed point and approximately
1043 the same local dynamics around the relevant fixed point, then we defined the relevant fixed point as
1044 being functionally the same across inputs; and therefore, the dynamical motif was shared across input
1045 conditions.

1046 Alternatively, if the interpolation between $\alpha = 0$ and $\alpha = 1$ resulted in a bifurcation of the fixed
1047 point structure, then we defined the dynamical motifs to be distinct. We highlight that our definition of
1048 distinct motifs was limited in that a different path for consecutive input interpolation might not result
1049 in a bifurcation. It will be of great interest to explore ambiguous cases of shared and distinct motifs in
1050 future work.

1051 1.8 Effective Input Modulation

1052 In previous work, it was identified that relaxation dynamics of the network state could contextually
1053 integrate stimulus inputs [5]. We identified some networks that additionally contextually amplified
1054 stimulus inputs (Supplementary Figure 6). In order to deconstruct how the signal from one input
1055 is contextually amplified, we look at the first order Taylor Series approximation of the state update
1056 around a particular input, \mathbf{u}^* , and its associated fixed point, \mathbf{h}^* :

$$\frac{\partial F}{\partial \mathbf{h}}(\mathbf{h}^*, \mathbf{u}^*)\delta \mathbf{h} + \frac{\partial F}{\partial \mathbf{u}}(\mathbf{h}^*, \mathbf{u}^*)\delta \mathbf{u} \quad (14)$$

1057 The network received contextual rule input during the context period and moved toward a stable fixed
1058 point. We took this stable fixed point during the context period to be the initial conditions for the sub-
1059 sequent stimulus period. We modelled the initial stimulus input in the $\Delta u_{stimulus} = u_{stimulus} - u_{context}$
1060 term without changing $\frac{\partial F}{\partial \mathbf{u}}(\mathbf{h}_{context}, \mathbf{u}_{context})$ at the context dependent fixed point. We calculate the
1061 input response for each stimulus condition for trials spanning $[0, 2\pi)$ by calculating the norm of the
1062 dot product of

$$\left\| \frac{\partial F}{\partial \mathbf{u}}(\mathbf{h}_{context}, \mathbf{u}_{context}) \Delta u_{stimulus} \right\|_2. \quad (15)$$

1063 We found that there was task specific amplification of stimulus modality inputs, where the modality
1064 1 (2) input response was larger for tasks where the network must respond according to modality 1
1065 (2).

1066 1.9 Task variance analysis

1067 In order to examine the contributions of unit variance to computation in each task period, we used a
1068 modified version of the task variance analysis described previously [1]. We ran the network forward
1069 for a set of possible stimulus conditions on the task of interest. For example, in the Delayed Response
1070 tasks, we presented the network with trials where $\theta_{stimulus}$ ranged from $[0, 2\pi)$. In the Decision Making
1071 tasks we ran the same network with $\theta_{stimulus}$ ranging from $[0, 2\pi)$ and coherences ranging from
1072 0.005 to 0.2. We then computed the variance across possible stimulus conditions for each unit on each
1073 task period through time. This was a deviation from [1] in two ways. Firstly, we computed task period
1074 variance across stimulus conditions for all task periods separately because we considered each task
1075 period as a separate dynamical landscape. Secondly, we computed variance through time rather than
1076 averaging across time, because we were interested in the dynamics rather than static representations.
1077 Variance during the fixation period was low so we excluded the fixation period from this analysis,
1078 and we study it separately in Figure 4b-c. Private noise (ξ in equation 4) was set to zero for this
1079 analysis to eliminate the effect of recurrent noise. This analysis was a useful method to uncover unit
1080 contributions to network computations because our networks were activity regularized. Given activity
1081 regularization, any deviations from zero activity were costly for the network to produce and therefore
1082 likely beneficial for task computation. The result of this analysis was a matrix composed of columns
1083 of units and rows of task periods, where each index quantified the participation of a given unit to the
1084 computation during a given task period. We refer to this matrix as the variance matrix.

1085 Correlations between variance matrices were computed by first sorting task period rows according to
1086 one reference network. A correlation matrix for each network was computed by finding the Pearson
1087 correlation between rows of the variance matrix for that network separately. Each correlation matrix
1088 was compared to every other correlation matrix by first flattening the upper triangle of entries in
1089 each correlation matrix. We then calculated the Pearson Correlation between this vector and the same
1090 vector associated with each trained network. Both trained and untrained networks were compared to

1091 trained networks in order to determine whether the structure in trained networks emerged due to the
1092 input structure or due to learned dynamical motifs.

1093 1.10 Clusters

1094 We sorted rows and columns of the variance matrix [see Section 1.9 for details] according to similarity
1095 using the Ward variance minimization algorithm [6]. This algorithm produced a dendrogram that
1096 shows the hierarchical distance between rows or columns of the task variance matrix. In order to obtain
1097 discrete clusters, we identified the optimal distance threshold for each dendrogram by computing the
1098 silhouette score on the basis of intracluster and intercluster distances. The silhouette score of an unit
1099 i was $(d_{i,\text{inter}} - d_{i,\text{intra}}) / \max(d_{i,\text{intra}}, d_{i,\text{inter}})$, where $d_{i,\text{intra}}$ was the average distance of this unit with
1100 other units in the same cluster, and $d_{i,\text{inter}}$ was average distance between this unit and units in the
1101 nearest cluster. The silhouette score of a clustering scheme was the average silhouette score of all
1102 units. A higher silhouette score meant a better clustering. We computed the silhouette score for the
1103 number of clusters ranging from 3 to 40. The optimal number of clusters k was determined by choosing
1104 the k with the highest silhouette score. Clustering results were robust to clustering method and to the
1105 network hyperparameters that we explored.

1106 1.11 Lesions

1107 We lesioned a network unit by setting its projection weights to 0 for all recurrent and output units.
1108 When we lesioned a particular network cluster, we lesioned all units within that cluster.

1109 1.12 Transfer Learning

1110 Networks were pre-trained on a subset of tasks as described previously, where W_{in} , W_{rec} , W_{out} and
1111 bias vectors b_{in} , b_{out} were learned over the course of training [see Section 1.3 for details]. After this
1112 initial stage of training, the network was trained on a held out task. In this second phase of training, all
1113 network connections were held fixed except for the rule input weights of the held out task. That meant
1114 that in the second phase of learning, only a vector $\mathbf{u}_{\text{rule}^*}$ of size N_{rec} within W_{in} changed.

1115 In Supplementary Figure 7, we wanted to understand the relationship between how well this transfer
1116 learning approach worked and whether the held out task required learning of novel dynamical motifs.
1117 We first quantified the extent to which a task required a unique dynamical motif by comparing rows
1118 in the variance matrix [see Section 1.9 for details]. We sorted rows according to a reference network
1119 and computed the correlation matrix for the variance atlas in each network across all hyperparameter
1120 settings in Figure 3b. We then took the average across all correlation matrices. We used this aver-
1121 age correlation matrix to inform the average relationship across task periods for all networks that we
1122 examined. For each task, we identified the maximum correlation to other tasks for each task period.
1123 The most unique task period was that which had the lowest maximum correlation to other task per-
1124 iodls. Our hypothesis was that tasks with lower correlation required unique dynamical motifs whereas
1125 tasks with higher correlation could be shared across tasks. We then quantified how well our transfer
1126 learning method performed for each task. We first trained a network on all but one task in the first
1127 stage of learning and then on the held out task in the second stage of learning. We compared the cost
1128 during training each task in the second stage of transfer learning to a single task network. Single task
1129 networks were trained to perform the task of interest with all weights and biases plastic. We compared
1130 the cost at two different points in the training process, (1) early in training in order to determine the
1131 benefit from starting with previously learned dynamical motifs and (2) late in training to determine the
1132 cost of freezing all weights except the rule input for the task of interest. We then plotted the difference

1133 in cost at these two separate time points against our metric for unique task periods. The fast learning
1134 benefit was smaller and the long term cost was negative for tasks with unique dynamical motifs.

1135 Methods References

- 1136 [1] Guangyu Robert Yang, Madhura R Joglekar, H Francis Song, William T Newsome, and Xiao-Jing Wang.
1137 Task representations in neural networks trained to perform many cognitive tasks. *Nature neuroscience*,
1138 22(2):297, 2019.
- 1139 [2] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference
1140 on Learning Representations (ICLR)*, 2015.
- 1141 [3] H Francis Song, Guangyu R Yang, and Xiao-Jing Wang. Reward-based training of recurrent neural networks
1142 for cognitive and value-based tasks. *Elife*, 6:e21492, 2017.
- 1143 [4] Matthew Golub and David Sussillo. Fixedpointfinder: A tensorflow toolbox for identifying and character-
1144 izing fixed points in recurrent neural networks. *Journal of Open Source Software*, 3(31):1003, 2018.
- 1145 [5] Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computa-
1146 tion by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78–84, 2013.
- 1147 [6] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical
1148 association*, 58(301):236–244, 1963.