

Separating the *what* and *how* of compositional computation to enable reuse and continual learning

Appendix

A Formalization of compositional tasks and task designs

We propose a formalization of shared compositional structures in sequence-to-sequence (seq2seq) tasks and show that the formulation is expressive enough to capture many commonly used tasks in the literature. In each trial of a seq2seq task, the learner receives an input time sequence $\mathbf{s}_{t=1,\dots,T}$ and needs to produce a target output sequence $\mathbf{y}_{t=1,\dots,T}$. Different tasks (indexed by $c \in \mathbb{Z}^+$) are distinguished from each other by their input distributions as well as the underlying input-to-target rules, together encoded in $p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T} | c)$. For notational convenience let $\mathbf{q}_t \equiv [\mathbf{s}_t, \mathbf{y}_t] \in \mathbb{R}^{D_q}$.

We model this distribution over task-observables by introducing a set of latent variables resulting in the joint distribution

$$p(\mathbf{q}_{1:T}, z_{1:T}, x, c) = p(x|c)p(z_1|c) \prod_{t=1}^{T-1} p(z_{t+1}|z_t, c) \prod_{t=1}^T p(\mathbf{q}_t|z_t, x) p(c). \quad (6)$$

We initially provide intuition about our modeling choices, before formally specifying the distributions for the tasks of the main paper in section [A.1](#) and [A.2](#) below.

To model different tasks with a shared compositional structure, we first observe that in many tasks used in neuroscience experiments, each trial can be temporally segmented into discrete epochs, which we denote as $z_t \in \mathbb{Z}^+$. Each epoch has its distinctive input and target statistics, though they may exhibit complex temporal dependencies across epochs throughout the trial. To illustrate this, we can consider a simple task testing the learner’s ability to memorize: During a stimulus epoch, inputs provide information about a trial-specific latent variable (e.g., a dot on the screen indicating an angle θ). The inputs eventually turn off, and the learner needs to maintain the relevant information in memory until a cue solicits a θ -dependent response (e.g., a saccade in the direction of the angle). This example corresponds to the MemoryPro task from the main paper.

One might initially attempt to model this epoch dependence as a simple Hidden Markov Model (HMM), where each epoch has its own observation model $p(\mathbf{q}_t|z_t)$. However, this approach would fail to capture that the \mathbf{y}_t during the response epoch is coupled to the \mathbf{s}_t from the stimulus epoch via the shared trial-specific latent variable. In our example above, the statistics of the stimulus epoch and response epoch are coupled since they are both dependent on θ . To take this cross-epoch dependency into account, we explicitly model the “stimulus condition”, indexed by the trial variable x . In the specific example above, there may be a list of possible directions and x selects one to use as θ . Thus, in our model of cognitive tasks, the observed inputs and target responses depend not only on the latent epoch z_t but also a second latent variable x , which additionally parameterizes the observation models of all epochs.

We model tasks as being compositionally related if we can use the same underlying set of task epochs and conditional distributions over $\mathbf{q}_t|z_t, x$ to describe them. For a simple example of different tasks sharing compositional structure, we can consider the MemoryPro task described above, and construct a second task, MemoryAnti, which shares the same stimulus epoch but requires a response in the opposite direction of θ . Here, the dependence of the response epoch on θ is different (and MemoryAnti will therefore have a different response epoch relative to MemoryPro), but the statistics of the stimulus epoch are identical across the two tasks. The differences in what epochs contribute to each task and how one epoch transitions to the next is captured via task-dependent Markovian transitions over epochs $p(z_{t+1}|z_t, c)$.

A.1 Generative model

To make our model of neuroscience tasks tractable and applicable to training RNNs, we make some simplifying choices for its components. For all tasks, $p(x|c)$ is assumed to be a uniform distribution over some finite set of size N_x . $p(z_{1:T}|c)$ is assumed to be a Markov process with the initial state distribution and transition probabilities determined by c . Finally, the observation model $p(\mathbf{q}_t|z_t, x)$ is

assumed to be a multivariate Gaussian with a (z_t, x) -dependent mean. Under these assumptions, our model can be seen as an HMM with c -dependent latent-state dynamics and x -dependent emission models for all latent states. If there is only one task and one possible x value in all trials, the model reduces to a standard HMM. In the more general case, it can be viewed as a mixture (across tasks identities) of HMMs with Gaussian Mixture emissions (across trial variables x).

In summary, for trials $r = 1, \dots, N_{trials}$ we generate tasks as

$$\begin{aligned}
c_r &\sim p(c) && \text{task identity} && c_r \in \{1, \dots, N_c\} \\
z_1^r | c_r &\sim \text{Discrete}(\mathbf{\Pi}^c) && \text{epoch identity} && z_t^r \in \{1, \dots, N_z\} \\
z_t^r | z_{t-1}^r = z', c_r = c &\sim \text{Discrete}(\mathbf{\Lambda}_{:,z'}^c) \\
x_r | c_r &\sim \text{Discrete}\left(\frac{1}{N_x} \mathbf{1}\right) && \text{trial variable} && x_r \in \{1, \dots, N_x\} \\
\mathbf{q}_t^r | z_t^r = z, x_r = x &\sim \mathcal{N}(\bar{\mathbf{q}}_{z,x}, \sigma^2 I) && \text{inputs and targets} && \mathbf{q}_t^r \in \mathbb{R}^d
\end{aligned}$$

This model thus allows us to specify a set of N_c tasks sharing a compositional structure as follows. Let N_z denote the number of epochs shared among these tasks. The composition of the c -th task is specified by its epoch-transition parameters: the transition probabilities $\mathbf{\Lambda}^c \in \mathbb{R}^{N_z \times N_z} : \Lambda_{z,z'}^c = p(z_t = z | z_{t-1} = z', c)$ and initial probabilities $\mathbf{\Pi}^c \in \mathbb{R}^{N_z} : \Pi_z^c = p(z_1 = z | c)$. The observation model of each epoch is specified by the means corresponding to different x values, $\{\bar{\mathbf{q}}_{z,x} \in \mathbb{R}^{D_q}\}_{x=1, \dots, N_x}$, where $\bar{\mathbf{q}}_{z,x}$ is the mean of the multivariate Gaussian $p(\mathbf{q}_t | z_t = z, x)$. Thus, altogether an ensemble of N_c tasks composed from N_z epochs is specified by the tuple $(\{\bar{\mathbf{q}}_{z,x}\}_{z=1, \dots, N_z; x=1, \dots, N_x}, \{(\mathbf{\Lambda}^c, \mathbf{\Pi}^c)\}_{c=1, \dots, N_c})$, where the first component specifies the epochs and the second specifies how they are used to compose the tasks.

A.2 Expressing common cognitive and motor tasks in our framework

Below, we provide explicit details of the distributions and parameters used to generate our task set. We follow the same epoch notation introduced in Table 1 of the main paper. We initially describe the temporal structure in terms of epochs of each task, and then define epoch-specific distributions.

All tasks start with the *fixation* epoch (F), which presents no stimulus and an active fixation cue. The required output is to maintain fixation without response. In all the non-response epochs (*stimulus* S , *memory* M and *decision stimuli* S_{DM}), the fixation cue is on and the required output is to maintain fixation without producing a response. During the response epochs ($R_P, R_A, R_{M,P}, R_{M,A}, R_{DM,P}, R_{DM,A}$), the fixation cue turns off and the learner needs to stop fixation and produce a response ϕ according to some rule, as described below. Without loss of generality, we considered 8 possible stimulus conditions per task ($N_x = 8$) and generate x i.i.d. from a uniform distribution for the 8 possible values.

Delayed response tasks (DelayPro, DelayAnti). After F , the stimulus epoch (S) presents an angle θ , chosen from $\{0, \pi/4, \dots, 7\pi/4\}$ depending on x . The stimulus presentation stays on during the ensuing response epoch (R_P for DelayPro or R_A for DelayAnti), where the target output becomes $\phi = \theta$ (R_P) or $\phi = \theta + \pi$ (R_A).

Memory-guided response tasks (MemoryPro, MemoryAnti). After S , stimulus presentation disappears in the memory epoch (M). During the ensuing response epoch ($R_{M,P}$ for MemoryPro, $R_{M,A}$ for MemoryAnti), there is still no stimulus presentation and the learner must produce a response based on the memorized θ : $\phi = \theta$ ($R_{M,P}$) or $\phi = \theta + \pi$ ($R_{M,A}$).

Decision making tasks (DMPPro, DMAnti). After F , a decision stimuli epoch (S_{DM}) presents two stimuli simultaneously. $\theta = 0$ with strength γ in input dims 1, 2 and $\theta = \pi$ with strength γ' in input dims 3, 4. The strengths (γ, γ') are determined from a set of pairs by x and scale the input channels. During the ensuing response epoch ($R_{DM,P}$ for DMPPro, $R_{DM,A}$ for DMAnti), the stimuli persist and the required output ϕ is the direction of the stimulus with a higher strength ($R_{DM,P}$) or the one with a lower strength ($R_{DM,A}$).

The epoch structure of the different tasks is encoded via $\mathbf{\Lambda}^c$. Altogether, the 6 tasks can be described as compositions using a shared pool of 10 epochs. Given the epoch identity, the conditional distributions of the inputs and responses are generated as follows:

epoch	$\bar{\mathbf{s}}_{z,x}$	$\bar{\mathbf{y}}_{z,x}$
F, M	$[0, 0, 0, 0, 0]$	$[0, 0, 0]$
S	$[\cos \theta, \sin \theta, 0, 0, 0]$	$[0, 0, 0]$
R_P	$[\cos \theta, \sin \theta, 0, 0, 1]$	$[\cos \theta, \sin \theta, 1]$
R_A	$[\cos \theta, \sin \theta, 0, 0, 1]$	$[\cos(\theta + \pi), \sin(\theta + \pi), 1]$
$R_{M,P}$	$[0, 0, 0, 0, 1]$	$[\cos \theta, \sin \theta, 1]$
$R_{M,A}$	$[0, 0, 0, 0, 1]$	$[\cos(\theta + \pi), \sin(\theta + \pi), 1]$
S_{DM}	$[\gamma \cos \theta, \gamma \sin \theta, \gamma' \cos \theta', \gamma' \sin \theta', 1]$	$[1, 0, 0]$
$R_{DM,P}$	$[\gamma \cos \theta, \gamma \sin \theta, \gamma' \cos \theta', \gamma' \sin \theta', 1]$	$[\cos \phi, \sin \phi, 1]$, where $\phi = \mathbf{1}_{\gamma > \gamma'} \theta + \mathbf{1}_{\gamma' > \gamma} \theta'$
$R_{DM,A}$	$[\gamma \cos \theta, \gamma \sin \theta, \gamma' \cos \theta', \gamma' \sin \theta', 1]$	$[\cos \phi, \sin \phi, 1]$, where $\phi = \mathbf{1}_{\gamma < \gamma'} \theta + \mathbf{1}_{\gamma' < \gamma} \theta'$

Where $\mathbf{1}$ is an indicator that takes on value 1 if the subscript is true and 0 otherwise. This fully specifies the conditional distributions $p(\mathbf{q}_t | z_t, x)$ for each task epoch with $\bar{\mathbf{q}}_{z,x} = [\bar{\mathbf{s}}_{z,x}, \bar{\mathbf{y}}_{z,x}]$. For the DM tasks, we restrict the stimulus values to two locations $\theta = 0, \theta' = \pi$, but pick $N_x = 8$ different combinations of possible (γ, γ') pairs. Note that while we generate the input and response distributions using stimulus values θ to follow convention from the literature [11, 14], each value of θ (or (γ, γ') in the DM tasks) maps onto a different x value. For σ we used 0.05; for (γ, γ') , x selects from $[0.5, 1], [1, 2], [0.5, 2], [0.2, 1.5], [1, 0.5], [2, 1], [2, 0.5], [1.5, 0.2]$.

Note that while we have followed many conventions from previous work in the task design, these previous approaches tend to implement each task individually but with related distributional assumptions [11, 13, 14, 16]. Modeling shared structure across tasks through an explicit, shared generative framework for an entire task family is novel. While this was not the focus of the main paper, it is worth noting that access to a description of shared statistical structure in the input and target response pairs of each task forms an important baseline for expectations of shared statistical structure across tasks in the solution emerging after RNN training [11, 13-15].

B Online learning and inference of compositional task structures

All algorithms discussed here are implemented in the [code repository](#) under `task_model.py`.

B.1 Online learning and inference

In this section, we provide additional details on the algorithms developed for performing posterior inference over the latent variables of the task model, and online (one-trial-at-a-time) learning of the task model.

Inference. We can perform exact inference by utilizing a message passing scheme similar to that used for performing inference in classic HMM models.

Let $\alpha_t^r(z, x, c) = p(\mathbf{q}_{1:t}^r, z_t^r = z | x_r = x, c_r = c)$ denote the forward (filtering) message for a given task. During a filtering pass, we compute

$$\alpha_1^r(z, x, c) = p(z_1^r = z | c_r = c) p(\mathbf{q}_1^r | z_1^r = z, x_r = x) \quad (7)$$

$$\alpha_{t+1}^r(z, x, c) = \left(\sum_{z'=1}^{N_z} \alpha_t^r(z', x, c) p(z_{t+1}^r = z | z_t^r = z', c_r = c) \right) p(\mathbf{q}_{t+1}^r | z_{t+1}^r = z, x_r = x) \quad (8)$$

Marginalizing the forward message at the final time-step allows us to compute the marginal likelihood over observations

$$p(\mathbf{q}_{1:T}^r | x_r = x, c_r = c) = \sum_{z=1}^{N_z} \alpha_T^r(z, x, c) \quad (9)$$

$$(10)$$

Let $\beta_t^r(z, x, c) = p(\mathbf{q}_{(t+1):T}^r | z_t^r = z, x_r = x, c_r = c)$ denote the backwards (smoothing) message.
 During the smoothing pass, we compute

$$\beta_T^r(z, x, c) = 1 \quad (11)$$

$$\beta_t^r(z, x, c) = \sum_{z'=1}^{N_z} p(z_{t+1}^r = z' | z_t^r = z, c_r = c) p(\mathbf{q}_{t+1}^r | z_{t+1}^r = z', x_r = x) \beta_{t+1}^r(z', x, c) \quad (12)$$

Given these quantities, we can compute the joint posterior over z_t^r , x_r and c_r as

$$\gamma_t^r(z, x, c) = p(z_t^r = z, x_r = x, c_r = c | \mathbf{q}_{1:T}^r) \quad (13)$$

$$= \frac{p(\mathbf{q}_{1:t}^r, z_t^r = z | x_r = x, c_r = c) p(\mathbf{q}_{(t+1):T}^r | z_t^r = z, x_r = x, c_r = c) p(x_r = x, c_r = c)}{\sum_{x,c} p(\mathbf{q}_{1:T}^r | x_r = x, c_r = c) p(x_r = x, c_r = c)} \quad (14)$$

$$= \frac{\alpha_t^r(z, x, c) \beta_t^r(z, x, c) p(x_r = x, c_r = c)}{\sum_{x,c} \sum_{z'=1}^{N_z} \alpha_T^r(z', x, c) p(x_r = x, c_r = c)} \quad (15)$$

Note that the filtering and smoothing passes can be done for each value of x and c in parallel. Finally,
 for later use in the online learning algorithm we also compute the joint posterior over z_{t-1}^r , z_t^r , x_r
 and c_r as

$$\xi_t^r(z, z', x, c) = p(z_{t-1}^r = z, z_t^r = z', x_r = x, c_r = c | \mathbf{q}_{1:T}^r) \quad (16)$$

$$= \frac{\alpha_{t-1}^r(z, x, c) p(z_t^r = z' | z_{t-1}^r = z, c_r = c) p(\mathbf{q}_t^r | z_t^r = z', x_r = x) \beta_t^r(z', x, c) p(x_r = x, c_r = c)}{\sum_{x,c} p(\mathbf{q}_{1:T}^r | x_r = x, c_r = c) p(x_r = x, c_r = c)} \quad (17)$$

Online learning. Learning aims to recover parameters of the generative model,
 $(\{\bar{\mathbf{q}}_{z,x}\}_{z=1,\dots,N_z; x=1,\dots,N_x}, \{(\mathbf{\Lambda}^c, \mathbf{\Pi}^c)\}_{c=1,\dots,N_c})$, up to a permutation over z and x . Performing
 parameter learning with EM requires computing the expected counts of visiting particular epochs
 or transitioning across epochs across all trials. When all trials are available as a batch, this involves
 sums over the quantities computed during inference for each trial. To make notation more compact,
 we denote X as the set of sufficient statistics needed to update the set of model parameters θ . Let
 $\theta^{(i,k)}$ denote the learned parameters after seeing the i -th trial and running k EM iterations. Let
 $X^{r,(i,k)}$ denote the single-trial statistics on trial r , computed using parameters $\theta^{(i,k)}$. When the
 learning algorithm has access to all trials throughout learning (batch EM), the updates take the form

$$\theta^{(i,k+1)} = f(S^{(i,k)}(X)) \quad S^{(i,k)}(X) = \sum_{r=1}^i X^{r,(i,k)}. \quad (18)$$

For example, for the parameter estimates for the transition matrix, this takes the form

$$\Lambda_{czz'}^{i,k+1} = \frac{\sum_{t=1}^T \sum_{x=1}^{N_x} \Xi_{czz'}^{i,k}(t)}{\sum_{t=1}^T \sum_{x=1}^{N_x} \sum_{z''=1}^{N_z} \Xi_{czz''}^{i,k}(t)}. \quad (19)$$

with

$$\Xi_{czz'}^{i,k}(t) = \sum_{r=1}^i p(c_r = c, x_r = x, z_{t-1}^r = z, z_t^r = z' | \mathbf{q}_{1:T}^r; \theta^k). \quad (20)$$

In general, the batch EM parameter updates take well-known forms for HMMs and GMM-HMMs,
 which is why we only give an example here in the interest of brevity.

When trials are only available one-trial-at-a-time and cannot be revisited (online EM), the sums of
 expected sufficient statistics across trials have to be approximated and updated after each trial instead,
 leading to an approximation to the batch updates. We propose the following update rule to perform
 parameter learning online

$$\theta^{(i,k+1)} = (1 - \eta_{\text{params}}) \theta^{(i-1,K)} + \eta_{\text{params}} f(S_{\text{online}}^{(i,k)}(X)). \quad (21)$$

$$S_{\text{online}}^{(i,k+1)}(X) = (1 - \eta_{\text{stats}}) S_{\text{online}}^{(i-1,K)}(X) + X^{i,(i,k)}. \quad (22)$$

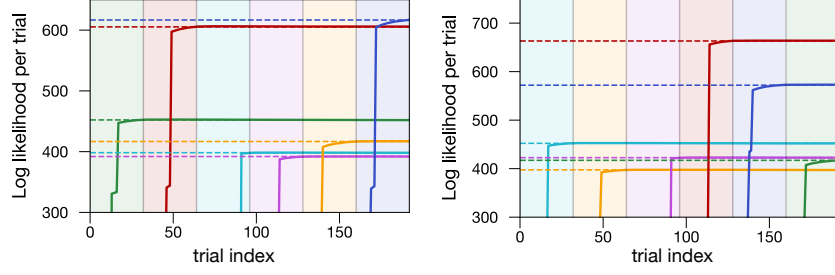


Figure S.1: Online continual learning of task structures with some example task orderings. See Fig 2 for legends.

143 where K denotes the number of iterations per trial. After learning the i -th trial, we only need to store
 144 $\theta^{(i,K)}$ and $S_{online}^{(i,K)}(X)$, taking the form

$$\theta^{(i,K)} = \sum_r^i (1 - \eta_{params})^{i-r} \eta_{params} f(S_{online}^{(r,K)}(X)) + (1 - \eta_{params})^{i+1} \theta_{init}. \quad (23)$$

$$S_{online}^{(i,K)}(X) = \sum_r^i (1 - \eta_{stats})^{i-r} X^{r,(r,K)}. \quad (24)$$

145 We summarize this online learning approach in Algorithm 1 and show performance for different
 146 training orders (supplementing Figure 2 in the main paper) in Figure S.1.

Algorithm 1 High-level Overview of the Incremental EM algorithm.

```

 $\theta \leftarrow \text{init.}$  ▷ Initialize parameter estimates.
 $S(X) \leftarrow \mathbf{0}$  ▷ These are the estimated sums of sufficient statistics across trials
for  $r = 1, \dots, N_{trials}$  do
   $\theta \leftarrow \text{incremental\_initialize}(\theta, q_{1:T_r}^r, c_r)$  ▷ See B.2.
   $\hat{\theta} \leftarrow \theta$  ▷ Create a temporary copy of the parameters
   $\hat{S}(X) \leftarrow \text{None}$ 
  for  $k = 1, \dots, K$  do
     $X \leftarrow \text{getstats}(q_{1:T_r}^r, c_r, \hat{\theta})$  ▷ Compute sufficient statistics of this trial with current
    params.
     $\hat{S}(X) \leftarrow (1 - \eta_{stats})S(X) + X$  ▷ Decay the sums from the previous trial and add new
    stats
     $\hat{\theta} \leftarrow \theta \odot (1 - \eta_{params} M_{gate}) + \eta_{params} f(\hat{S}(X)) \odot M_{gate}$  ▷ Incremental
    update of params. using the modified sums.  $M_{gate}$  is a binary mask controlling which parameters
    are updated (see B.3).
  end for
   $\theta \leftarrow \hat{\theta}$ 
   $S(X) \leftarrow \hat{S}(X)$ 
end for

```

147 B.2 Incremental initialization of model parameters

148 Even for simple models such as the Gaussian Mixture Model, EM is susceptible to local optima [45].
 149 To avoid convergence to bad local optima, it is important to obtain good initializations for the model
 150 parameters. For simple time-series models such as the HMM, this is typically achieved by collapsing
 151 the sequences across time, performing clustering (e.g. with K-means [46]), and using the resulting
 152 cluster centers as the initial observation-model parameters, given each latent discrete state.

153 In our model, we need to initialize the underlying cluster centers of the observation model,
 154 $\{\hat{q}_{z,x}\}_{z=1,\dots,\hat{N}_z, x=1,\dots,N_x}$. Here, \hat{N}_z reflects the fact that the total number of epochs in the en-
 155 tire dataset is unknown *a priori*. We overcome this by setting up a large number of ‘slots’ ($\hat{N}_z \geq N_z$),

156 exceeding the likely total number of epochs in the task family. Our setting adds two significant
 157 challenges relative to simple HMMs: First, each trial contains only a few epochs and a specific
 158 stimulus condition (x value). Therefore, clustering each trial can only initialize a small subset of the
 159 $N_x N_z$ mixture components of the observation model. Second, since there is one stimulus condition
 160 x per trial, the mixture components $\bar{q}_{z,x}$ explaining a given trial must be allocated to the same x ,
 161 but different epoch states z . Initialization schemes based on simple clustering approaches on the
 162 entire trial (such as that outlined for HMMs above) would be agnostic to this structure and fail to
 163 provide a feasible initial set of parameters. To overcome these issues, we introduce an ‘incremental
 164 initialization’ scheme, which is applied to all the estimated means $\{\hat{q}_{z,x}\}$ before learning each trial.

165 To introduce the scheme, we first introduce the notion of the putative z, x , denoted \tilde{z}, \tilde{x} . This is to
 166 highlight that learning the correct $\{\bar{q}_{z,x}\}$ does not require inferring the real z, x in the generative
 167 process but only requires them to be correct up to a permutation. Let \tilde{N}_z, \tilde{N}_x denote the number of
 168 epochs and task slots in the learning algorithm, respectively. Note that these do not need to be set
 169 as the correct N_z, N_x – they can simply be large integers. We do assume that the system knows the
 170 correct number of x values, N_x . The learner keeps track of $\{\hat{q}_{\tilde{z},\tilde{x}}\}_{\tilde{z}=1,\dots,\tilde{N}_z,\tilde{x}=1,\dots,\tilde{N}_x}$ as well as two
 171 Boolean-valued tables, $F_{c,x}$ of size $\tilde{N}_c \times N_x$ and $F_{x,z}$ of size $\tilde{N}_x \times \tilde{N}_z$. The two tables keep track
 172 of which combinations of c, \tilde{x} and \tilde{x}, \tilde{z} have been encountered.

173 When the r th trial is observed and the learner has access to $(\mathbf{q}_{1:T_r}^r, c_r)$, the scheme is threefold:

- 174 1. We perform K-means clustering on the entire sequence $\mathbf{q}_{1:T_r}^r$. This gives us a set of cluster
 175 means. Since we assume the epochs to have piecewise constant inputs and mild noise, the
 176 means correspond to the different epochs that appeared in this trial. The challenge now is to
 177 assign \tilde{z}, \tilde{x} to these means, and to use them to initialize $\hat{q}_{\tilde{z},\tilde{x}}$ accordingly.
- 178 2. All cluster means from step (1) should be assigned the same \tilde{x}_r . We check the means against
 179 $\{\hat{q}_{\tilde{z},\tilde{x}}\}$ and decide on \tilde{x}_r according to a set of rules and $F_{c,x}$. The c_r, \tilde{x}_r pair is marked
 180 ‘encountered’ in $F_{c,x}$.
- 181 3. Given \tilde{x}_r from step (2), we treat the cluster means not found in $\{\hat{q}_{\tilde{z},\tilde{x}}\}_{\tilde{x}=\tilde{x}_r}$ as unfamiliar
 182 \tilde{z}, \tilde{x} , meaning that they represent previously unseen \tilde{z}, \tilde{x} combinations and should be used
 183 to initialize. Each center is assigned a different \tilde{z} that has not been encountered (according
 184 to $F_{x,z}$). The \tilde{z}, \tilde{x}_r pairs are marked as ‘encountered’ in $F_{x,z}$.

185 B.3 Gated updates to parameters

186 Since each trial contains information about only one task and the few epochs it is associated with, it
 187 does not make sense to update parameters related to other tasks and epochs. For Λ^c , we simply gate
 188 it such that only the transition matrix corresponding to the current task (the label of which is given) is
 189 updated. For $\{\hat{q}_{z,x}\}$, we infer which epochs appeared in this trial using the posterior $p(z_{1:T}^r | \mathbf{q}_{1:T}^r)$.
 190 Only epochs with a sufficiently high chance of appearance have their $\{\hat{q}_{z,x}\}$ updated.

191 B.4 Epoch identifiability in our set of tasks

192 For the particular set of tasks we considered, since F, M epochs have identical $\bar{q}_{z,x}$ for all x , they
 193 are indistinguishable. Thus, our learning algorithm will combine them into a single epoch, which
 194 we denote as F/M . Instead of learning the 10 epochs in the generative process for our set of tasks,
 195 the task-model will end up learning 9 epochs, including the combined F/M epoch. In terms of the
 196 transition matrices, this creates a complication for MemoryPro, MemoryAnti tasks where both F
 197 and M epochs appear in each trial. The learned transitions will not be deterministic in the sense that
 198 the F/M epoch may transition to either the stimulus epoch or the response epoch. The ‘ground-truth’
 199 parameters used for plotting in Fig. 2 and Fig. S.1 refer to the optimal parameters with a merged F/M
 200 epoch. In future work, it will be interesting to investigate how epochs that are computationally distinct
 201 (e.g. holding still in F vs. holding still while maintaining a memory in M) but map onto the same
 202 observations may be distinguished, e.g. via feedback from the downstream network implementing
 203 the different computations for each epoch.

204 C RNN architecture and hyperparameter settings

205 C.1 Default RNN architecture and task parameters

Parameter	Value
α	0.1
σ_r	0.05
ϕ	ReLU
number of hidden units	256
rank of U_z, V_z	3
input noise std	$\sqrt{2/\alpha} \sigma_{in}$, where $\sigma_{in} = 0.01$
minimum duration of a epoch	5 time steps
$p(z_{t+1} = z_t c)$	0.9

207 C.2 Default training protocol

208 We used a batch size of 256 and trained each task for 1000 batches unless otherwise specified.

209 For the context-modulated RNN, the learning rate for the weights associated with each context z was
 210 initially set to $\eta_z = 0.001$. After training each task c , η_z was multiplied by a decay factor $\gamma = 0.5$ for
 211 any context with $p(z | c) > 0.001$. During training on task c , L_2 weight regularization was applied
 212 with a coefficient of 10^{-5} for contexts with $p(z | c) > 0.001$, and set to 0 for all other contexts.

213 For baseline algorithms using standard (“vanilla”) RNN architectures, the learning rate was set to 0.01,
 214 and the L_2 weight regularization coefficient was set to 10^{-5} . Parameter choices were determined by
 215 a coarse grid search.

216 C.3 Loss function and performance measure

217 The loss function is a weighted mean square error similar to [11, 14]. $L := \langle m_{i,t} (y_{i,t} - \hat{y}_{i,t})^2 \rangle_{i,t}$,
 218 where i is the index of the output units, $m_{i,t} = 1$ for response epochs and $m_{i,t} = 0.2$ for all other
 219 epochs.

220 A trial is considered correct if the network maintained fixation for all time steps before the fixation
 221 cue turns off, and responded to the correct direction for time steps in the response epoch. If the
 222 activity of the fixation output exceeds 0.5, the network is considered to have broken fixation. The
 223 network’s response direction is considered correct if its angular difference from the target direction is
 224 less than $\pi/10$. Average performance and test loss were calculated on 200 test trials for each task.

225 D Supplementary results

226 D.1 Additional results on transfer learning

227 We provide additional results on transfer learning. Figure S.2 and Figure S.3 supplement Figure S.a
 228 and b, respectively, by showing results for all task pairs. Figure S.4 shows the lack of backward
 229 transfer when training with the OWP algorithm [13], where the test loss of previously learned tasks
 230 did not decrease when training on subsequent tasks with overlapping epochs.

231 D.2 Results with other hyperparameter choices

232 We observed improved performance of our continual learning algorithm as we increased the rank
 233 of U_z and V_z (denoted by r) (Figure S.5a,b, compared with Figure 4f). Using $r = 3, 5$, and 10 in
 234 ContextRNN results in 34587, 43803, and 66843 trainable parameters, respectively — all fewer
 235 than the 69379 parameters of a general RNN with the same number of hidden units. With $r = 3$,
 236 ContextRNN performed worse with tanh than with ReLU activation, but this gap was closed at $r = 10$

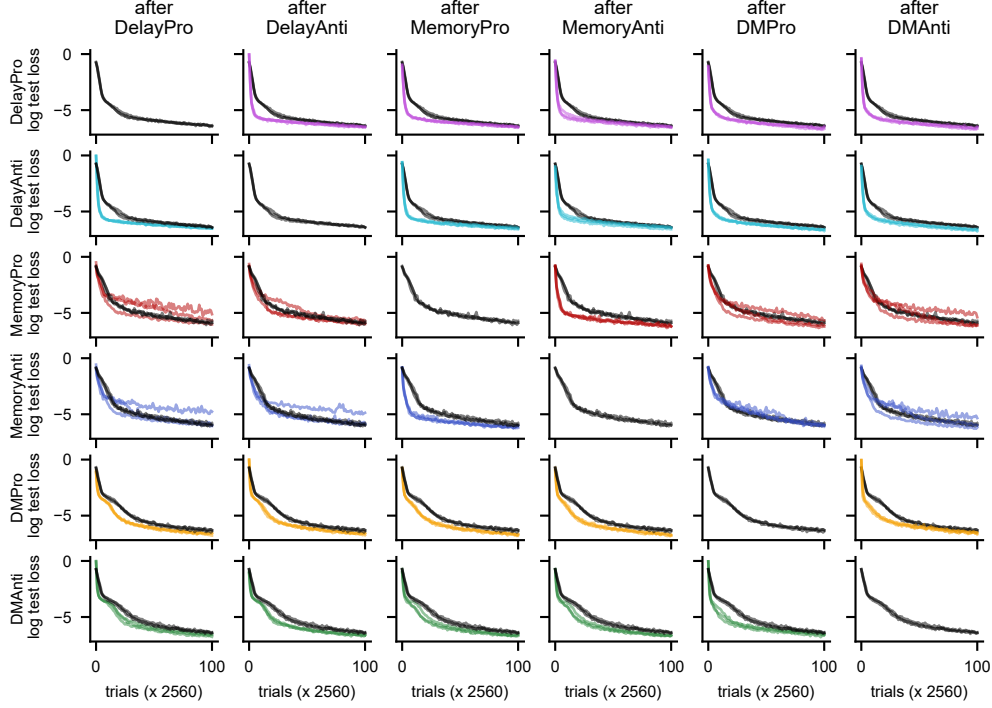


Figure S.2: Additional results on forward transfer. Colored curves show the log test loss of task B (indicated by the row label) when trained after task A (column label). For comparison, black curves in each row show the log test loss of task B when trained from scratch. We plot results across three random seeds for each task order.

(Figure S.5c, d). Similarly, performance with $r = 3$ and $\alpha = 0.2$ was worse than with $\alpha = 0.1$, but this gap was also closed at $r = 10$ (Figure S.5e, f).

E Other continual learning approaches

E.1 Elastic Weight Consolidation

Elastic Weight Consolidation (EWC) [30], slows down learning rates for network weights deemed important for previous tasks. This is achieved by adding a regularization term to the training loss function. For a loss $\mathcal{L}(\theta)$, the EWC objective is given as

$$\mathcal{L}^{EWC}(\theta) = \mathcal{L}(\theta) + \frac{\lambda}{2} \sum_i F_i (\theta_i - \theta_i^*)^2 \quad (25)$$

Here F_i is an importance weight that ties the i th parameter value θ_i to its value θ_i^* at the end of training on the previous task. The important weights are computed as the diagonal of the Fisher Information matrix F evaluated at the parameter values θ_i^* . In Figure 4b, λ was determined by a coarse grid search and set to 10^5 .

E.2 Orthogonal Weight Projection

The original learning rule in Duncker et al. [13] was derived for a parameterization of network dynamics of the form

$$\dot{\mathbf{x}} = -\mathbf{x} + \phi(\mathbf{W}^{\text{rec}}\mathbf{x} + \mathbf{W}^{\text{in}}\mathbf{s}) \quad (26)$$

given an element-wise activation function $\phi(\cdot)$. The learning rule was intended to maintain the stimulus/response relationship of previous task by applying a set of projection matrices to the gradient used to update the network weights over learning. The projection matrices were intended to remove

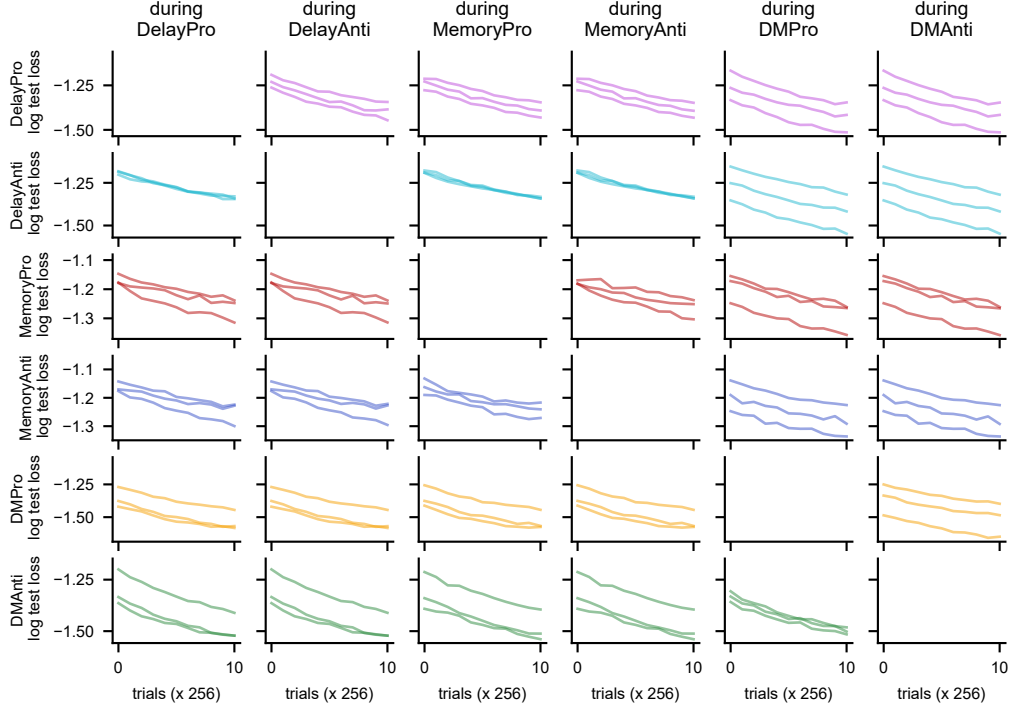


Figure S.3: Additional results on backward transfer. The log test loss of the previously trained task A (indicated by the row label) continue to decrease during subsequent training on another task B (column label). Each task is trained with 2560 trials and we plot results across three random seeds for each task order.

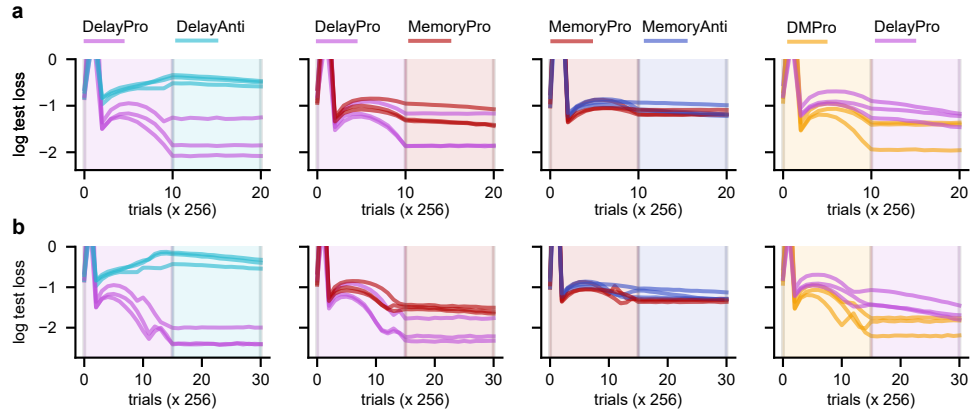


Figure S.4: No backward transfer with OWP. **a**: The log test loss during sequential training of two tasks with the OWP algorithm, each tasks is trained for 2560 trials. The loss of the previous tasks does not decrease after switching to a new task, indicating no backward transfer. Results are shown for three random seeds per task order. **b**: same as **a** but with each task trained for 3840 trials.

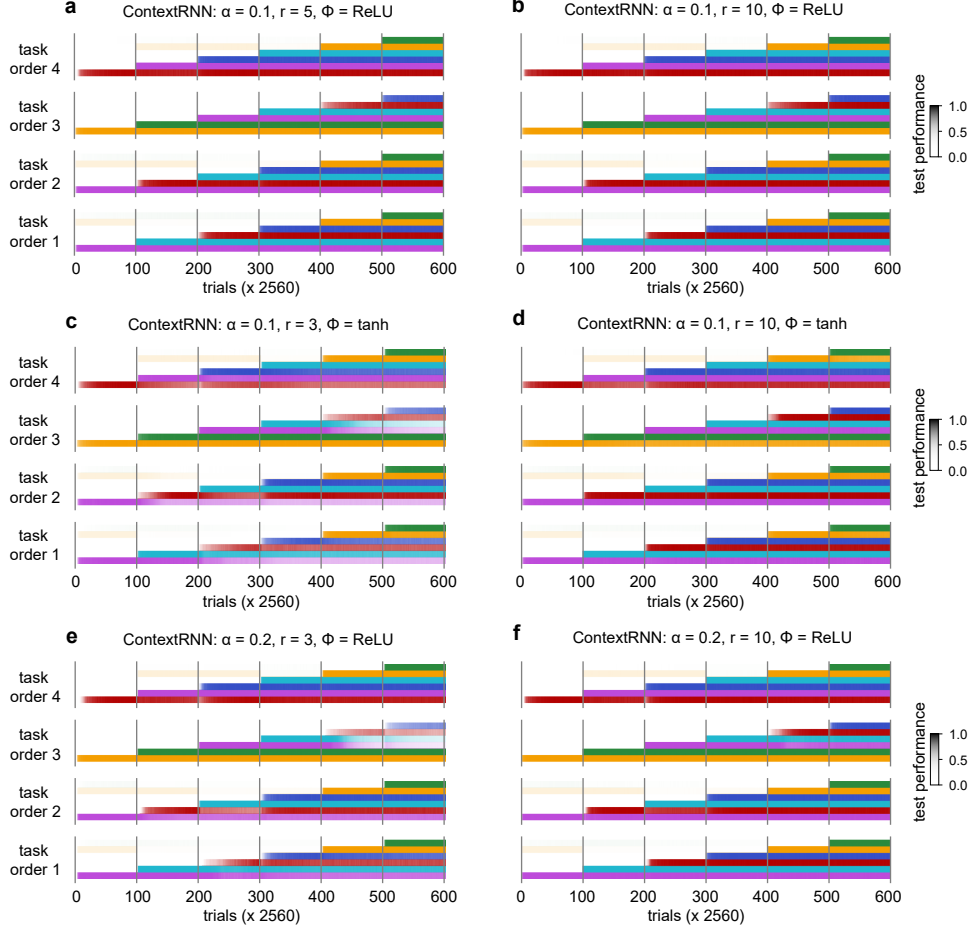


Figure S.5: Continual learning performance with different hyperparameter choices. **a**: Color-coded test performance during sequential training of four different task orders. Each row color-codes the average test performance across five random seeds of a specific task over training. RNNs used for this panel have $\alpha = 0.1, r = 5, \phi = \text{ReLU}$. **b-f**: same as **a** but with a different hyperparameter choice labeled by the title.

254 directions from the weight update that would interfere with previous tasks and were defined as follows.

255 Letting $\mathbf{z}_t^{c,r} = \begin{bmatrix} \mathbf{x}_t^{c,r} \\ \mathbf{s}_t^{c,r} \end{bmatrix}$ denote the concatenated network state and input state at time t of trial r on

256 task c , $\mathbf{Z}_{1:c} = [\mathbf{z}_1^{1,1}, \dots, \mathbf{z}_T^{c,r}]$ the collection of all time points and trials on tasks 1 through c , and

257 $\mathbf{W} = [\mathbf{W}^{\text{rec}} \mathbf{W}^{\text{in}}]$ the concatenated weight matrices, we define the projection matrices

$$\mathbf{P}_1^{1:c} = \left(\frac{1}{\lambda} \mathbf{Z}_{1:c} \mathbf{Z}_{1:c}^T + \mathbf{I} \right)^{-1} \quad (27)$$

$$\mathbf{P}_2^{1:c} = \left(\frac{1}{\lambda} \mathbf{W} \mathbf{Z}_{1:c} \mathbf{Z}_{1:c}^T \mathbf{W}^T + \mathbf{I} \right)^{-1} \quad (28)$$

258 and the modified learning update as

$$\Delta \mathbf{W}^{\text{CL}} \propto \mathbf{P}_2^{1:c} \nabla_{\mathbf{W}} \mathcal{L} \mathbf{P}_1^{1:c} \quad (29)$$

259 where $\nabla_{\mathbf{W}} \mathcal{L}$ is the derivative of the loss on the new task with respect to the network weights. An
260 analogous set of projection matrices and modified learning update is used for the readout weights.

261 To facilitate direct comparisons with our approach, we adapted the learning rule to a modified setting,
262 where the RNN dynamics are expressed as

$$\dot{\mathbf{h}} = -\mathbf{h} + \mathbf{W}^{\text{rec}} \phi(\mathbf{h}) + \mathbf{W}^{\text{in}} \mathbf{s} \quad (30)$$

263 given the same element-wise activation function $\phi(\cdot)$. While the two parameterizations in (26) and
 264 (30) are generally considered equivalent [47], the linear intuition used to motivate the approach of
 265 [13] should be exact in (30). With $\mathbf{x}_t = \phi(\mathbf{h}_t)$, $\mathbf{Z}_{1:c}$ is unchanged, but we instead use

$$\mathbf{P}_2^{1:c} = \left(\frac{1}{\lambda} \mathbf{H}_{1:c} \mathbf{H}_{1:c}^\top + I \right)^{-1} \quad (31)$$

266 where $\mathbf{H}_{1:c} = [\mathbf{h}_1^{1,1}, \dots, \mathbf{h}_T^{c,r}]$. While this is very similar to the version in Duncker et al. [13], the
 267 projections matrix now only depends implicitly on the \mathbf{W} of previous tasks. We performed all
 268 comparisons using this modified learning rule.

269 F Code repository

270 We have uploaded an anonymized code repository with all code used for training and some examples
 271 at https://anonymous.4open.science/r/context_inference_rnn-8176.