GRAPH CONVOLUTIONAL MEMORY USING TOPOLOGICAL PRIORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Solving partially-observable Markov decision processes (POMDPs) is critical when applying reinforcement learning to real-world problems, where agents have an incomplete view of the world. We present graph convolutional memory (GCM), the first hybrid memory model for solving POMDPs using reinforcement learning. GCM uses either human-defined or data-driven topological priors to form graph neighborhoods, combining them into a larger network topology using dynamic programming. We query the graph using graph convolution, coalescing relevant memories into a context-dependent belief. When used *without* human priors, GCM performs similarly to state-of-the-art methods. When used *with* human priors, GCM outperforms these methods on control, memorization, and navigation tasks while using significantly fewer parameters.

1 INTRODUCTION

Reinforcement learning (RL) was designed to solve *fully observable* Markov decision processes (MDPs) (Sutton & Barto, 2018, Chapter 3), where an agent knows its true state – a property that rarely holds in the real world. Problems where agent state is ambiguous, incomplete, noisy, or unknown can be modeled as partially-observable MDPs (POMDPs). RL guarantees optimal policy convergence for POMDPs when a *belief* is maintained over an episode (Cassandra et al., 1994). Storing information and retrieving it later for belief estimation is known as *memory* (Moreno et al., 2018).

Memory models in RL tend to be either general or task-specific. General memory comes from the field of sequence learning, with recurrent neural networks (RNNs), transformers, or memory augmented neural networks (MANNs) as notable examples. General memory assumes a sequential ordering, but makes no other assumptions about the inputs and can be applied to any POMDP. These models excel in supervised learning, but tend to be costly in terms of training time and number of parameters. RL exacerbates these problems with its sparse and noisy learning signal (Beck et al., 2020).

The substantial cost of training general memory drives many to design task-specific memory for RL applications, like Chaplot et al. (2020); Parisotto & Salakhutdinov (2017); Gupta et al. (2017); Lenton et al. (2021) which build 2D or 3D maps for navigation, or Li et al. (2018a) which utilizes dosing information to inform a tree search over hospital patient states. Task-specific memory is built around human-defined prior knowledge, and aimed at solving a specific task. The downside of this memory is that it must be implemented by hand for each specific task. Many RL applications would benefit from task-specific memory, but the implementation is non-trivial: outside of the core RL research community there are chemists (Zhou et al., 2017), painters (Huang et al., 2019), or roboticists (Morad et al., 2021) trying to solve field-specific problems using RL.

This paper is the first to propose a *hybrid* memory model. Our memory model, termed Graph Convolutional Memory (GCM)[1] is applicable to any partially-obserable RL task, but utilizes task-specific *topological priors*. In other words, GCM builds a graph structure from topological priors, which are either human-designed or learned entirely from data. GCM has a simple interface, enabling users to develop topological priors suited to their specific tasks in a few lines of code (Sec. 2.2). Fig. 1

[1] GCM is available at https://anonymous.4open.science/r/graph-conv-memory-55AD/README.md

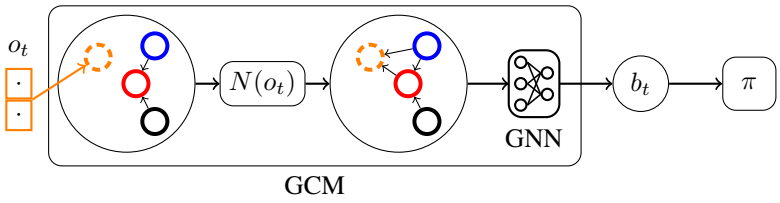


Figure 1: GCM flowchart for an incoming observation o_t. GCM places o_t as a node in a graph, and computes its neighborhood $N(o_t)$, and then updates the edge set. Task-specific topological priors are defined via the neighborhood. A convolutional GNN queries the graph for belief state b_t. A policy π uses the belief for decision making. Compared to transformers or DNCs, GCM is architecturally simple.

presents an overview of our method. GCM builds a graph, and defines local neighborhoods using said priors, resulting in an expressive graph topology via dynamic programming. The edge structure induced by the graph provides interpretability, and facilitates simple debugging. We leverage the computational efficiency and representational power of graph neural networks (GNNs) to extract contextualized beliefs from the graph. GCM can be applied to any sequence learning problem, but in this paper we focus on RL. In our experiments, we show that GCM uses significantly fewer parameters than RNNs, MANNs, or transformers, but performs comparably without any human priors. With human-designed priors, GCM outperforms the general memory used in our experiments.

2 Graph Convolutional Memory

We model a POMDP following Kaelbling et al. (1998) with tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \Omega, \mathcal{O})$. At time t we enter hidden state $s_t \in \mathcal{S}$ and receive observation $o_t \sim \mathcal{O}(s_t) : S \to \Omega$. We sample action $a_t \in \mathcal{A}$ from policy π and follow transition probabilities $\mathcal{T}(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ to the next state s_{t+1}, receiving reward $R(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We learn π to maximize the expected cumulative discounted reward subject to discount factor γ: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$. In an MDP, the policy uses the true state $\pi(s_t) : \mathcal{S} \to \mathcal{A}$, but in a POMDP our policy uses the *belief* state $\pi(b_t) : \mathcal{B} \to \mathcal{A}$. Our goal in this paper is to construct a latent belief state b_t using memory function M and memory state m_t:

$$(b_t, m_t) = \mathrm{M}(o_t, m_{t-1}). \tag{1}$$

2.1 Model Description

We implement GCM following Eq. 1 using Alg. 1. GCM stores a collection of experiences over an episode, with each experience represented by an observation vertex o and associated neighborhood $N(o)$. We query the set of experiences using a graph neural network (GNN) to produce a context-dependent belief $b_t|o_t$.

Algorithm 1 Graph Convolutional Memory

procedure $\mathrm{M}(o_t, m_{t-1})$
  $V, E \leftarrow m_{t-1}$ # Unpack memory
  $V \leftarrow V \cup o_t$ # Add observation
  $E \leftarrow E \cup \{(o_t, o_i)\}_{i \in N(o_t)}$ # Update edges
  $Z \leftarrow \mathrm{GNN}_\theta(V, E)$ # Get embedding
  $b_t \leftarrow Z\,[t]$ # At current vertex
  $m_t \leftarrow V, E$ # Pack into memory
  return b_t, m_t # Belief and memory
end procedure

In detail, at time t, we insert vertex o_t into the graph, producing $m_t = (V_t, E_t)$ where $V_t = (o_1, \ldots o_t)$ and $E_t : V_t \times V_t$. We determine the neighborhood $N(o_t)$ using *topological priors* defined in Sec. 2.2, and update the edges following:

$$E_t = E_{t-1} \cup \{(o_t, o_i) \mid i \in N(o_t)\} \tag{2}$$

We query the graph for context-dependent information using a GNN with layers $h \in \{1 \ldots \ell\}$. We convolve over $o_1, \ldots o_t$ to produce hidden representations $z_1^h, \ldots z_t^h$ for each hidden layer, propagating information from the h^{th}-degree neighbors of o_t into z_t^h. After collecting and integrating data across the ℓ^{th}-degree neighborhood, we output z_t^ℓ as the belief. This provides a mechanism for fast and relevant feature aggregation over memory graphs, depicted in Fig. 2.

As an example, let us consider a visual navigation task where the neighborhood consists of the previous observation index $N(o_t) = \{t-1\}$. Let o_1, o_2, o_3 represent "chair", "wall", and "table", respectively. The first GNN layer combines o_1, o_2 into a "chair-room" embedding and o_2, o_3 into a

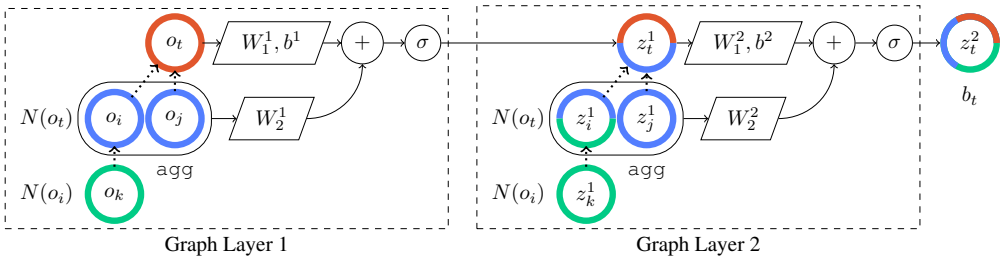


Figure 2: The two-layer 1-GNN used in all our experiments. Colors denote mixing of vertex information and dashed lines denote directed edges, forming neighborhoods $N(o_t), N(o_i)$. The current observation o_t and aggregated neighboring observations o_i, o_j pass through fully-connected layers $(W_1^1, b^1), (W_2^1)$ before summation and nonlinearity σ, resulting in the first hidden state z_t^1 (Eq. 3). We repeat this process at o_j, o_k, o_l to form hidden states z_j^1, z_k^1, z_l^1. The second layer combines embeddings of the first layer and the second-layer hidden state z_t^2 is output as the belief state b_t. Additional layers increase the GNN receptive field.

"table-room" embedding. The second layer combines "chair-room" and "table-room" into "dining-room", and outputs "dining-room" as the belief. We found the 1-GNN defined in Morris et al. (2019) empirically outperformed graph isomorphism networks (Xu et al., 2019) and the original graph convolutional network (Kipf & Welling, 2017). GCM can utilize any GNN, but our GNNs are built from the 1-GNN convolutional layer defined as:

$$z_t^h = \sigma\left[W_1^h z_t^{h-1} + b^h + W_2^h \texttt{agg}\left(\left\{z_i^{h-1} | i \in N(o_t)\right\}\right)\right] \tag{3}$$

with σ representing a nonlinearity and $z_t^0 = o_t, z_i^0 = o_i$ for the base case. At each layer h, weights and biases W_1^h, b^h produce a root vertex embedding while W_2^h generates a neighborhood embedding using vertex aggregation function $\texttt{agg}$. Separate weights allow the 1-GNN to weigh each h^{th}-degree neighborhood's contribution to the belief, ignoring the neighborhood and decomposing into an MLP for empty or uninformative neighborhoods. The root and neighborhood embeddings are combined to produce the layer embedding z_t^h (Fig. 2). Notice, the weights W_1^h, b^h in Eq. 3 form an MLP, so GCM does not require an MLP preprocessor like other memory models (Mnih et al., 2016).

2.2 Topological Priors

We use the shorthand $N(o_t)$ to define the open neighborhood of o_t over vertices V_t, in edge-list format. We compute $N(o_t)$ using the union of one or more *topological priors* $\Phi_i : \Omega^t \to 2^{V_{t-1}}$, as in

$$N(o_t) : V \to 2^{V_{t-1}} \coloneqq \bigcup_{i=1}^{k} \Phi_i(V_t). \tag{4}$$

Breaking down the graph connectivity problem into easier neighborhood-forming subtasks is a form of *dynamic programming*. The priors are task-specific, because for different tasks, we often want different connectivity. For example, in control problems, associating memories temporally could be beneficial in learning a dynamics model. In navigation, spatial connectivity could aid with loop closures. We have implemented spatial, temporal, latent similarity, and other topological priors in Tab. 1, but GCM can utilize any mapping from vertices to a neighborhood. We provide task-specific examples for medicine and aerospace in Appendix B.

Learning Topological Priors We stress the importance of human knowledge in forming the graph topology, but there are cases where we have no information about the problem, or where designing a topological prior is non-trivial. A naïve formulation of prior learning via gradient descent is challenging due to a large number of possible edges represented by boolean values. One option is to use a fully-connected graph with weighted edges similar to Veličković et al. (2018), but this would hinder interpretability of the graph (Jain & Wallace, 2019) and be computationally expensive.

Instead, we propose to learn a probability distribution over all edges, from which we sample to explore the large edge space. We do not know the ideal neighborhood size, so we must learn this as well. The Gumbel-Softmax Estimator (Jang et al., 2016) enables differentiable sampling from

Prior Description	$\Phi(V)$ Definition
Empty: o_t has no neighbors and GCM decomposes into an MLP.	$\emptyset$ (5)
Dense: Connects o_t to all other observations $o_1 \dots o_{t-1}$.	$\{1, 2, \dots t-1\}$ (6)
Temporal: Similar to the temporal prior of an LSTM, where each observation o_t is linked to some previous $t-c$ observation.	$\{t-c\}$ (7)
Spatial: Connects observations taken within c meters of each other, useful for problems like navigation. Let $\mathrm{p}(\cdot)$ extract the position from an observation.	$\left\{ i \,\middle\vert\, \begin{array}{c} \lvert\lvert \mathrm{p}(o_i) - \mathrm{p}(o_t) \rvert\rvert_2 \leq c \\ \text{and } 0 < i < t \end{array} \right\}$ (8)
Latent Similarity: Links observations in a non-human readable latent space (e.g. autoencoders). Various measures like cosine or L_2 distance may be used depending on the space. e is an encoder function, d is a distance measure, and c is user-defined.	$\left\{ i \,\middle\vert\, \begin{array}{c} \mathrm{d}\left(\mathrm{e}\left(o_i\right), \mathrm{e}\left(o_t\right)\right) < c \\ \text{and } 0 < i < t \end{array} \right\}$ (9)
Identity: Connects observations where two values are identical, useful in discrete domains where inputs are related. a, b are indexing functions ($\mathrm{a} = \mathrm{b}$ may hold).	$\left\{ i \,\middle\vert\, \begin{array}{c} \mathrm{a}(o_i) - \mathrm{b}(o_t) = 0 \\ \text{and } 0 < i < t \end{array} \right\}$ (10)

Table 1: Knowledge-based priors for GCM

a categorical distribution using the reparameterization trick from Kingma & Welling (2014). We leverage this to build a multinomial distribution across all possible edges, and use the maximum a posteriori (MAP) estimate to form the neighborhood (Fig. 3).

Assume we want to sample between one and K edges at time t for the neighborhood $N(o_t)$. First, we compute logits l_i for all possible edges $(o_i, o_t), i \in \{1, \dots t-1\}$ in Eq. 11, using a neural network ϕ_θ, positional encoding e_p from Vaswani et al. (2017), and concatenation operator $||$. Then, we sample Gumbel noise $g_i^k, k \in \{1, \dots K\}$ using the inverse CDF of the Gumbel distribution and uniform random sampling $\mathcal{U}(0, 1)$ in Eq. 12. Using Gumbel noise g_i^k, we build a multinomial distribution from K Gumbel-Softmax distributions X_k (Eq. 13) which we "sample" by taking the MAP (argmax) (Eq. 14). Note that we are not actually sampling from a single categorical distribution, but taking the MAP over many perturbed categorical (i.e. multinomial) distributions. This approach is formalized as follows:

$$l_i = \phi_\theta \left(\mathrm{e}_p(o_i) \,||\, \mathrm{e}_p(o_t)\right) \tag{11}$$

$$g_i^k = -\log(-\log u); u \sim \mathcal{U}(0, 1) \tag{12}$$

$$P(X_k) = \left\{ \frac{\exp\left(\log l_i + g_i^k\right)}{\sum_{j=1}^{t-1} \exp\left(\log l_j + g_i^k\right)} \,\middle\vert\, 1 \leq i < t \right\} \tag{13}$$

$$\Phi(V_t) \coloneqq \left\{ \arg\max P(X_k) \mid 1 \leq k \leq K \right\}. \tag{14}$$

Random sampling helps our method explore more neighborhood possibilities while still providing boolean values for edges. Since we are sampling with replacement, the neighborhood size can vary from 1 to K, depending on the kurtosis of the learned distribution. For tasks where information is replicated over many observations, Eq. 14 can learn a flat distribution, sampling many distinct edges and building a large neighborhood. In cases where a few key observations contain salient information, we can learn a peaked distribution to more reliably sample these edges.

We do not run into the interpretability trap of transformers explained in Jain & Wallace (2019), because we learn a distribution of edges rather than attention weights over the edges. In a transformer, an infinitesimal perturbation in attention weights can cause a significant change in model output. Our model learns a distribution, where an infinitesimal perturbation corresponds to an infinitesimal shift of probability mass. An infinitesimally-shifted distribution will not significantly impact the drawn samples, or in turn the learned prior output. Furthermore, the learned prior should be robust to noisy distributions, since the distributions themselves are constantly perturbed with Gumbel noise during learning. We note this sampling methodology might also be useful for designing differentiable hard and sparse self-attention in transformers.

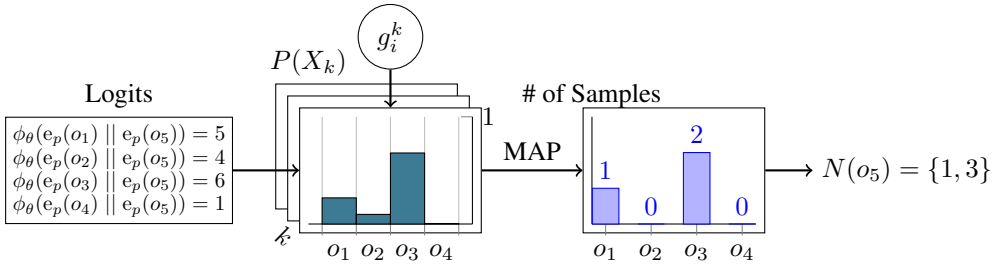


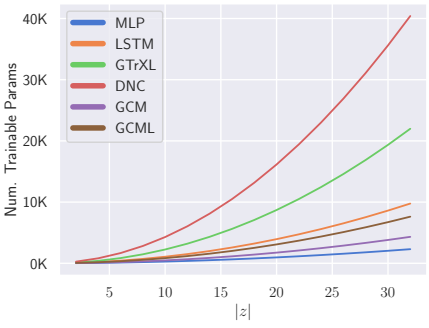
Figure 3: Generating one to K edges in a non-deterministic manner, using our learned topological prior. We present an example of the learned prior with $K = 3$ at $t = 5$. An MLP computes logits over previous vertices to produce three distributions $X_k; k \in \{1, 2, 3\}$, perturbed with Gumbel noise g_i^k. We compute the MAP for each X_k, resulting in three samples which form a two-edge neighborhood $N(o_5)$ containing vertices o_1, o_3. This process is fully-differentiable and trained end-to-end with GCM.

3 Experiments

We evaluate GCM on control (Fig. 5a), non-sequential long-term recall (Fig. 5b), and indoor navigation (Fig. 5c). We run three trials for each memory model across all experiments and report the mean reward per train batch with a 90% confidence interval. We test six contrasting models, and base our evaluation on the hidden size of the memory models, denoted as $|z|$ in Fig. 4. Nearly all hyperparameters are Ray RLlib defaults, tuned for RLlib's built-in models.[2] Appendix C defines all training environment details and Appendix D contains a complete list of hyperparameters.

We compare GCM to an MLP and four alternative memory models in all our experiments. The **MLP** model is a two-layer feed-forward neural network using $\tanh$ activation. It has no memory, and forms a performance lower bound for the memory models. The **LSTM** memory model is a MLP followed by an LSTM cell, the standard model for solving POMDPs (Mnih et al., 2016). **GTrXL** is a MLP followed by a single-head GRU-gated transformer XL. The **DNC** is an MLP followed by a neural computer with an LSTM-based memory controller. Our memory model, **GCM**, uses a two-layer 1-GNN using $\tanh$ activation with sum (cartpole and concentration) and mean (navigation) neighborhood aggregation. The **GCML** is the GCM with the learned topological prior (Eq. 14) with $K = 5$, and ϕ_θ as a two-layer MLP using ReLU and LayerNorm (Ba et al., 2016).

Partially Observable Cartpole Our first experiment evaluates memory in the control domain. We use a partially observable form of cartpole-v0, where the observation corresponds to positions rather than velocities (Fig. 5a). We optimize our policy using proximal policy optimization (PPO) Schulman et al. (2017). The equations of motion for the cartpole system are a set of second-order differential equations containing the position, velocity, and acceleration of the system (Barto et al., 1983). Using this information, we use GCM with temporal priors, i.e., $N(o_t) = \{t-1, t-2\}$ (Tab. 1) and present results in Fig. 6.



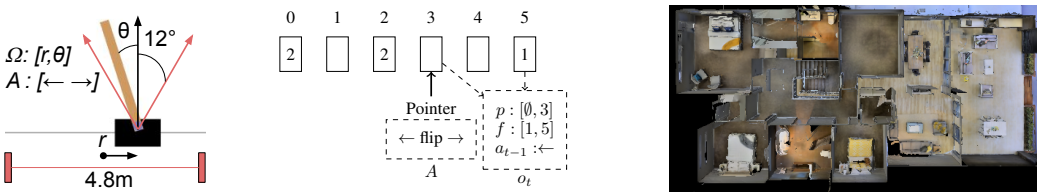
(a)

Model	Meaning of $\|z\|$
MLP	Layer size
LSTM	Size of hidden and cell states
GTrXL	Size of the attention head and position-wise MLP
DNC	LSTM size, word width, and number of memory cells
GCM	Size of the graph layers
GCML	Size of graph layers and ϕ_θ layers

(b)

Figure 4: (a) The number of trainable parameters per memory model, based on the hidden size $|z|$. GCM uses much fewer parameters than other memory models. (b) The meaning of $|z|$ with respect to each memory model, as used in all our experiments.

[2]The MLP, LSTM, DNC, and GTrXL are standard Ray RLlib (Liang et al., 2018) implementations written in Pytorch (Paszke et al., 2019). We implement GCM using Pytorch Geometric (Fey & Lenssen, 2019), and integrate it into RLlib.



(a) Partially obs. cartpole (b) Concentration card game (c) Habitat 3D simulation

Figure 5: Visualizations of our experiments. (a) The classic cartpole control problem, but where $\dot{r}, \dot{\theta}$ are hidden. (b) An example state from the long-term non-sequential recall environment with six cards. The observation space o_t contains the value and index of pointer card p and last flipped card f, as well as previous action a_{t-1}. (c) The top-down view of the 3D scene used in our navigation experiment.

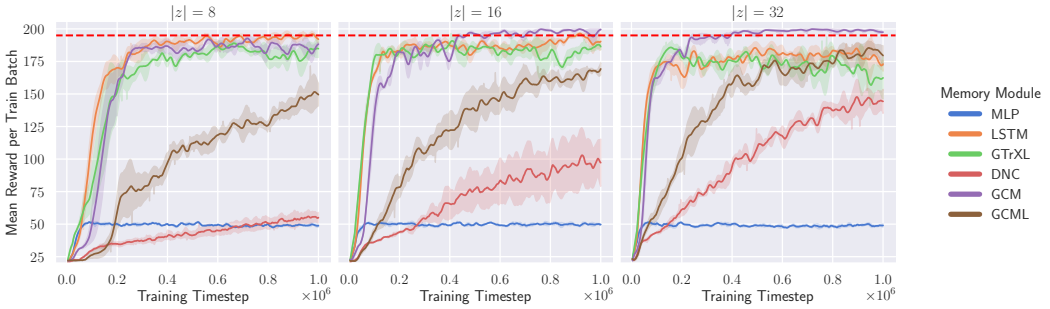


Figure 6: Stateless cartpole, where the agent must derive velocity from past observations. OpenAI considers fully-observable cartpole solved at a reward of 195 (dashed red line), which only GCM can reliably reach in partially-observable cartpole. Results represent the mean and 90% confidence interval over three trials.

Concentration Card Game Our next experiment evaluates non-sequential and long-term recall with the concentration card game.[3] Unlike reactionary cartpole, this experiment tests memorization and recall over longer time periods. We vary the number of cards $n \in \{8, 10, 12\}$ with episodes lengths of $50, 75, 100$ respectively. All models have $|z| = 32$ and train using PPO. We use GCM with temporal priors for short-term memory and an additional value identity prior between the face-up card and the card at the pointer, using function $v : \Omega \to \mathbb{N}$:

$$N(o_t) = \{t-1, t-2\} \cup \{i | v(o_t) = v(o_i)\}. \tag{15}$$

In other words, when GCM flips a card face up, it recalls if it has seen that card in the past. We present the results in Fig. 7.

Navigation The final experiment evaluates spatial reasoning with a navigation task. We use the Habitat simulator with the validation scene from the 2020 Habitat Challenge (Fig. 5c). We train for 10M timesteps using IMPALA (Espeholt et al., 2018), examining $z \in \{8, 16, 32\}$ across all models. Fig. 8 is an ablation study across multiple topological priors and GCM setups. We evaluate the effectiveness of empty, dense, temporal, spatial, and learned priors (formally defined in Tab. 1). We also examine whether the larger receptive field induced by the second-degree neighborhood is helpful with the FlatLocal and FlatGlobal entries. The FlatLocal GCM uses the spatial prior and replaces the second GNN layer with a fully-connected layer, restricting GCM to its first-degree neighborhood. The FlatGlobal GCM is the FlatLocal GCM, but with an increased-distance spatial prior such that the first-degree neighborhood includes the first and second-degree neighborhoods of the spatial entry. Fig. 9 compares GCM with learned and spatial topological priors against other memory baselines.

[3]Rules for concentration are available at: https://en.wikipedia.org/wiki/Concentration_(card_game)

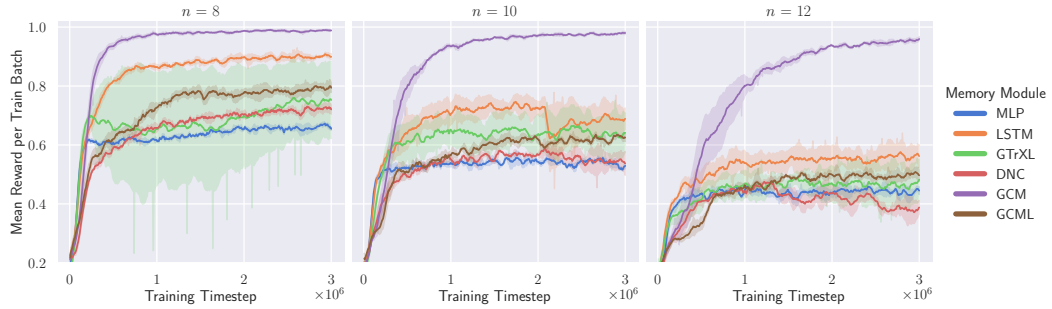


Figure 7: Results from concentration with hidden size $|z| = 32$, where n is the number of cards. This tests the agent's long-term non-sequential memory. The agent receives a small reward for matching a pair of cards, receiving a cumulative reward of one for matching all pairs. Episode lengths are 50, 75, and 100 respectively. Results are averaged over three trials and the shaded area represents the 90% confidence interval.

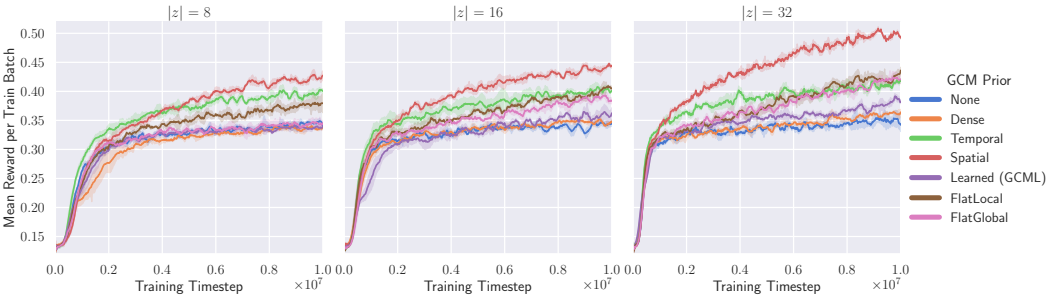


Figure 8: We compare various GCM priors across hidden sizes $|z|$ for the navigation problem. Since navigation is a spatial problem, the spatial prior performs best. This shows the importance of selecting good priors. Results are averaged over three trials and the shaded area represents the 90% confidence interval.

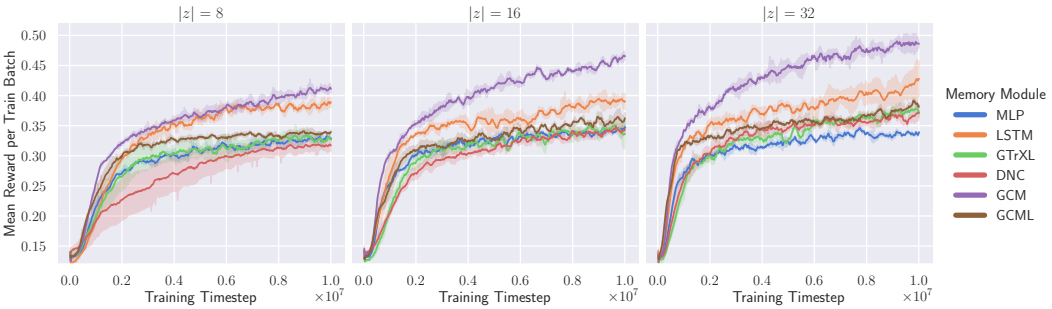


Figure 9: We compare GCM to other memory baselines for the navigation problem. $|z|$ denotes the hidden size used across all models. Results are averaged over three trials and the shaded area represents the 90% confidence interval.

4 DISCUSSION

The versatility of GCM compared to other models stems from its representation of experiences as a graph. This allows it to access specific observations from the past, bypassing the limited temporal range of LSTM. By using a multilayer GNN to reason over this graph of experiences, GCM can build embeddings hierarchically, unlike transformers. The importance of hierarchical reasoning is demonstrated experimentally in Fig. 8, where the GCM outperforms the FlatGlobal GCM, which merges the first and second-degree neighborhood from the spatial prior into a first-degree neighborhood. To build some intuition about why this is the case, consider an observation graph in a navigation experiment. Reasoning over this structure hierarchically can break down the task into manageable subtasks. The first GNN layer can fuse neighborhood viewpoints to represent local surroundings, and the second layer can combine its neighborhood of local surroundings into regions for planning. On the other hand, in a flat representation, information about the relationships between individual observations is lost. The flat model receives unstructured observations, and must learn to differentiate nearby observations from distant ones.

Like Beck et al. (2020), we find sequence learning is much harder in RL than supervised learning. This is particularly clear in Fig. 7, where introducing one more pair of cards decreases reward. Although general memory models can learn optimal policies in theory, this was not the case given our timescales. The LSTM performs well but does not reliably solve (i.e. reach 195 reward) stateless cartpole, even with small 2-dimensional observation and action spaces, and a large number of inner and outer PPO iterations (Heess et al. (2015), Fig. 6). Even though transformers significantly outperform LSTMs in supervised learning (Vaswani et al., 2017), their added complexity seems to hinder them in RL, at least at single-GPU scales. The memory search space over all past observations is huge, and determining which observations are useful greatly reduces what the memory model must learn.

GCM's graph structure can utilize external information about which experiences are relevant, greatly reducing the search space. Human intuition is an incredibly useful tool that cannot be easily leveraged by transformers, RNNs, or MANNs. This is the key contribution of our work – a prior defined by a few lines of code can significantly boost RL performance. GCM provides an easy way to embed this intuition, using more general priors (Tab. 1) or task-specific priors (Sec. 2.2). For more complex problems where human intuition falls short, GCML can learn a prior. Then, the resulting human-interpretable graph of observations can inform the development of new, human-derived topological priors (Appendix A).

In our experiments, we use simple environments to demonstrate how model-dependent memory connectivity affects performance. Models like LSTM work nearly as well as GCM on problems like cartpole where a temporal prior makes sense (Fig. 6), but the gap widens on the concentration environment where non-temporal priors are more suitable (Fig. 7). The navigation ablation study (Fig. 8) demonstrates how using a suboptimal topological prior can negatively impact performance – the dense prior (a fully-connected graph) performs nearly as poorly as the empty prior (no edges at all) in Fig. 8. Exploratory trials of combining human priors with learned priors resulted in performance greater than GCML alone, but not as good as GCM with human priors. Future work could evaluate these mixed priors on more complex problems over longer timescales, where human priors are less useful.

Across all experiments, GCM with human expertise received significantly more reward than the next best competitor. We believe that this is remarkable, considering that GCM uses notably fewer parameters than the other models (Fig. 4). GCML performance was generally less than LSTM, and most similar to the transformer performance across our experiments. Caveat emptor: we tackled simple tasks using smaller models, due to our limited computational capacity. These conclusions might not hold for those who train markedly larger models for billions of timesteps. Given more compute and longer episodes, we suspect the transformer and possibly GCML would outperform LSTM, like in Parisotto et al. (2019).

5 RELATED WORK

Memory in Reinforcement Learning We classify RNNs, MANNs, transformers, and related memory models as general memory. RNN-based architectures, such as long short-term memory

(LSTM) (Hochreiter & Schmidhuber, 1997) and the gated recurrent unit (GRU) (Chung et al., 2014) are used heavily in RL to solve POMDP tasks (Oh et al., 2016; Mnih et al., 2016; Mirowski et al., 2017). RNNs update a recurrent state by combining an incoming observation with the previous recurrent state. Compared to transformers and similar methods, RNNs fail to retain information over longer episodes due to vanishing gradients (Li et al., 2018b). By connecting relevant experiences directly and doing a single forward pass, GCM sidesteps the vanishing gradient issue.

MANNs address limited temporal range of RNNs (Graves et al., 2014). Unlike RNNs, MANNs have addressable external memory. The differentiable neural computer (Graves et al., 2016) (DNC) is a fully-differentiable general-purpose computer that coined the term MANN. In the DNC, a RNN-based memory controller uses content-based addressing to read and write to specific memory addresses. The MERLIN MANN (Wayne et al., 2018) outperformed DNCs on navigation tasks. The implementations of the MANNs are much more complex than RNNs. In contrast to transformers or RNNs, MANNs are much slower to train, and benefit from more compute.

The transformer is the most ubiquitous implementation of self-attention (Vaswani et al., 2017). Until the gated transformer XL (GTrXL), transformers had mixed results in RL due to their brittle training requirements (Mishra et al., 2018). The GTrXL outperforms MERLIN, and by extension, DNCs in Parisotto et al. (2019). When learning topological priors, our GCML borrows concepts from transformers such as positional encodings. The self-attention module in a transformer can be implemented using a single graph attention layer over a fully-connected graph (Joshi, 2020). Unlike self-attention in transformers, GCML is hard, sparse, and hierarchical. GCM does not use attention weights, nor a fully-connected graph.

Similar to our work, Savinov et al. (2018) build an observation graph, but specifically for navigation tasks, and do not use GNNs. Wu et al. (2019) use a probabilistic graphical model to represent spatial locations during indoor navigation. Eysenbach et al. (2019); Emmons et al. (2020) build a state-transition graph similar to our observation graph for model-based RL, but use A* and other methods to evaluate the graph.

Graph Neural Networks GNNs are most easily understood using a message-passing scheme (Gilmer et al., 2017), where each vertex in a graph sends and receives latent messages from its neighborhood. Each layer in the GNN learns to aggregate incoming messages into a hidden representation, which is then shared with the neighborhood. Convolutional graph neural networks (Kipf & Welling, 2017) are a subcategory of GNNs and a generalization of convolutional neural networks (CNNs) to the graph domain. Convolutional GNNs tend to be efficient in both the computational and parameter sense due to their use of sliding filters and reliance on batched sums and matrix multiplies.

Our method shares some similarities with graph attention networks (Veličković et al., 2018), namely the pairwise-vertex MLP to compute edge significance. Graph RNNs (Ruiz et al., 2020) are a generalization of RNNs to graph inputs with a fixed number of time-varying vertices, and tackle an entirely different problem than GCM. Chen et al. (2019); Li et al. (2019); Chen et al. (2020) apply GNNs to RL for task-specific problems. Beck et al. (2020) implement feature aggregation for RL in a similar fashion to GNNs. The aggregated memory operator by Zweig et al. (2020) combines a GNN with a RNN to navigate a graph of states using reinforcement learning. To date, GCM is the only *task-agnostic* RL memory model to utilize GNNs.

6 Conclusion

GCM provides a framework to embed task-specific priors into memory, without writing task-specific memory from the ground up. Embedding custom toplogical priors in the graph is trivial: it involves implementing a boolean function that determines whether or not observation o_i is useful for decision making at time t. This simple feature makes GCM very versatile. Moreover, with GCML we showed that GCM can also be purposed to *learn* topological priors *without* human input, and in this case performs similarly to a transformer. When even basic domain knowledge is available (e.g., when the problem is spatial, or when it follows Newton's laws) GCM outperforms transformers, LSTMs, and DNCs, while using significantly fewer parameters.

7 ETHICS STATEMENT

GCM is entirely open-source and free to use for your own purposes, we just ask you cite our work if it was helpful to you, and that you not use it in any application that harms others. This research was funded via public and private grants for application in intelligent navigation, multiagent RL, and GNN-based systems. It is possible GCM could be used to solve RL tasks for nefarious purposes.

8 REPRODUCABILITY STATEMENT

The most up-to-date version of GCM as well as all edge selectors are publicly available on GitHub. Please follow the GitHub README.md for how to install and use GCM.

Please see README.md in the supplemental material for how to replicate our precise experimental setup using Docker, and how to rerun the experiments. We use the Ray RLlib open-source implementations of PPO and IMPALA, and provide all hyperparameters in Appendix D. All experimental trials were run three times, and we represent the mean and 90% confidence interval for all experiments. Our experiments are non-deterministic due to asynchronous GPU operations, but we took great care to select hyperparameters that result in consistent and reproducible reward curves, as evident in the relatively low-variance confidence interval in our results. Environmental details, such as observation and action spaces are explained in Appendix C.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. jul 2016. URL https://arxiv.org/abs/1607.06450v1http://arxiv.org/abs/1607.06450.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Jacob Beck, Kamil Ciosek, Sam Devlin, Sebastian Tschiatschek, Cheng Zhang, and Katja Hofmann. AMRL: Aggregated Memory For Reinforcement Learning. *International Conference on Learning Representations (ICLR)*, pp. 1–14, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. jun 2016.

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pp. 1023–1028, 1994.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, 2018. doi: 10.1109/3DV.2017.00081.

Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to Explore using Active Neural SLAM. In *International Conference on Learning Representations (ICLR)*. arXiv, apr 2020.

Fanfei Chen, John D. Martin, Yewei Huang, Jinkun Wang, and Brendan Englot. Autonomous Exploration Under Uncertainty via Deep Reinforcement Learning on Graphs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. arXiv, jul 2020.

Kevin Chen, Juan Pablo de Vicente, Gabriel Sepúlveda, Fei Xia, Alvaro Soto, Marynel Vázquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, 2019. doi: 10.15607/rss.2019.xv.010.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

Scott Emmons, Ajay Jain, Michael Laskin, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning. In *Advances in Neural Information Processing Systems*, volume 2020-Decem, 2020.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Boron Yotam, Firoiu Vlad, Harley Tim, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *35th International Conference on Machine Learning, ICML 2018*, volume 4, 2018.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric. *arXiv*, 2019. URL http://arxiv.org/abs/1903.02428.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *34th International Conference on Machine Learning, ICML 2017*, volume 3, pp. 2053–2070. PMLR, jul 2017. ISBN 9781510855144. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. oct 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, oct 2016. doi: 10.1038/nature20101.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pp. 7272–7281. Institute of Electrical and Electronics Engineers Inc., nov 2017. doi: 10.1109/CVPR.2017.769.

Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. dec 2015. URL https://arxiv.org/abs/1512.04455v1http://arxiv.org/abs/1512.04455.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Zhewei Huang, Shuchang Zhou, and Wen Heng. Learning to paint with model-based deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, pp. 8708–8717, 2019. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00880.

Sarthak Jain and Byron C. Wallace. Attention is not explanation. In *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, volume 1, pp. 3543–3556, 2019. ISBN 9781950737130. doi: 10.18653/v1/N19-1357.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, nov 2016. URL http://arxiv.org/abs/1611.01144.

Chaitanya Joshi. Transformers are Graph Neural Networks, feb 2020. URL https://graphdeeplearning.github.io/post/transformers-are-gnns/.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. ISSN 00043702. doi: 10.1016/s0004-3702(98)00023-x.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

Daniel Lenton, Stephen James, Ronald Clark, and Andrew J. Davison. End-to-End Egospheric Spatial Memory. In *International Conference on Learning Representations*, sep 2021. URL http://arxiv.org/abs/2102.07764.

Dong Li, Qichao Zhang, Dongbin Zhao, Yuzheng Zhuang, Bin Wang, Wulong Liu, Rasul Tutunov, and Jun Wang. Graph attention memory for visual navigation, may 2019.

Luchen Li, Matthieu Komorowski, and Aldo A. Faisal. The Actor Search Tree Critic (ASTC) for Off-Policy POMDP Learning in Medical Decision Making. *arXiv preprint arXiv:1805.11548*, 2018a. URL http://arxiv.org/abs/1805.11548.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466, 2018b. ISBN 9781538664209. doi: 10.1109/CVPR.2018.00572.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *35th International Conference on Machine Learning, ICML 2018*, volume 7, pp. 4768–4780, 2018.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Volodymyr Mnih, Adria Puigdomenech Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, pp. 2850–2869, 2016. ISBN 9781510829008.

Steven D. Morad, Roberto Mecca, Rudra P.K. Poudel, Stephan Liwicki, and Roberto Cipolla. Embodied Visual Navigation with Automatic Curriculum Learning in Real Environments. *IEEE Robotics and Automation Letters*, 6(2):683–690, apr 2021. doi: 10.1109/LRA.2020.3048662.

Pol Moreno, Jan Humplik, George Papamakarios, Bernardo Avila Pires, Lars Buesing, Nicolas Heess, and Theophane Weber. Neural belief states for partially observed domains. In *NeurIPS 2018 workshop on Reinforcement Learning under Partial Observability*, 2018.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 4602–4609, 2019. doi: 10.1609/aaai.v33i01.33014602.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of Memory, Active Perception, and Action in Minecraft. Technical report, jun 2016.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, Vancouver, feb 2017. arXiv.

Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning, oct 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated Graph Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 68:6303–6318, 2020. ISSN 19410476. doi: 10.1109/TSP.2020.3033962.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, 2019. doi: 10.1109/ICCV.2019.00943.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pp. 5999–6009, 2017.

Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised Predictive Memory in a Goal-Directed Agent. *arXiv*, mar 2018.

Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Bayesian relational memory for semantic visual navigation. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, pp. 2769–2779. Institute of Electrical and Electronics Engineers Inc., oct 2019. doi: 10.1109/ICCV.2019.00286.

Keyulu Xu, Stefanie Jegelka, Weihua Hu, and Jure Leskovec. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. Optimizing Chemical Reactions with Deep Reinforcement Learning. *ACS Central Science*, 3(12):1337–1344, 2017. ISSN 23747951. doi: 10.1021/acscentsci.7b00492.

Aaron Zweig, Nesreen K Ahmed, Ted Willke, and Guixiang Ma. Neural Algorithms for Graph Navigation. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, pp. 1–11, oct 2020. URL https://openreview.net/pdf?id=sew79Me0W0c.

A INTERPRETING MEMORY GRAPHS

One of the strengths of GCM is that the memory is an interpretable graph. We can see precisely which past observations contribute to the current belief. In this section, we provide an example memory graph from the cartpole and navigation experiments using the learned topological prior (GCML).

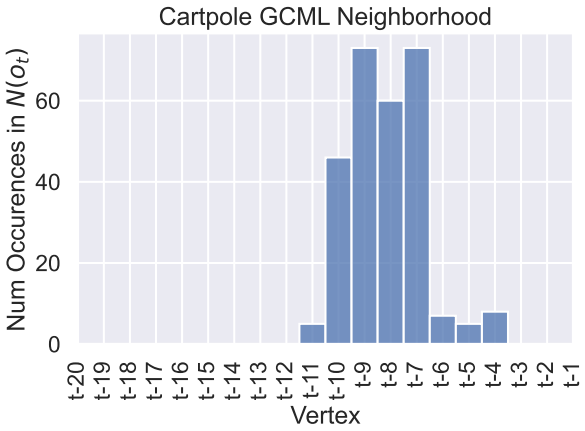


Figure 10: What do GCML neighborhoods look like for stateless cartpole? At each episodic timestep $t > 20$, we record the neighborhood $N(o_t)$ GCML produces relative to t. We plot the accumulation of all neighborhoods over an episode, using $|z| = 32, K = 5$. We see that GCML learns a temporal prior, where each timestep uses observations from 7 to 10 timesteps ago. Surprisingly, GCML does not use the preceding observation, suggesting that the simulation contains high-frequency variations which prevent an accurate estimation of velocity between consecutive timesteps. Perhaps a human-defined prior with $\{t-7\}$ would produce a smoothed signal, leading to better performance than our $\{t-1, t-2\}$ prior. With $K = 5$, the maximum possible neighborhood size is 5. The mean neighborhood size is 1.54, demonstrating that GCML can learn a peaked distribution, resulting in sparse graphs.

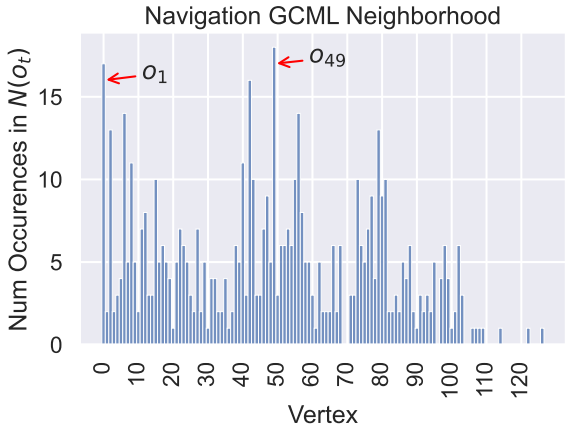
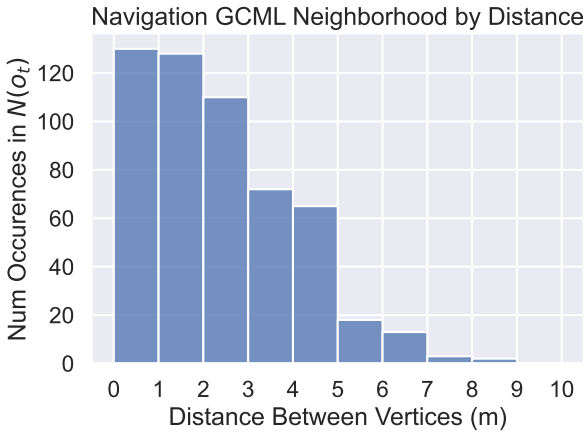
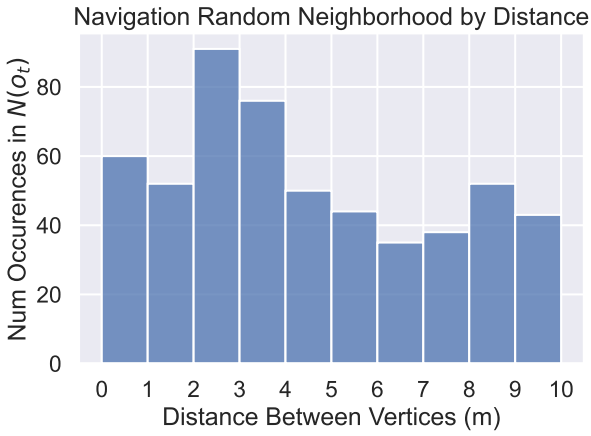


Figure 11: What do GCML navigation neighborhoods look like? At each episodic timestep, we record the neighborhood $N(o_t)$ GCML produces. We plot the accumulation of all neighborhoods over an episode, using $|z| = 32, K = 5$. Unlike the Fig. 10, vertices here are labeled in an absolute fashion (when they occurred). We see there are a few "key" observations (red arrows), such as the start vertex o_1 or o_{49}, similar to the use of keyframes in visual SLAM. Unlike stateless cartpole, information in the navigation task is spread over many observations. With $K = 5$, the maximum neighborhood size is 5 and the mean neighborhood size is 4.23, illustrating a flat distribution, and resulting in a denser graph.

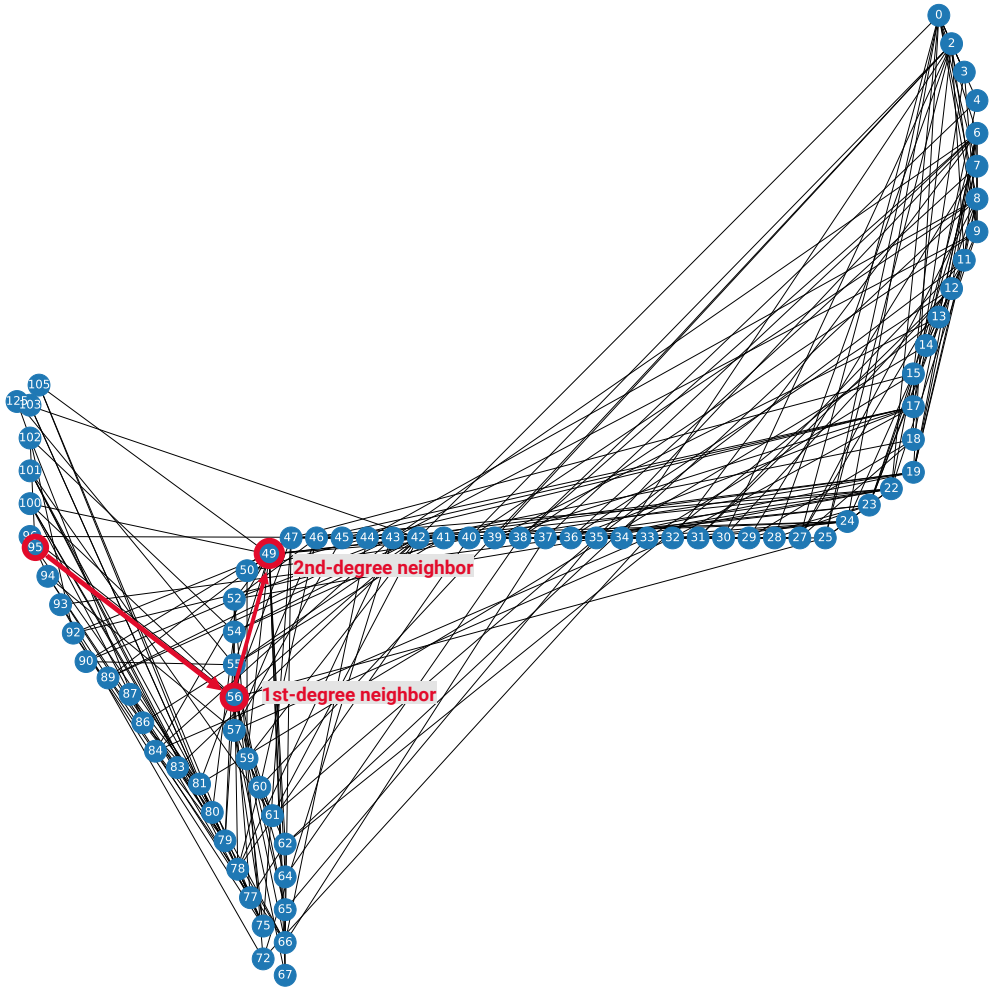


(a)

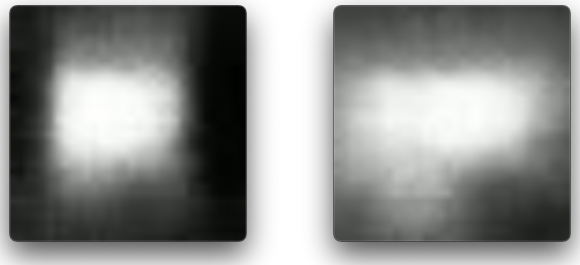


(b)

Figure 12: What do GCML navigation neighborhoods look like in the spatial domain? (a) At each episodic timestep, we record the neighborhood $N(o_t)$ GCML produces. We plot the accumulation of all neighborhoods over an episode, using $|z| = 32, K = 5$. We bin each vertex in the neighborhood $N(o_t)$ by its distance to the vertex o_t. We see GCML looks to be spatially-biased, heavily prioritizing nearby vertices over further vertices. (b) We shuffle the edge indices from a to see if the task itself is biased towards shorter distances (e.g. maybe agent spends lots of time in a single room 4m^2 in size, and all vertices are within 4m). We find this is not the case, and that the vertices span a large distance. a and b together prove that GCML indeed learns a spatial prior.



(a)



(b) (c)

Figure 13: (a) A visual representation of the memory graph from the episode used in Fig. 11 and Fig. 12. Vertices are labeled by timestep and projected into a 2D-plane based on their physical location. Overlapping vertices (e.g. when the agent rotates in place) are not shown. Here, we find clues for why o_{49} was so heavily favored in Fig. 11. We reconstruct the o_{49} depth image and find it was precisely when the agent left a narrow corridor (b) to enter a large living room (c). As it explores the living room ($o_{50} - o_{94}$), it localizes itself relative o_{49}, using past information to exit through a different door than it entered (o_{95}), maximizing exploration reward. We trace the belief at o_{95} through first-degree neighbor o_{56} to second-degree neighbor o_{49} in red. This introspection leads us to believe we should pay attention to doorways when designing topological priors for navigation.

B TOPOLOGICAL PRIOR EXAMPLES

To demonstrate how simple it is to write topological priors, we provide two examples of task specific priors in Pytorch. First, imagine a hospital agent where key observations occur when the medicines administered to the patient change. This could be implemented as such

```
import torch
class MedicalPrior(torch.nn.Module):
  def forward(self, V):
    # V is V_t (contains o_t)
    meds = get_meds_from_observation(V[:-1])
    N = torch.where(meds != meds.roll(1,0))
    return N
```

where `get_meds_from_observation` slices a tensor of observations to extract the medications administered at each timestep. Next, assume we are learning a dynamics model for an aircraft in flight. We want to exclude all past observations after a bird strike takes out an engine, and learn a new reduced-capability dynamics model

```
import torch
class AircraftPrior(torch.nn.Module):
  max_deviation = 8 # meters per second squared
  def forward(self, V):
    errors = get_dyn_model_error(V[:-1]) # Obs err using dyn model
    latest_damage_event = (errors > max_deviation).nonzero()[-1,0]
    N = torch.range(latest_damage_event, V.shape[0])
    return N
```

where `get_dyn_model_error` returns the acceleration disparity between a learned dynamics model and sensor measurements for each timestep.

C Environmental Details

C.1 Stateless Cartpole

We do not change any of the default environment settings from the OpenAI defaults: we use an episode length of 200, with a reward of 1 for each timestep survived. The episode ends when the pole angle is greater than 12 degrees or the cart position is further than 2.4m from the origin. Actions correspond to a constant force in either the left or right directions. We use the OpenAI gym definition (mean reward of 195) Brockman et al. (2016) to determine if the agent is successful. The value loss coefficient hyperparameter was used from the RLlib cartpole example, and thus differs from the non-cartpole default.

C.2 Concentration Game

The agent is given $n/2$ pairs of shuffled face-down cards, and must flip two cards face up. If the cards match, they remain face up, otherwise they are turned back over again. Once the player has matched all the cards, the game ends. We model the game of memory using a *pointer*, which the player moves to read and flip cards (Fig. 5b). The observation space consists of the pointer (card index and card value) and the last flipped (if any) face-up card. Cards are represented as one-hot vectors. The agent receives a reward for each pair it matches, with a cumulative reward of one for matching all the cards.

C.3 Navigation

The navigation experiment operated on the CVPR Habitat 2020 challenge (Savva et al., 2019) validation scene from the MP3D dataset (Chang et al., 2018). We used the same list of agent start coordinates as used during the challenge. 32×32 depth images with range $[0.5, 5\text{m}]$ and a 79 degree field of view were compressed into a 64-dimensional latent representations using a 6 layer (3 encoder, 3 decoder) convolutional β-VAE Kingma & Welling (2014) with $\beta = 0.01$, ReLU activation, and batch normalization. The β-VAE was pretrained until convergence using random actions. The full observations consists of agent heading and coordinates relative to the current start location, the previous action, and the latent VAE representation. The agent can rotate 30 degrees in either direction or move 0.25m forward. The agent receives a reward of 0.01 for exploring a new area of radius 0.20m.

D EXPERIMENT HYPERPARAMETERS

The following hyperparameters used for each experiment across all memory modules. We generally reduce the learning rate as well as increase the batch size to produce more consistent reward curves. Nearly all other hyperparameters are Ray RLlib defaults.

Term	Value	RLlib Default
Navigation IMPALA		
Decay factor γ	0.99	✓
Value function loss coef.	0.5	✓
Entropy loss coef.	0.001	0.01
Gradient clipping	40	✓
Learning rate	0.005	✓
Num. SGD iters	1	✓
Experience replay ratio	1:1	0:1
Batch size	1024	500
GAE λ	1.0	✓
V-trace ρ	1.0	✓
Cartpole PPO		
Decay factor γ	0.99	✓
Value function loss coef.	1e-5	✓(cartpole-specific)
Entropy loss coef.	0.0	✓
Value function clipping	10.0	✓
KL target	0.01	✓
KL coefficient	0.2	✓
PPO clipping	0.3	✓
Value clipping	0.3	✓
Learning rate	5e-5	✓
Num. SGD iters	30	✓
Batch size	4000	✓
Minibatch size	128	✓
GAE λ	1.0	✓
Concentration PPO		
Decay factor γ	0.99	✓
Value function loss coef.	1.0	✓
Entropy loss coef.	0.0	✓
Value function clipping	10.0	✓
KL target	0.01	✓
KL coefficient	0.2	✓
PPO clipping	0.3	✓
Value clipping	0.3	✓
Learning rate	3e-4	5e-5
Num. SGD iters	30	✓
Batch size	4000	✓
Minibatch size	4000	128
GAE λ	1.0	✓