ABSTRACT

Transformers have increasingly outperformed gated RNNs in obtaining new state-of-the-art results on supervised tasks involving text sequences. Inspired by this trend, we study the question of how Transformer-based models can improve the performance of sequential decision-making agents. We present the Working Memory Graph (WMG), an agent that employs multi-head self-attention to reason over a dynamic set of vectors representing observed and recurrent state. We evaluate WMG in two partially observable environments, one that requires complex reasoning over past observations, and another that features factored observations. We find that WMG significantly outperforms gated RNNs on these tasks, supporting the hypothesis that WMG’s inductive bias in favor of learning and leveraging factored representations can dramatically boost sample efficiency in environments featuring such structure.

1 INTRODUCTION

Because of their ability to process sequences of data, gated Recurrent Neural Networks (RNNs) have been widely applied to natural language processing (NLP) tasks such as machine translation. In the RNN-based approach of Sutskever et al. [2014], an encoder RNN maps an input sentence to a series of internal hidden state vectors. The encoder’s final hidden state is copied into a decoder RNN, which then generates another sequence of hidden states that determine the selection of output tokens in the target language. This model can be trained to translate sentences, but translation quality deteriorates on long sentences where long-term dependencies become critical. Reasoning that this drop in performance is due to the limited representational capacity of an RNN’s hidden state vector, Bahdanau et al. [2014] boosted translation quality by applying an attention mechanism to create paths serving as shortcuts from the input to the output sequences, routing information outside the linear chain of the RNN’s hidden states. Similar attention mechanisms have since gained wide usage, culminating in the Transformer model [Vaswani et al., 2017] which replaces the RNN with many short paths of self-attention. Since then, Transformers have outperformed RNNs on many NLP tasks (Devlin et al., 2018).

We seek to leverage these intuitions to improve the ability of Reinforcement Learning (RL) agents to reason over long time horizons in Partially Observable Markov Decision Processes (POMDPs) (Kaelbling et al., 1998). In a POMDP, a single observation \( \text{Obs}_t \) is not sufficient to identify the latent environment state \( s_t \). Thus the agent must reason over the history of past observations in order to select the best action for the current step. A simple strategy employed by DQN (Mnih et al., 2015) is to condition the policy on the \( N \) most recent observations \( \pi(a_t|\text{Obs}_{t-N}...\text{Obs}_t) \). But in complex environments, the sufficient number \( N \) may be large, highly variable, and unknown.

To address this issue, gated RNNs such as LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Chung et al., 2015) use internal, recurrent state vectors which in theory can maintain information from past observations (Hausknecht & Stone, 2015; Oh et al., 2016). However, in practice, these methods are limited by the single path of information flow defined by the linear chain of RNN hidden states. As in NLP, we hypothesize that providing alternative paths for information will be advantageous to RL agents. Building on this intuition, we introduce the Working Memory Graph (WMG), a Transformer-based agent that uses self-attention to provide a multitude of shortcut paths for information to flow between the past observations and the current action.

In addition to providing many paths for information, Transformers are also well suited for handling variably-sized inputs such as words in a sentence. Although most Reinforcement Learning environ-
Figure 1: **Left:** BabyAI rewards the agent (red triangle) for performing the task given by the instruction. **Right:** Our Transformer-based model (WMG) leverages factored observations and shortcut recurrence to solve this partially observable BabyAI task (PickupLoc) in far fewer environment interactions than a GRU-based agent using the same factored observations.

ments provide fixed-sized feature spaces, certain environments have observations spaces amenable to factorization. As a motivating example, consider the BabyAI environment (Chevalier-Boisvert et al., 2018) depicted in Figure 1 (left). The native observation space is the agent’s field of view, a 7x7 grid, shown in lighter grey. This observation can be efficiently represented by a set of factors describing the types, colors, and relative \(x\) and \(y\) coordinates of all visible objects:

\[
\text{([green, key, 1, 3], [grey, box, 2, 1], [green, ball, 2, 2], [red, key, 3, 0])}
\]

This factored observation is more compact than the native observation, but will vary in size depending on the number of objects in view. Motivated by prior work on factored representations (Russell & Norvig, 2009) and factored MDPs (Boutilier et al., 2000, 2001), we explore the idea of encoding factored observations as input to Transformer-based agents. In particular, we compare how factored observations affect the learning speed of Transformer and RNN-based agents.

**Our contributions are twofold:** First we introduce the Working Memory Graph (WMG), a Transformer-based agent implementing a novel form of shortcut recurrence which we demonstrate to be effective at complex reasoning over long-term dependencies. Second, we identify the synergy between Transformer-based RL architectures and factored observations, demonstrating that by virtue of its Transformer-style self-attention, WMG is able to effectively leverage factored observations to learn high-performing policies given an order of magnitude fewer environment interactions than alternative architectures. To preview our findings, Figure 1 (right) shows an example of the dramatic boosts in sample efficiency obtained through the combination of shortcut recurrence and Transformer-based processing of factored observations.

2 **Working Memory Graph**

Broadly, WMG incorporates an inductive bias in favor of learning and leveraging factored representations, including both observed and unobserved (latent) factors. Observed factors are represented by multiple input vectors called percepts. Latent factors are represented by multiple recurrent vectors called concepts. Instead of handling long-range dependencies over time by applying self-attention to a long history of observations, for which the quadratic computational cost could be prohibitively expensive, WMG relies on its much more limited set of concepts to represent long-range dependencies. The term Working Memory Graph is motivated by the relatively limited size of WMG’s self-attention computation graph, in loose analogy with the cognitive science term working memory, which refers to a cognitive system that holds a limited amount of information for use in mental processing. (Miller, 1956)

WMG introduces shortcut recurrence, which replaces a gated RNN’s single path of information flow with a network of shorter self-attention paths. As illustrated in Figure 2 (right), WMG’s shortcut
recurrence applies multi-head self-attention to a dynamic set of hidden state vectors, the aforementioned concepts, to simultaneously represent multiple latent factors or aspects of partially observable environments. Formally, each concept vector defines one row in a concept matrix $C \in \mathbb{R}^{n_C \times d_C}$, where $n_C$ is the number of concepts maintained by WMG and $d_C$ is the dimension of each concept vector. On each time step, the oldest concept is replaced by a new one.

WMG applies self-attention to observations by introducing multiple observation input vectors, percepts, as depicted in Figure 2 (right). In our experiments, a single percept encodes either an entire observation from a window of recent observations, or one factor (such as a green key in BabyAI) of a factored observation. On each time step, WMG receives a formatted observation consisting of a variable number of $(n_P)$ percept vectors forming a percept matrix $P \in \mathbb{R}^{n_P \times d_P}$, and a core vector $c \in \mathbb{R}^{d_c}$ that contains any other observation information (such as any non-factored portions of the current observation). The core, percept and concept vectors are stacked into one matrix for input to WMG’s Transformer operation:

$$T_{\text{in}} = \begin{bmatrix} c W_{\text{core}} + b_{\text{core}} \\ P W_{\text{per}} + b_{\text{per}} \\ C W_{\text{con}} + b_{\text{con}} \end{bmatrix}, \quad C' = \begin{bmatrix} C & I \end{bmatrix}$$

where $W_{\text{core}} \in \mathbb{R}^{d_c \times d_T}$, $W_{\text{per}} \in \mathbb{R}^{d_P \times d_T}$ and $W_{\text{con}} \in \mathbb{R}^{(d_C + n_C) \times d_T}$ are embedding matrices with corresponding bias vectors $b \in \mathbb{R}^{d_T}$ broadcast over rows, and each concept is concatenated with a one-hot age vector. Closely following the encoder architecture of Vaswani et al. (2017), WMG’s Transformer operation takes the input matrix $T_{\text{in}} \in \mathbb{R}^{n_T \times d_T}$ and returns an output matrix $T_{\text{out}} \in \mathbb{R}^{n_T \times d_T}$, where $n_T = 1 + n_P + n_C$ is the number of input (or output) nodes, and $d_T$ is the size of each node vector. The oldest concept is replaced by a new concept vector generated as a non-linear function of the core node’s output vector $h = T_{\text{out}}$.

$$C = \begin{bmatrix} \tanh(h W_C + b_C) \\ C_{-n_C} \end{bmatrix}, \quad W_C \in \mathbb{R}^{d_T \times d_C}, \quad b_C \in \mathbb{R}^{d_C}$$
Figure 3: **Shortcut Recurrence**: In a Gated RNN, for the first observation $O_{bs_1}$ to affect the agent’s output $h_8$, information must pass through 8 gating operations and 7 intervening hidden states $a$-$g$. In contrast, in a WMG, many possible paths lead from the first observation to the output $h_8$. The highlighted path requires only three passes through $W$, and is stored unchanged for several time steps in concept vectors $a$ and $d$. This example illustrates how WMG’s concept nodes provide shorter paths for information to flow forward and gradients to flow backward.

The trainable parameters $\theta$ of WMG and its Transformer layers are trained end-to-end through backpropagation of a policy-gradient loss maximizing the cumulative expected return $J(\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t|h_t; \theta) \cdot A^\pi(Obs_t, a_t) + \beta \nabla_\theta H(\pi(h_t; \theta)) \right]$$

where $\pi(a|h_t; \theta)$ denotes WMG’s policy head operating on hidden state $h_t$, $H$ is the entropy of the policy’s action distribution, and $\beta$ controls the strength of the entropy regularization term. To reduce the variance of gradient estimates, we use the advantage actor-critic algorithm described by Mnih et al. (2016), which estimates the advantage $A^\pi(s_t, a_t)$ using a $\gamma$-discounted $k$-step return as follows:

$$A^\pi(Obs_t, a_t) = \left( \sum_{i=0}^{k-1} \gamma^i r_{t+i} \right) + \gamma^k V^\pi(h_{t+k}) - V^\pi(h_t; \theta)$$

where $V(h_t; \theta)$ denotes WMG’s state-value head, which is trained to minimize the squared difference between the $k$-step return and the current value estimate: $||\left( \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V^\pi(h_{t+k}) \right) - V(h_t)||_2^2$, and $k$ is upper-bounded by the number of time steps ($t_{max}$) in the actor’s current update window.
To summarize WMG’s operation, Figure 3 compares the flow of information through a gated RNN and through WMG, illustrating how WMG’s concept vectors latch information unchanged for multiple time steps to create shorter paths for information flow in both the forward and backward passes.

3 RELATED APPROACHES

Having explained how WMG operates, we distinguish it from related work: Prior approaches have used attention for memory access (Graves et al., 2016; Oh et al., 2016) or self-attention to process individual observations (Zambaldi et al., 2019; Vinyals et al., 2019). These approaches all used LSTM-based recurrence over time. In contrast, WMG obviates the need for gated recurrence by applying self-attention to a network of concept vectors which are persisted through time.

Other Transformer-based models handle partial observability using state vectors analogous to WMG’s concepts, but with different state-update schedules: RMC (Santoro et al., 2018) updates all state vectors on every time step, while RIMs (Goyal et al., 2019) enforces sparsity by updating exactly half of the state vectors (called RIMs) on each step. WMG replaces only one concept on each time step in order to maximize the persistence of latched concept vectors and thereby extend the reach of the shortcut paths that they create from inputs to outputs. And unlike WMG, RMC and RIMs use gated RNNs to update their state vectors.

Unlike the other models discussed here, the Gated Transformer-XL (Anonymous, 2019) addresses partial observability by feeding hundreds of past observations at once into the Transformer. By contrast, in order to mitigate the $O(N^2)$ computational cost of self-attention, WMG computes self-attention over a comparatively small number of concepts which capture and maintain the relevant aspects of past observations.

4 EXPERIMENTS

In our experiments, we aim to (1) evaluate WMG’s ability to reason over long time spans in a setting of high partial observability, and (2) understand how factored representations may be effectively utilized by WMG. To address these questions we present results on two environments: a novel Pathfinding task which requires complex reasoning over past observations, and the BabyAI domain (Chevalier-Boisvert et al., 2018) which involves changing goals, partial observability, and observations that can be readily factored. To foreshadow our results, the Pathfinding task demonstrates the effectiveness of WMG’s shortcut recurrence, and BabyAI demonstrates that WMG leverages factored observations to deliver very large gains in sample efficiency.

4.1 PATHFINDING TASK

The Pathfinding task is designed to evaluate WMG’s ability to perform complex reasoning over past observations. Figure 4 depicts the incremental construction of a directed graph over nodes identified by unique pattern vectors which are randomly generated on every episode. (See Appendix A for the graph-construction algorithm and other details.) On odd time steps the agent observes two pattern nodes to be linked, and on even steps the agent must indicate whether or not a directed path exists from one given pattern to another. As this cycle repeats, the graph grows larger and the agent must perform an increasing number of reasoning steps to confirm or deny the existence of a path between arbitrary nodes. Because the observation only contains incremental information and the entirety of the graph is never directly observed, the agent must leverage information from previous observations to infer connectivity between nodes.

For example, consider step 4 of Figure 4. To determine whether a path exists from green to yellow, the agent must recall and combine information from steps 1 and 3. Similarly, on step 12, if the agent were asked about the existence of a path from cyan to yellow, answering correctly without guessing would require piecing together information from three non-contiguous time steps. Since the actual quiz on step 12 asks whether a path exists from green to blue, the agent must reason over many past observations to determine that no such path exists.

Each pattern is a vector of D real numbers drawn randomly from the interval -1 to 1. A binary value is added to the observation vector to indicate whether the current step is a quiz step, bringing
Figure 4: **Left:** **One episode of the Pathfinding task,** in this work, consists of 12 time steps with a maximum graph size $N = 7$ patterns. The boxes with rounded corners illustrate the observations for the given time steps, where a question mark identifies the step as a quiz step rather than a construction step. The box colors represent distinct pattern vectors. **Right:** **Results on Pathfinding:** Each plotted point is the percentage of reward on quiz steps received by the agent over the previous 10k time steps, averaged over 100 independent training runs. Bands display one standard deviation. (See Table 9 for more details.)

The size of the observation space is $2D + 1$, where $D = 7$ for our experiments. The action space consists of two actions, defined as yes or no. If the agent answers correctly on a quiz step, it receives a reward of 1; otherwise, it receives a reward of 0. The quiz questions are constructed to guarantee that each answer (yes or no) is correct half the time, so agents that act randomly or have no memory will obtain 50% of possible reward in expectation.

WMG is configured with concept nodes to handle the partial observability but no percept nodes, since we are not using this task to explore factored observations. The number of concept nodes is a tuned hyperparameter, equal to 16 in this experiment. (See Table B for all settings.) Each observation is passed directly to WMG’s core node, and WMG generates a new concept on each time step. We compare WMG’s performance to several baselines. Each Depth-$n$ baseline is a hand-coded algorithm demonstrating the performance obtained using perfect memory of past observations and reasoning over paths up to $n$ steps long. For example, Depth-2 remembers all previous construction steps, and reasons over all paths of depth 2. Finally, in order to understand the effectiveness of concept nodes at capturing past information, we evaluate a full-history, non-recurrent version of WMG by removing the concept nodes and giving it all past observations on each time step, each one passed to a separate percept node.

As shown in Figure 4, the GRU-based agent exceeds Depth-1 performance, but remains well short of Depth-2 performance after 20 million steps of training (environment interactions). In contrast, both versions of the WMG agent nearly reach Depth-3 performance, demonstrating a greater ability to perform complex reasoning over past observations. The best performance is achieved by the nr-WMG with full-history, which has no need for recurrence. But the full WMG (with concepts) is nearly as sample efficient as this perfect-memory baseline. These results indicate that shortcut recurrence enables WMG to learn to store and utilize essential information from past Pathfinding observations in a more effective manner than a GRU.

### 4.2 BabyAI Environment

In order to understand how factored representations may be effectively utilized by WMG, we study BabyAI, a domain whose observation space is amenable to factorization. BabyAI [Chevalier-Boisvert et al., 2018] is a partially observable, 2D grid-world containing objects that can be viewed and moved by the agent. Unlike most RL environments, BabyAI features text instructions that specify the goal the agent needs to achieve, such as “pick up the green box.”

We focus on five BabyAI levels, for which the environment consists of a single 6x6 room, as shown in Figure 5 (left). Despite the apparent simplicity of a single-room domain, learning to solve it
can often take model-free RL agents hundreds of thousands of environment interaction steps. The agent’s action space consists of 7 discrete actions: Move forward, Turn left, Turn Right, Pick up, Drop, Toggle, and Done. An episode ends after 64 time steps, or when the agent achieves the goal, for which it receives a reward of 1. In Level 1 (GoToObj), the room contains only one object. The agent completes the mission by moving to an adjacent square and pointing toward the object. In Level 2, the target object is always a red ball, and seven grey boxes are present as distractors. In Level 3, the distractors may be any of the 3 object types and 6 colors. If one of the distractors happens to be a red ball, the agent is rewarded for reaching it. In Level 4, the instruction specifies the color and type of the target object. This is the first level in which the text instruction contains valuable information. (See Table 12 for instruction templates.) Level 5 increases the difficulty of Level 4 in two ways. First, the agent must not only reach the target object, but must also pick it up. Second, if multiple qualifying target objects are present, the agent is given the initial relative location of the true target, such as “behind you”. 

![Factored Observation](image)

**Instruction:** 
Go to the yellow box

| Part of observation | Variable assignments | Node |
|---------------------|----------------------|------|
| Factored image      | color=green, type=key, X=3, Y=1 | percept |
| Factored image      | color=grey, type=box, X=1, Y=2 | percept |
| Factored image      | color=green, type=ball, X=2, Y=2 | percept |
| Factored image      | color=red, type=key, X=0, Y=3 | percept |
| Factored image      | vertical wall X=-2 | core |
| Factored image      | horizontal wall Y=4 | core |
| Factored instruction| command=go to, article=the, color=yellow, type=box, loc=None | core |
| Additional info     | orientation=west, last action=move forward | core |

Figure 5: One completely factored observation, where each variable assignment corresponds to a one-hot vector in the full observation vector. Since the number of objects in an observation can vary, each object’s vectors are concatenated then passed to a percept node. All other one-hot vectors from the observation are concatenated then passed to the core node. X & Y coordinates refer to a frame of reference with the agent at the origin, pointed in the positive Y direction. The agent always observes one vertical wall and one horizontal wall.

Each agent observation in BabyAI consists of a text instruction, an image, and the agent’s orientation. The image’s native format is a 7x7 array of cell descriptors (not pixels) identifying three attributes of each cell: type, color, and open/closed/locked (referring to doors, which are not found in these 5 levels). To study factored observations in BabyAI, we define a factored representation, depicted in Figure 5. In our experiments the text instruction is always factored, but the image is formatted in multiple ways: (1) 7x7x3, the native BabyAI image array; (2) flat, the native 7x7x3 array flattened to one vector; (3) factored image, as described in Figure 5. (Note that when a factored image is passed to a GRU, it must first be flattened and padded to form a fixed-length vector.)

To determine whether WMG can leverage factored observations more effectively than gated RNNs in BabyAI, we evaluate the following agents: (1) WMG is the full, recurrent WMG model, with percepts mapped to observation factors, (2) nr-WMG is an ablated, non-recurrent version of WMG with no concepts, (3) GRU is a GRU model, and (4) CNN+GRU uses a CNN to process the native 7x7x3 image, followed by a GRU. This CNN is one of the two CNN models provided in the BabyAI open source code (Chevalier-Boisvert et al., 2018).

4.2.1 RESULTS

**Factored Observations:** The largest performance differences in Table 1 stem from the choice of factored versus flat or native image formats. Notably, WMG with factored images can achieve sample efficiencies 10x greater (on Level 3) than CNN+GRU using the native 7x7 image format. However, factored observations alone are not sufficient for sample efficiency: WMG utilizes factored images much more effectively than a GRU on Levels 2-5. This result supports our hypothesis that Transformer-based models are particularly well suited for operating on set-based inputs like factored observations, and large gains in sample efficiency are observed as a result.
Table 1: **BabyAI sample efficiency**: the amount of training (shown here in thousands of environment interactions) required for a model to solve 99% of 10,000 episodes. Hyperparameters were first tuned on each model/format/level combination separately, then each reported result was computed as the median sample efficiency over 100 additional training runs. Dashes indicate that no model reliably reached a solution rate of 99% within 6 million training steps (environment interactions). Note that Chevalier-Boisvert et al. (2018) report sample efficiencies in terms of episodes rather than environment interactions. (See Table 12.)

| Model | Image format | WMG | nr-WMG | GRU | WMG | GRU | CNN+GRU |
|-------|--------------|-----|--------|-----|-----|-----|---------|
|       |               | factored |        | factored | flat | flat | native 7x7x3 |
| 1 - GoToObj | factored | 1.6 | 1.4 | 1.7 | 15.0 | 19.0 | 10.6 |
| 2 - GoToRedBallGrey | 6.7 | 5.2 | 24.6 | 29.0 | 31.0 | 22.3 |
| 3 - GoToRedBall | 16.0 | 23.6 | 174.4 | 92.0 | 124.6 | 204.9 |
| 4 - GoToLocal | 59.7 | 71.3 | 2,241.6 | 1,379.9 | 1,799.4 | —— |
| 5 - PickupLoc | 222.3 | 253.0 | —— | —— | —— | —— |

**Concept Nodes**: Without factored observations, WMG-flat slightly outperforms GRU-flat, suggesting that shortcut recurrence implemented by the WMG’s concept nodes compares favorably to the GRU’s gated recurrence. With the benefit of factored observations, the non-recurrent ablation of WMG (nr-WMG) performs slightly better than the full WMG on the simplest two levels. But for the more challenging levels 3-5, WMG’s concept vectors prove to be of benefit for WMG with factored observations.

**Early vs Late instruction fusion**: Interestingly, within our training limit of 6 million environment interactions, CNN+GRU is unable to learn to solve the levels (4 & 5) where instructions carry important information. We suspect this is because the CNN processes just the image while the factored instruction is passed directly to the GRU, skipping the CNN. By contrast, the baseline BabyAI agent uses FiLM layers to integrate the processing of the image with the text instruction. Both WMG and GRU models can process the image and instruction together in all levels of processing. This early fusion appears to allow all WMG and GRU models to solve Level 4.

In summary, the two WMG models with factored images were the only agents able to solve Level 5, and they learned to do so in approximately the same number of interactions that CNN-GRU required to solve Level 3. These drastic differences in sample efficiency serve to highlight the potential gains that can be achieved by RL agents equipped to utilize factored observations.

While WMG’s sample efficiencies dramatically exceed the RL benchmarks published with the BabyAI domain (Chevalier-Boisvert et al., 2018), often by two orders of magnitude (Table 12), it’s important to note that these sets of results are not directly comparable. Our experiments all used factored text instructions, and each model’s hyperparameters were tuned for each level separately, while the BabyAI benchmark agent was trained on all levels using the single hyperparameter configuration provided in the BabyAI release. Because of these differences, our experiments should not be interpreted as a new state-of-the-art on the standard BabyAI tasks.

### 4.2.2 Hyperparameter Sensitivity

To evaluate WMG’s sensitivity to hyperparameter selection, we applied the tuned hyperparameter settings from Level 4 to new training runs on all other levels. Figure 6 shows moderate degradations in performance for all models. In particular, when the hyperparameter values tuned on Level 4 are used in Level 5 training runs, none of the models reach a 99% solution rate within 1 million training steps, but WMG with factored observations reaches higher levels of performance than the other models. Broadly, these results indicate that WMG is no more sensitive to hyperparameter settings than the baseline agents.

### 5 Conclusion and Future Work

We designed the Working Memory Graph to investigate how Transformer-based models can improve the performance of RL agents. In order to effectively leverage factored observations, WMG applies
Transformer-style self-attention to arbitrary numbers of percept vectors mapped directly to observed factors. And in order to represent multiple latent aspects of partially observable environments, without incurring large quadratic computational costs of self-attention over long histories, WMG incorporates a form of recurrence that creates shortcut paths of self-attention over a dynamic set of hidden states, called concepts.

We compared WMG’s performance to that of gated RNNs in two partially observable environments, one focused on complex reasoning over long-term dependencies, and one focused on reasoning over factored observations. In these experiments, WMG outperforms gated RNNs by wide margins. In particular, our results demonstrate that when factored observations are available, sample efficiency can be dramatically boosted by passing the factors separately to WMG percepts, instead of entangling the factors through concatenation into fixed-length vectors for processing by a gated RNN.

To clarify certain limitations of this version of WMG, we outline three potential enhancements:

**Flexible concept lifetimes:** In the work reported here, each new concept automatically replaces the oldest. A more flexible and adaptive concept-deletion scheme may improve WMG’s ability to model latent aspects in the environment. For instance, concept vectors that receive more attention than others may be the ones most worth keeping around for longer. Deleting a concept only when its recently-received attention falls below a certain threshold would allow the number of concept vectors to fluctuate somewhat over time, depending on the needs of the situation.

**Graph edge content:** As in the original Transformer, WMG applies input vectors to the nodes in its computation graph, but not to the edges between them. To better represent graph-structured data, [Veličković et al., 2017] contemplated incorporating edge-specific data into Graph Attention Networks as future work. By harnessing the richer representational abilities of graph structures over set structures, a similar extension of WMG may allow it to better model complex relations among observed and latent factors in the environment.

**Memory vectors:** Various forms of external memory have been proposed in recent years. [Graves et al., 2016; Munkhdalai et al., 2019] Memory vectors retrieved from such stores could be fed to dedicated WMG memory nodes, in addition to the current concept and percept nodes, to further extend the range and flexibility of an agent’s effective time horizon.

**Acknowledgments**

The authors wish to thank Alekh Agarwal and Xiaodong Liu for many valuable discussions.
REFERENCES

Anonymous. Stabilizing transformers for reinforcement learning, 2019. URL https://openreview.net/forum?id=SyxKrySYFr Under review, International Conference on Learning Representations, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by joint learning to align and translate, 2014. URL http://arxiv.org/abs/1409.0473 Accepted at ICLR 2015 as oral presentation.

Craig Boutilier, Raymond Reiter, Mikhail Soutchanski, and Sebastian Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 355–362. AAAI Press, 2000. ISBN 0-262-51112-6. URL http://dl.acm.org/citation.cfm?id=647288.721273.

Craig Boutilier, Ray Reiter, and Bob Price. Symbolic dynamic programming for first-order mdps. In Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI’01, pp. 690–697, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-812-5, 978-1-55860-812-2. URL http://dl.acm.org/citation.cfm?id=1642090.1642184.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. CoRR, abs/1810.08272, 2018. URL http://arxiv.org/abs/1810.08272.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS’15, pp. 2980–2988, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969442.2969572.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. CoRR, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Y. Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. CoRR, abs/1909.10893, 09 2019. URL http://arxiv.org/abs/1909.10893.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badía, 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, October 2016. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature20101.

Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. CoRR, abs/1507.06527, 2015. URL http://arxiv.org/abs/1507.06527.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Artif. Intell., 101(1-2):99–134, May 1998. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X. URL http://dx.doi.org/10.1016/S0004-3702(98)00023-X.
Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980  3rd International Conference for Learning Representations, San Diego, 2015.

George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. The Psychological Review, 63(2):81–97, March 1956. URL http://www.musanim.com/miller1956/

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, February 2015. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature14236.

Volodymyr Mnih, Adrià Puigdomènec Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML), pp. 1928–1937, 2016.

Tsendsuren Munkhdalai, Alessandro Sordoni, Tong Wang, and Adam Trischler. Metalearned neural memory. CoRR, abs/1907.09720, 2019. URL http://arxiv.org/abs/1907.09720.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, pp. 2790–2799. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045684.

Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594.

Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. CoRR, abs/1806.01822, 2018. URL http://arxiv.org/abs/1806.01822.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS’14, pp. 3104–3112, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969033.2969173.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. CoRR, abs/1710.10903, 2017. URL http://arxiv.org/abs/1710.10903.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, pp. 1–5, 2019.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Deep reinforcement learning with relational inductive biases, 2019.
Appendices

A  PATHFINDING ENVIRONMENT DETAILS

A.1  GRAPH CONSTRUCTION ALGORITHM

The Pathfinding graph is constrained to be a polytree (singly-connected, directed acyclic graph) at each step of the episode, up to a maximum size of $N$ patterns, each of size $D$, connected by $N - 1$ links. At the start of each episode, the graph contains a single random pattern. On each construction step, the environment links one new pattern to the graph through the following procedure (drawing all random numbers from uniform distributions):

1. Create a new random pattern $a_i \in (-1, +1)^D$.
2. Randomly choose one existing pattern in the graph.
3. Create a new link, choosing a random direction, between the two patterns.

A.2  QUIZ-GENERATION ALGORITHM

On each quiz step, the environment first decides whether the correct answer should be 0 or 1 by sampling a binary value from a discrete uniform distribution. Then the environment draws uniform-random ordered pairs of nodes from the current graph until finding a pair that satisfies the desired answer. The observation is then constructed by concatenating the two pattern vectors and a value of 1 to mark this observation as a quiz.

A.3  Depth-n BASELINE ALGORITHM

The hand-coded baseline agent is configured with a depth parameter $n$. As new pattern pairs are revealed on graph-construction time steps, the agent maintains a growing vector of all patterns seen, and a growing matrix of directed path lengths from every observed pattern to every other. A path length of zero indicates that no path exists from the first pattern to the second. On each quiz step, the agent looks up from the matrix the path length $\text{len}$ for the ordered pair of patterns in the observation. If $0 < \text{len} \leq n$, the agent chooses the yes action. Otherwise, the agent chooses the no action.

B  TRAINING DETAILS AND HYPERPARAMETERS

Table 2: Fixed settings and options used for all experiments, except for the replicated baselines in Table 12.

| Settings and options                          | Values                      |
|----------------------------------------------|-----------------------------|
| Dropout                                      | None                        |
| Learning rate schedule                       | Constant learning rate      |
| Non-linearities                              | ReLU, tanh                  |
| Parallel training workers                    | 1                           |
| Optimizer                                    | Adam [Kingma & Ba, 2014]    |
| Parameter initialization, biases             | 0                           |
| Parameter initialization, non-bias weights    | Kaiming uniform [He et al., 2015] |
| Reward shaping                               | None                        |
| Training algorithm                           | Advantage actor-critic [Mnih et al., 2016] |
| Weight decay regularization                  | None                        |
Table 3: Tuned hyperparameter settings for Pathfinding experiments.

|                          | WMG | nr-WMG | GRU |
|--------------------------|-----|--------|-----|
| Actor-critic hidden layer size | 128 | 128 | 512 |
| Actor-critic $t_{max}$    | 16  | 16    | 16  |
| Adam eps                 | 1e-06 | 1e-08 | 1e-08 |
| Discount factor $\gamma$ | 0.5 | 0.6   | 0.5 |
| Entropy term strength $\beta$ | 0.01 | 0.005 | 0.02 |
| Gradient clipping threshold | 16.0 | 16.0 | 4.0 |
| GRU observation embedding size | | | 256 |
| GRU size                 | 384 |
| Learning rate            | 0.00016 | 0.00016 | 0.0001 |
| Reward scale factor      | 2.0 | 1.0 | 0.5 |
| WMG attention head size  | 12  | 16 |
| WMG attention heads      | 6   | 6 |
| WMG concept nodes        | 16  | 0  |
| WMG concept size         | 128 |
| WMG hidden layer size    | 12  | 32 |
| WMG layers               | 4   | 4 |

Table 4: Tuned hyperparameter settings for BabyAI Level 1 - GoToObj.

|                          | WMG factored | nr-WMG factored | GRU factored | GRU flat | CNN+GRU native 7x7x3 |
|--------------------------|--------------|-----------------|--------------|----------|----------------------|
| Actor-critic hidden layer size | 2048 | 4096 | 4096 | 4096 | 2048 |
| Actor-critic $t_{max}$    | 1 | 1 | 6 | 16 | 4 |
| Adam eps                 | 0.0001 | 1e-08 | 1e-08 | 1e-10 | 0.0001 |
| CNN hidden channel size 1 | 16 |
| CNN hidden channel size 2 | 40 |
| CNN hidden channel size 3 | 192 |
| Discount factor $\gamma$ | 0.98 | 0.9 | 0.7 | 0.6 | 0.9 | 0.8 |
| Entropy term strength $\beta$ | 0.002 | 0.05 | 0.01 | 0.005 | 0.02 | 0.02 |
| Gradient clipping threshold | 256.0 | 1024.0 | 512.0 | 512.0 | 128.0 |
| GRU observation embed size | 1024 | 512 | 512 |
| GRU size                 | 96 | 512 | 96 |
| Learning rate            | 0.0001 | 4e-05 | 0.0004 | 0.0001 | 0.0001 | 0.0004 |
| Reward scale factor      | 4.0 | 32.0 | 32.0 | 8.0 | 32.0 | 8.0 |
| WMG attention head size  | 24 | 16 | 16 |
| WMG attention heads      | 4 | 10 | 12 |
| WMG concept nodes        | 1 | 0 | 1 |
| WMG concept size         | 64 | | 256 |
| WMG hidden layer size    | 64 | 64 | 32 |
| WMG layers               | 4 | 4 | 1 |
| Hyperparameter | WMG factored | nr-WMG factored | GRU factored | WMG flat | GRU flat | CNN+GRU native 7x7x3 |
|----------------|--------------|----------------|-------------|----------|----------|----------------------|
| Actor-critic hidden layer size | 4096 | 2048 | 4096 | 4096 | 4096 | 64 |
| Adam $\epsilon_{max}$ | 8 | 6 | 16 | 1 | 1 | 1 |
| CNN hidden channel size 1 | 1e-06 | 1e-08 | 1e-10 | 1e-10 | 1e-06 | 0.0001 |
| CNN hidden channel size 2 | 0.8 | 0.9 | 0.8 | 0.9 | 0.9 | 0.95 |
| CNN hidden channel size 3 | 0.01 | 0.02 | 0.01 | 0.005 | 0.005 | 0.02 |
| Discount factor $\gamma$ | 1024.0 | 512.0 | 128.0 | 64.0 | 64.0 | 12 |
| GRU observation embed size | 4096 | 2048 | 4096 | 2048 | 256 | 24 |
| GRU size | 96 | 512 | 96 | 512 | 64 | 192 |
| Learning rate | 0.0001 | 0.00025 | 0.0001 | 2.5e-05 | 2.5e-05 | 0.0004 |
| Reward scale factor | 8.0 | 4.0 | 4.0 | 4.0 | 4.0 | 2.0 |
| WMG attention head size | 64 | 48 | 64 | 48 | 64 | 12 |
| WMG attention heads | 4 | 1 | 3 | 1 | 3 | 3 |
| WMG concept nodes | 1 | 0 | 8 | 0 | 8 | 8 |
| WMG concept size | 32 | 64 | 32 | 64 | 64 | 16 |
| WMG hidden layer size | 16 | 24 | 384 | 24 | 384 | 128 |
| WMG layers | 3 | 3 | 1 | 3 | 1 | 1 |

Table 6: Tuned hyperparameter settings for BabyAI Level 3 - GoToRedBall.

| Hyperparameter | WMG factored | nr-WMG factored | GRU factored | WMG flat | GRU flat | CNN+GRU native 7x7x3 |
|----------------|--------------|----------------|-------------|----------|----------|----------------------|
| Actor-critic hidden layer size | 4096 | 2048 | 4096 | 4096 | 4096 | 4096 |
| Adam $\epsilon_{max}$ | 1 | 2 | 3 | 1 | 2 | 3 |
| CNN hidden channel size 1 | 1e-12 | 0.0001 | 1e-06 | 0.0001 | 1e-06 | 0.01 |
| CNN hidden channel size 2 | 0.95 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 |
| CNN hidden channel size 3 | 0.05 | 0.9 | 0.05 | 0.9 | 0.02 | 0.02 |
| Discount factor $\gamma$ | 128.0 | 128.0 | 128.0 | 128.0 | 32.0 | 32.0 |
| GRU observation embed size | 4096 | 2048 | 2048 | 4096 | 256 | 256 |
| GRU size | 192 | 512 | 192 | 512 | 64 | 192 |
| Learning rate | 2.5e-05 | 6.3e-05 | 6.3e-05 | 2.5e-05 | 2.5e-05 | 0.0004 |
| Reward scale factor | 8.0 | 4.0 | 8.0 | 4.0 | 4.0 | 4.0 |
| WMG attention head size | 128 | 32 | 128 | 32 | 128 | 32 |
| WMG attention heads | 2 | 8 | 12 | 8 | 12 | 12 |
| WMG concept nodes | 2 | 0 | 16 | 0 | 16 | 16 |
| WMG concept size | 128 | 256 | 128 | 256 | 256 | 128 |
| WMG hidden layer size | 64 | 32 | 128 | 32 | 128 | 128 |
| WMG layers | 4 | 4 | 1 | 4 | 1 | 1 |
Table 7: Tuned hyperparameter settings for BabyAI Level 4 - GoToLocal.

| WMG | nr-WMG | GRU | WMG | GRU |
|-----|--------|-----|-----|-----|
| factored | factored | GRU | flat | flat |
| Actor-critic hidden layer size | 2048 | 2048 | 1024 | 512 | 4096 |
| Actor-critic $t_{max}$ | 6 | 3 | 3 | 6 | 4 |
| Adam eps | 1e-12 | 0.01 | 1e-06 | 1e-08 | 1e-12 |
| Discount factor $\gamma$ | 0.5 | 0.6 | 0.95 | 0.5 | 0.9 |
| Entropy term strength $\beta$ | 0.1 | 0.1 | 0.1 | 0.02 | 0.02 |
| Gradient clipping threshold | 512.0 | 512.0 | 256.0 | 256.0 | 512.0 |
| GRU observation embed size | | | 1024 | 512 | |
| GRU size | | | 128 | 96 | |
| Learning rate | 6.3e-05 | 0.0001 | 4e-05 | 2.5e-05 | 4e-05 |
| Reward scale factor | 32.0 | 16.0 | 8.0 | 16.0 | 2.0 |
| WMG attention head size | 128 | 64 | 24 |
| WMG attention heads | 2 | 4 | 16 |
| WMG concept nodes | 8 | 0 | 16 |
| WMG concept size | 32 | | 64 |
| WMG hidden layer size | 32 | 48 | 16 |
| WMG layers | 4 | 3 | 2 |

Table 8: Tuned hyperparameter settings for BabyAI Level 5 - PickupLoc.

| WMG | nr-WMG |
|-----|--------|
| factored | factored |
| Actor-critic hidden layer size | 512 | 2048 |
| Actor-critic $t_{max}$ | 12 | 12 |
| Adam eps | 1e-10 | 1e-10 |
| Discount factor $\gamma$ | 0.7 | 0.8 |
| Entropy term strength $\beta$ | 0.02 | 0.05 |
| Gradient clipping threshold | 512.0 | 512.0 |
| Learning rate | 0.0001 | 6.3e-05 |
| Reward scale factor | 8.0 | 8.0 |
| WMG attention head size | 24 | 48 |
| WMG attention heads | 10 | 6 |
| WMG concept nodes | 8 | 0 |
| WMG concept size | 32 | |
| WMG hidden layer size | 128 | 96 |
| WMG layers | 2 | 2 |
### C ADDITIONAL EXPERIMENTAL RESULTS

Table 9: Additional details for the Pathfinding experimental results in Figure 4 (right).

| Models & algorithms | Final performance | Trainable parameters | Training speed |
|---------------------|-------------------|----------------------|----------------|
| Depth-(n-1) baseline | 100.0% of reward |                      |                |
| Depth-3 baseline    | 99.7% of reward   |                      |                |
| Depth-2 baseline    | 97.6% of reward   |                      |                |
| Depth-1 baseline    | 86.9% of reward   |                      |                |
| nr-WMG, full-history| 99.6% of reward   | 204,963              | 96 steps/sec   |
| WMG                 | 99.6% of reward   | 132,507              | 91 steps/sec   |
| GRU                 | 94.7% of reward   | 1,139,459            | 291 steps/sec  |

Table 10: Number of trainable parameters, in thousands, for the BabyAI models in Table 1.

| BabyAI level          | WMG factored | nr-WMG factored | GRU factored | WMG flat | GRU flat | CNN+GRU native 7x7x3 |
|-----------------------|--------------|-----------------|--------------|----------|----------|----------------------|
| 1 - GoToObj           | 636          | 1,864           | 1,572        | 2,053    | 4,170    | 393                  |
| 2 - GoToRedBallGrey   | 2,997        | 258             | 3,723        | 2,116    | 10,075   | 140                  |
| 3 - GoToRedBall       | 3,418        | 2,217           | 3,749        | 3,229    | 15,126   | 709                  |
| 4 - GoToLocal         | 2,235        | 1,960           | 1,137        | 2,022    | 1,479    | ——                   |
| 5 - PickupLoc         | 879          | 2,007           | ——           | ——       | ——       | ——                   |

Table 11: Training steps per second on a fixed machine, for the BabyAI models in Table 1.

| BabyAI level          | WMG factored | nr-WMG factored | GRU factored | WMG flat | GRU flat | CNN+GRU native 7x7x3 |
|-----------------------|--------------|-----------------|--------------|----------|----------|----------------------|
| 1 - GoToObj           | 38           | 28              | 146          | 111      | 86       | 149                  |
| 2 - GoToRedBallGrey   | 58           | 113             | 147          | 35       | 18       | 88                   |
| 3 - GoToRedBall       | 18           | 32              | 78           | 25       | 20       | 87                   |
| 4 - GoToLocal         | 44           | 48              | 132          | 54       | 134      | ——                   |
| 5 - PickupLoc         | 81           | 84              | ——           | ——       | ——       | ——                   |
Table 12: BabyAI baseline agent sample efficiencies, defined as the amount of training (in either episodes or environment interaction steps) required for the agent to solve 99% of random episodes within 64 steps. The published results are the means of the min & max RL sample efficiencies reported in Table 3 of [Chevalier-Boisvert et al., 2018]. The replicated results are the medians over 10 training runs, using the code and default hyperparameter settings from the open source release of the BabyAI baseline agent. All numbers are in thousands.

| BabyAI level | Instruction template | Published (episodes) | Replicated (episodes) | Replicated (interactions) |
|--------------|----------------------|----------------------|-----------------------|--------------------------|
| 1 - GoToObj  | GO TO color object   | ——                   | 19                    | 333                      |
| 2 - GoToRedBallGrey | GO TO RED BALL     | 16                   | 16                    | 282                      |
| 3 - GoToRedBall | GO TO RED BALL      | 272                  | 283                   | 3,674                    |
| 4 - GoToLocal | GO TO color object  | 971                  | 1,064                 | 16,422                   |
| 5 - PickupLoc | PICK UP color object loc | 2,977             | 1,557                 | 25,574                   |