EPISODIC REINFORCEMENT LEARNING WITH ASSOCIATIVE MEMORY

Guangxiang Zhu\textsuperscript{1}; Zichuan Lin\textsuperscript{2}; Guangwen Yang\textsuperscript{2}, Chongjie Zhang\textsuperscript{1}
\textsuperscript{1}Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
\textsuperscript{2}Department of Computer Science and Technology, Tsinghua University, Beijing, China
guangxiangzhu@outlook.com, linzcl6@mails.tsinghua.edu.cn, ygw@tsinghua.edu.cn, chongjie@tsinghua.edu.cn

Abstract

Sample efficiency has been one of the major challenges for deep reinforcement learning. Non-parametric episodic control has been proposed to speed up parametric reinforcement learning by rapidly latching on previously successful policies. However, previous work on episodic reinforcement learning neglects the relationship between states and only stored the experiences as unrelated items. To improve sample efficiency of reinforcement learning, we propose a novel framework, called Episodic Reinforcement Learning with Associative Memory (ER-LAM), which associates related experience trajectories to enable reasoning effective strategies. We build a graph on top of states in memory based on state transitions and develop a reverse-trajectory propagation strategy to allow rapid value propagation through the graph. We use the non-parametric associative memory as early guidance for a parametric reinforcement learning model. Results on navigation domain and Atari games show our framework achieves significantly higher sample efficiency than state-of-the-art episodic reinforcement learning models.

1 Introduction

Deep reinforcement learning (RL) has achieved remarkable performance on extensive complex domains (Mnih et al., 2015; Lillicrap et al., 2016; Silver et al., 2016; Schulman et al., 2017). Deep RL research largely focuses on parametric method, which usually depends on a parametrized value function. The parametric approaches are quite sample inefficient and requires several orders of magnitude more training samples than a human. This is because gradient-based updates are incremental and slow, and has global impacts on parameters, which leads to catastrophic inference issue.

Recently, episodic reinforcement learning has attracted much attention for improving sample efficiency of deep reinforcement learning, such as model-free episodic control (MFEC) (Blundell et al., 2016), neural episodic control (NEC) (Pritzel et al., 2017), ephemeral value adjustments (EVA) (Hansen et al., 2018), and episodic memory deep q-networks (EMDQN) (Lin et al., 2018). Episodic control is inspired by the psychobiological and cognitive studies of human memory (Sutherland & Rudy, 1989; Marr et al., 1991; Lengyel & Dayan, 2008; Botvinick et al., 2019) and follows the idea of instance-based decision theory (Gilboa & Schmeidler, 1995). It builds a non-parametric episodic memory to store past good experiences, and thus can rapidly latch onto successful policies when encountering with states similar to past experiences.

However, most of the existing breakthroughs have been focusing on episodic memory and leave the association of memory largely unstudied. Previous work usually uses a tabular-like memory and experiences are stored as unrelated items. Studies in psychology and cognitive neuroscience (Kohonen, 2012; Anderson & Bower, 2014) discover that associative memory found in hippocampus plays an important role in human activities, which associates past experiences by remembering relationship between them. Inspired by this, we propose a novel associative memory based reinforcement learning framework to improve the sample-efficiency of reinforcement learning, called Episodic Reinforcement Learning with Associative Memory (ER-LAM), which associates related experience trajectories to enable reasoning effective strategies. We store the best historical values for
memorized states like episodic memory, and maintain a graph on top of these states based on state transitions at the same time. Then we develop an efficient reverse-trajectory propagation strategy to allow the values of new experiences to rapidly propagate to all memory items through the graph. Finally, we use the fast-adjusted non-parametric high values in associative memory as early guidance for a parametric RL agent so that it can rapidly latch on states that previously yield high returns instead of waiting for many slow gradient updates.

To illustrate the superiority of the associative memory in reinforcement learning, consider a robot exploring in a maze to seek out the apple (at place G), as shown in Figure 1. It collects two trajectory experiences starting from place A and B, respectively. All the states of trajectory A (the top blue dash line) receive no reward because the agent terminates at a state with a non-zero reward (at place C), while in trajectory B (the bottom blue dash line) the final non-zero reward of catching an apple (at place G) back-propagates through the whole path. Episodic memory keeps a high value at the intersection of two trajectories (the door) when taking actions toward lower-right corner while recording zero values at the other states in trajectory A. If an episodic memory based robot starts from place A, it will wander around A because there are no positive values indicating the way to goal. Thus based on the episodic memory, the robot may eventually take a policy like the green line after multiple attempts. However, if the robot adopts associative memory, the high value in the door collected from trajectory B will be further propagated to the start point A and thus the robot can correctly take the red-line policy.

To some extent, our associative memory is equivalent to automatic augmentation of counterfactual combinatorial trajectories in memory. Thus, our framework significantly improves the sample-efficiency of reinforcement learning. Comparisons with state-of-the-art episodic reinforcement learning methods show that ERLAM is substantially more sample efficient for general settings of reinforcement learning. In addition, our associative memory can be used as a plug-and-play module and is complementary to other reinforcement learning models, which opens the avenue for further researches on associative memory based reinforcement learning.

2 BACKGROUND

In the framework of reinforcement learning [Sutton & Barto, 1998], an agent learns a policy to maximize its cumulative rewards by exploring in a Markov Decision Processes (MDP) environment. An MDP is defined by a tuple \( (S, A, P, R, \gamma) \), where \( S \) is a finite set of states, \( A \) is a finite set of actions available to the agent, \( P : S \times A \times S \to \mathbb{R} \) defines the transition probability distribution, \( R \) is the reward function, and \( \gamma \in (0, 1) \) is the discount factor. At each time step \( t \), the agent observes state \( s_t \in S \), selects an action \( a_t \in A \) according to its policy \( \pi : S \to A \), and receives a scalar reward \( r_t \). In the setting of finite horizon, the accumulated discounted return is calculated as,

\[
R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}
\]

where \( T \) is the episode length and goal of the agent is to maximize the expected return for each state \( s_t \).

The state-action value function \( Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a] \) is the expected return for executing action \( a \) on state \( s \) and following policy \( \pi \) afterwards. DQN [Mnih et al., 2015] parameterizes this action-value function by deep neural networks \( Q_\theta(s, a) \) and use Q-learning [Watkins & Dayan, 1992] to learn it to rank which action at is best to take in each state \( s_t \) at time step \( t \). The parameters of the value network \( \theta \) are optimized by minimizing the \( L_2 \) difference between the networks output \( Q_\theta(s, a) \) and the Q-learning target \( y_t = r_t + \gamma \max_a Q_\theta(s_{t+1}, a_t) \), where \( \theta \) are parameters of a target network that is a older version of the value network and updated periodically. DQN uses an off-policy learning strategy, which samples \((s_t, a_t, r_t, s_{t+1})\) tuple from a replay buffer for training.
DQN as a typical parametric reinforcement learning method suffers from sample inefficiency because of slow gradient-based updates. Thus episodic reinforcement learning is proposed to speed up the learning process by a non-parametric episodic memory. Episodic reinforcement learning enables fast learning by modeling the hippocampal instance-based learning. The key idea is to store good past experiences in a tabular-based non-parametric memory and rapidly latch onto past successful policies when encountering with similar states, instead of waiting for many steps of optimisation.

3 RELATED WORK

Deep Reinforcement Learning Our method is closely related to DQN (Mnih et al., 2015). As the seminal work of deep reinforcement learning, DQN learns a deep neural network for state-action value function by gradient back-propagation and conduct parametric control. Following this line, a large number of extensions have been proposed to improve the learning efficiency of the parametric model. Double DQN (Van Hasselt et al., 2016) alleviates the over-estimation issue of Q-Network. Dueling network (Wang et al., 2015) separates Q-Network to two streams which predict state value and advantage value respectively and achieves better generalization across actions. Prioritized experience replay (Schaul et al., 2015b) changes the sampling priority of each training sample according to its learning error. Apart from these prior improvement, many algorithms have been proposed to accelerate reward propagation and back up mechanism. Optimality Tightening method (He et al., 2016) combines the strength of DQN with a constrained optimization approach to rapidly propagate close-by rewards. $Q^*(\lambda)$ (Harutyunyan et al., 2016) and Retrace ($\lambda$) (Munos et al., 2016) incorporate on-policy samples into off-policy learning targets. Noisy Net (Fortunato et al., 2017) adds noise to the parametric model during learning to improve the exploration ability. Distributional RL (Bellemare et al., 2017) learns the value function as a full distribution instead of a expected value. Different from these work, we focus on combining non-parametric memory and parametric model in this paper, thus our method is complementary to these prior extensions and can be combined with them seamlessly.

Episodic Reinforcement Learning Our work is also related to episodic reinforcement learning. Model-free episodic control (Blundell et al., 2016) uses a completely non-parametric model that keeps the best Q values of states in a tabular-based memory and replays the sequence of actions that so far yielded the highest return from a given start state. At the end of each episode, the Q values in memory are updated by the greater of the existing values and the accumulated discounted returns in the current episode. In the execution stage, the agent selects actions according to a k-nearest-neighbours lookup in the memory table. Recently, a number of extensions have been proposed to integrate episodic control with parametric DQN. Neural episodic control (Pritzel et al., 2017) develops end-to-end episodic control by a differentiable neural dictionary to generate semi-tabular representation as slow-changing keys and then retrieves fast-updating values by context-based lookup for action selection. To better leverage trajectory nature of experience, ephemeral value adjustments method (Hansen et al., 2018) proposes to further leverage trajectory information from replay buffer to propagate value through time and produce trajectory-centric value estimates. Our method differs from EVA in that we associate memory by a graph, thus we can leverage not only intra-episode but also inter-episode information. Episodic memory deep q-networks (Lin et al., 2018) distills the information of episodic memory into a parametric model by adding a regularization term in objective function and significantly boosts up the performance of DQN. Different from these prior work which adopt either tabular memory or semi-tabular memory, our work builds a graph on memory items based on their relationship to from an associative memory.

Graph Based Methods in Deep Reinforcement Learning Recently, several work have also been proposed to use graph for planning in deep reinforcement learning. Eysenbach et al. (2019) builds a directed graph directly on top of states in replay buffer and run graph search to find the sequence of waypoints, leading to many easier sub-tasks and thus improve learning efficiency. Huang et al. (2019) abstracts state space as a small-scale map which allows it to run high-level planning using pairwise shortest path algorithm. Different from these prior work which use graph for planning, our method reorganize episodic memory by a graph to allow faster reward propagation. In addition, these graph based models rely on goal-conditioned RL (Kaelbling [1993] Schaul et al. [2015a]) and only demonstrate their performance in navigation-like problems, while our approach is intended for general RL settings.
Exploration Efficient exploration is a long-standing problem in reinforcement learning. Prior works have proposed guiding exploration based on criteria such as intrinsic motivation (Stadie et al., 2015), state-visitation counts (Jang et al., 2017), thompson sampling and bootstrapped models (Chapelle & Li, 2011; Osband et al., 2016), optimism in the face of uncertainty (Kearns & Singh, 2002), parameter-space exploration (Plappert et al., 2017; Fortunato et al., 2017). Recently, Oh et al. (2018) proposed self-imitation learning (SIL) and found that exploiting past good experiences can indirectly drive deep exploration. In their work, agent imitates its own decisions in the past only when such decisions resulted in larger returns than expected. Similar to SIL, EMDQN (Lin et al., 2018) learns from episodic memory to replay past best decisions, therefore incentivizing exploration. In our method, we build associative memory through a graph, which enhances the exploitation of past good experiences and thus can indirectly encourage deeper exploration than EMDQN (Lin et al., 2018).

4 Episodic Reinforcement Learning with Associative Memory

4.1 Associating episodic memory as a graph

Similar with previous episodic reinforcement learning, we adopt an episodic memory to maintain the historically highest values \( Q_{EC}(\phi(s), a) \) of each state-action pair, where \( \phi \) is an embedding function and can be implemented as a random projection or variational auto-encoders (VAE) (Kingma & Welling 2013). When receiving a new state, the agent will look up in the memory and update the values of states according to the following equation,

\[
Q_{EC}(\phi(s_t), a_t) \leftarrow \begin{cases} 
\max(R_t, Q_{EC}(\phi(s_t), a_t), R_t) & \text{if } (\phi(s_t), a_t) \in Q_{EC} \\
R_t & \text{otherwise.}
\end{cases}
\]  

(1)

However, episodic memory stores states as unrelated items and does not make use of relationship between these items. To fully exploit information in episodic memory, we further build a directed graph \( G \) on top of items in the episodic memory to form an associative memory, as shown in Figure 2. In this graph, each node corresponds to a memory item that records the embedded vector of a state \( \phi(s) \) and we leverage transitions of states to bridge the nodes. The graph are defined as,

\[
G = (V, E), V = \phi(s), E = \{ s \rightarrow s' \mid (s, a, s') \text{ is stored in memory} \}. 
\]  

(2)

Given a sampled trajectory, we temporarily add each state to the graph. We add directed edges from the given state to every other previously memorized state that is successor of it under a certain action. Our associative memory reorganizes the episodic memory and connects these fragmented states that previously yielded high returns by a graph. We rewrite these stored values \( Q_{EC}(\phi(s), a) \) as \( Q_{G}(\phi(s), a) \) in our graph augmented episodic memory. In addition, we adopt a strategy of discarding the least recently used items when the memory is full.

4.2 Propagating values through associative memory

Typical deep RL algorithms sample experience tuples uniformly from the replay buffer to update value function. However, the way of sampling tuples neglects the trajectory nature of an agent’s experience (i.e., one tuple occurs after another and so information of the next state should be quickly propagated into the current state). EVA (Hansen et al., 2018) encourages faster value propagation by introducing trajectory-centric planning (TCP) algorithm. Nonetheless, EVA only propagates value through the current episode, which we refer to as intra-episode propagation. Our insight here is that a same state might appear in different trajectories and such join points can help connect different various trajectories. Therefore, we explicitly build the graph between states from different trajectories in memory and thus allows inter-episode value propagation.

Since the graph over states is complicated (e.g., not a tree structure), value propagation over such graph is always slow. To accelerate the propagating process, we propagate values using the sequential property. The pseudo code of value propagation is shown in Algorithm 1. Our general idea is to update the values of the graph in the reverse order of each trajectory. Specifically, when adding to a new state to the memory, we record the sequential step ID \( t \) of the state at current trajectory. For memory associating, we first sort the elements in memory by their sequential step IDs in descending order, and propagate the value from states with large sequential step ID to small one for
Figure 2: Comparison of episodic memory and associative memory.

several iterations until $Q_{G}$ values converge. At each update, we get all successor state-action pairs $(s', a')$ of the current one $(s, a)$ and current reward $r$ according to the graph $G$ and apply max operation on successor action $a'$ to propagate the values to current state-action pair. Formally, our graph augmented memory is updated as follow:

$$Q_{G}(\phi(s), a) \leftarrow r + \gamma \max_{a'} Q_{G}(\phi(s'), a').$$

(3)

Since most of states at the beginning are similar across different episodes, our reverse order updating strategy can efficiently propagate all the values of the graph. In addition, as we show in Theorem 1, our graph-based value propagation algorithm can converge to a unique optimal point. The proofs are shown in Appendix A.

**Theorem 1.** Denote the Bellman backup operator in Equation 3 as $B : \mathbb{R}^{|S| \times |A|} \rightarrow \mathbb{R}^{|S| \times |A|}$ and a mapping $Q^0 : S \times A \rightarrow \mathbb{R}^{|S| \times |A|}$ with $|S| < \infty$ and $|A| < \infty$, and define $Q^{k+1} = BQ^k$. Repeated application of the operator $B$ for our graph-based state-action value estimate $\tilde{Q}_G$ converges to a unique optimal value $Q_G^*$.

In the previous episodic reinforcement learning with no graph built, only the values of exactly the same or similar states can be updated. This is because in the typical update rule of episodic memory as shown in Eq. 1, the relationship between states has been neglected. That is, episodic memory does not leverage the information of edges $E$ in our graph $G$. Consequently, stored values in episodic memory often violate bellman equation. On the contrary, our associative memory allows efficient value propagation through the edges of the graph to compute the more accurate values for each state.

### 4.3 Learning with Associative Memory

Building associative memory can be viewed as a way of augmenting counterfactual experiences. As shown in Figure 2, the same states might appear in $N > 1$ trajectories. Vanilla episodic memory maps such states to highest values among $N$ trajectories, while our associative memory regards such states as join points to connect different trajectories, leading to totally $N^2$ trajectories. This is equivalent to sample more combinatorial trajectories from environments, and thus can significantly improve sample efficiency of RL algorithms.

Our associative memory can be applied to both learning and control phase. In this paper, we use our associative memory as a guidance for the learning of Q function. The overall framework is shown as Figure 3. Specifically, we use the associative memory as regularization term of objective function to supervise the learning of Q-network. The Q-network is learned by minimized the following objective function:

$$L_\theta = \mathbb{E}_{(s,a,s',r) \sim D} \left[ (r + \gamma \max_a Q_{\hat{\theta}}(s', a) - Q_{\theta}(s,a))^2 + \lambda (Q_{\hat{\theta}}(\phi(s), a) - Q_{\theta}(s,a))^2 \right],$$

(4)

where $\lambda$ is the weight of the regularization term, $\theta$ represents parameters of parametric Q-network. Similar with DQN [Mnih et al. 2015], we also adopt a target network parameterized by $\hat{\theta}$ to stabi-
Figure 3: Overall framework of ERLAM.

lize the learning process. Through the combination of parametric and non-parametric term, we can efficiently guide the learning of a conventional Q-network by the fast-adjusted high values in associative memory so that the agent can rapidly latch on strategies that previously yield high returns instead of waiting for many steps of slow gradient update. The pseudo code of our method is shown in Algorithm 2.

Algorithm 2 ERLAM: Episodic Reinforcement Learning with Associative Memory

| D: Replay buffer          |
| G: Graph (Associative memory) |
| T_e: Trajectory length of e-th episode |
| K: Associate frequency |

**for** Episode number e = 1 . . . E **do**

**for** t = 1 . . . T_e **do**

Receive initial observation s_t from environment with state embedding h_t = \( \phi(s_t) \)

\( a_t \leftarrow \epsilon \)-greedy policy based on \( Q_\theta(s_t, a) \)

Take action \( a_t \), receive reward \( r_t \) and next state \( s_{t+1} \)

Append \((s_t, a_t, r_t, s_{t+1})\) to \( D \)

if \( t \mod \text{update_freq} == 0 \) **then**

Sample training experiences \((s, a, r, t)\) from \( D \)

Retrieve \( Q_G(\phi(s), a) \) from associative memory

Update parameter \( \theta \) using Eq. 4

**end if**

**end for**

**for** t = T_e . . . 1 **do**

\( R_t \leftarrow r_t + \gamma R_{t+1} \), if \( t < T_e \); \( R_t \leftarrow r_t \), if \( t = T_e \)

Append \((h_t, a_t, r_t, t, R_t)\) to \( G \) if \((h_t, a_t) \not\in G\)

Update \( Q_G \) using Eq. 1 if \((h_t, a_t) \in G\)

**end for**

if \( e \mod K == 0 \) **then**

Run Algorithm 1 to update \( Q_G \)

**end if**

**end for**

4.4 CONNECTION TO GRAPH-BASED DEEP REINFORCEMENT LEARNING

When the general RL setting used in our approach degenerates to a setting of navigation-like task that is usually adopted by goal-conditional RL [Kaelbling [1993] Schaul et al. [2015a]], the update target of associative memory in Eq. 3, \[ y = r + \gamma \max_{a'} Q_G(\phi(s'), a') \] can be rewritten as,

\[ y = \begin{cases} 
  r & \text{if } s' \text{ is a terminal state,} \\
  \gamma \max_{a'} Q_G(\phi(s'), a') & \text{otherwise.} 
\end{cases} \] (5)

Optimizing with target in Eq. 5 is equivalent to finding the shortest path in the graph of all states. In this case, algorithm 1 is analogous to Bellman-Ford algorithm [Bellman [1958]], which is proved
that the value can converge in limited iterations. In the context of goal-conditional RL, some graph-based methods [Huang et al., 2019; Eysenbach et al., 2019] also calculated shortest path. They focus on a graph of waypoints learned by goal-conditional RL instead of memorized states that previously yield high returns. In addition, they use a parametric approach for value approximation, while we develop a non-parametric approach to improve sample efficiency of a parametric RL agent.

5 EXPERIMENTS

5.1 EXPERIMENT SETTING

We follow the same setting for network architecture and all hyper-parameters as DQN (Mnih et al., 2015). The raw images are resized to an 84 × 84 grayscale image \( s_t \) and 4 consecutive frames are stacked into one state. The Q value network alternates convolutions and ReLUs followed by a 512-unit fully connected layer and an output layer whose size is equal to the number of actions in each game. Denote \( \text{Conv}(W, F, S) \) as the convolutional layer with the number of filters \( W \), kernel size \( F \) and stride \( S \). The 3 convolutional layers can be indicated as \( \text{Conv}(32,8,4) \), \( \text{Conv}(64,4,2) \), and \( \text{Conv}(64,3,1) \). We used the RMSProp algorithm (Tieleman & Hinton, 2012) with learning rate \( \alpha = 0.00025 \) for gradient descent training. The discount factor \( \gamma \) is set to 0.99 for all games. We use annealing \( \epsilon \)-greedy policies from 1.0 to 0.1 in training stage while fixing \( \epsilon = 0.05 \) during evaluation.

For hyper-parameters of associative memory, we set the value of \( \lambda \) as 0.1 and associate frequency \( K \) as 10 in the navigation domain, Monster Kong. In Atari games, we use the same settings for all games. The value of \( \lambda \) is 0.3 and associate frequency \( K \) is 50. The memory size is set as 1 million. We use random projection technique and project the states into vectors with dimension of \( d = 4 \). For efficient table lookup, we build a kd-tree for these low-dimension vectors.

![Maps of Monster Kong](image)

**Figure 4:** Maps of Monster Kong. Compared to MonsterKong1, MonsterKong2 has a different goal and MonsterKong3 has a totally different MDP.

5.2 RESULTS ON NAVIGATION DOMAIN

We first test our model on the navigation domain, which contributes to clearly demonstrate the superiority of our algorithm and understand the contribution of associative memory. We use a video game Monster Kong from Pygame Learning Environment (PLE; Tasfi, 2016) to set up the navigation experiments. In this game, the goal of the agent is to approach the princess with actions \( \text{up, down, left, right, jump} \) and \( \text{noop} \) from random starting positions. The agent will win with an extra reward +1 when touching the princess, and lose when hitting the thorns (silver triangles). We run ERLAM on three maps of Monster Kong (see Figure 4) and compare it with EMDQN and DQN.

As shown in Figure 5 the sample-efficiency of ERLAM significantly outperforms EMDQN and DQN. ERLAM with only 10M samples can gain higher scores than EMDQN with 80M samples on map MonsterKong2 and MonsterKong3. Then, we inspect the value estimation of Q networks and the stored values in memory to provide insight into our reinforcement learning results. We plot the average values of states in associative memory (orange line in bottom row of Figure 5) during the training process of ERLAM. To better understand the contribution of value propagation process in associative memory, we maintain a memory without value propagation (which amounts
Figure 5: Learning curves of ERLAM, EMDQN and DQN on Monster Kong. The top row compares the average scores per episode between all models. The bottom row shows state-action value estimates by associative memory, episodic memory, and Q networks when running ERLAM. The black dash line represents the actual discounted state-action values of the best learned policy.

to episodic memory, shown as green line in bottom row of Figure 5, in the meanwhile and compare the state-action values of it to associative memory. As expected, the values after value propagation of associative memory grow higher, which indicates associative memory provides a better non-parametric lower bound of Q value compared to episodic memory. Values estimated by associative memory are closer to the true values of optimal policy (black dash line) and capable to guide the learning of Q network (blue line). We further visualize and compare the execution policies according to associative memory and episodic memory to gain a deeper understanding of their connections. We study a case in Figure 6. We observe that the policy provided by associative memory (yellow dash line) is exactly the combination of two policies in episodic memory (blue line and red line) and such a combinatorial trajectory is not a real trajectory in replay buffer. This result suggests that the value propagation in associative memory enables automatic augmentation of counterfactual combinatorial trajectories, which accounts for the improvement of sample efficiency in ERLAM.

5.3 Results on Atari Games

To further evaluate the sample efficiency of ERLAM on a diverse set of games, we conduct experiments on the benchmark suite of Atari games from the Arcade Learning Environment (ALE; Bellemare et al. (2013)), which offer various scenes to test RL algorithms over different settings. We largely follow the training and evaluation protocol as (Mnih et al., 2015). We train our agents for 10 epochs, each containing 1 million frames, thus 10 million frames in total. For each game, we evaluate our agent at the end of every epoch for 0.5 million frames, with each episode up to 18000 frames and start the game with up to 30 no-op actions to provide random starting positions for the agent.

In our experiments, we compare ERLAM with episodic reinforcement learning baselines, MFEC (Blundell et al., 2016), NEC (Pritzel et al., 2017), EMDQN (Lin et al., 2018), EVA (Hansen et al., 2018), as well as an ablation (i.e., DQN with no associative memory). MFEC directly uses the non-parametric episodic memory for action selection, while NEC, EMDQN and EVA combine non-parametric episodic memory and a parametric Q-network. Different from previous work, ERLAM adopts associative memory to guide the learning of a Q-network.
Figure 7: Comparison between ERLAM (i.e., DQN with associative memory) and EMDQN (i.e., DQN with episodic memory) measured in improvements of scores over DQN as shown in Eq. 6. Bars indicate how much each algorithm outperforms the DQN (i.e., DQN with no memory) agent. 0% means the performance is equal to DQN and higher is better.

Table 1: Performance comparisons on mean and median human-normalized scores as in (Mnih et al., 2015). All agents are trained using 10 million frames except for EMDQN which is trained with 40 million frames.

<table>
<thead>
<tr>
<th></th>
<th>DQN</th>
<th>A3C</th>
<th>Prior. DQN</th>
<th>MFEC</th>
<th>NEC</th>
<th>EVA</th>
<th>EMDQN(40M)</th>
<th>ERLAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>83.6</td>
<td>40.1</td>
<td>116.6</td>
<td>77.7</td>
<td>106.1</td>
<td>172.2</td>
<td>250.6</td>
<td>515.4</td>
</tr>
<tr>
<td>Median</td>
<td>16.0</td>
<td>6.9</td>
<td>32.3</td>
<td>40.9</td>
<td>53.3</td>
<td>39.2</td>
<td>95.5</td>
<td>103.5</td>
</tr>
</tbody>
</table>

We tested ERLAM on 25 popular and challenging Atari games. To evaluate our approach, we follow Wang et al. (2015) and measure improvement in percentage in score over the better of human and DQN agent scores for both ERLAM and EMDQN:

\[
\text{Score}_{\text{Agent}} - \text{Score}_{\text{DQN}} = \frac{\max\{\text{Score}_{\text{Human}}, \text{Score}_{\text{DQN}}\} - \text{Score}_{\text{Random}}}{\max\{\text{Score}_{\text{Human}}, \text{Score}_{\text{DQN}}\}}.
\]

To test the sample efficiency of our method, we limit our training data to 10 million frames and compare with state-of-the-art results on episodic RL (i.e., EMDQN (Lin et al., 2018)), which are trained with 40 million frames and reported in their original paper. The results are shown in Figure 7. We found that even though our agent uses 4 time less training samples than EMDQN, ERLAM still outperforms EMDQN on 17 games. It demonstrates that associative memory can efficiently guide the learning of a parametric RL agent and our framework of combining associative memory with parametric RL can achieve significantly better sample efficiency than existing RL algorithms. We also compare the overall performance (mean and median) of ERLAM with other methods in Table 1. We found that ERLAM significantly outperforms all baselines over most games. For the games where ERLAM does not perform very well, we summarize the reasons as follows. First, ERLAM is good at improving the sample-efficiency in near-deterministic environments but may suffer from overestimation in highly stochastic environments, such as Tutankham. Second, since representations learning is not the focus of this paper, we simply use the naive random projection as the state representations in memory. Random projection is only used for dimension reduction and does not contain useful high-level feature or knowledge (e.g., objects and relations). Thus in some games with rare revisited states, there are not enough joint nodes in our graph and our algorithm does not perform well, such as FishingDerby and Jamesbond.

To gain better understanding of our superior performance, we further plot learning curves (Figure 8) on four games, which include three general good cases (Atlantis, BattleZone, StarGunner) and a bad case (BankHeist) to demonstrate when associative memory works extremely well and when it is not particularly effective. In addition, we plot the average values of states in memory (Figure...
for better revealing the performance difference on game scores. Across most games, ERLAM is significantly faster at learning than EMDQN and DQN, but ERLAM only has a slightly better performance than EMDQN on BankHeist. The reasons lie in two folds. Firstly, there are more crossed experiences on Atlantis, BattleZone, StarGunner than BankHeist, and thus on the first three games the values computed by associative memory are significantly larger than those in episodic memory. Secondly, we observe that the background objects in BankHeist have abnormally changeable appearance and complex behaviors, which are intractable for memory based methods (e.g., MFE, NEC, EMDQN and ERLAM) especially with a simple random projection embedding function for state feature abstraction (we also discussion this in Conclusion Section). It also accounts for the reason why ERLAM and EMDQN have similar performance with DQN.

Figure 8: Examples of learning curves on 10 million frames compared with EMDQN and DQN. The top row shows average scores per episode and the bottom row shows average values of states in memory. One Plot Number is equivalent to about 30K frames. Note that 0 indicates the first million.

We also add experiments to verify our superior performance benefits from associative memory rather than representations (e.g., random projection). As shown in Appendix Figure 9, DQN with only random projections as inputs has a much worse performance than ERLAM and the vanilla DQN, which suggests that it is associative memory that matters.

6 CONCLUSION

In this paper, we propose a biologically inspired sample efficient reinforcement learning framework, called Episodic Reinforcement Learning with Associative Memory (ERLAM). Our method explicitly organizes memorized states as a graph. We develop an efficient reverse-trajectory propagation strategy to allow the values of new experiences rapidly propagate to all memory items through the graph. Experiments in navigation domain and Atari games demonstrate that our proposed framework can significantly improve sample efficiency of current reinforcement learning algorithms.

In the future, there are some interesting research directions that can be pursued within our proposed framework. Firstly, in this paper, following the work of [Blundell et al., 2016] and [Lin et al., 2018], our state embedding function $\phi$ is implemented as random projection. It is possible to incorporate advanced representation learning approaches that can capture useful features into our framework to support more efficient memory retrieval and further boost up the performance. Secondly, existing episodic reinforcement learning algorithms mainly focus on value-based methods. It will be an interesting future work to extend the episodic memory to policy gradient methods. Thirdly, we instantiate our associative memory in the learning phase in this paper. However, associative memory can also be used in explicit episodic control to further enhance exploitation. Fourthly, at the current stage, ERLAM, as a kind of episodic RL approach, is only good at improving sample-efficiency in near-deterministic environments. To deal with completely stochastic environments, our model can be potentially extended by storing distribution of Q values [Bellemare et al., 2017, Dabney et al., 2018] instead of the maximum Q value in the associative memory.
REFERENCES


A THEORETICAL CONVERGENCE

Proof. Note that our graph-based value propagation is similar to the proof of value iteration (Bellman [1966], Bertsekas et al. [1995], Sutton & Barto [2011]). For any estimate of our graph-based action-value function $\hat{Q}_G$

\[ B\hat{Q}_G(s, a) = R(s, a) + \gamma \max_{a' \in A} \sum_{s' \in S} P_G(s'|s, a) \hat{Q}_G(s', a'), \]

where $P_G(s'|s, a)$ defines transition probability given graph $G$. For any action-value function estimates $\hat{Q}_G^1$, $\hat{Q}_G^2$,

\[ |B\hat{Q}_G^1(s, a) - B\hat{Q}_G^2(s, a)| \]
\[ = \gamma \max_{a' \in A} \sum_{s' \in S} P_G(s'|s, a) \hat{Q}_G^1(s', a') - \max_{a' \in A} \sum_{s' \in S} P_G(s'|s, a) \hat{Q}_G^2(s', a') \]
\[ \leq \gamma \max_{a' \in A} \sum_{s' \in S} P_G(s'|s, a) |\hat{Q}_G^1(s', a') - \hat{Q}_G^2(s', a')| \]
\[ = \gamma \max_{s \in S, a \in A} |\hat{Q}_G^1(s, a) - \hat{Q}_G^2(s, a)| \]

So the contraction property of Bellman operator holds:

\[ \max_{s \in S, a \in A} |B\hat{Q}_G^1(s, a) - B\hat{Q}_G^2(s, a)| \leq \gamma \max_{s \in S, a \in A} |\hat{Q}_G^1(s, a) - \hat{Q}_G^2(s, a)| \tag{7} \]

For the fixed point $Q^*_G$, we have:

\[ \max_{s \in S, a \in A} |B\hat{Q}_G(s, a) - B\hat{Q}_G^*(s, a)| \leq \gamma \max_{s \in S, a \in A} |\hat{Q}_G(s, a) - \hat{Q}_G^*(s, a)| \implies \hat{Q}_G \to Q^*_G \tag{8} \]

Therefore, we prove that our graph-based value propagation algorithm can converge to unique optimal value. 

\[ \square \]
### Raw Scores on Atari Games

Table 2: Raw scores on Atari games at 10 million frames. All agents are trained using 10 million frames except for EMDQN which is trained with 40 million frames.

<table>
<thead>
<tr>
<th>Game</th>
<th>DQN</th>
<th>A3C</th>
<th>Prior.DQN</th>
<th>MPEC</th>
<th>NEC</th>
<th>EVA</th>
<th>EMDQN(40M)</th>
<th>ERLAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>634.80</td>
<td>415.50</td>
<td>800.50</td>
<td>1717.70</td>
<td>3460.60</td>
<td>1007.93</td>
<td>1662.00</td>
<td>2070.85</td>
</tr>
<tr>
<td>Amidar</td>
<td>126.80</td>
<td>96.30</td>
<td>99.10</td>
<td>370.90</td>
<td>811.30</td>
<td>231.19</td>
<td>374.10</td>
<td>980.47</td>
</tr>
<tr>
<td>Assault</td>
<td>1489.50</td>
<td>720.80</td>
<td>1339.90</td>
<td>510.20</td>
<td>599.90</td>
<td>550.77</td>
<td>2566.80</td>
<td>3230.18</td>
</tr>
<tr>
<td>Atlantis</td>
<td>14210.50</td>
<td>3638.00</td>
<td>12579.10</td>
<td>95499.40</td>
<td>51208.00</td>
<td>180367.20</td>
<td>290953.30</td>
<td>359530.00</td>
</tr>
<tr>
<td>BankHeist</td>
<td>29.30</td>
<td>15.80</td>
<td>70.10</td>
<td>163.70</td>
<td>343.30</td>
<td>4022.45</td>
<td>348.00</td>
<td>702.92</td>
</tr>
<tr>
<td>BattleZone</td>
<td>6961.00</td>
<td>2354.20</td>
<td>13500.00</td>
<td>19053.60</td>
<td>13345.50</td>
<td>14000.47</td>
<td>28300.00</td>
<td>33995.24</td>
</tr>
<tr>
<td>BeamRider</td>
<td>3741.70</td>
<td>450.20</td>
<td>3249.60</td>
<td>749.60</td>
<td>749.60</td>
<td>1914.30</td>
<td>5980.90</td>
<td>7116.67</td>
</tr>
<tr>
<td>Boxing</td>
<td>31.30</td>
<td>2.50</td>
<td>64.70</td>
<td>10.70</td>
<td>72.80</td>
<td>58.43</td>
<td>89.30</td>
<td>87.77</td>
</tr>
<tr>
<td>ChopperCommand</td>
<td>827.20</td>
<td>1036.70</td>
<td>1426.50</td>
<td>3075.60</td>
<td>5070.30</td>
<td>1612.93</td>
<td>3106.70</td>
<td>4172.83</td>
</tr>
<tr>
<td>CrazyClimber</td>
<td>66061.60</td>
<td>70103.50</td>
<td>76574.10</td>
<td>9892.20</td>
<td>34344.00</td>
<td>90656.27</td>
<td>107038.70</td>
<td>106538.71</td>
</tr>
<tr>
<td>Defender</td>
<td>2877.90</td>
<td>4596.00</td>
<td>3486.40</td>
<td>10052.80</td>
<td>6126.10</td>
<td>2890.44</td>
<td>14408.00</td>
<td>705833.33</td>
</tr>
<tr>
<td>DemonAttack</td>
<td>5541.90</td>
<td>346.80</td>
<td>6503.60</td>
<td>1081.80</td>
<td>641.40</td>
<td>504.52</td>
<td>5603.10</td>
<td>11065.76</td>
</tr>
<tr>
<td>Enduro</td>
<td>364.90</td>
<td>0.00</td>
<td>1125.80</td>
<td>0.00</td>
<td>1.40</td>
<td>1106.35</td>
<td>659.00</td>
<td>912.73</td>
</tr>
<tr>
<td>FishingDerby</td>
<td>81.60</td>
<td>89.50</td>
<td>84.20</td>
<td>90.30</td>
<td>72.20</td>
<td>68.10</td>
<td>8.40</td>
<td>30.20</td>
</tr>
<tr>
<td>Frodo</td>
<td>339.10</td>
<td>218.90</td>
<td>711.30</td>
<td>925.10</td>
<td>2747.40</td>
<td>1005.44</td>
<td>596.30</td>
<td>3193.83</td>
</tr>
<tr>
<td>Hero</td>
<td>1050.70</td>
<td>4598.20</td>
<td>5164.50</td>
<td>14767.70</td>
<td>16265.30</td>
<td>12075.89</td>
<td>7247.80</td>
<td>13615.00</td>
</tr>
<tr>
<td>Jamesbond</td>
<td>165.90</td>
<td>31.50</td>
<td>203.80</td>
<td>244.70</td>
<td>376.80</td>
<td>252.18</td>
<td>586.70</td>
<td>518.42</td>
</tr>
<tr>
<td>Keul</td>
<td>6015.10</td>
<td>3627.60</td>
<td>6700.70</td>
<td>4555.20</td>
<td>5179.20</td>
<td>4030.04</td>
<td>7796.30</td>
<td>7755.80</td>
</tr>
<tr>
<td>KongFuMaster</td>
<td>17166.10</td>
<td>6634.60</td>
<td>21456.20</td>
<td>12906.50</td>
<td>30568.10</td>
<td>25005.15</td>
<td>23890.00</td>
<td>20353.49</td>
</tr>
<tr>
<td>Riverraid</td>
<td>3144.90</td>
<td>2312.60</td>
<td>4871.80</td>
<td>4195.00</td>
<td>5498.10</td>
<td>4026.74</td>
<td>7728.30</td>
<td>8138.66</td>
</tr>
<tr>
<td>RoadRunner</td>
<td>7285.40</td>
<td>759.90</td>
<td>24746.60</td>
<td>5432.10</td>
<td>12661.40</td>
<td>28194.17</td>
<td>27856.60</td>
<td>37318.63</td>
</tr>
<tr>
<td>Robotank</td>
<td>14.60</td>
<td>2.40</td>
<td>8.50</td>
<td>7.30</td>
<td>11.10</td>
<td>15.13</td>
<td>5.30</td>
<td>25.44</td>
</tr>
<tr>
<td>Seaquest</td>
<td>618.70</td>
<td>514.10</td>
<td>1192.20</td>
<td>711.60</td>
<td>1015.30</td>
<td>1714.15</td>
<td>4235.90</td>
<td>2693.96</td>
</tr>
<tr>
<td>StarGunner</td>
<td>604.80</td>
<td>613.60</td>
<td>1131.40</td>
<td>14834.90</td>
<td>1171.40</td>
<td>2006.22</td>
<td>23933.30</td>
<td>9432.97</td>
</tr>
<tr>
<td>Tutankham</td>
<td>148.70</td>
<td>108.30</td>
<td>194.00</td>
<td>86.30</td>
<td>121.60</td>
<td>171.33</td>
<td>148.00</td>
<td>146.94</td>
</tr>
<tr>
<td>YarsRevenge</td>
<td>7614.10</td>
<td>9953.00</td>
<td>9228.50</td>
<td>5956.70</td>
<td>21490.50</td>
<td>11010.90</td>
<td>13236.70</td>
<td>14259.26</td>
</tr>
</tbody>
</table>

Figure 9: Learning curves on 10 million frames compared with ERLAM, DQN with random project and vanilla DQN. The top row shows average scores per episode.