2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

The Effects of Memory Replay in Reinforcement Learning

Ruishan Liu
Department of Electrical Engineering
Stanford University
Stanford, USA
ruishan@stanford.edu

James Zou
Department of Biomedical Data Science
Stanford University
Stanford, USA
jamesyzou@gmail.com

***Abstract*—Experience replay is a key technique behind many recent advances in deep reinforcement learning. Allowing the agent to learn from earlier memories can speed up learning and break undesirable temporal correlations. Despite its widespread application, very little is understood about the properties of experience replay. How does the amount of memory kept affect learning dynamics? Does it help to prioritize certain experiences? In this paper, we address these questions by formulating a dynamical systems ODE model of Q-learning with experience replay. We derive analytic solutions of the ODE for a simple setting. We show that even in this very simple setting, the amount of memory kept can substantially affect the agent's performance—too much or too little memory both slow down learning. Moreover, we characterize regimes where prioritized replay harms the agent's learning. We show that our analytic solutions have excellent agreement with experiments. Finally, we propose a simple algorithm for adaptively changing the memory buffer size which achieves consistently good empirical performance.**

***Index Terms*—reinforcement learning, memory replay**

I. Introduction

In reinforcement learning (RL), the agent observes a stream of experiences and uses each experience to update its internal beliefs. For example, an experience could be a tuple of (state, action, reward, new state), and the agent could use each experience to update its value function via TD-learning. In standard RL algorithms, an experience is immediately discarded after it's used for an update. Recent breakthroughs in RL leveraged an important technique called experience replay (ER), in which experiences are stored in a memory buffer of certain size; when the buffer is full, oldest memories are discarded. At each step, a random batch of experiences are sampled from the buffer to update agent's parameters. The intuition is that experience replay breaks the temporal correlations and increases both data usage and computation efficiency [5].

Combined with deep learning, experience replay has enabled impressive performances in AlphaGo [12], Atari games [7], etc. Despite the apparent importance of having a memory buffer and its popularity in deep RL, relatively little is understood about how basic characteristics of the buffer, such as its size, affect the learning dynamics and performance of the agent. As an example, we train an RL agent to play the MountainCar game under the standard deep Q-Network (DQN) algorithm. The only parameter we vary is the size of the memory buffer, as shown in Fig. 1. Even in this simple game, we find that the agent's performance (validation score after 2000 learning episodes, averaged for 50 new games) depends non-monotonically on the memory size. Both too little and too much memory can substantially degrade the agent's learning. In practice, a memory buffer size is determined by heuristics that are not well understood and then is fixed for the agent.

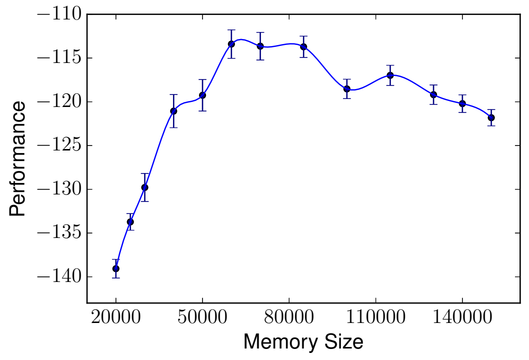


Fig. 1: Illustration of agent's performance vs. memory size for MountainCar Game with DQN. Error bar is standard deviation.

Prioritized experience replay (pER) is a modification of ER whereby instead of uniformly choosing experiences from the buffer to use in update, the agent is more likely to sample experiences that are "surprising" [8] [11]. pER is empirically shown to improve the agent's performance compared to the regular ER, but we also lack a good mathematical model of pER.

A. Contributions.

In this paper, we perform a first rigorous study of how the size of the memory buffer affects the agent's learning behavior. We develop an ODE model of experience replay and prioritized replay. In a simple setting, we derive analytic solutions characterizing the agent's learning dynamics. These solutions directly quantify the effects of memory buffer size on the learning rate. Surprisingly, even in this simple case with no value function model mismatch, memory size also has a non-monotonic effect on learning speed. Too much or too little memory both can slow down learning. Moreover, prioritized replay could also slow down learning. We confirm these theoretical predictions with experiments. This motivated us to develop a simple adaptive experience replay (aER) algorithm to automatically learn the memory buffer size as the agent is learning its other parameters. We demonstrate that aER consistently improves agent's performance.

B. Related works.

The memory replay technique has been widely implemented in RL experiments currently and is shown to have a good performance for different algorithms such as actor-critic RL algorithms [18], DQN algorithms [6], [7], and double Q-learning algorithms [16]. To further make good use of experience, prioritized methods are proposed for RL algorithms [8], [9]. The main idea of prioritization is to sample transitions that lead to larger value change in RL more frequently. The probability of selecting an experience is determined by the relative magnitude of the temporal difference error (TD-error). This has been reported to be effective in many experiments [8], [11], [17]. Measures other than TD-error are also in literature to weight experience; examples include rewards [15] and the transition property [10].

The performance of RL with experience replay is similar to the batch RL but in an incremental way [2]–[4]. Another approach to reuse data in RL is called model-learning or Dyna architecture, which builds a model to simulate and generate new data [13], [14]. This method, however, induces both extra computation cost and modeling error for the data.

II. A Dynamical System Model of Experience Replay

In an RL task, an agent takes actions a, observes states x and receives rewards r in sequence during its interaction with the environment. The goal is to learn a strategy which leads to best possible reward. A standard learning framework for the agent is to use the action-value function to learn optimal behavior and perform action selection. The optimal action-value $Q(x,a)$ is defined as the maximum expected return when the agent starts from state x and takes first action a. It satisfies

$$\begin{aligned} Q(x,a) &= \mathbb{E}\left[\sum_{i=0}^{K}\gamma^i r(x_i,a_i)\middle| x_0=x, a_0=a\right] \\ &= r(x,a)+\gamma\sum_{y\in X}P(x,a,y)\sup_{a'\in A}Q(y,a'), \end{aligned} \tag{1}$$

where $r(x,a)$ is the reward function, γ $(0\leqslant\gamma<1)$ denotes the discount factor, and $P(x,a,y)$ is the state transition probability kernel, defined as the probability of moving from state x to state y under action a.

In practice, the state space is usually large and the function approximation is adopted to estimate the action-value function $Q(x,a;\theta)$; DQN is an example of this approach. At learning time t, we compute the TD-error $\delta(t,t')$ based on the data collected at time t' $(t'\leqslant t)$ as

$$\begin{aligned} \delta(t,t') = r(t') &+ \gamma\cdot\max_{a'\in A}Q[x(t'+\Delta t),a';\theta(t)] \\ &- Q[x(t'),a(t');\theta(t)], \end{aligned} \tag{2}$$

where Δt is the time step. For standard RL algorithms, only the most recent transition is visited and $t'=t$, while for the ER approach, experience data are reused and $t'<t$.

Then the commonly-used TD-learning method updates the weight θ according to

$$\theta(t+\Delta t)-\theta(t)=\alpha(t)\delta(t,t')\nabla_\theta Q[x(t'),a(t');\theta(t)]\Delta t, \tag{3}$$

where $\alpha(t)$ is the learning rate. When $\Delta t=1$, Eq. (2) and Eq. (3) give the standard TD update at discrete learning step t in real experiments. The pseudo-code for a typical RL task with experience replay is given by Algorithm 1.

Algorithm 1 Reinforcement Learning with Experience Replay

```
Require: memory size N, minibatch size $m$, step size $\alpha$, discount factor $\gamma$, total steps $T$, initial weights $\theta_0$, update policy $\pi_\theta$
1: Initialize replay memory BUFFER with capacity N
2: Observe initial state $x_0$
3: for $t = 1$ to $T$ do
4:     Take action $a_t \sim \pi_\theta(x_t)$
5:     Observe $r_t$ and $x_{t+1}$
6:     Store transition $(x_t, a_t, r_t, x_{t+1})$ in memory BUFFER
7:     Sample $m$ transitions $(x_i, a_i, r_i, x_{i+1})$ randomly from BUFFER
8:     Compute TD-error $\delta_i = r_i + \gamma \max_a Q(x_{i+1}, a; \theta) - Q(x_i, a_i; \theta)$
9:     Update weights
           $\theta = \theta + (\alpha/m)\sum_{i=i_1}^{i_m} \delta_i \nabla_\theta Q(x_i, a_i; \theta)$
10: end for
```

In ER, recent transitions are stored in a replay memory with the capacity N, and a minibatch of data is randomly chosen for TD-learning. The effect of the memory buffer can not be extracted from the ER algorithm itself, and the hidden mechanism is hard to perceive only with experiments in a black box. Thus we derive an ODE model to simulate the learning process. General results are obtained numerically and even analytically, confirmed as good matches with experiments. This analytic approach enables us to systematically analyze how the replay memory affects the learning process and what is the principle behind it.

Proposition 1. *In the limit where the time step size Δt goes to 0 and the number of samples from the memory buffer at each step increases, the parameter dynamics under experience replay converges to*

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t}=\frac{\alpha(t)}{n(t)}\int_{t-n(t)}^{t}\delta(t,t')\nabla_\theta Q[x(t'),a(t');\theta(t)]\mathrm{d}t', \tag{4}$$

where $n(t)$ is the memory size and $\delta(t, t')$ is the TD-error given in Eq. (2). The dynamics of the agent's state converges to

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \int_X [y - x(t)] P\left(x(t), a(t), y\right) \mathrm{d}y, \tag{5}$$

where the action $a(t)$ is selected according to a policy π: $a(t) = \pi[x(t)]$

Derivation details are provided in the Appendix.

Now we are able to analyze the learning process, more specifically, the weights $\theta(t)$ and state $x(t)$ as a function of learning time, based on Eq. (4) and Eq. (5). No experiment is needed and the influence of the memory buffer or other parameters can be analyzed explicitly from the analytical solutions. Our theoretical model is further validated by experiments based on ER algorithms.

Prioritized replay (pER) proposes to speed up the learning process by sampling the experience transitions according to a non-uniform probability distribution. One commonly used model is to parametrize the probability $p(t_i)$ for selecting data collected at learning time t_i according to its TD-error $\delta(t, t_i)$ as

$$p(t_i) = \frac{|\delta(t, t_i)|^\beta}{\sum_{t_j} |\delta(t, t_j)|^\beta}, \tag{6}$$

where β is a constant exponent. The only difference between ER and pER is in how to sample experiences. Taking $\beta = 2$ as an instance, the dynamic equation for weights under pER is given by

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \frac{\alpha(t) \int_{t-n(t)}^{t} \delta^3(t, t') \nabla_\theta Q[x(t'), a(t'); \theta(t)] \mathrm{d}t'}{\int_{t-n(t)}^{t} \delta^2(t, t') \mathrm{d}t'}. \tag{7}$$

III. Analysis of Memory Effects in a Simple Setting

Starting from a toy game we call LineSearch, we analytically solve the ODEs (4) and (26) to get the learning dynamics and quantify the effects of memory. We further characterize settings when pER helps or hinders learning tasks compared to ER. In the RL experiments, the learning time t is discretized into integer learning step t and the time step size Δt is 1. We show that our theoretical predictions have excellent agreement with experiments.

A. Model setup.

We first define a simple game LineSearch, for which the space of agent state x is one dimensional, the reward function is linear $r(x) = \beta_1 x + \beta_2$, and the action is binary $a \in \{v, -v\}$, where v is a constant. In a transition, the next state is determined by adding the action value to the current state, i.e., $x(t + \Delta t) = x(t) + a(t)$. When the discount factor γ is set as 0 (non-zero γ setting gives similar agent's behavior and will be addressed later), the real action-value function is

$$Q_{\mathrm{real}}(x, a) = \beta_1 \cdot (x + a) + \beta_2. \tag{8}$$

When there is no model mismatch, the action-value function of the agent is

$$Q_{\mathrm{agent}}(x, a; \theta) = \theta_1 \cdot (x + a) + \theta_2. \tag{9}$$

At $t = 0$, θ_1 and θ_2 are randomly initialized. As the agent performs TD update, we are interested in how quickly the θ's approach the true β's. A natural evaluation metric is $\Delta\theta_1$ and $\Delta\theta_2$ defined as

$$\Delta\theta_1 = \theta_1 - \beta_1 \text{ and } \Delta\theta_2 = \theta_2 - \beta_2 \tag{10}$$

The agent learns well if $\Delta\theta_1$ and $\Delta\theta_2$ approach 0, and performs badly when the amplitude of $\Delta\theta_1$ or $\Delta\theta_2$ is large.

Under a greedy policy, the evolution of the agent's state is

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{\theta_1}{|\theta_1|} v. \tag{11}$$

With the initial state denoted as x_0, the evolution of the two metrics is derived from Eqs. (2)-(11) as

$$\frac{\mathrm{d}\Delta\theta_1(t)}{\mathrm{d}t} = -(b_{10} + b_{11} t + b_{12} t^2) \Delta\theta_1(t) - (b_{20} + b_{21} t) \Delta\theta_2(t) \tag{12a}$$

$$\frac{\mathrm{d}\Delta\theta_2(t)}{\mathrm{d}t} = -(b_{20} + b_{21} t) \Delta\theta_1(t) - b_{22} \Delta\theta_2(t), \tag{12b}$$

where $b_{10} = \alpha(n^2 v^2/3 + x_0^2 - n v x_0)$, $b_{11} = \alpha(2 v x_0 - n v^2)$, $b_{11} = \alpha v^2$, $b_{20} = \alpha(x_0 - nv/2)$, $b_{21} = \alpha v$, and $b_{22} = \alpha$. Here our discussion is set in the $\theta_1 > 0$ region, similar study could be carried out when $\theta_1 < 0$. Throughout this section, we choose the initial state $x_0 = -5$, the action amplitude $v = 0.01$, and the initial metrics $\Delta\theta_1^0 = -0.1$ and $\Delta\theta_2^0 = 0.5$.

The learning process in theory can then be calculated based on the dynamic equations of the metrics, i.e., Eqs. (12). It should be noted that the weights θ are obtained at the same time as $\theta_1 = \Delta\theta_1 + \beta_1$ and $\theta_2 = \Delta\theta_2 + \beta_2$.

The theoretical learning curves for both RL and pRL settings are depicted in Fig. 2a and Fig. 2b, with the learning rate α being 0.05. The two metrics $\Delta\theta_1$ and $\Delta\theta_2$ are represented by the solid blue and orange curve, respectively.

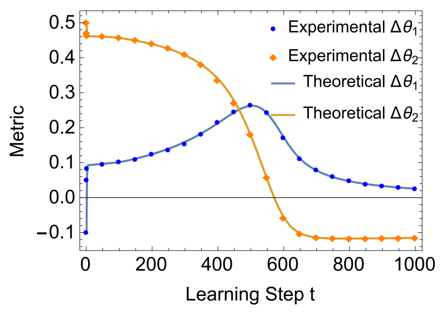
To demonstrate the validity of our theoretical solution, we also performed experiments on LineSearch following ER and pER algorithms with minibatch size being 5. For illustration, we plot the experimental results in Fig. 2a and Fig. 2b, where the blue dots stand for $\Delta\theta_1$ and the orange square denotes $\Delta\theta_2$. Our theoretical prediction has excellent agreement with experiment results.

B. Effects of memory size.

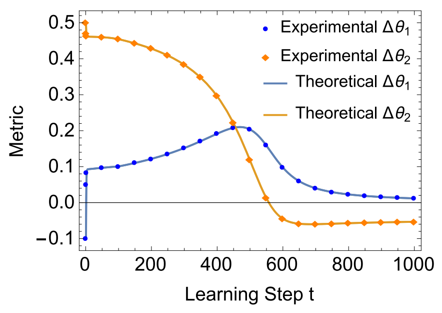
By solving the ODEs, we found that the memory setting has a non-monotonic effect on the RL performance. We are able to extract the mechanism behind this phenomenon from the analytic expressions.

With θ_2 being fixed as the real value $\theta_2 \equiv \beta_2$, the metric $\Delta\theta_1(t)$ is solved analytically

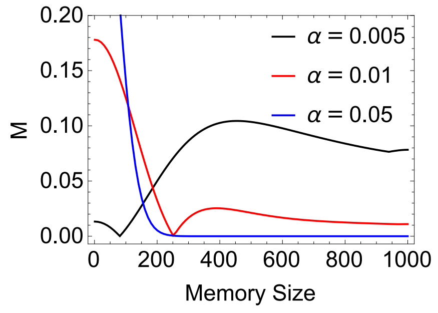
$$\begin{aligned}\Delta\theta_1(t) =& \Delta\theta_1^0 e^{-\alpha\left[\frac{v^2}{3} t^3 + \frac{v(2x_0 - Nv)}{2} t^2\right]} \\ & \cdot e^{-\alpha\left[\left(x_0^2 - N v x_0 + \frac{N^2 v^2}{3}\right) t - \frac{1}{18} N^2 v (Nv - 9x_0)\right]},\end{aligned} \tag{13}$$



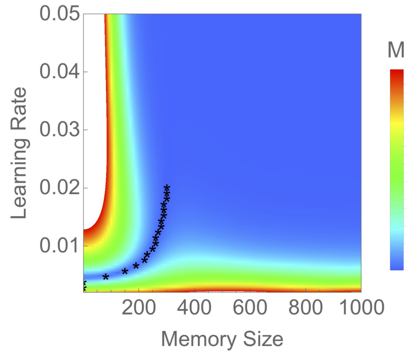
(a) ER.



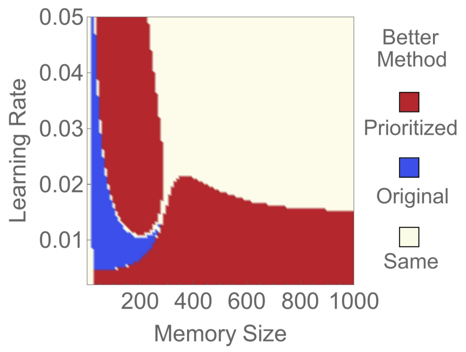
(b) pER.



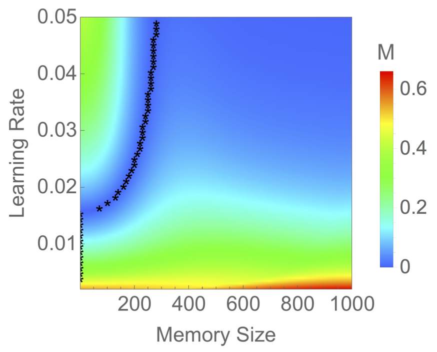
(c) M vs. memory size for ER.



(d) ER with $\gamma = 0$.



(e) Compare ER and pER



(f) ER with $\gamma = 0.7$.

Fig. 2: (a,b) Learning curves for two metrics $\Delta\theta_1 = \theta_1 - \beta_1$ and $\Delta\theta_2 = \theta_2 - \beta_2$, where $\theta_{1(2)}$ and $\beta_{1(2)}$ are the weights of the agent and the real weights, respectively. The scattered blue dots and orange squares represent the experimental results for $\Delta\theta_1$ and $\Delta\theta_2$, using (a) ER algorithm and (b) pER algorithm. The blue and orange curves are the numerical solutions for $\Delta\theta_1$ and $\Delta\theta_2$ based on our theoretical model. *For both ER and pER, our theoretical predictions (curves) have excellent agreement with the experimental results (dots) on the learning dynamics.* (c) Dependence of final absolute metric sum $M = |\Delta\theta_1(t = 1000)| + |\Delta\theta_2(t = 1000)|$ on memory size for different learning rate α. Note that smaller M stands for better performance. Here we use the original setting when the discount factor $\gamma = 0$. (d,f) Contour plot of measure M as a function of memory size and learning rate, with the discount factor (d) $\gamma = 0$ and (f) $\gamma = 0.7$. The stars denote the optimal memory sizes given learning rate values. The plots in (c) corresponds to the situations in (d) when the learning rate is 0.005, 0.01, and 0.05. (e) The red (blue) region stands for the situations when pER (ER) works better. The rest white region is the situations when the two settings behave similarly. More precisely, the absolute difference of M for the ER and pER is less than 1.5×10^{-3} in the white area. Here (c-f) are plotted based on theory predictions and also fit experiments well, similar to (a,b).

The metric converges exponentially to 0, but the rate of convergence (the exponent) has non-monotonic dependence on the memory size N.

In contrast, when θ_1 is fixed to the correct value, $\theta_1 \equiv \beta_1$, the metric $\Delta\theta_2(t)$ evolves according to

$$\Delta\theta_2(t) = \Delta\theta_2^0 e^{-\alpha t} \tag{14}$$

In this case, the choice of memory size has no effect on the learning behavior.

Now we turn to a more general situation when the updates of the two weights are coupled together. To examine the performance, we define a measure $M = |\Delta\theta_1(t = 1000)| + |\Delta\theta_2(t = 1000)|$, i.e., the sum of the absolute values of the two metrics at the learning step 1000, the end of the game. The smaller the measure M is, the better the agent learns. Fig. 2d plots the dependence of M on the memory size N and the learning rate α.

The learning performance is affected non-monotonically by the memory size for $\alpha < 0.02$, while a monotonic relation is observed for $\alpha > 0.02$, as shown in Fig. 2d. For example, an optimal memory size around 250 exists for $\alpha = 0.01$, denoted by the red curve in Fig. 2c. In contrast, the measure M experiences a monotonic decrease with the growth of memory size for $\alpha = 0.05$, plotted by the blue curve in Fig. 2c.

The influence of the memory setting in RL arises from the trade-off between the overshooting and the weight update. Here the term overshooting describes the phenomenon when some of the weights are updated in the wrong direction. For example, in Fig. 2a, θ_1 actually moves further away from β_1 during learning steps 200 to 500; θ_2 also overshoots at $t = 600$ and incurs negative bias. We first address the settings with small learning rate. When the memory size is small, the learning process is more likely to overshoot because of the limited memory capacity; when the replay memory is enlarged, the overshooting effect is mitigated. In the meantime, with the increase of the memory size, the averaged weight update first becomes slow then slightly accelerates. The balance between these two contributions leads to the non-monotonic nature. When the learning rate is large, there is still a trade-off between overshooting and increasing weight update. However, the latter can not counteract the former because of the quick convergence induced by the large TD update.

C. Performance of prioritized replay.

We further compare pER and ER, and discuss how the memory buffer affects pER based on our theoretical model. Fig. 2b plots the learning curve for pER, which exhibits a similar property as for ER in Fig. 2a. We compare the performance of RL and pRL algorithms in Fig. 2e, where blue (red) regions stand for cases when ER (pER) is better, and white areas represent situations when the two algorithms perform similarly. It is shown that pER performs relatively worse when the memory size is small, particularly when the learning rate is not large. This is also attributed to the trade-off between the overshooting and quick weight update. For small memory size, the overshooting effect is more serious under the prioritized sampling, while for a large memory, the prioritized agents update the weight quicker, which leads to a faster convergence.

D. Nonzero discount factor.

The discount factor is set as 0 in the previous subsections for simplicity. Here we show that the learning dynamics with $\gamma > 0$ is qualitatively similar to the case when $\gamma = 0$ in the LineSearch game. With a nonzero discount factor γ, i.e., considering the long-term effect, the real action-value function under the greedy policy is

$$\begin{aligned}Q_{\text{real}}(x,a) &= \mathbb{E}\left[\sum_{i=0}^{K}\gamma^i[\beta_1(x_0+a_0+i|\beta_1|v) + \beta_2]\middle| x_0=x, a_0=a\right]\\ &= \frac{1-\gamma^{K+1}}{1-\gamma}[\beta_1(x+a)+\beta_2] + v|\beta_1|\left[\frac{\gamma-\gamma^{K+1}}{(1-\gamma)^2} - \frac{K\gamma^{K+1}}{1-\gamma}\right]\\ &\approx \frac{\beta_1}{1-\gamma}(x+a) + \frac{\beta_2}{1-\gamma} + \frac{\gamma v|\beta_1|}{(1-\gamma)^2},\end{aligned} \tag{15}$$

where K stands for the total training steps afterwards before the game ends. The approximation in Eq. (15) is valid for large K and $\gamma < 1$. For instance, with the discount factor $\gamma = 0.9$, the contribution of the term γ^K after 100 steps is $\gamma^{100} = 0.00003$.

When there is no model mismatch, the action-value function of the agent is

$$Q_{\text{agent}}(x,a;\theta) = \theta_1 \cdot (x+a) + \theta_2, \tag{16}$$

Algorithm 2 Adaptive Memory Size Reinforcement Learning with Experience Replay

```
Require: initial memory size $N_0$, discount factor $\gamma$, total steps $T$, number of checked oldest transitions $n_{\text{old}}$ and memory adjustment internal $k$.
    Initialize replay memory BUFFER with capacity $N = N_0$ and set $|\delta_{\text{old}}| = 0$
    for $t = 1$ to $T$ do
        Take action $a_t$, observe reward $r_t$ and next state $x_{t+1}$, and store transition $(x_t, a_t, r_t, x_{t+1})$.
        Do TD updates as in standard ER algorithm.
        if mod(t, k) = 0 and memory BUFFER is full then
            Compute $|\delta_{\text{old}}|' = \sum_{i=t-N+1}^{t-N+n_{\text{old}}} |r_i + \gamma \max_a Q(x_{i+1}, a; \theta) - Q(x_i, a_i; \theta)|$
            if $|\delta_{\text{old}}|' > |\delta_{\text{old}}|$ or $N = k$ then
                Enlarge the memory $N = N + k$
                $|\delta_{\text{old}}| = |\delta_{\text{old}}|'$
            else
                Shrink the memory $N = N - k$, delete the oldest $k$ transitions in BUFFER
                Compute $|\delta_{\text{old}}| = \sum_{i=t-N+k+1}^{t-N+k+n_{\text{old}}} |r_i + \gamma \max_a Q(x_{i+1}, a; \theta) - Q(x_i, a_i; \theta)|$
            end if
        end if
    end for
```

Then the evolution of the two weights are derived together with Eqs. (4)-(11) as

$$\begin{aligned}\frac{\mathrm{d}\theta_1(t)}{\mathrm{d}t} =& -(b_{20}+b_{21}t)[(1-\gamma)\theta_2(t) - \gamma v|\theta_1(t)| - \beta_2]\\ &- (b_{10}+b_{11}t+b_{12}t^2)\left[(1-\gamma)\theta_1(t) - \beta_1\right]\end{aligned} \tag{17}$$

$$\begin{aligned}\frac{\mathrm{d}\theta_2(t)}{\mathrm{d}t} =& -(b_{20}+b_{21}t)\left[(1-\gamma)\theta_1(t) - \beta_1\right]\\ &- b_{22}\left[(1-\gamma)\theta_2(t) - \gamma v|\theta_1(t)| - \beta_2\right],\end{aligned} \tag{18}$$

where $b_{10} = \alpha(n^2v^2/3 + x_0^2 - nvx_0)$, $b_{11} = \alpha(2vx_0 - nv^2)$, $b_{11} = \alpha v^2$, $b_{20} = \alpha(x_0 - nv/2)$, $b_{21} = \alpha v$, and $b_{22} = \alpha$.

We find that the ODEs for $\gamma > 0$ have similar form as $\gamma = 0$. Correspondingly, the results obey similar principles, as shown in Fig. 2f. Here the step size is $\alpha = 10^{-3}$, the real weights θ_1^r and θ_2^r are 0.1 and 0.5, and the initial weights θ_1^0 and θ_2^0 are 0 and 1.

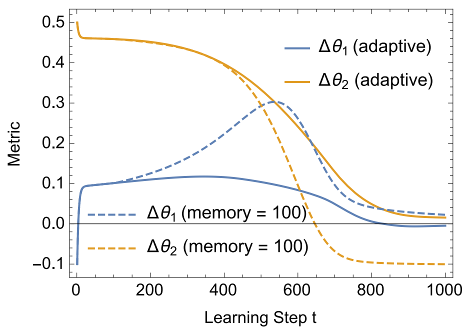
IV. Adaptive Memory Size Algorithm

The analysis from the previous section motivated us to develop a new algorithm that allows the agent to adaptively adjust the memory size while it is learning other parameters.

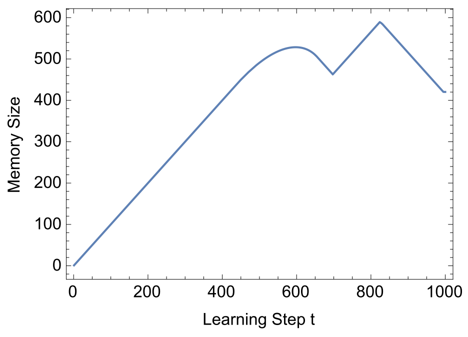
A. Intuition.

The lesson from Section III is that an useful adaptive algorithm should increase the memory capacity when the overshooting effect dominates and shrink the memory buffer if the weight update becomes too slow.

We use the *change* in absolute TD error of the oldest memories in the buffer as a proxy for whether the agent



(a) Learning curve for the LineSearch game, when the memory is learned adaptively from 100 (solid) vs. fixed at 100 (dashed). Metric equal to 0 is optimal.



(b) Adaptive memory for the LineSearch game starting from memory size of 100.

Fig. 3: Learning curve and adaptive memory for LineSearch.

is overfitting to more recent memories. The intuition is that if the TD error magnitude for the oldest transitions in the buffer starts to increase—i.e. the old data violate the Bellman equation more severely as the agent learns—then this is a sign that the agent might be overshooting and overfitting for the more recent experiences. In this case, we enlarge the memory buffer to ensure that the older experiences are kept longer to be used for future updates. On the other hand, if the TD error magnitude for the oldest transitions in the buffer starts to decrease over time, then the older memories are likely to be less useful and the agent decreases the memory buffer to accelerate the learning process.

B. *Algorithm description.*

The memory size is adaptively changed according to the TD error magnitude change of the oldest transitions, characterized by $|\delta_{\text{old}}|' - |\delta_{\text{old}}|$. Here $|\delta_{\text{old}}|$ and $|\delta_{\text{old}}|'$ are defined as the sum of the absolute TD errors of the oldest n_{old} transitions in memory, where n_{old} is a hyperparameter which denotes the number of old data we choose to examine. $|\delta_{\text{old}}|$ is first calculated. After k steps, we derive $|\delta_{\text{old}}|'$ and compare it with $|\delta_{\text{old}}|$. More specifically, every k steps, if the absolute TD error magnitude sum for the old transitions decreases, i.e., $|\delta_{\text{old}}|' < |\delta_{\text{old}}|$, the memory shrinks, otherwise increases, as given in Algorithm 2, denoted as aER.

C. *Performance on LineSearch.*

We first analyze how aER works for the LineSearch game. With the learning rate $\alpha = 0.01$, the agent adaptively adjusts its memory capacity from 100 as depicted in Fig. 3b. Compared to the setting with fixed memory size as 100, the agent updates weights more effectively and the overshooting effect is mitigated, indicated by Fig. 3a.

D. *Performance on OpenAI Games with DQN.*

We further evaluated the algorithm on three standard RL benchmarks that we downloaded from OpenAI Gym, which are CartPole, MountainCar and Acrobot.

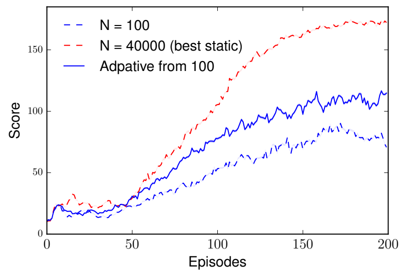
We used DQN with fully connected neural network (NN) for the value function approximation, where the NN is one layer for CartPole and two layers for MountainCar and Acrobot. Here we randomly initialize the weights, and set the minibatch size to be 50 and memory adjustment internal $k = 20$. The checked old transitions cover half of the initial memory size, out of which we randomly sample 50 (1000) experiences to approximate $|\delta_{\text{old}}|'$ for CartPole (MountainCar and Acrobot). The discount factor γ is set to be 0.9 for CartPole and 0.99 for MountainCar and Acrobot. The learning rate α is 10^{-3}, 6×10^{-4} and 10^{-3} for CartPole, MountainCar and Acrobot.

The adaptive memory algorithm achieved better performance compared to having a fixed-size replay buffer in all three games.

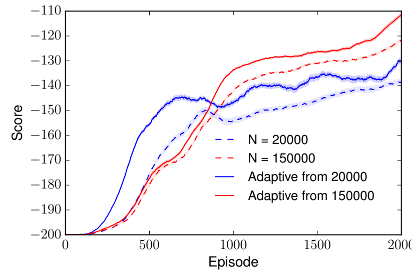
First, for the CartPole game which starts with an initial memory size of 100, the agent speeds up its learning from adaptive adjustments of the memory size, as shown in Fig. 4a and Fig. 4d where each curve is averaged for 100 new games. Here a larger static memory is always better and the best performance is achieved when no experience is discarded, corresponding to the full size $40,000$. With the aER, the agent correctly learns to increase its memory size from a small initial size of 100. We note that in 71% of the trials the aER outperforms the averaged static memory strategy.

In the second example, MountainCar, Fig. 1 indicates an optimal fixed memory size around 60,000. Both smaller and larger static memory sizes reduce learning. In this setting, aER learns to increase the memory size if the initial memory is small and it also learns to decrease the capacity if the initial memory is too large, as indicated by Fig. 4b and Fig. 4e where the result is averaged for 100 new games. 62% and 86% trials with aER outperforms the averaged fixed memory results for the initial size of 20,000 and 150,000, respectively. We note that the dynamical change of memory size from 150,000 also enables the agent to earn 2 more points than the best static result at the end of the game.

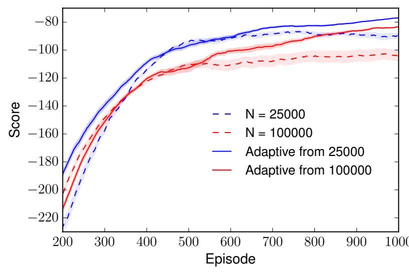
Last, aER also demonstrates good performance in the Acrobot game where a relatively smaller fixed memory is preferred, as plotted in Fig. 4c and Fig. 4f where each curve is averaged for 100 new games. When the initial buffer size is 25,000 and 100,000, aER outperforms the averaged fixed memory approach in 100% and 95% of the experiments, respectively. Here the agent learns to accelerate its learning by dynamically decreasing the memory size.



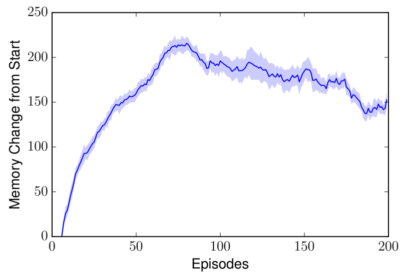
(a) Learning curve for CartPole, when the memory is adaptively learned starting from 100, and fixed as 100 and 40000, i.e., no experience is discarded.



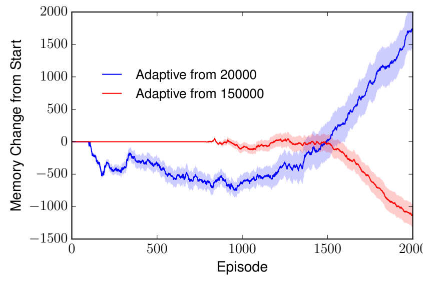
(b) Learning curve for MountainCar, when the memory is fixed as 20000 and 150000, and adaptively learned starting from the size of 20000 and 150000.



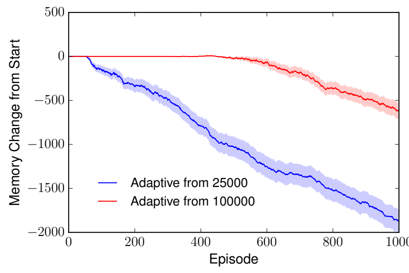
(c) Learning curve for Acrobot, when the memory is fixed as 25000 and 100000, and adaptively learned starting from the size of 25000 and 100000.



(d) Adaptive memory change for CartPole starting from memory size of 100.



(e) Adaptive memory compared to the initial size 20000 and 150000 for MountainCar.



(f) Adaptive memory compared to the initial size 25000 and 100000 for Acrobot.

Fig. 4: (a-c) Learning curves for CartPole, MountainCar and Acrobot, which are averaged for 100 new games. (d-f) Adaptive memory change from the initial size, corresponding to (a-c). The shaded areas indicate the standard error of the mean for the 100 experiments.

E. *Additional considerations.*

Little extra computation cost is needed to carry out the aER. Taking the CartPole game for example, only every 20 steps, we need to compute one or two forward passes of neural network without backpropogation. The examined batch is in the same size as the sampling minibatch for TD update.

The simple aER algorithm has a tendency to shrink the memory, but already shows good performance in experiments. The goal of the TD learning process is to diminish the TD error amplitude for all data, so the updated $|\delta_{\text{old}}|'$ is more likely to be less than $|\delta_{\text{old}}|$ in average sense. One possible solution is to change the criterion for shrinking the memory buffer to be $|\delta_{\text{old}}|' < |\delta_{\text{old}}| - \epsilon$, where ϵ could be a predefined constant, or learned online such as from the averaged TD error amplitude change through the whole dataset and the previous $|\delta_{\text{old}}| - |\delta_{\text{old}}|'$ value.

V. Discussion

Our analytic solutions, confirmed by experiments, demonstrate that the size of the memory buffer can substantially affect the agent's learning dynamics even in very simple settings. Perhaps surprisingly, the memory size effect is non-monotonic even when there is no model mismatch between the true value function and the agent's value function. Too little or too much memory can both slow down the speed of agent's learning of the correct value function. We developed a simple adaptive memory algorithm which evaluates the usefulness of the older memories and learns to automatically adjust the buffer size. It shows consistent improvements over the current static memory size algorithms in all four settings that we have evaluated. There are many interesting directions to extend this adaptive approach. For example, one could try to adaptively learn a prioritization scheme which improves upon the prioritized replay. This paper focused on simple settings in order to derive clean, conceptual insights. Systematic evaluation of the effects of memory buffer on large scale RL projects would also be of great interest.

Appendix

A. *Derivation of the ODEs*

To simulate the learning process of ER, we derive an ODE model, which enables us to systematically analyze the learning properties and the influence of the replay memory. The theoretical predictions exhibit excellent agreement with experimental results.

In the ODE model, the discrete learning step t is interpolated into continuous learning time t. Under the continuous approximation, the dynamic equation for the weights at learning time t based on the data collected at time t' ($t' \leqslant t$)

is approximated as

$$\begin{aligned}\left[\frac{\mathrm{d}\theta(t)}{\mathrm{d}t}\right](t') &= \lim_{\Delta\to 0}\frac{\theta(t+\Delta)-\theta(t)}{\Delta} \\ &\approx \frac{\theta(t+\Delta t)-\theta(t)}{\Delta t} \\ &= \alpha(t)\delta(t,t')\nabla_\theta Q[x(t'),a(t');\theta(t)].\end{aligned} \tag{19}$$

Eq. (3) is utilized in the derivation.

In the RL experiments, the discrete learning step size is $\Delta t = 1$. The agent's state evolution is estimated as

$$\begin{aligned}\frac{\mathrm{d}x(t)}{\mathrm{d}t} &= \lim_{\Delta\to 0}\frac{x(t+\Delta)-x(t)}{\Delta} \\ &\approx \frac{x(t+\Delta t)-x(t)}{\Delta t} \\ &= \int_X [y-x(t)]P\left(x(t),a(t),y\right)\mathrm{d}y.\end{aligned} \tag{20}$$

Validation of Continuous Approximation. The validity of our ODE approximation follows from standard results in stochastic approximation. We state the precise conditions and guarantees here.

Consider a general discrete stochastic system described by

$$\theta_{n+1} = \theta_n + \alpha_n h(\theta_n, w_n), \tag{21}$$

where $\theta_n \in \mathbb{R}^d$ is the state of the system, α_n is the step-size, w_n is a random variable that captures noise, and h is a function. In our work, θ is the continuous state parameter of the agent and h is the TD update function. Stochastic approximation states that under certain regularity conditions, the discrete system is *well approximated* by the ODE $\frac{d\theta(t)}{dt} = \bar{h}(\theta(t))$, where $\bar{h}(\theta) = E_w[h(\theta, w)]$. To make the appoximation precise, we need a few standard assumptions: (A1) h is Lipshitz; (A2) α_n satisfies $\sum_n \alpha_n = \infty$ and $\sum_n \alpha_n^2 < \infty$; (A3) w_n is a Martingale difference sequence; and (A4) θ_n is bounded.

Theorem 2. *Given A1-A4, almost surely the sequence $\{\theta_n\}$ generated by Eqn. 21 converges to the solution trajectory of $\frac{d\theta(t)}{dt} = \bar{h}(\theta(t)), t \geq 0$. See Chapter 2 of Borkar, "Stochastic Approximation: a Dynamical Systems Viewpoint." [1].*

ODEs with memory replay. With a memory buffer, the weights are updated based on a minibatch of data, instead of a single transition as in Eq. (19). Sampling from the whole memory, the expected weight gradient is

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \int_{t-n(t)}^{t} \mathrm{d}t' p(t,t')\left[\frac{\mathrm{d}\theta(t)}{\mathrm{d}t}\right](t'), \tag{22}$$

where $n(t)$ is the memory size and $p(t,t')$ is the probability of sampling data collected at learning time t' when the weights are updated at time t.

For ER, the experiences are randomly selected for TD-learning, which implies

$$p(t,t') = \frac{1}{n(t)}. \tag{23}$$

Together with Eq. (19) and Eq. (22) we derive

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \frac{\alpha(t)}{n(t)}\int_{t-n(t)}^{t} \delta(t,t')\nabla_\theta Q[x(t'),a(t');\theta(t)]\mathrm{d}t'. \tag{24}$$

For pER, the experiences are sampled according to a probability distribution given in Eq. (6). Under the continuous approximation, the probability is parametrized as

$$p(t,t') = \frac{|\delta(t,t')|^\beta}{\int_{t-n(t)}^{t} |\delta(t,t')|^\beta \mathrm{d}t'}. \tag{25}$$

When $\beta = 2$, combined with Eq. (19) and Eq. (22), we obtain

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \frac{\alpha(t)\int_{t-n(t)}^{t} \delta^3(t,t')\nabla_\theta Q[x(t'),a(t');\theta(t)]\mathrm{d}t'}{\int_{t-n(t)}^{t} \delta^2(t,t')\mathrm{d}t'}. \tag{26}$$

References

[1] V. S. Borkar. *Stochastic approximation: a dynamical systems viewpoint*, volume 48. Springer, 2009.

[2] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[3] S. Kalyanakrishnan and P. Stone. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 94. ACM, 2007.

[4] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[5] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[8] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

[9] J. Peng and R. J. Williams. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[10] X. B. Peng, G. Berseth, and M. van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[13] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

[14] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 528–536. AUAI Press, 2008.

[15] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

[16] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[17] H. Van Seijen and R. S. Sutton. Efficient planning in mdps by small backups. 2013.

[18] P. Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.