The Effects of Memory Replay in Reinforcement Learning

Ruishan Liu [1] James Zou [1]

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.

1. 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 (Lin, 1992).

Combined with deep learning, experience replay has enabled impressive performances in AlphaGo (Silver et al., 2016), Atari games (Mnih et al., 2015), 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 dynamic and performance of the agent. In practice, a memory buffer size is determined by heuristics and then is fixed for the agent.

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" (Moore & Atkeson, 1993)(Schaul et al., 2015). pER is empirical shown to improve the agent's performance compared to the regular ER, but we also lack a good mathematical model of pER.

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 has a non-monotonic effect on learning rate. 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.

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 (Wawrzyński, 2009),

[1] Stanford University, CA, USA. Correspondence to: Ruishan Liu <ruishan@stanford.edu>, James Zou <jamesyzou@gmail.com>.

Presented at the ICML 2017 Workshop on Principled Approaches to Deep Learning, Sydney, Australia, 2017. Copyright 2017 by the author(s).

deep Q-Network (DQN) algorithms (Mnih et al., 2013; 2015), and double Q-learning algorithms (Van Hasselt et al., 2016). To further make good use of experience, prioritized methods are proposed for RL algorithms (Moore & Atkeson, 1993; Peng & Williams, 1993). 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 (Moore & Atkeson, 1993; Van Seijen & Sutton, 2013; Schaul et al., 2015). Measures other than TD-error are also in literature to weight experience; examples include rewards (Tessler et al., 2016) and the transition property (Peng et al., 2016).

The performance of RL with experience replay is similar to the batch RL but in an incremental way (Lagoudakis & Parr, 2003; Kalyanakrishnan & Stone, 2007; Ernst et al., 2005). Another approach to reuse data in RL is called model-learning or Dyna architecture, which builds a model to simulate and generate new data (Sutton, 1990; Sutton et al., 2008). This method, however, induces both extra computation cost and modeling error for the data.

2. 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 $Q(x, a; \theta)$; deep Q-Network (DQN) is an example of this approach. At learning step t, the commonly-used TD-learning method updates the weight θ according to

$$\begin{aligned} \theta(t+1) =& \theta(t) + \alpha(t) \Big[r(t') + \gamma \cdot \max_{a' \in A} Q[x(t'+1), a'; \theta(t)] \\ & - Q\Big[x(t'), a(t'); \theta(t) \Big] \Big] \times \nabla_\theta Q[x(t'), a(t'); \theta(t)], \end{aligned} \tag{2}$$

where $\alpha(t)$ is the step size. Here the data collected at learning step t' is utilized to do the TD update. For standard RL algorithms, only the most recent transition is visited and $t' = t$, while for the ER approach, experience data are reused and $t' < t$.

Algorithm 1 Reinforcement Learning with Experience Replay

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

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.

The ODE model corresponds to a continuous interpolation of the discrete learning step t ($t = 0, 1, 2, ...$). This continuous approximation works well when the step size α is not too large, i.e., there is no dramatical change for the weights within a few steps. This criterion is often met in real experiments. More details of the ODE derivation is in the Appendix.

Under the continuous approximation, the dynamic equation for the weights is

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \alpha(t)\nabla_\theta Q[x(t'), a(t'); \theta(t)]\Big[- Q[x(t'), a(t'); \theta(t)] + r(t') + \gamma \cdot \max_{a' \in A} Q[x(t'+1), a'; \theta(t)]\Big], \tag{3}$$

The agent's state evolution can also be estimated as

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \int_X P\left[x(t), a(t), y\right] y \mathrm{d}y \tag{4}$$

Here the action $a(t)$ is selected according to a policy π

$$a(t) = \pi[x(t)] \tag{5}$$

Together with Eq. (4), the state evolving trajectory can be depicted. For instance, with an $\epsilon-$greedy policy, the state movement obeys

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \int_X y\mathrm{d}y \Bigg[\frac{\epsilon \int_A P\left[x(t), a, y\right] \mathrm{d}a}{\int_A \mathrm{d}a} + (1-\epsilon) P[x(t), \arg\max_{a \in A} Q[x(t), a; \theta(t)]), y]\Bigg] \tag{6}$$

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 parameter dynamics under experience replay is

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \frac{m\alpha(t)}{n(t)} \int_{t-n(t)}^{t} \mathrm{d}t' \nabla_\theta Q[x(t'), a(t'); \theta(t)] \Big[r(t') + \gamma \cdot \max_{a \in A} Q[x(t'+1), a; \theta(t)] - Q[x(t'), a(t'); \theta(t)]\Big] \tag{7}$$

where m is the minibatch size and $n(t) \leqslant N$ is the memory size. Eq. (7) can be viewed as giving the expected gradient value of the parameter updates at each time step.

Now we are able to analyze the learning process, more specifically, the weights $\theta(t)$ and state $x(t)$ as a function of learning step, based on Eq. (4) and Eq. (7). 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 for selecting transition i as

$$P(i) = \frac{|\delta_i|^\beta}{\sum_j |\delta_j|^\beta}, \tag{8}$$

where β is a constant exponent and $\delta_i = r_i + \gamma \max_a Q(s_{i+1}, a; \theta) - Q(s_i, a_i; \theta)$ is the TD-error. 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

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \frac{m\alpha(t) \int_{t-n}^{t} \delta^3(t') \nabla_\theta Q[x(t'), a(t'); \theta(t)] \mathrm{d}t'}{\int_{t-n}^{t} \delta^2(t') \mathrm{d}t'} \tag{9}$$

where $\delta(t') = r(t') + \gamma \cdot \max_{a \in A} Q[x(t'+1), a; \theta(t)] - Q[x(t'), a(t'); \theta(t)]$.

3. Analysis of a Simple Example

Starting from a toy game we call LineSearch, we analytically solve the ODEs (7) and (9) 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. Finally, we show that our theoretical predictions have excellent agreement with experiments.

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., $x(t+1) = x(t) + a(t)$. 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

$$\begin{aligned} Q_{\mathrm{real}}(x, a) &= r(x+a) \\ &= \beta_1 \cdot (x+a) + \beta_2 \end{aligned} \tag{10}$$

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

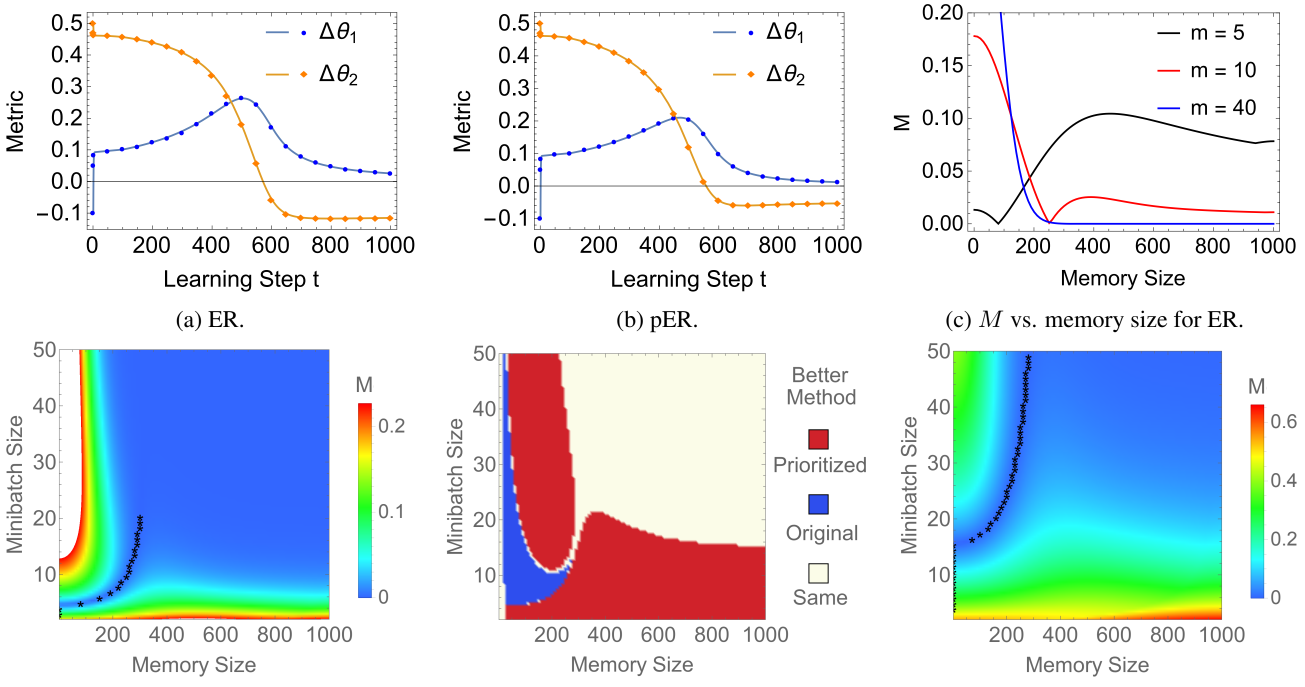
$$Q_{\mathrm{agent}}(x, a; \theta) = \theta_1 \cdot (x+a) + \theta_2, \tag{11}$$

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

$$\Delta\theta_1 = \theta_1 - \beta_1 \text{ and } \Delta\theta_2 = \theta_2 - \beta_2 \tag{12}$$

The agent learns well if $\Delta\theta_1$ and $\Delta\theta_2$ approach 0, and performs badly when $\Delta\theta_1$ or $\Delta\theta_2$ is large. Under a greedy policy, the evolution of the agent's state is

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{\theta_1}{|\theta_1|} v, \tag{13}$$



(a) ER.

(b) pER.

(c) M vs. memory size for ER.

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

(e) Compare ER and pER

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

Figure 1. (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. (c) Dependence of final absolute metric sum $M = |\Delta\theta_1(t = 1000)| + |\Delta\theta_2(t = 1000)|$ on memory size for different minibatch sizes. Note that smaller M stands for better performance. Here we use the original setting when the discount factor $\gamma = 0$; m indicate batch size. (d,f) Contour plot of measure M as a function of memory size and minibatch size, with the discount factor (d) $\gamma = 0$ and (f) $\gamma = 0.5$. The stars denote the optimal memory sizes given minibatch values. The plots in (c) corresponds to the situations in (d) when the minibatch size is 5, 10, and 40. (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).

With the initial state denoted as x_0, the evolution of the two metrics is derived from Eqs. (7)-(13) as

$$\frac{\mathrm{d}\Delta\theta_1(t)}{\mathrm{d}t} = -(b_{10}+b_{11}t+b_{12}t^2)\Delta\theta_1(t)-(b_{20}+b_{21}t)\Delta\theta_2(t) \tag{14}$$

$$\frac{\mathrm{d}\Delta\theta_2(t)}{\mathrm{d}t} = -(b_{20} + b_{21}t)\Delta\theta_1(t) - b_{22}\Delta\theta_2(t), \tag{15}$$

where $b_{10} = m\alpha(n^2v^2/3+x_0^2-nvx_0)$, $b_{11} = m\alpha(2vx_0-nv^2)$, $b_{11} = m\alpha v^2$, $b_{20} = m\alpha(x_0 - nv/2)$, $b_{21} = m\alpha v$, and $b_{22} = m\alpha$. 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., Eq. (14) and Eq. (15). 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 detailed analytical solution and discussion for the ODEs Eq. (14) and Eq. (15) are given in Appendix.

The theoretical learning curves for both RL and pRL settings are depicted in Fig. 1a and Fig. 1b, with the minibatch size m being 5 and the step size α being 0.01. 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. For illustration, we plot the experimental results in Fig. 1a and Fig. 1b, 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.

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

$$\begin{aligned}\Delta\theta_1(t) =& \Delta\theta_1^0 e^{-m\alpha\left[\frac{v^2}{3}t^3+\frac{v(2x_0-Nv)}{2}t^2\right]} \\ &\cdot e^{-m\alpha\left[\left(x_0^2-Nvx_0+\frac{N^2v^2}{3}\right)t-\frac{1}{18}N^2v(Nv-9x_0)\right]},\end{aligned} \tag{16}$$

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

$$\Delta\theta_2(t) = \Delta\theta_2^0 e^{-m\alpha t} \tag{17}$$

In this case, the choice of memory size has no effect on the learning behavior. Detailed explanation and analysis are in Appendix.

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 two metrics absolute values at the learning step 1000, the end of the game. The smaller the measure M is, the better the agent learns. Fig. 1d plots the dependence of M on the memory size N and the minibatch size m, with the step size $\alpha = 10^{-3}$.

The learning performance is affected non-monotonically by the memory size for $m < 20$, while a monotonic relation is observed for $m > 20$, as shown in Fig. 1d. For example, an optimal memory size around 250 exists for $m = 10$, denoted by the red curve in Fig. 1c. In contrast, the measure M experiences a monotonic decrease with the growth of memory size for $m = 40$, plotted by the blue curve in Fig. 1c.

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. 1, θ_1 actually moves further away from β_1 during times 200 to 500; θ_2 also overshoots at $T = 600$ and incurs negative bias. We first address the settings with small minibatches. When the memory size is also 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. With the increase of the memory size, the averaged weight update first becomes slowly then slightly accelerates. The balance between these two contributions leads to the non-monotonic nature. When m 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. Analytical expressions and numerical results are combined for illustrations in Appendix.

Performance of prioritized replay. We further compare pER and ER, and discuss how the memory buffer affects pER based on our theoretical model. Fig. 1b plots the learning curve for pER, which exhibits a similar property as for ER in Fig. 1a . We compare the performance of RL and pRL algorithms in Fig. 1e, 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 minibatch size 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. Demonstrations are given in Appendix.

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

$$\begin{aligned}Q_{\text{real}}(x,a) =& \mathbb{E}\Bigg[\sum_{i=0}^{K}\gamma^i[\beta_1(x_0+a_0+i|\beta_1|v) \\ &\qquad + \beta_2]\Bigg| x_0 = x, a_0 = a\Bigg] \\ =& \frac{1-\gamma^{K+1}}{1-\gamma}[\beta_1(x+a)+\beta_2] \\ &+ v|\beta_1|\left[\frac{\gamma-\gamma^{K+1}}{(1-\gamma)^2} - \frac{K\gamma^{K+1}}{1-\gamma}\right] \\ \approx& \frac{\beta_1}{1-\gamma}(x+a) + \frac{\beta_2}{1-\gamma} + \frac{\gamma v|\beta_1|}{(1-\gamma)^2},\end{aligned} \tag{18}$$

where K stands for the total training steps afterwards before the game ends. The approximation in Eq. (18) 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

$$Q_{\text{agent}}(x,a;\theta) = \theta_1 \cdot (x+a) + \theta_2, \tag{19}$$

Then the evolution of the two weights are derived together with Eqs. (7)-(13) as

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

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

where $b_{10} = m\alpha(n^2v^2/3+x_0^2-nvx_0)$, $b_{11} = m\alpha(2vx_0-nv^2)$, $b_{11} = m\alpha v^2$, $b_{20} = m\alpha(x_0-nv/2)$, $b_{21} = m\alpha v$, and $b_{22} = m\alpha$. The learning curve is attached in Appendix.

We find that the ODEs for $\gamma > 0$ have similar form as $\gamma = 0$. Correspondingly, the results obey similar principles, as shown in Fig. 1f. 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.

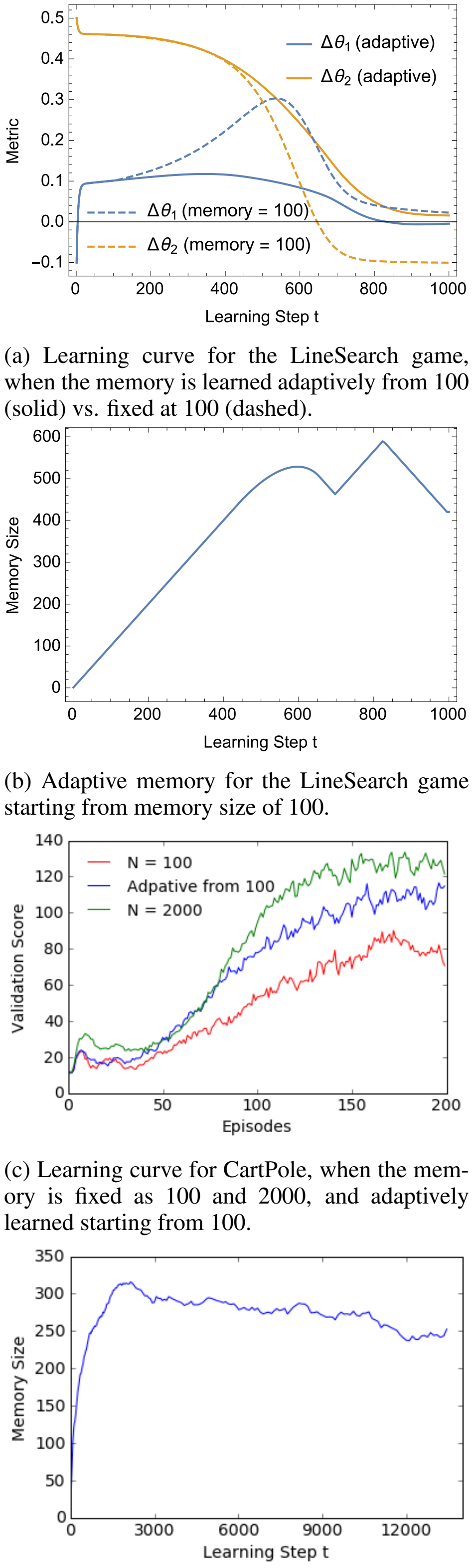
4. Adaptive Memory Size Algorithm

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.

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 old 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 change of 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 1, denotes as aER.

The absolute TD error updates for old transitions $|\delta_{\text{old}}|' - |\delta_{\text{old}}|$ are used as a metric for the balance between the overshooting effect and quick weight update. After k steps of weight updates, if the TD error magnitude for the old transitions goes up, which means the old data violate the Bellman equation even worse, the old data should be kept longer for adequate updates in the future. If $|\delta_{\text{old}}|' < |\delta_{\text{old}}|$, the oldest transitions are abandoned to mitigate possible overshooting effect.

Performance on LineSearch and CartPole. We first analyze how aER works for the LineSearch game. With the minibatch size $m = 10$ and the step size $\alpha = 10^{-3}$, the agent adaptively adjusts its memory capacity from 100 as



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

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

(c) Learning curve for CartPole, when the memory is fixed as 100 and 2000, and adaptively learned starting from 100.

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

Figure 2. Learning curve and adaptive memory for LineSearch and CartPole.

Algorithm 2 Adaptive Memory Size Reinforcement Learning with Experience Replay

```
Input: initial memory size $N_0$, minibatch size m, step size $\alpha$, discount factor $\gamma$, total steps $T$, initial weights $\theta_0$, update policy $\pi_\theta$, number of checked oldest transitions $n_{\text{old}}$ and memory adjustment internal $k$.
Initialize replay memory BUFFER with capacity $N = N_0$ and set $|\delta_{\text{old}}| = 0$
Observe initial state $x_0$
for $t = 1$ to $T$ do
    Take actions, observe, store transitions and do TD updates as in 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
```

depicted in Fig. 2b. 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. 2a.

Next we evaluated the algorithm on CartPole, a standard RL benchmark that we downloaded from OpenAI Gym. The game starts with the pole being upright and the cart locating at the middle of two boundaries. The agent fails and gets a reward -1 if the pole falls more than 40 degrees or the cart hits the walls. The agent earns a reward of 1 if it survives to time 200 in one episode. We used DQN with a one-layer fully connected neural network for the value function approximation. aER achieved good performance in CartPole. Here we randomly initialize the weights and set the minibatch size $m = 50$, number of checked old transitions $n_{\text{old}} = m = 50$, memory adjustment internal $k = 20$, discount factor $\gamma = 0.9$ and step size $\alpha = 2 \times 10^{-5}$. Starting with an initial memory size of 100, the agent speeds up its learning from adaptive adjustments of the memory size, as shown in Fig. 2c and Fig. 2d where each curve is averaged for 100 new games. At the start of the training, the agent learns to increase the memory size for a quicker weight update. In the last 50 episodes, the learning process is accelerated with a slowly shrinking memory, compared with the fixed memory size cases. Although an ultra-large memory like 100000 could do a little bit better at last, the result of adaptive algorithm is already satisfying and also memory-saving with the maximum capacity of around 300.

Evaluation Little extra computation cost is needed to carry out the aER. Taking the cartpole game defined above 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. This rough version of 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.

5. 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 promising improvements over the current static memory size algorithms. 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.

A. Analytic solutions of the ODEs of the learning dynamics

Utilizing our model, we are able to analyze the learning properties and the influence of the replay memory systematically. How to choose replay memory settings, such as memory size and minibatch size, are further discussed.

A.1. Solutions for 1D weights

In this subsection, we start with the simplest case when the weight are one-dimensional to get some basic intuition and to prepare for more complexed cases which will be addressed later.

Fix the intercept θ_2 We first consider the setting when the intercept θ_2 in Eq. (11) is fixed. In this case, the approximated action-value function of the agent Q_{agent} is

$$Q_{\text{agent}}(x, a; \theta) = \theta \cdot (x + a) + \beta_2 \tag{22}$$

Note that the real action-value function is $Q_{\text{real}}(x, a; \theta) = \beta_1 \cdot (x + a) + \beta_2$. The weight is initialized to be θ^0, and the initial metric is $\Delta\theta^0 = \theta^0 - \beta_1$ correspondingly.

The metric as a function of learning step $\Delta\theta(t)$ is calculated analytically as

$$\Delta\theta(t) = \Delta\theta^0 e^{-k(t)}, \tag{23}$$

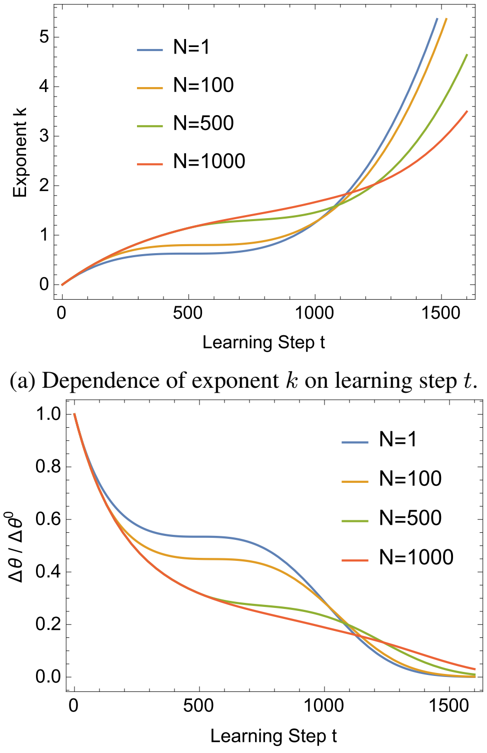
where the exponent $k(t)$ is given by

$$k(t) = m\alpha[\frac{v^2}{9}t^3 + \frac{x_0 v}{2}t^2 + x_0^2 t] \quad (t \leqslant N) \tag{24a}$$

$$k(t) = m\alpha\left[\frac{v^2}{3}t^3 + \frac{v(2x_0 - Nv)}{2}t^2 - \frac{1}{18}N^2 v\left(Nv - 9x_0\right) + \left(x_0^2 - Nvx_0 + \frac{N^2v^2}{3}\right)t\right] \quad (t > N) \tag{24b}$$

The learning process consists of two parts. In the beginning, The exponent k gradually grows from 0, described by Eq. (24a). After the replay memory gets full, the memory buffer becomes a sliding window and the exponent evolves according to Eq. (24b). The metric $\Delta\theta$ approaches the desired value 0 exponentially and the exponent is a cubic function of the learning step t.

In Fig. 3, we present the learning curves of the exponent k and the metric $\Delta\theta$ for different memory sizes N. Here the minibatch size m is 5 and the step size α is 2×10^{-5}. In the whole learning process, the exponent grows monotonically and the metric $\Delta\theta$ decreases monotonically to 0, indicated by Eq. (23).



(a) Dependence of exponent k on learning step t.

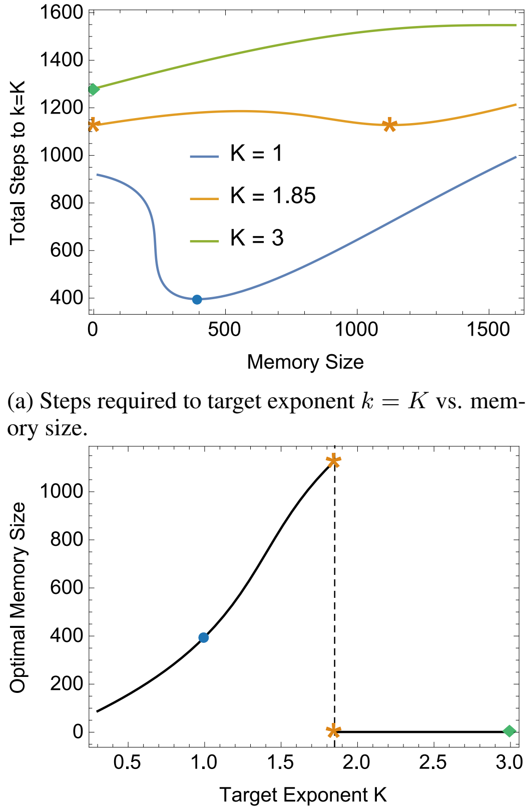
(b) Metric $\Delta\theta/\Delta\theta^0$ vs. learning step t.

Figure 3. Learning curve for the exponents k and the metric $\Delta\theta/\Delta\theta^0 = e^{-k}$. The memory sizes N are 1, 50, 500, 1000.

We further study how the replay memory setting affects the learning performance. In practice, the total training time for an agent to learn a certain amount of knowledge is used to represent its performance. Here the total steps required to reach $k = K$ is chosen to stand for the learning ability of the agent. The value K is our desired exponent value. For instance, if $K = 3$, the agent is thought to learn well when its metric is $\Delta\theta = \theta - \beta_1 = 0.05\Delta\theta^0$. Note that an agent is viewed as a better learner if it uses less training time, i.e., less learning steps, to achieve $k = K$.

First, it is always more beneficial to have a larger minibatch in this case. On the one hand, the exponent k is strictly proportional to the minibatch size m. On the other hand, the exponent monotonically increases during the whole learning process. Thus, the larger minibatch an agent has, the faster it learns. It should be mentioned that in real experiments, the minibatch cannot be too large, cause one gradient step size is required to be small to guarantee the validity of TD-error update, and the gradient step size is proportional to the minibatch size in the defined ER algorithm.

Second, the learning ability of the agent, represented by the total steps an agent takes from the initial exponent to $k = K$, has nonmonotonic dependence on the memory size, as plotted by Fig. 4a. As the learning proceeds, the weight update from the currently collected transition, i.e., $\alpha\delta_i\Delta_\theta Q(x_i, a_i; \theta)$, first decreases, and then grows after the



(a) Steps required to target exponent $k = K$ vs. memory size.

(b) Optimal meomroy size vs. target exponent

Figure 4. How memory size affects performance in fixed intercept case.

moment when the state increases to be above 0. In the same time, for any transition in this setting, the weight update is always in the correct direction towards the real value β_1. That is, a larger weight update is always preferable.

In the beginning of the game, older transitions in the memory replay contribute more to the weight update and a larger memory is more desirable. As soon as the agent reaches the region where the weight update from its current state is large enough, compared to the average of its past experience, only updating the current transition becomes a good strategy. As illustrated by the blue curve in Fig. 3, for memory size $N = 1$, it grows most slowly at start, the velocity of its exponent increases gradually surpasses other conditions with different memory size, and grows most rapidly among all possible choices of N after learning step $t = 1000$. Thus, the optimal memory size increases with the growth of the target exponent K, abruptly falls to 1 at a certain K,and remains to be 1 thereafter, as demonstrated in Fig. 4b.

The prioritized method always outperforms the uniformly selection approach in the fixed-intercept case. It is due to the fact that, a larger absolute TD-error corresponds to a larger derivative of the action-value function with respect to the weight, and furthermore it corresponds to a larger weight update. As mentioned above, a larger weight update is always beneficial. In the prioritized setting, transitions with larger TD-error value are more frequently selected, leading to a faster convergence to real weight value. It can also be demonstrated analytically as the difference between the exponent for the prioritized setting $k_{\mathrm{pri}}(t)$ and for the original one $k(t)$ is written as

$$k_{\mathrm{pri}}(t) - k(t) = c(t)v^3t^3\left[\left(2vt + \frac{15}{4}x_0\right)^2 + \frac{15}{16}x_0^2\right] \quad (t \leqslant N) \tag{25}$$

$$k_{\mathrm{pri}}(t) - k(t) = c'(t)v^3N^3\left[\left(2vt - \frac{15}{4}(vt + x_0)\right)^2 + \frac{15}{16}(vt + x_0)^2\right] \quad (t > N), \tag{26}$$

where $c(t) > 0$ and $c'(t) > 0$. It can be easily observed that $k_{\mathrm{pri}}(t) - k(t) > 0$ at any learning step t, thus prioritized learning converges more quickly than the original setting, as illustrated by Fig. 5.

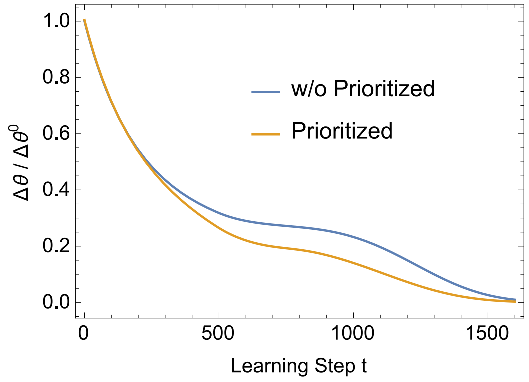


Figure 5. The learning curve for the metric $\Delta\theta$ under conditions with prioritized method (orange curve) and without prioritized method (blue curve) in fixed intercept case. Here the memory size is 500, batch size is 5 for illustration purpose.

Fix the slope θ_1 Another possible one-dimensional action-value function weight is to fix the slope θ_1 in Eq. (11), which means

$$Q_{\mathrm{agent}}(x, a; \theta) = \beta_1 \cdot (x + a) + \theta \tag{27}$$

With the initial weight θ^0, the initial metric is given by $\Delta\theta^0 = \theta^0 - \beta_2$, considering the real action-value function is $Q_{\mathrm{real}}(x, a; \theta) = \beta_1 \cdot (x + a) + \beta_2$.

The metric $\Delta\theta(t) = \theta(t) - \beta_2$ is obtained analytically as

$$\Delta\theta(t) = \Delta\theta^0 e^{-m\alpha t} \tag{28}$$

Equation (28) suggests that the metric also decreases to the target value 0 exponentially, but the exponent is proportional to the learning step t. In strong contrast, the exponent is third-order polynomial with respect to t in the fixed-intercept case. Thus, the learning is generally slower for this fixed-slope case than the fixed-intercept case. For example, when the minibatch size is 5 and the step size is 5×10^{-5}, it takes the fixed-slope agent 30000 learning steps to achieve $\Delta\theta/\Delta\theta^0 = 0.05$, while the fixed-intercept agent only needs less than 1200 steps to learn.

The learning process and results are totally independent of the initial state, velocity of state changing, and most importantly, the memory size. The learning dynamics is fully described by the step size, the minibatch size and the initial weights. This is due to the fact that all transitions are identically useful in fixed-slope situation. At any learning step, it can be easily proved that the TD-error and the weight update are the same for all transition with all possible state values. The selection of data for update no longer matters, so different replay memory settings have same performance.

Similarly, the prioritized method has the same learning results as the original setting, cause all transitions are equal in the sense of weight update. This can be confirmed by theoretical calculation, which yields that the exponent for the prioritized setting $k_{\text{pri}}(t)$ and the original one $k(t)$ satisfy

$$k_{\text{pri}}(t) \equiv k(t). \tag{29}$$

A.2. Solution for the full model

From Eq. (14) and Eq. (15), the dynamic equation for the metric $\Delta\theta_1(t)$ is given by

$$\begin{aligned}0 =& \frac{\mathrm{d}^2\Delta\theta_1(t)}{\mathrm{d}t^2} + \frac{c_{10}+c_{11}t+c_{12}t^2+c_{13}t^3}{d_0+d_1t}\frac{\mathrm{d}\Delta\theta_1(t)}{\mathrm{d}t}\\ &+ \frac{c_{00}+c_{01}t+c_{02}t^2+c_{03}t^3}{d_0+d_1t}\Delta\theta_1(t) \qquad (t \leqslant N)\end{aligned} \tag{30a}$$

$$\Delta\theta_1(t=0) = \theta_1^0 - \beta_1 \tag{30b}$$

$$\frac{\mathrm{d}\Delta\theta_1(t=0)}{\mathrm{d}t} = -m\alpha x_0^2(\theta_1^0 - \beta_1) - m\alpha x_0(\theta_2^0 - \beta_2) \tag{30c}$$

$$\begin{aligned}0 =& \frac{\mathrm{d}^2\Delta\theta_1(t)}{\mathrm{d}t^2} + \frac{g_{10}+g_{11}t+g_{12}t^2+g_{13}t^3}{h_0+h_1t}\frac{\mathrm{d}\Delta\theta_1(t)}{\mathrm{d}t}\\ &+ \frac{g_{00}+g_{01}t+g_{02}t^2}{h_0+h_1t}\Delta\theta_1(t) \qquad (t \geqslant N)\end{aligned} \tag{31a}$$

$$\Delta\theta_1(t=N^+) = \Delta\theta_1(t=N^-) \tag{31b}$$

$$\frac{\mathrm{d}\Delta\theta_1(t=N^+)}{\mathrm{d}t} = \frac{\mathrm{d}\Delta\theta_1(t=N^-)}{\mathrm{d}t} \tag{31c}$$

After $\Delta\theta_1(t)$ is obtained, another metric $\Delta\theta_2(t)$ can be derived based on Eq. (14). Here the parameters are

$$c_{00} = 12\alpha m v x_0^2 \tag{32a}$$

$$c_{01} = 6\alpha m v^2 x_0 \tag{32b}$$

$$c_{02} = 2\alpha^2 m^2 v^2 x_0 + 4\alpha m v^3 \tag{32c}$$

$$c_{03} = \alpha^2 m^2 v^3 \tag{32d}$$

$$c_{10} = 24\alpha m x_0^3 + 24\alpha m x_0 - 12v \tag{32e}$$

$$c_{11} = 12\alpha m v + 36\alpha m v x_0^2 \tag{32f}$$

$$c_{12} = 20\alpha m v^2 x_0 \tag{32g}$$

$$c_{13} = 4\alpha m v^3 \tag{32h}$$

$$d_0 = 24x_0 \tag{32i}$$

$$d_1 = 24v \tag{32j}$$

$$\begin{aligned}g_{00} =& \alpha m v\Big[N^2v^2(4-\alpha m N) + 2Nvx_0(\alpha m N - 12)\\ &+ 24x_0^2\Big]\end{aligned} \tag{33a}$$

$$g_{01} = 2\alpha m v^2\left[Nv(\alpha m N - 12) + 24x_0\right] \tag{33b}$$

$$g_{02} = 24\alpha m v^3 \tag{33c}$$

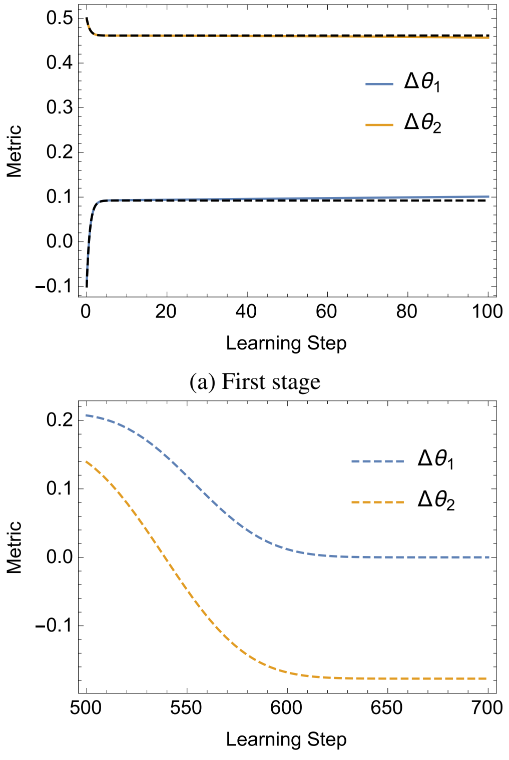
$$\begin{aligned} g_{10} =& 4\Big[v\left(\alpha m N\left(N^2v^2+3\right)+6\right)-\alpha m x_0\left(5N^2v^2+6\right) \\ &+9\alpha m N v x_0^2-6\alpha m x_0^3\Big] \end{aligned} \tag{33d}$$

$$g_{11} = -4\alpha m v\left(5N^2v^2-18Nvx_0+18x_0^2+6\right) \tag{33e}$$

$$g_{12} = 36\alpha m v^2\left(Nv-2x_0\right) \tag{33f}$$

$$g_{13} = 24\alpha m v^3 \tag{33g}$$

$$h_0 = 24x_0-12Nv \tag{33h}$$



(a) First stage

(b) Last stage

Figure 6. Approximated analytical solutions for two metrics. The dashed curves are analytical results from (a) Eq. (34) and (b) Eq. (36). (a) The solid curves represent the numerical solution.

Approximated analytical solutions can be derived. For Fig. 1a, the approximated learning functions in the first and last stage are plotted in Fig. 6.

In the beginning part, the metrics are estimated as

$$\begin{aligned} \Delta\theta_1(t) =& \frac{\Delta\theta_1^0+\Delta\theta_1^0x_0^2e^{\alpha m t\left(-x_0^2-1\right)}-\Delta\theta_2^0x_0}{x_0^2+1} \\ &+\frac{\Delta\theta_2^0x_0e^{\alpha m t\left(-x_0^2-1\right)}}{x_0^2+1} \end{aligned} \tag{34a}$$

$$\begin{aligned} \Delta\theta_2(t) =& \frac{\Delta\theta_1^0x_0e^{\alpha m t\left(-x_0^2-1\right)}+\Delta\theta_2^0e^{\alpha m t\left(-x_0^2-1\right)}}{x_0^2+1} \\ &+\frac{\Delta\theta_2^0x_0^2-\Delta\theta_1^0x_0}{x_0^2+1} \end{aligned} \tag{34b}$$

As illustrated by the dashed curve in Fig. 6a, this estimation fits the real solution well when the learning step is less than 100.

Equation (34) indicates that the weights changes rapidly to

$$\Delta\theta_1(t) \to \Delta\theta_1^0-\frac{x_0\left(\Delta\theta_2^0+\Delta\theta_1^0x_0\right)}{x_0^2+1} \tag{35a}$$

$$\Delta\theta_2(t) \to \Delta\theta_2^0-\frac{\Delta\theta_2^0+\Delta\theta_1^0x_0}{x_0^2+1} \tag{35b}$$

After the swift change, the metrics remain constant for a short period, as shown in Fig. 6a. Note that the value changing behavior happens so fast that the values given in Eq. (35) do not rely on the memory size, policy, minibatch size and even the step size.

In the last stage of the learning, the metrics are approximated as

$$\Delta\theta_1(t) = \Delta\theta_1^1e^{-\frac{1}{3}\alpha m t^3v^2} \tag{36a}$$

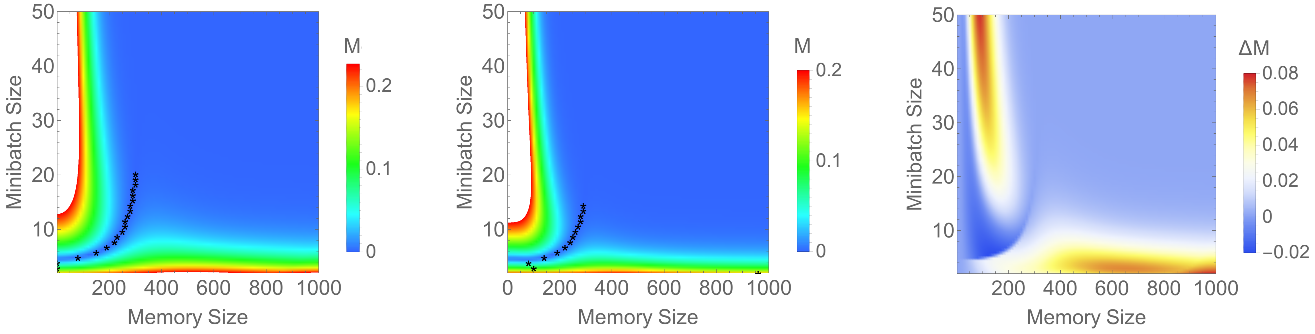
$$\begin{aligned} \Delta\theta_2(t) =& \frac{3^{2/3}\Delta\theta_1^1\sqrt[3]{\alpha m v^2}\Gamma\left(\frac{2}{3},\frac{1}{3}mt^3v^2\alpha\right)3\Delta\theta_2^1v}{3v} \\ &-\frac{3^{2/3}\Delta\theta_1^1\Gamma\left(\frac{2}{3}\right)\sqrt[3]{\alpha m v^2}}{3v}, \end{aligned} \tag{36b}$$

where $\Delta\theta_1^1$ and $\Delta\theta_2^1$ are estimated values for $\Delta\theta_1$ and $\Delta\theta_2$ when the agent enters the last stage. The learning curve in the last stage is illustrated in Fig. 6b. Here the guess for $\Delta\theta_1^1$ and $\Delta\theta_2^1$ can vary a lot. In the rigorous result, we observe a delayed effect, which is due to the fact that the agent does not fully enter the last stage in our experiment time scale and the neglected terms for the derivation of Eq. (36) also contribute to the result.

From Eq. (36) we observe that the weights finally approach

$$\Delta\theta_1(t) \to 0 \tag{37a}$$

$$\Delta\theta_2(t) \to \frac{3\Delta\theta_2^1v-3^{2/3}\Delta\theta_1^1\Gamma\left(\frac{2}{3}\right)\sqrt[3]{\alpha m v^2}}{3v} \tag{37b}$$



(a) ER.

(b) pER.

(c) Compare ER and pER.

Figure 7. Contour plot of final absolute metric sum $M = |\Delta\theta_1(t = 1000)| + |\Delta\theta_2(t = 1000)|$ as a function of memory size and minibatch size for (a) ER and (b) pER algorithms. Smaller final total metric stands for better performance. The stars denote the optimal memory sizes given minibatch values. (c) The contour plot of the difference of the final absolute metric sum M for the original setting and the prioritized setting as a function of memory size and minibatch size, i.e., the result of subtracting Fig. 1d by Fig. 7b. The positive value represents that the prioritized method is useful, while the negative value denotes that the prioritized method is harmful

It should be addressed that the metric $\Delta\theta_2$ does not converge to 0 because the agent changes the state unidirectionally and the movement finally reaches a balance with the weight update. Thus, in order to make the metric as near 0 as possible at last, it is crucial for the agent to have small value of Eq. (37b) when it enters the final stage of the learning process.

Here we illustrate in details how the memory setting affects the learning performance from the trade-off between the overshooting and the weight update. When the memory size is small, overshooting is more likely to take place owing to the limited memory size. As depicted by the solid curves in Fig. 8b when the memory size is 100, the metric $\Delta\theta_2$ is gradually fitted from positive to negative, while $\Delta\theta_1$ remains to be positive after the first stage. From Eq. (37b) we know that, in this case, the absolute value which $\Delta\theta_2$ approaches is large, confirmed by Fig. 8b. With the growth of memory size, the overshooting effect is mitigated. In this case, the weight update is first decelerated and then slightly accelerates. The optimal memory size is reached around 250, as depicted by Fig. 8c. When the memory size continues to increase, the agent suffers little overshooting issue. The convergence is slower than the optimal one because of the smaller weight update, as shown in Fig. 8d.

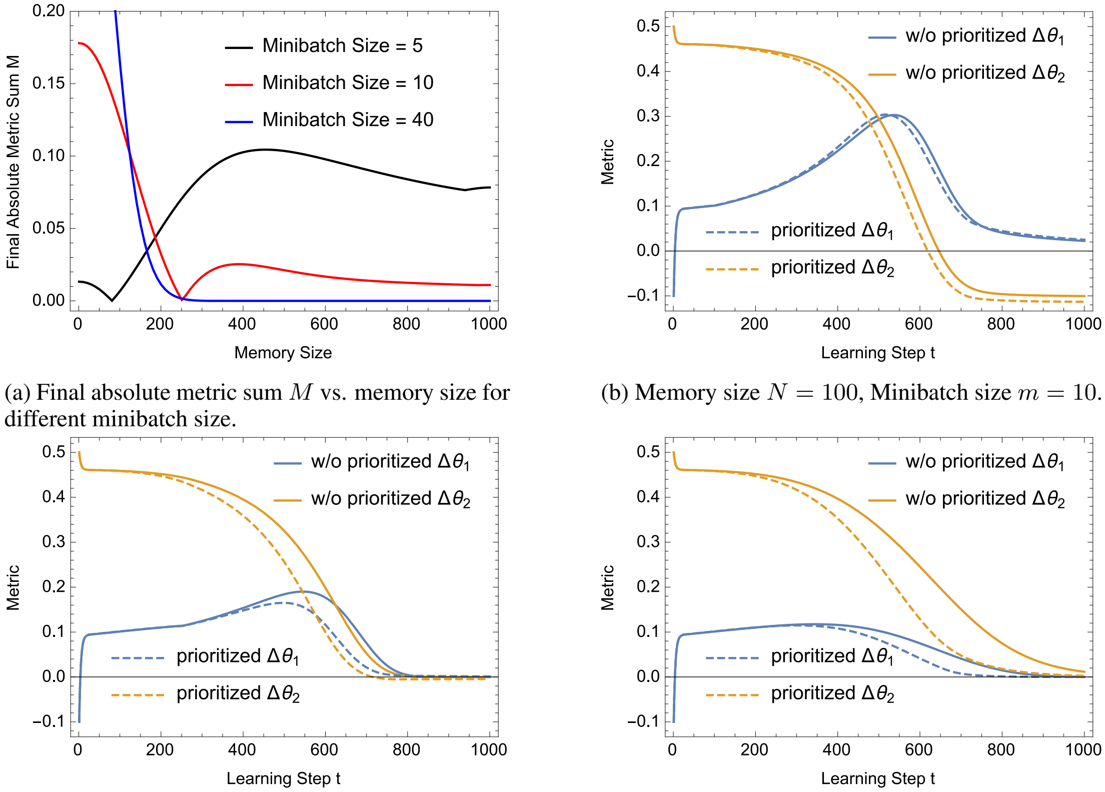
B. Effects of memory size in prioritized replay

Here we analyze the effects of the memory replay on the prioritized methods. Fig. 7b presents the dependence of the final absolute metric sum M on memory size and minibatch size, which exhibits a similar behavior as in the original setting and the causes are also similar. M has a nonmonotonic dependence on memory given the batch size less than 15, and decreases monotonically with the increase of memory. Fig. 7c depicts how the difference of M between the original and prioritized settings depend on memory size and minibatch size. A positive difference value means that the prioritized setting is better. Fig. 1e is derived from it.

In principle, the pER algorithm is found to perform relatively worse than the ER when both the memory size and minibatch size is small, as indicated by Fig. 7c. This could also be explained with the trade-off between the overshooting and quick weight update, similar to the situation in Fig. 8a. As plotted in Fig. 8b and Fig. 8d, for the memory size N of 100, the prioritized setting makes the overshooting even worse; while for $N = 1000$, the weights are quickly updated and the prioritized agents converge faster. It should be noted that there are more complicated situations in this two-dimensional-weight situation. For $\Delta\theta_2$, prioritized scheme always results in a faster weight update, while for $\Delta\theta_1$, this does not necessarily hold true according to the definition of the weight update. For example, in Fig. 8c, before the learning step 500, the non-prioritized case actually learns faster than the prioritized case.

C. Nonzero discount factor

The theoretical model when $\gamma = 0$ also fits well with the experiments' result, as illustrated in Fig. 1, where the minibatch size m is 5, the step size α is 0.01, the discount factor γ is 0.5, the real weights β_1 and β_2 are 0.1 and 0.5, and the initial weights θ_1^0 and θ_2^0 are 0 and 1.



(a) Final absolute metric sum M vs. memory size for different minibatch size.

(b) Memory size $N = 100$, Minibatch size $m = 10$.

(c) Memory size $N = 250$, Minibatch size $m = 10$.

(d) Memory size $N = 1000$, Minibatch size $m = 10$.

Figure 8. (b-d) Learning curve for the two metrics $\Delta\theta_1$ and $\Delta\theta_2$ with or without prioritized methods.

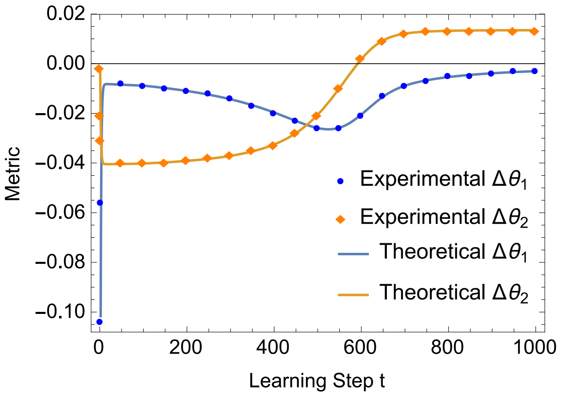


Figure 9. Learning curve for two metrics $\Delta\theta_1 = \theta_1 - \theta_1^r$ and $\Delta\theta_2 = \theta_2 - \theta_2^r$, where $\theta_{1(2)}$ and $\theta_{1(2)}^r$ are the weights of the agent and the real weights, respectively. The scattered blue dots and orange squares represent the experimental results for $\Delta\theta_1$ and $\Delta\theta_2$, based on the ER algorithm. The blue and orange curve are the theoretical solutions for $\Delta\theta_1$ and $\Delta\theta_2$.

References

Ernst, Damien, Geurts, Pierre, and Wehenkel, Louis. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Kalyanakrishnan, Shivaram and Stone, Peter. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pp. 94. ACM, 2007.

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

Lin, Long-Ji. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Moore, Andrew W and Atkeson, Christopher G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

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

Peng, Xue Bin, Berseth, Glen, and van de Panne, Michiel. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

Sutton, Richard S, Szepesvári, Csaba, Geramifard, Alborz, and Bowling, Michael. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 528–536. AUAI Press, 2008.

Tessler, Chen, Givony, Shahar, Zahavy, Tom, Mankowitz, Daniel J, and Mannor, Shie. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Van Seijen, Harm and Sutton, Richard S. Efficient planning in mdps by small backups. 2013.

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