2019 IEEE International Workshop on Signal Processing Systems

Memory Reduction through Experience Classification for Deep Reinforcement Learning with Prioritized Experience Replay

Kai-Huan Shen* and Pei-Yun Tsai†
Department of Electrical Engineering
National Central University, Jhong-Li, 32001, Taiwan.
*shovockhs@gmail.com, †pytsai@ee.ncu.edu.tw

***Abstract*—Prioritized experience replay has been widely used in many online reinforcement learning algorithms, providing high efficiency in exploiting past experiences. However, a large replay buffer consumes system storage significantly. Thus, in this paper, a segmentation and classification scheme is proposed. The distribution of temporal-difference errors (TD errors) is first segmented. The experience for network training is classified according to its updated TD error. Then, a swap mechanism for similar experiences is implemented to change the lifetimes of experiences in the replay buffer. The proposed scheme is incorporated in the Deep Deterministic Policy Gradient (DDPG) algorithm, and the Inverted Pendulum and Inverted Double Pendulum tasks are used for verification. From the experiments, our proposed mechanism can effectively remove the buffer redundancy and further reduce the correlation of experiences in the replay buffer. Thus, better learning performance with reduced memory size is achieved at the cost of additional computations of updated TD errors.**

***Index Terms*--Reinforcement learning (RL), deep determinis-tic policy gradient (DDPG), prioritized experience replay (PER).**

I. Introduction

Due to the explosive growth of computation power and massive parallelism offered by GPU, reinforcement learning (RL) techniques have once again gained popularity. The Deep Q-Network (DQN) algorithm [1], which combines neural network and Q-learning, creates the possibility of deep reinforcement learning [2]. With neural networks, we can extract high-level features from raw sensory data and enable applications such as computer vision, natural language processing, and other detection as well as recognition technologies. However, for training DQN, uncorrelated data are preferred and required in many stochastic gradient-based algorithms. To support an online learning agent that receives a stream of highly-correlated state transitions, experience replay, first proposed in [3], is essential. With the experience replay technique, the temporal correlations could be broken by mixing together the more and less recent experiences, and the past experiences could be possibly used multiple times. These properties expedite and stabilize the training of DQN [1].

As the complexity of the scenario that deploys reinforcement learning agents gets higher, the online learning algorithm requires a larger replay buffer. How to use the experiences in the replay buffer efficiently becomes a critical issue. Studies [1] and [4] addressed that uniform sampling method does not differentiate important experiences and the past experiences may be overwritten due to the limitation of the finite memory size. Researchers in [1] and [4] also suggested developing a method similar to prioritized sweeping [5] to promote important experiences. Therefore, during the learning process, the agent is encouraged to learn from successful attempts and failed tumbles instead of learning from uniformly sampled experiences in the replay buffer. The priority of the experience in the replay buffer is given according to their absolute temporal-difference errors (TD errors). In addition, the highest priority is assigned to the latest transition. The experiences with high priority are replayed more frequently to strengthen their effects for learning. The experiment of prioritized experience replay (PER) incorporated with double DQN in the Atari environment [6] outperformed the previous results significantly.

Although the highest priority is given, PER does not guarantee the immediate replay of the latest experience. In addition, the hyperparameter of the buffer size must be carefully selected and is task-dependent [7]. To ensure the agent obtaining the up-to-date information, combined experience replay (CER) which always includes the latest experience in the mini-batch has been proposed [7]. Given a sufficiently large buffer, PER or CER aims to utilize the experiences in the replay buffer efficiently. However, Schaul *et al.* further pointed out in [6] that the memory size could be reduced if an explicit control mechanism can be developed to drop the well-learned experiences and to reserve those with high errors.

In light of the above, in this paper, we propose a classification mechanism (CLA) to achieve the goal of buffer size reduction. To verify the proposed scheme, it is combined with the continuous control algorithm – Deep Deterministic Policy Gradient (DDPG) [8], which is an extension of DQN. DDPG integrates the conventional policy gradient approach [9] with the actor-critic algorithm [10] and shows its distinct performance in various continuous control problems. The proposed PER plus CLA scheme has the following features.

- The distribution of TD errors is partitioned into different segments. The experiences with TD errors belonging to the same segment are regarded as similar importance. The classifier is also updated periodically to prevent from using outdated TD error distribution.

978-1-7281-1927-4/19/$31.00 ©2019 IEEE

- It changes the regular lifetime of the experiences in the replay buffer by the swap mechanism. The experiences with higher TD errors have higher probability to be swapped due to the properties of PER and the classifier.
- By further reducing the correlation of the experiences in the replay buffer, the buffer size can be reduced.

From the experimental results, we demonstrate that incorporating PER plus CLA, the DDPG algorithm can learn in a more stable way within a shorter training period than that with only PER [11] given a smaller replay buffer size.

The rest of this paper is organized as follows. We illustrate DDPG, PER and the proposed CLA algorithms in Section II. The simulation results and comparisons are given in Section III. The paper is summarized in Section IV.

II. Prioritized Experience Replay with Classification Mechanism in DDPG

Unlike DQN, DDPG can generate continuous and deterministic action according to the state information and thus it can be applied in the robot control applications. In the following, integration of PER and CLA in DDPG will be illustrated sequentially.

A. DDPG with Prioritized Experience Replay

Reinforcement learning is commonly modeled as a Markov Decision Process (MDP), which is described by the state space, $\mathcal{S}$, the action space $\mathcal{A}$, and the real-value reward. A policy function $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$ that guides the behavior of the agent maps the states to the probability distribution over the actions. The reward function is defined by $R_s^a : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, where $s \in \mathcal{S}$, and $a \in \mathcal{A}$. The goal of the agent is to learn a policy maximizing the long-term returns. The action-value function describing the expected return under a given policy starting from the state s_t with action a_t takes the form of

$$Q_\pi(s_t, a_t) = \mathbb{E}[\sum_{i=t}^{T} \gamma^{i-t} R_{s_i}^{a_i} | s_t, a_t], \tag{1}$$

where t is the time step. With the Bellman equation and the deterministic policy $\mu : \mathcal{S} \rightarrow \mathcal{A}$, the optimal action-value function is expressed as

$$Q(s_t, a_t) = \mathbb{E}[R_{s_t}^{a_t} + \gamma Q(s_{t+1}, \mu(s_{t+1})]. \tag{2}$$

Based on the concept of deterministic policy gradient [12], two kinds of networks are constructed to accommodate the operations to the continuous state-action space [8]. The critic network learns the action-value function $Q(s, a|\theta^Q)$ while the actor network learns the deterministic actor policy $\mu(s|\theta^\mu)$, where θ^Q and θ^μ are the network parameters [12]. Similar to DQN, the target network and evaluation network are employed for both the actor and the critic. The target networks $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ are first initialized by $\theta^{Q'}$=θ^Q and $\theta^{\mu'}$=θ^μ and are updated in a soft manner during the training. The evaluation network is used to predict the long-term return. The critic evaluation network is trained for minimizing the loss function $L(\theta^Q)$ defined as the mean of TD errors in a mini-batch of size K,

$$\delta_i = (\mathcal{R}_{s_i}^{a_i} + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) - Q(s_i, a_i|\theta^Q))^2, \tag{3}$$

$$L(\theta^Q) = \frac{1}{K} \sum_i \delta_i, \tag{4}$$

and the policy gradient for the actor evaluation network is

$$\nabla_{\theta^\mu} J \approx \frac{1}{K} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}. \tag{5}$$

Prioritized experience replay introduces priority of experience j as p_j, for $j = 1, 2, \cdots, N$, which can be given by [6]

$$p_j = |\delta_j| + \epsilon \quad \text{or} \quad \frac{1}{\text{rank}(j)}, \tag{6}$$

where ϵ is a small positive constant to prevent p_j from being zero. The probability of sampling experience j then becomes

$$P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}, \tag{7}$$

where the exponent α adjusts the degree of the prioritization.

To avoid the bias introduced by the prioritization, importance-sampling (IS) weight is used for compensation, which is expressed as [6]

$$w_j = \left(\frac{1}{N} \cdot \frac{1}{P(j)}\right)^\beta, \tag{8}$$

where N is the buffer size. The exponent β corresponds to the degree of correction. The weight is further normalized by $1/(\max_j w_j)$ for stability reason and $w_j \delta_j$ is used in the loss function in (4) instead of δ_j.

B. Classification Scheme

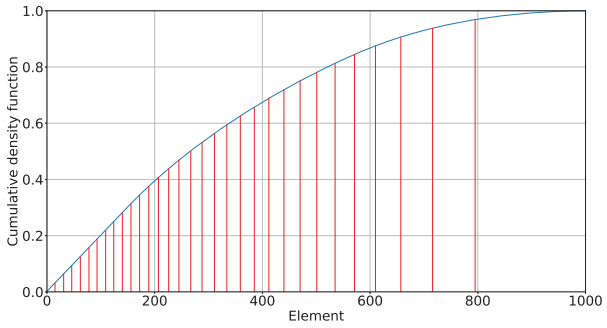


Figure 1. Cumulative density function of prioritized experience sampling probability in the replay buffer of size 10^3.

Let the priorities of the experiences in the replay buffer be sorted in descending order. Fig. 1 shows the cumulative density function (CDF) of the prioritized sampling probability versus the sorted elements according to their probability $P(j)$, where the replay buffer N is set to 10^3. Given the mini-batch size K equal to 32, the entire CDF is partitioned into 32 regions and their boundaries are indicated by the vertical red lines in the figure. One experience in each region is sampled for training. It is clear that the smaller the region is, the higher the sampling probability for the experiences inside becomes. Therefore, the high sampling probability can be assigned to those experiences with large TD errors because they are allocated in the small region.

However, the experiences are stored in the replay buffer in a circular manner in the conventional prioritized experience replay algorithm. The fair replacement strategy leads to an equal lifetime of N steps of these experiences. Besides, the correlation of the replayed experiences is also influenced by the buffer size. Consequently, if a proper control mechanism can be developed to change the lifetime of the experiences and break the correlation further, the redundancy of similar experiences can be removed and the buffer size can be reduced.

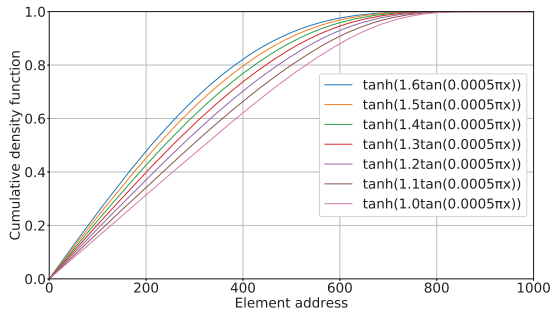


Figure 2. Function approximation for cumulative density function for memory size 10^3.

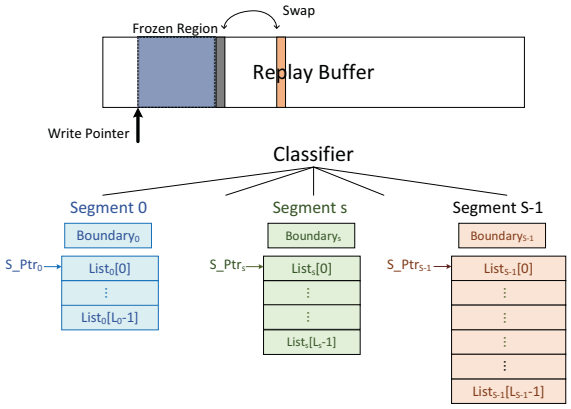


Figure 3. Segmentation of the replay buffer content.

Extended from the concept of PER, which partitions the CDF into K regions, a classifier i s d esigned. T he T D error distribution is classified i nto S s egments. I n o rder t o differentiate the replacement probability of the experiences with smaller *TD errors* (priority) and larger *TD errors*, as shown in Fig. 2, the hyperbolic tangent function is used for imbalanced segmentation,

$$\mathcal{F}(i) = \tanh(C\tan(\frac{0.5\pi}{N}i)), \tag{9}$$

where N is the replay buffer size. Fig. 2 shows the curve of $\mathcal{F}(i)$ with different parameter C for $N = 1000$. Note that in contrast to PER, the TD errors are sorted in ascending order in the classifier. Then, segment s contains the sorted elements lying in the CDF range of $[s/S, (s+1)/S)$. Thus, more elements are classified into segments corresponding to large TD errors and fewer elements are allocated in the segments associated with small TD errors. Experiences in the same segments are regarded as similar because of their similar TD errors. As shown in Fig. 3, the buffer addresses of the elements belonging to the segment s are saved in a list specified by the list size L_s. Besides, the lower boundary of TD errors LB_s of segment s is also recorded. Initially, the segment pointer, S_Ptr_s, points to the address corresponding to the oldest experience.

The DDPG algorithm with prioritized experience replay and the proposed classification mechanism is given in Algorithm 1. The segmentation procedure is described in Algorithm 2 as well as the classification and swap procedure in Algorithm 3. At first, the agent warms up as usual. Thus, line 1 to line 11 are the same as the conventional DDPG [11].

Every time when the memory is full or N experiences are written in, the segmentation procedure is activated, which is indicated in line 12. For PER mini-batch calculation, K experiences are fetched separately from the buffer and thus we can acquire the TD error of that experience with the up-to-date network parameters as shown in line 13 to line 18. In line 17, the TD error is then classified so that its segment index s can be derived. Details of the function are stated in Algorithm 3.

To change the life time of that experiences in the replay buffer, it replaces another similar experience, which is selected from the same segment according to the pointer S_Ptr_s. Thus, we swap the positions of two experiences in the replay buffer D. Note that a frozen region with length $N/5$ is set in the replay buffer. The content in the frozen region is not swapped and is going to be overwritten. The range is specified from the current write pointer κ. If either of the experiences is located in the frozen region, the swap is not allowed. In the extreme case, if the frozen region is equal to the buffer size N, then the lifetimes of all the experiences become N, equal to the conventional PER scheme. If no frozen region exists, there could be some experiences lasting too long without being overwritten due to the swap mechanism. Hence, with proper setting for the size of frozen region, each experience will fade out eventually.

The frozen region shifts when a new experience is acquired from the environment. Hence, if an experience is going to enter the frozen region, the following two conditions can change it owing to swapping.

- First, it is selected by PER for the mini-batch gradient calculation. Because PER tends to pick up the experiences with high TD errors, those high-priority experiences can be preserved with a higher probability due to the swap. Another old experience belonging to the same segment derived from the classification procedure will be in place of it.
- Secondly, it is pointed by S_Ptr_s and the updated TD error belongs to the same segment s. In this case, the lifetime of the new experience is shortened.

If an experience is swapped several times, its lifetime in the buffer is lengthened effectively. The probability of experiences being swapped is about $2K/N$. Fig. 4 shows the swap cause of the experience next to the frozen region with higher or lower

TD errors according to the information from the classifier. As shown in the figure, the swap probabilities is close to the aforementioned value for $N = 1200$. Also, the experiences with higher TD errors has high probability to swap.

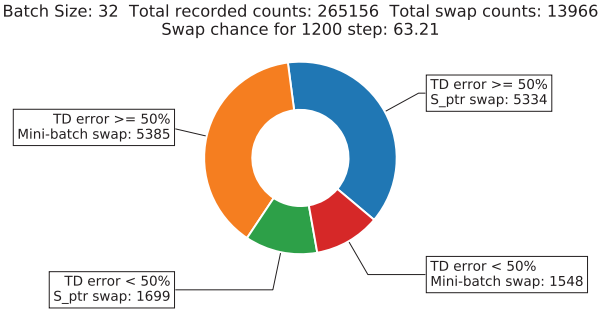


Figure 4. Swap cause of the experience next to the frozen region with $N = 1200$ and $K = 32$ in Inverted Double Pendulum task.

Fig. 5 shows the change of the lifetime of the experiences in the replay buffer. The lifetime of the element from being inserted in the buffer to being replaced by a new experience is drawn. If a conventional buffer access scheme is adopted, all the experiences have an equal lifetime of $N(= 669)$ steps. With the proposed CLA mechanism, the lifetime varies and even can be lengthened to several thousand steps. It also implies that the experiences with more than N steps apart can coexist in the buffer and the correlation of the experiences in the replay buffer is significantly reduced, too. The average lifetime of the most recent N steps is also depicted as the blue curve. In Fig. 5, the exploration is ceased around step 25283 together with the CLA scheme. We can see the lifetime of all the experiences gradually become N steps.

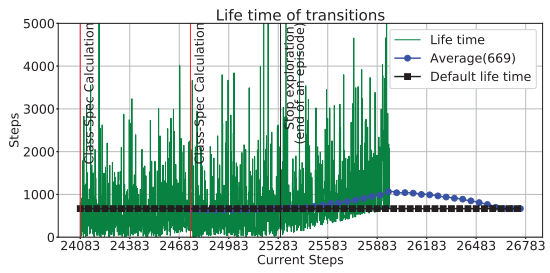


Figure 5. Life time of the experiences.

III. Simulations and Comparisons

A. Simulation Setup

The simulations are carried out based on OpenAI Gym [13] wrapped MuJoCo [14]. The Inverted Pendulum task and Inverted Double Pendulum task, which are classical continuous control problems, are adopted. In the Inverted Pendulum task, a cart with a pendulum attached on the pivot point moves along the horizontal finite-length bar while in the Inverted Double Pendulum task, the pendulum has two poles connected with a joint in the middle. Both tasks aim to keep the pendulum balanced by applying a horizontal force on the cart as the action.

In these two environments, the observation spaces are 4-tuple and 11-tuple, respectively, and the action spaces are defined by a force with values within ± 3 and ± 1, respectively. After applying the action, in the Inverted Pendulum task, the agent receives reward +1 if the pendulum does not fall, otherwise, the episode is terminated and the reward becomes 0. Besides, the maximum number of steps in an episode is 10^3 and if the average reward of 950.0 in 100 consecutive episodes is obtained, the task is regarded as solved. On the other hand, the reward in the Inverted Double Pendulum task is more informative as it is calculated by the states of the pendulum. The task is solved when the average reward of 9100.0 in 100 episodes is received.

In all the simulations, the learning rates for the critic network and actor network are 10^{-3} and 10^{-4}, respectively. The discount factor γ is set to 0.99, the soft update rate τ is 10^{-2}, and the minibatch size K is 64. The network contains two hidden layers, 400 nodes in the first layer and 300 nodes in the second layer. The PER parameters α is fixed to 0.6, but β, initialized by 0.4, is varying with an increment of 6.666×10^{-6} for every mini-batch sample. Besides, $S = 16$ and the size of frozen region is set to $N/5$. The Ornstein-Uhlenbeck process is employed as the exploration noise added to the output of the actor network and Adamax [15] is adopted to accelerate the training process.

B. Results and Comparisons

One memory record for saving a state transition is 10 and 24 *double-precision* words in Inverted Pendulum and Inverted Double Pendulum task, respectively. The memory overhead of using the proposed CLA scheme can be expressed in *double-precision* words, given as

$$N_{\text{Extra}} = S \cdot \overbrace{\frac{\lceil \log_2(N) \rceil}{64}}^{\text{poionter}} + \overbrace{N}^{\text{list}} + \overbrace{S-1}^{\text{boundary}} . \tag{10}$$

The computation overhead for the proposed scheme can be estimated as follows. Extra inference and sorting efforts are required for every N steps if the PER plus CLA technique are adopted. However, when the memory size is large, the segmentation boundary may becomes gradually outdated as the network parameters change. Hence, a variable $0 < \phi \leq 1$ is introduced and only the latest ϕN experiences are considered for segmentation. Then, the size of each list is scaled up by $1/\phi$. The partial mapping can reduce the sorting and inference complexity. The comparison between PER and PER with CLA schemes is shown in TABLE I. To distinguish different memory sizes in two schemes, M is used for PER.

TABLE I. Overhead Estimation

	PER per M steps	PER+CLA per N steps
Memory Size (unit: 1 transition)	M	$N + N_{\text{Extra}}$
Inference Times	M	$(1+\phi)N$
Sorting Complexity	M	N and ϕN

Algorithm 1 DDPG with Prioritized Experience Replay and Classification Scheme

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize circular replay buffer D to size N, minibatch K, soft update learning rate τ and first transition priority $p_1 = 1$
Initialize κ=0 // Write pointer for circular replay buffer D
for episode = 1 to M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1 to T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) with maximal priority $p_t = max_{i<t} p_i$ in D by address pointer κ
        GenerateSegmentation
        for i = 1 to K **do**
            Sample transition i according to $P(i) = p_i^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight w_i and TD error δ_i
            Update transition priority $p_i \leftarrow \delta_i$
            ClassificationSwap
        end for
        Update the critic by minimizing the loss: $\frac{1}{K} \sum_i w_i \cdot \delta_i$
        Update the actor policy using the sampled policy gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{K} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$
    end for
end for

Algorithm 2 Segmentation

procedure GENERATESEGMENTATION
    if $\kappa == N - 1$ **then**
        Calculate TD errors δ_l for $0 \le l < N$
        Sort δ_l in ascending order
        Categorize δ_l to the corresponding segment s
        Update length L_s of segment s for $0 \le s < S$
        Update lower boundary LB_s
        Save element addresses in $List_s[0 : L_s - 1]$
        Set $S\_Ptr_s = 0$ for all s
    end if
end procedure

Algorithm 3 Classification and Swap

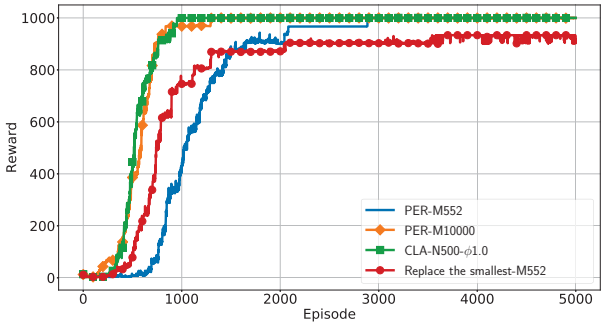
procedure CLASSIFICATIONSWAP
    if $LB_s \le \delta_i < LB_{s+1}$ **then**
        $addr_{cla} = List_s[S\_Ptr_s]$
        $addr_t = addr_{cla}$
        while $addr_{cla} \in [\kappa : \kappa + N/5]$ **do**
            S_Ptr_s points to the next element circularly
            $addr_{cla} = List_s[S\_Ptr_s]$
            if $addr_t == addr_{cla}$ **then**
                $addr_{cla} = -1$
                break
            end if
        end while
        S_Ptr_s points to the next element circularly
    end if
    if $addr_{cla} \neq -1$ and $addr_i \notin [\kappa : \kappa + N/5]$ **then**
        Swap($D[addr_{cla}], D[addr_i]$)
    end if
end procedure

Fig. 6 shows the averaged learning performances over 30 independent runs for the PER scheme and the PER with CLA scheme of the Inverted Pendulum and the Inverted Double Pendulum tasks. Considering the memory overhead in the PER with CLA scheme, we use two buffer sizes in the agents that only incorporates the PER scheme, a typical value of 10000 and the similar total memory size M as $N + N_{\text{Extra}}$ in the PER with CLA scheme.

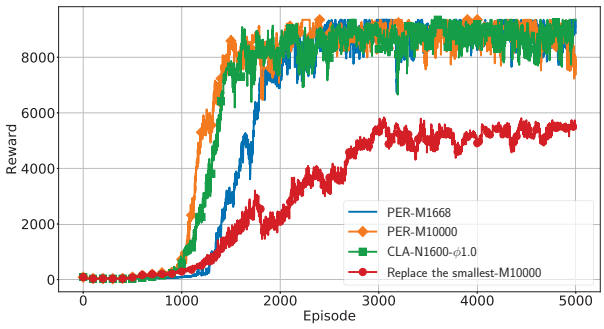
In the Inverted Double Pendulum, a memory size denoted by "M10000" is equivalent to 1920 KBytes and the proposed PER with CLA scheme utilizes a total of 134 KBytes. In addition, to show the effectiveness of the classification, a special scheme, which always replaces the experience with the smallest TD error in the replay buffer is also depicted in the figure. It is clear that our proposed scheme accelerates learning speed compared to the PER only scheme given the similar memory size. In Fig. 7, we show the average reward versus memory size for different schemes and parameter ϕ at a certain episode. From the comparison, we can see that the PER with CLA scheme not only achieves memory size reduction but also expedites the learning performance.

IV. CONCLUSION

In this paper, we propose the CLA scheme as an add-on to PER for reducing memory requirement and stabilizing the training of DDPG agent with additional computations for the

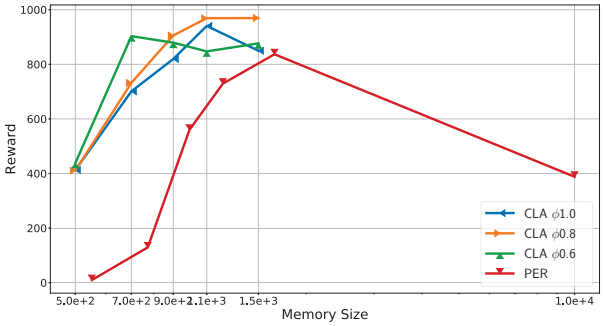


(a)

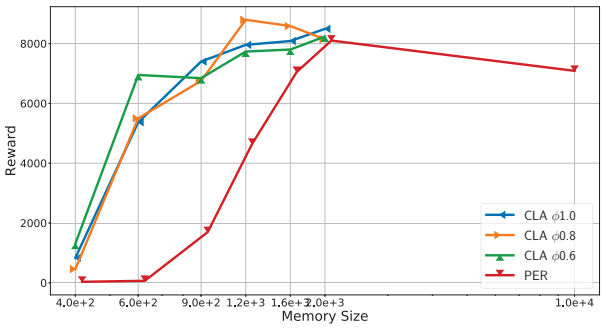


(b)

Figure 6. Averaged learning performance for (a) Inverted Pen-dulum and (b) Inverted Double Pendulum tasks.



(a)



(b)

Figure 7. Averaged rewards versus buffer sizes for (a) Inverted Pendulum and (b) Inverted Double Pendulum tasks.

updated TD error information. The proposed CLA mechanism consists of two procedures. The segmentation procedure partitions the TD error distribution into several segments by a hyperbolic tangent function. The classification a nd swap procedure changes the lifetimes of the experience in the replay buffer. From the simulations, we can see the proposed scheme can reduce the memory size and also accelerate the training process successfully. The reduction of the huge replay buffer can benefit the implementation of the DRL in the end devices and is helpful to the scenarios with constrained memory size.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *NIPS Deep Learning Workshop*, 2013.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, pp. 26–38, 2017.

[3] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, S. Ostrovski, Georg andPetersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb 2015.

[5] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, no. 1, pp. 103–130, Oct 1993.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, Nov 2016.

[7] S. Zhang and R. S. Sutton, "A Deeper Look at Experience Replay," *arXiv e-prints*, p. arXiv:1712.01275, Dec 2017.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, Sep 2015.

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, Jun 2000, pp. 1057–1063.

[10] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, Apr 2003.

[11] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen, "A novel ddpg method with prioritized experience replay," in *2017 IEEE International Conference on Systems, Man, and Cybernetics SMC 2017*, Oct 2017, pp. 316–321.

[12] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 32, no. 1, 22–24 Jun 2014, pp. 387–395.

[13] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," 2018.

[14] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 5026–5033.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.