2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
Banff Center, Banff, Canada, October 5-8, 2017

A Novel DDPG Method with Prioritized Experience Replay

Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, Chunlin Chen
Department of Control and Systems Engineering
School of Management and Engineering
Nanjing University, Nanjing 210093, China
Email: clchen@nju.edu.cn

***Abstract*—Recently, a state-of-the-art algorithm, called deep deterministic policy gradient (DDPG), has achieved good performance in many continuous control tasks in the MuJoCo simulator. To further improve the efficiency of the experience replay mechanism in DDPG and thus speeding up the training process, in this paper, a prioritized experience replay method is proposed for the DDPG algorithm, where prioritized sampling is adopted instead of uniform sampling. The proposed DDPG with prioritized experience replay is tested with an inverted pendulum task via OpenAI Gym. The experimental results show that DDPG with prioritized experience replay can reduce the training time and improve the stability of the training process, and is less sensitive to the changes of some hyperparameters such as the size of replay buffer, minibatch and the updating rate of the target network.**

***Index Terms*—Deep Deterministic Policy Gradient, Deep Reinforcement Learning, Prioritized Experience Replay**

I. Introduction

In the conventional continuous control domains, where the action space is large and actions are real-valued, letting agents directly learn from pixel-level information to accomplish complex manipulation tasks is very difficult [1], [2]. Recently, a novel deep reinforcement learning algorithm, called deep deterministic policy gradient (DDPG) [1], has achieved good performance in many simulated continuous control problems by integrating deterministic policy gradient algorithm [3] with some tricks introduced in making computers play Atari games [4]-[6]. It uses a replay buffer to remove the correlations existing in the input experiences (i.e. experience replay mechanism and an experience is a four-element tuple (s_t, a_t, r_t, s_{t+1})) [7], [8] and exploits target network approaches to stabilize the training process. As an essential part of DDPG, the experience replay significantly affects the performance and the speed of the learning process via the experiences selected to train the neural network.

In the experience replay mechanism, it uses a finite-size memory called a replay buffer to store previous experiences and randomly selects a fixed number of experiences (i.e., a minibatch of experiences) to update the neural network at a time step. It is clear that temporal correlations existing in the replayed experiences have been greatly weakened by mixing recent experiences with old experiences to update the network. However, the experience replay mechanism is based on the intuition that all experiences stored in the replay buffer are of equal importance and therefore it uniformly samples a minibatch of experiences to perform the update of the network. This intuition goes against the common phenomenon that when people learn to do something, experiences with big rewards, very successful attempts or painful lessons will become consistently fresh in their memory during the learning process and therefore these experiences are more valuable compared with others. During the learning process, instead of replaying all experiences uniformly, we tend to frequently replay experiences associated with more successful attempts and similar ideas can be referred to [9], [10]. Experiences with terrible tumbles are also frequently replayed to help us better realize the consequence of the wrong actions, correct them aggressively in the corresponding conditions and avoid making the same mistakes again. Hence, in this paper we consider the values of different experiences and propose to apply prioritized sampling or prioritized experience replay mechanism [11] to improve the DDPG algorithm, while the original experience replay mechanism adopted in DDPG algorithm is called uniform sampling or uniform experience replay.

The prioritized experience replay mechanism works as follows. Firstly, we evaluate the learning value of the experiences by their absolute temporal difference errors (TD-errors), which have already been computed in many classical reinforcement learning algorithms like SARSA and Q-learning [12]. Then, by ranking the experiences in the replay buffer through the absolute value of their TD-errors, we more frequently replay those with high magnitude of TD-errors. This practice will inevitably change the state visitation frequency and thus introducing bias, which is corrected by importance-sampling weights [13]. The proposed algorithm is most similar to [14], whose application scenarios are more restrictive and limited to discrete action space. In this paper, we expand the applications from discrete action domains to continuous action domains, and the model is based on Tensorflow [15], [16]. The DDPG with prioritized sampling is tested with the inverted pendulum task on the platform called OpenAI Gym [17]. With comparison to the original DDPG method, the experimental results show that DDPG with prioritized experience replay not only achieves similar performance within shorter training time, but also demonstrates more stable training process and is less sensitive to the changes of some hyperparameters.

II. DDPG with Prioritized Experience Replay

In this section, deep reinforcement learning, DDPG and the idea of prioritized experience replay is first introduced, and then an integrated algorithm of DDPG with prioritized experience replay is presented.

A. Deep Reinforcement Learning and DDPG

In standard reinforcement learning scenarios, an agent interacts with the environment to maximize the long-term rewards. Typically, this interaction process is formulated as a Markov Decision Process (MDP) which is described by a four-element tuple (S,A,R,P), where S is the state space, A is the action space, $R: S\times A \rightarrow \mathbb{R}$ is the reward function and $P: S\times A\times A \rightarrow [0,1]$ is the transition probability. In this scenery, the target of the agent is to learn a policy $\pi: S\rightarrow A$ which maximizes the long-term rewards: $R_0 = \sum_{i=0}^{T} r(s_i,a_i)$, where T is the termination time step, $r(s_i,a_i)$ is the reward of executing action a_i in state s_i. Usually, this long-term reward is discounted by a factor γ as $R_0^{\gamma} = \sum_{i=0}^{T}\gamma^i r(s_i,a_i)$, where $\gamma \in (0,1)$. The action-value function is used to represent the expected long-term reward of executing action a_t in state s_t:

$$Q(s_t,a_t) = E[R_t^{\gamma}|s=s_t,a=a_t] = E[\sum_{i=t}^{T}\gamma^{i-t}r(s_i,a_i)]. \quad (1)$$

Bellman Equation is often used to find the optimal action-value function:

$$Q^*(s_t,a_t) = E[r(s_t,a_t)+\gamma \max_{a_{t+1}^{'}} Q^*(s_{t+1},a_{t+1}^{'})]. \quad (2)$$

However, the above approach is limited to occasions where both of the state space and the action space are discrete. In order to apply the above method to problems with a continuous state-action space, two deep neural networks is designed, i.e., the action-value network $Q(s_t,a_t,w)$ and the actor network $\mu(s_t,v)$, to approximate the action-value function $Q(s_t,a_t)$ as well as the actor function $\mu(s_t)$ where w and v are the network parameters, $\mu(s_t)$ is the function used to map the state s_t to a deterministic action a_t. Approaches combining conventional reinforcement learning algorithms (e.g. Q-learning) with deep neural networks are called deep reinforcement learning algorithms (e.g. Deep Q-Network [5]).

The training of the action-value network is based on minimizing the following loss function L(w):

$$L(w) = (r(s_t,a_t)+\gamma Q'(s_{t+1},a_{t+1},w) - Q(s_t,a_t,w))^2. \quad (3)$$

The parameters of the actor network are updated using policy gradient algorithm [3]:

$$\begin{aligned} &\nabla_v Q(s,a,w)|_{s=s_t,a=\mu(s_t,v)} \\ &= \nabla_a Q(s,a,w)|_{s=s_t,a=\mu(s_t,v)}\nabla_v \mu(s,v)|_{s=s_t}. \end{aligned} \quad (4)$$

Previously, the training of deep neural networks is difficult or even impossible for there is no theoretical guarantee and therefore the deep neural networks are prone to diverge in the training process. However, recent advances in deep learning areas (i.e., experience replay mechanism, target network approach and batch normalization) have helped overcome the difficulty. Experience replay mechanism is proposed to break the correlations between the input training experiences and the target network approach is exploited to give the training process a consistent target. In addition, batch normalization [18] is adopted to constrain the change of the network parameters. These approaches stabilize the training process as well as make the training of large non-linear neural networks possible.

B. Prioritized Experience Replay

The core idea of the prioritized experience replay is to more frequently replay experiences associated with very successful attempts or extremely awful performance. Therefore, the criterion of defining the value of the experiences is the central issue. Since in most reinforcement learning algorithms, TD-error is often used to update the estimation of the action-value function Q(s,a). The value of TD-error acts as the correction for the estimation and may implicitly reflect to what extent an agent can learn from the experience. The bigger the magnitude of absolute TD-error is, the more aggressive the correction for the expected action-value is. In this condition, experiences with high TD-errors are more likely to be of high value and associated with very successful attempts. Besides, experiences with big negative magnitudes of TD-errors are conditions where the agent behaves miserably and the states of these conditions are badly learned by the agent. More frequently replaying these experiences will help the agent gradually realize the true consequence of the wrong behavior in the corresponding states, as well as avoid making the wrong behavior in these conditions again, which can improve the overall performance. Therefore, these badly learned experiences are also considered to be of high value. In this paper, we select absolute TD-error $|\delta|$ of the experiences as the index to evaluate the value of the experiences. The computation of the TD-error δ_j of experience j is given as:

$$\delta_j = r(s_t,a_t)+\gamma Q'(s_{t+1},a_{t+1},w) - Q(s_t,a_t,w), \quad (5)$$

where $Q'(s_{t+1},a_{t+1},w)$ is the target action-value network parameterized by w. In many classical reinforcement learning algorithms like SARSA and Q-learning, such value has already been computed, which facilitates the evaluation process of the experiences. We define the probability of the sampled experience j as

$$P(j) = \frac{D_j^{\alpha}}{\sum_k D_k^{\alpha}}, \quad (6)$$

where $D_j = \frac{1}{rank(j)} > 0$, rank(j) is the rank of the experience j in the replay buffer with absolute TD-error being the criterion. Then parameter α controls to what extent the prioritization is used. The definition of the sampling probability can be seen as a method of adding stochastic factors in selecting experiences since even those with low TD-errors can still have a probability to be replayed, which guarantees the diversity of sampled experiences. Such diversity can help prevent the neural network from being over-fitting.

However, since we tend to more frequently replay experiences with high TD-errors, it will no doubt change the

Algorithm 1 DDPG with Prioritized Experience Replay

```
1: Initialize action-value network $Q(s_t,a_t,w)$, actor network $\mu(s_t,v)$ with $w$ and $v$ separately, replay buffer R with size S
2: Initialize target network $Q'(s_t,a_t,w)$, $\mu'(s_t,v)$ with $w$ and $v$ separately
3: Initialize maximum priority, parameters $\alpha$, $\beta$, updating rate of the target network $\lambda$, minibatch K
4: for episode = 1, M do
5:     Initialize a random noise process $\Phi$
6:     Observe initial state $s_1$
7:     for t = 1, H do
8:         Add noise $\Phi_t$ in the exploration policy and select action $a_t$ according to the new policy
9:         Obtain reward $r_t$ and new state $s_{t+1}$
10:        Store experience $(s_t,a_t,r_t,s_{t+1})$ in replay buffer R and set $D_t = \max_{i<t} D_i$
11:        if t > S then
12:            for j =1, K do
13:                Sample experience $j$ with probability $P(j)$
14:                Compute corresponding importance-sampling weight $W_j$ and TD-error $\delta_j$
15:                Update the priority of transition j according to absolute TD-error $|\delta_j|$
16:            end for
17:            Minimize the loss function to update action-value network: $L = \frac{1}{K}\sum_i w_i \delta_i^2$
18:            Compute policy gradient to update the actor network:$\nabla_v \mu|_{s_i} \approx \frac{1}{K}\sum_i \nabla_a Q(s_t,a_t,w)|_{s=s_i,a=\mu(s_i,v)} \nabla_v \mu(s_t,v)|_{s_i}$
19:            Adjust the parameters of the target network $Q'(s_t,a_t,w)$ and $\mu'(s_t,v)$ with an updating rate $\lambda$
20:        end if
21:    end for
22: end for
```

state visitation frequency. This change may cause the training process of the neural network prone to oscillate or even diverge. In order to handle this issue, importance-sampling weights are used in the computation of weight changes:

$$W_j = \frac{1}{S^\beta \cdot P(j)^\beta}, \tag{7}$$

where S is the size of the replay buffer, $P(j)$ is the probability of the sampled experience j, the parameter β controls to what extent the correction is used.

Based on the above introduced prioritized experience replay method, an integrated algorithm of DDPG with prioritized experience replay is shown as in Algorithm 1.

III. Simulated Experiments

In this section, experimental settings are first introduced and then the proposed algorithm is tested on a classical continuous control problem of an inverted pendulum task. Several groups of experimental results are given with comparison to the original DDPG method.

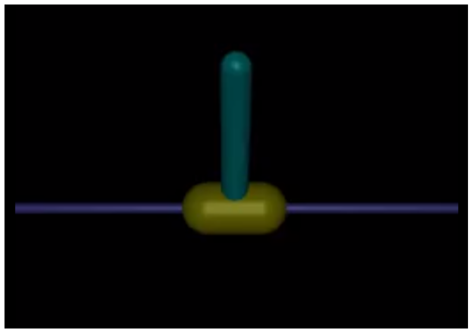
A. Experimental Settings

We use a similar low-dimensional neural network introduced in [1] and a momentum-based algorithm called Adam [19] is adopted to train the neural networks, with a learning rate of 10^{-4} for action-value network and 10^{-3} for the actor network. To make sure that the weights of the neural networks will not become too large, L_2 regularization item is included in the loss function to update the action-value network. The discount factor is set 0.99 and the update rate of the target network is set as 0.01. Two hidden layers are included in the low-dimensional networks (i.e. the action-value network and the actor network), with 400 hidden units for the first layer and 300 units for the second layer, respectively.

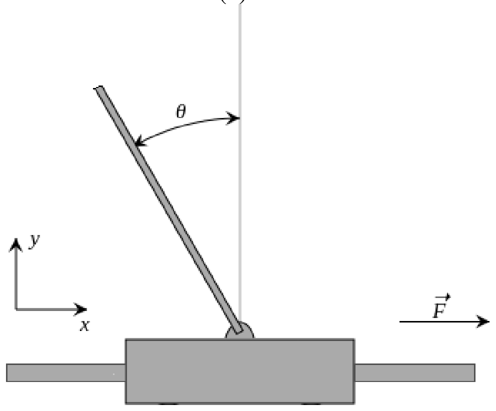
We randomly select values from a uniform distribution w.r.t the input of the layers for the weights of the two neural networks. However, the final layers' weights are uniformly selected between -3×10^{-3} and 3×10^{-3}. We use the Ornstein-Uhlenbeck process [20] to produce noise that is added in the exploration policy to help the agent explore the environment thoroughly, where the θ, σ is set 0.15 and 0.2, respectively. The value of the minibatch is set as 32 and the size of the replay buffer is set as 10^4. The storage structure of the replay buffer is a circular queue and the data structure of the priority queue is a binary heap to minimize the additional time cost for prioritized sampling. In the process of interacting with the environment, the agent receives low-dimensional vectors as the observation (i.e., state). These low-dimensional vectors are values of the speed, joint angles and coordinate information. MuJoCo [21], [22] is used as the simulator and OpenAI Gym is adopted to load the MuJoCo model. Learning speed, learning stability and sensitiveness to the value of some hyperparameters are used as three criterions to evaluate the performance of the algorithms. In this paper, we compare the performance of the original DDPG method, DDPG with prioritized experience replay and the best baseline of the OpenAI Gym in the inverted pendulum task.

B. Inverted Pendulum Task

The inverted pendulum task is a classical continuous control problem. As shown in Fig. 1, a cart on which a pendulum is fixed through the pivot point can move in the horizontal direction along the finite-length bar. A horizontal force can be used to keep the pendulum balanced as long as possible. The action is defined as the horizontal force added to the cart and the observation (i.e. state) of the agent is defined as a four-dimensional vector that represents the state of the pendulum. In the experiment, the agent is given a positive reward of +1 if the angle θ between the pendulum and the vertical line is kept within a desired degree at each time step. One training episode is autonomously terminated if θ is larger than the target or the cart moves out of the bar. We set the maximum number of the training steps in an episode to be 1,000 and the inverted pendulum task is deemed as solved when the agent get an average reward of 950.0 in 100 episodes consecutively. Theoretically, the maximum reward the agent can receive is 1,000.



(a)



(b)

Fig. 1: Inverted pendulum task. (a) the inverted pendulum task in the OpenAI Gym; (b) the mathematical model of the inverted pendulum.

As shown in Fig. 2, it is clear that DDPG with prioritized sampling outperforms DDPG with uniform experience replay regarding the training steps used to accomplish the inverted pendulum task. It takes 700 episodes for the prioritized experience replay method to accomplish the inverted pendulum task while around 1,300 episodes for the uniform sampling method to achieve the same performance. DDPG with prioritized sampling also outperforms the best method in the OpenAI Gym

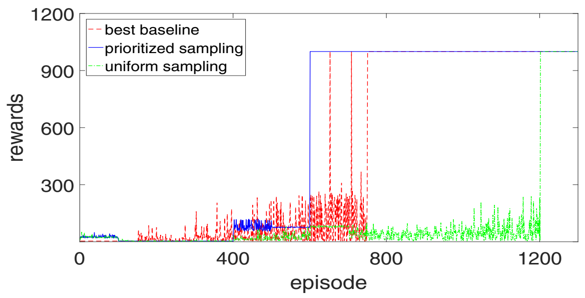


Fig. 2: Learning performance for the inverted Pendulum task regarding the accumulated rewards.

that needs 850 episodes to converge. In addition, fewer spines happen in the reward curve of prioritized experience replay compared with the other two algorithms, which demonstrates more stable performance in the training process for DDPG with prioritized experience replay mechanism.

Then we further study the distribution of the selected samples with different magnitudes of TD-errors under prioritized experience replay. The selected experiences are divided into three parts: high magnitude of TD-error, medium magnitude and low magnitude. Since the minibatch is 32 and the size of replay buffer is set 10^4 in the experiment, we divide the replay buffer into 32 segments and deem that first 8 segments (rank 1$\sim$98) are of high TD-error, last 8 segments (rank 3,544$\sim$10,000) are considered low TD-error and the rest (rank 99$\sim$3,543) is labeled medium TD-error (the determination of the boundaries of these segments is based on stratified sampling). In the experiment, we find that prioritized experience replay tends to choose those with medium and high magnitude of TD-error for they are of high value to the learning process of the agent. And also, prioritized sampling tends to choose more recent experiences for they are not as well estimated by the action-value function as others. Besides, experiences having low TD-errors are also included in the selected experiences, which implicitly shows the diversity of sampled experiences. To sum up, prioritized experience replay can not only focus on experiences with higher magnitudes of TD-error to help accelerate the training process, but also include those with low TD-errors to increase the sample diversity of training experiences.

In addition, we change the value of some hyperparameters (i.e., the size of the replay buffer S, minibatch K and the updating rate of the target network λ, see Algorithm 1) and try to test whether DDPG with prioritized sampling and DDPG with uniform sampling are robust enough to be insensitive to these changes. Three groups of experiments are carried out as follows.

1) Changing the size of the replay buffer: The size of the buffer is set as 10^4, 10^5 and 10^6, respectively. From Fig. 3 and Fig. 4, we find that DDPG with prioritized experience replay exhibits more robust performance compared with uniform one for its maximum average reward and the total training time remains to be 1,000 and 700 episodes separately although the

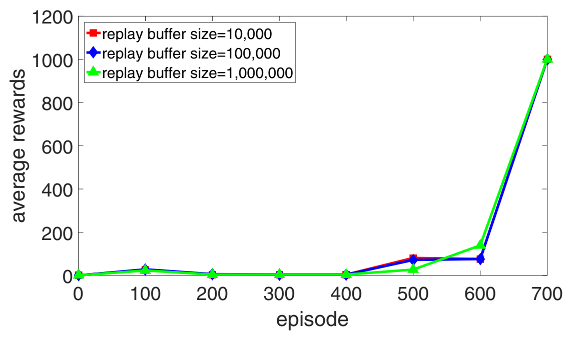


Fig. 3: Performance of DDPG with prioritized experience replay under different sizes of replay buffer.

size of the replay buffer changes. In contrast, the training time of the agent with uniform sampling increases from 1,300 episodes to 1,400 episodes when the size of the replay buffer increases from 10^4 to 10^6. One possible explanation is that because experiences are randomly selected to replay in uniform sampling method and increasing the size of the buffer is likely to cause more useless experiences to be selected to perform the network update. The agent has performed fairly well in these conditions and therefore these experiences are less beneficial to the learning process of the agent compared with those more valuable ones. As a result, the training time of the agent with uniform sampling increases.

2) Changing the value of minibatch: We set the size of minibatch as 16, 32 and 64, respectively. From Fig. 5, we find that three average reward curves of the prioritized sampling are almost overlapped, which means a stable performance can still be performed by the agent using prioritized experience replay. Nevertheless, total training time of the agent with uniform sampling increases and the learning process of the agent is suffering from more aggressive changes in terms of the average reward curves (see Fig. 6).

3) Changing the updating rate of the target network: The updating rate of the target network is set to be 0.01, 0.05 and 0.1, respectively. According to the experimental results, the training time of the agent is directly related with this hyperparameter. From Fig. 7 and Fig. 8, we find that when the updating rate of the target network changes from 0.01 to 0.1, the training time of the two algorithms declines to 600 episodes and 1,000 episodes, respectively. From the empirical perspective, this improvement is owing to the easiness of the training task. Specifically, a slight increase in the updating rate of the target network contributes to the converging speed of the network and thus reducing the training time. However, DDPG with prioritized sampling uses less training time and exhibits more stable performance in the training process compared to the uniform one.

According to all the above three groups of experiments, it can be concluded that DDPG with prioritized sampling is less affected by the changes of the hyperparameters and is more robust compared with DDPG with uniform sampling.

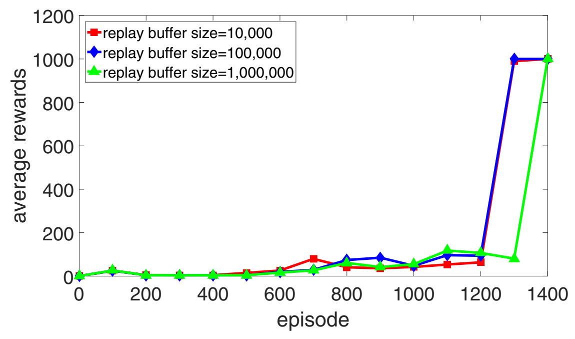


Fig. 4: Performance of DDPG with uniform experience replay under different sizes of replay buffer.

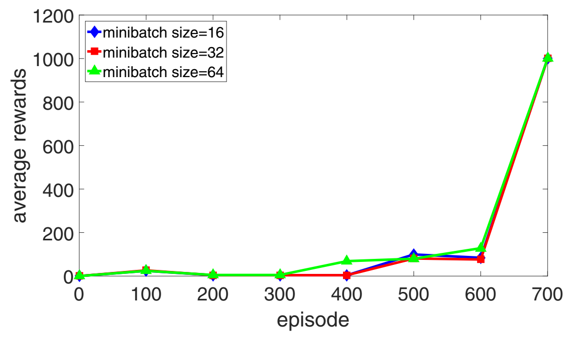


Fig. 5: Performance of DDPG with prioritized experience replay under different sizes of minibatch.

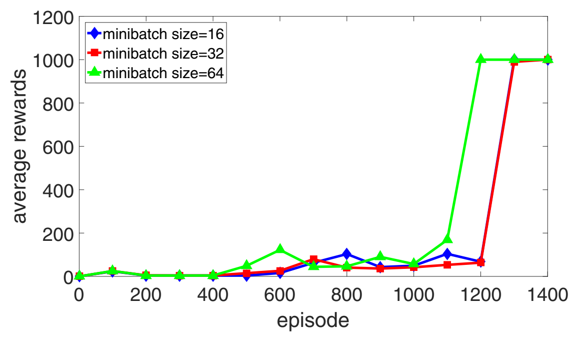


Fig. 6: Performance of DDPG with uniform experience replay under different sizes of minibatch.

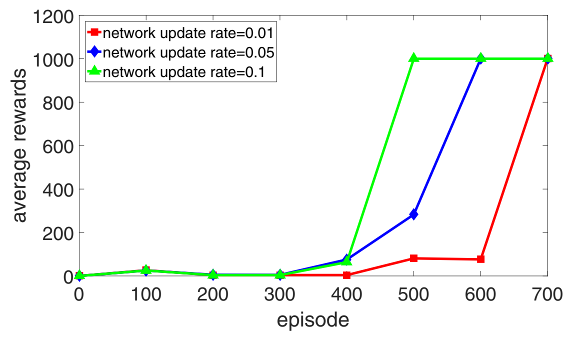


Fig. 7: Performance of DDPG with prioritized experience replay under different updating rates of the target network.

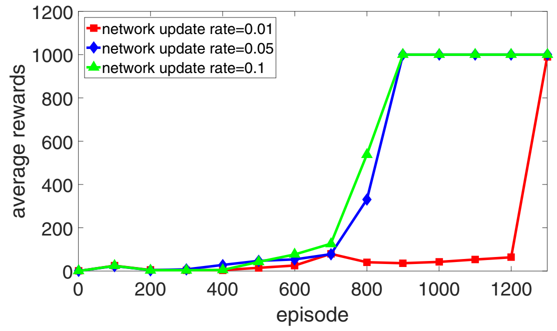


Fig. 8: Performance of DDPG with uniform experience replay under different updating rates of the target network.

IV. CONCLUSION

In this paper, the prioritized experience replay is proposed to replace the uniform experience replay in the DDPG algorithm. This idea is based on the intuition that some experiences have higher learning value compared with others and more frequently replaying these valuable experiences will improve the learning performance. The proposed algorithm with prioritized sampling helps shorten the total training time greatly as well as exhibit more stable learning process. In addition, prioritized experience replay is less affected by the changes of the hyperparameters and is more robust compared with the original DDPG algorithm.

However, more problems need to be further investigated in our future work. For example, a better criterion for evaluating the value of the experiences should be further investigated [14]. The proposed algorithm only concentrates on selecting which experience to replay and inevitably ignores the process of selecting which experience to discard. In addition, more experience selection and action selection methods for deep reinforcement learning may be inspired by quantum probability to design new exploration strategies [23]-[26]. We will focus on these problems in our future work and apply the proposed algorithm to more difficult tasks like hopper and walker [17].

ACKNOWLEDGMENT

This work was supported by the National Key Research and Development Program of China (No.2016YFD0702100) and by the National Natural Science Foundation of China (No.61432008).

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al, “Continuous control with deep reinforcement learning,” *Computer Science*, 8(6): A187, 2016.
[2] N. Heess, G. Wayne, D. Silver, et al, “Learning continuous control policies by stochastic value gradients,” *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 2944-2952, 2015.
[3] D. Silver, G. Lever, N. Heess, et al, “Deterministic Policy Gradient Algorithms,” *Proceedings of the International Conference on Machine Learning*, pp. 387-395, 2014.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, et al, “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint 1312.5602v1*, 19 December, 2013.
[5] V. Mnih, K. Kavukcuoglu, D. Silver, et al, “Human-level control through deep reinforcement learning,” *Nature*, 518(7540): 529-533, 2015.
[6] X. Guo, S. Singh, H. Lee, et al, “Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning,” *Proceedings of the International Conference on neural information processing systems*, pp.3338-3346, 2014.
[7] L. J. Lin, “Reinforcement learning for robots using neural networks,” *Carnegie Mellon University*, 1993.
[8] S. Adam, L. Busoniu, R. Babuska, “Experience Replay for Real-Time Reinforcement Learning Control,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Cybernetics*, 42(2): 201-212, 2012.
[9] A. C. Singer and L. M. Frank, “Rewarded outcomes enhance reactivation of experience in the hippocampus,” *Neuron*, 64: 910-921, 2009.
[10] C. G. McNamara, A. Tejero-Cantero, S. Trouche, et al, “Dopaminergic neurons promote hippocampal reactivation and spatial memory persistence,” *Nature neuroscience*, vol.17, pp. 1658-1660, 2014.
[11] A. W. Moore, C G. Atkeson, “Prioritized sweeping: Reinforcement learning with less data and less time,” *Machine Learning*, 13(1): 103-130, 1993.
[12] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[13] A. R. Mahmood, H. V. Hasselt, R. S. Sutton, “Weighted importance sampling for off-policy learning with linear function approximation,” *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 3014-3022, 2014.
[14] T. Schaul, J. Quan, I. Antonoglou, et al, “Prioritized Experience Replay,” *Proceedings of the IEEE International Conference on Learning Representations*, 2016.
[15] M. Abadi, A. Agarwal, P. Barham, et al, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467*, 2016.
[16] M. Abadi, P. Barham, J. Chen, et al, “TensorFlow: A system for large-scale machine learning,” *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.
[17] G. Brockman, V. Cheung, L. Pettersson, et al, “OpenAI Gym,” *CoRR*, 2016.
[18] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
[19] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv preprint arXiv:1412.6980 [cs.LG]*, 2017.
[20] G E. Uhlenbeck and L S. Ornstein, “On the Theory of the Brownian Motion,” *Revista Latinoamericana De Microbiologia*, 1930.
[21] E. Todorov, T. Erez, Y. Tassa, “MuJoCo: A physics engine for model-based control,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026-5033, 2012.
[22] E. Todorov, T. Erez, Y. Tassa, “Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX,” *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4397-4404, 2015.
[23] D. Dong, C. Chen, Z. Chen, “Quantum reinforcement learning,” *Lecture Notes in Computer Science*, 3611: 686-689, 2005.
[24] D. Dong, C. Chen, C. Zhang, Z. Chen, “Quantum Robot: Structure, Algorithms and Applications,” *Robotica*, 24(4): 513-521, 2006.
[25] D. Dong, C. Chen, H. X. Li, T. J. Tarn, “Quantum reinforcement learning,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(5): 1207-1220, 2008.
[26] C. Chen, D. Dong, H. X. Li, J. Chu, T. J. Tarn, “Fidelity-based Probabilistic Q-learning for Control of Quantum Systems,” *IEEE Transactions on Neural Networks and Learning Systems*, 25(5): 920-933, 2014.