Improving DDPG via Prioritized Experience Replay

Yuenan Hou
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
hy117@ie.cuhk.edu.hk

Yi Zhang
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
zy217@ie.cuhk.edu.hk

Abstract

Experience replay plays an important role in reinforcement learning. It reuses previous experiences to prevent the input data from being highly correlated. Recently, a deep reinforcement learning algorithm with experience replay, called deep deterministic policy gradient (DDPG), has achieved good performance in many continuous control tasks. However, it assumes the experiences are of equal importance and samples them uniformly, which obviously neglects the difference in the value of each individual experience. To improve the efficiency of experience replay in DDPG method, we propose to replace the original uniform experience replay with prioritized experience replay. We test the algorithms in five tasks in the OpenAI Gym, a testbed for reinforcement learning algorithms. In the experiment, we find that DDPG with prioritized experience replay mechanism significantly outperforms that with uniform sampling in terms of training time, training stability and final performance. We also find that our algorithm can achieve better performance in three tasks compared with some state-of-the-art algorithms like Q-prop, which directly proves the effectiveness of our proposed scheme. Our weekly report and codes are available at `https://github.com/cardwing/Codes-for-RL-Project`.

1 Introduction

Reinforcement learning (RL) is a process where an agent learns to interact with the environment to maximize the long-term rewards [19]. At each time step t, the agent observes state s_t, performs action a_t and receives a reward r_t. Conventional reinforcement learning algorithms like tabular Q-learning [19], have been proposed to handle problems with finite state-action space. To apply these methods to domains with infinite state space such as speech recognition [15] and image processing [2], features are hand crafted to represent the input high-dimensional states. The emergence of deep learning [6] has made the feature extraction process learned in an end-to-end manner.

However, deep learning can not be directly applied to RL scenarios since the states RL agent receives have strong temporal correlations. The strong correlation contradicts the independent requirement of deep learning algorithms which adopt gradient-based optimization algorithms to update the network parameters. Therefore, experience replay [3, 7] is proposed to break the temporal correlation between input samples. Experience replay utilizes a finite-size memory to store previous met samples. Then, at each iteration, a fixed number of samples are selected randomly from the memory to update the network.

With experience replay, deep reinforcement learning, which combines DL with RL, has achieved remarkable performance in many tasks. For instance, deep Q network (DQN) [14, 13] is proposed to play Atari 2600 games and surpass human players. As to domains with infinite action space, deep deterministic policy gradient (DDPG) [10] is designed to handle the continuous control tasks.

However, DDPG assumes all samples to be of equal importance and adopts uniform experience replay. This is irrational since it neglects the difference in the value of each individual sample. A

more rational sampling strategy should be utilized to replace the uniform experience replay. By referring to [16] which applied prioritized experience replay to DQN, we propose to adopt prioritized experience replay (PER) in DDPG. The difference between our method and [16] is that our application field has infinite action space and is more challenging than theirs which has discrete and finite action space. In the presented DDPG with PER, we first select the absolute temporal difference error as the measure to reflect the value of each individual sample. Meanwhile we are trying to find a better metric to represent the importance of samples for training. Besides, stochastic prioritization is adopted to select experiences to prevent over-fitting. Moreover, importance-sampling weights are used to correct the state visitation bias introduced by prioritized sampling.

Therefore, our contribution is to combine the prioritized experience replay mechanism with DDPG method in the continuous control domain where actions spaces are continuous. We test our algorithm in the inverted pendulum, inverted double pendulum, halfcheetah, walker and hopper task in the OpenAI Gym [4] which uses a physical engine called MuJoCo [20] as the simulator and our model is based on TensorFlow [1]. In the experiment, we find that compared with agents which use uniform sampling, agents utilizing prioritized experience replay approach can not only get more rewards in the long term, but also achieve more stable performance in the same training steps. To be more specific, prioritized sampling helps the agent get almost the same average rewards as the uniform sampling while it takes half of the training episodes as the uniform one.

2 Methodology

2.1 DRL and DDPG

A standard reinforcement learning process is described as follows: an agent interacts with its surrounding environment through a sequence of observations s_t, actions a_t and rewards r_t. Here, actions a_t are real-valued and rewards r_t are functions w.r.t s_t and a_t. We define the policy the agent uses to choose actions as $\pi = P_0(a_t|s_t)$. The target of the agent is to select actions that optimizes discounted future reward $R_t = \sum_{i=1}^{T} \gamma^{i-1} r_i$, $\gamma \in [0, 1]$ is the discounting factor. We define the optimal action-value function ($Q^*(s, a)$) as the maximum return under state s and action a, which is described as the following equation:

$$Q^*(s_t, a_t) = \max_{\pi} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...] \tag{1}$$

Many RL algorithms make use of the following equation to get the optimal action-value function, which is called Bellman equation:

$$Q^*(s_t, a_t) = E[r_t + \gamma \max_{a'_{t+1}} Q^*(s'_{t+1}, a'_{t+1})] \tag{2}$$

This equation is based on the following idea: the maximum return under state s_t and action a_t equals to the addition of the immediate reward and the maximum return under next state s_{t+1}. In order to get the optimal action-value function, we can use the Bellman equation to update the action-value function iteratively. This approach can approximate the action-value function well as time-step t approaches infinity [19] (i.e. $Q(s_t, a_t) \to Q^*(s_t, a_t)$ as $t \to \infty$).

Such approach is limited to the situations where states are low-dimensional and discrete for it needs to approximate action-value under every state and every action. It is a common practice to use a non-linear neural network as the function approximator to estimate the action-value function in high-dimensional state domains. We define the neural network which is designed for estimating the action-value function under state-action pair (s_t, a_t) as action-value network $Q(s_t, a_t; \theta_1)$, with network parameters θ_1. Using the neural network as the approximator has an added advantage: it can generalize to unobserved states, which can greatly expand the application fields of the action-value function.

However, reinforcement learning could be unstable because of the correlations present in the sequence of observations when a nonlinear function approximator such as a neural network with nonlinear rectified units is used to present the action-value function. The problem is addressed by using a novel variant of Q-learning [22] which consists of experience replay mechanism and iterative update. The former randomly samples the data and thereby removes correlations in the observation sequence and smooths the data distribution. The later iteratively updates the action-value towards the target value

to reduce the correlation with the target. The Q-learning update is performed on the action-value network by minimizing the following loss function:

$$L(\theta_1) = (r_t + \gamma Q'(s_{t+1}, a_{t+1}; \theta_1) - Q(s_t, a_t; \theta_1))^2 \tag{3}$$

where s_{t+1} is next state, $Q(s_t, a_t; \theta_1)$ is the action value of state-action pair (s_t, a_t) under network parameters θ_1. Instead of compute the full gradient of all the samples, stochastic gradient descent (SGD) which computes the gradient of a small part of all the samples is employed to reduce expensive and time-costly computation.

However, it is impossible to apply Q-learning method from low-dimensional action space to high-dimensional action space, because in high-dimensional action domains it is time-costly to find the optimal action a_{t+1} at every time t which maximizes the Q value under state s_{t+1}. Therefore we exploit the idea mentioned in [18], which is called DPG algorithm.

To be more specific, we use actor function $\mu(s_t; \theta_2)$ to map state s_t to a deterministic action a_t: $a_t = \mu(s_t; \theta_2)$. Similar to the definition of the Q network, actor network $\mu(s_t; \theta_2)$ is referred to the neural network which works as a function approximator to evaluate actor function $\mu(s_t)$, with network parameters θ_2. Therefore, we use two neural networks: action-value network $Q(s_t, a_t; \theta_1)$ and actor network $\mu(s_t; \theta_2)$ to approximate the optimal action-value function $Q(s_t, a_t; \theta_1)$ and actor function $\mu(s_t)$ respectively. The actor network is updated using the policy gradient [18] (i.e. the gradient of the policy's performance):

$$\begin{aligned} &\nabla_{\theta_2} Q(s, a; \theta_1)|_{s=s_t, a=\mu(s_t;\theta_2)} \\ &= \nabla_a Q(s, a; \theta_1)|_{s=s_t, a=\mu(s_t;\theta_2)} \nabla_{\theta_2} \mu(s; \theta_2)|_{s=s_t} \end{aligned} \tag{4}$$

Previously, non-linear neural network with thousands of parameters has been proved to be prone to diverge when trained and learning tends to be unstable in this condition. However, with recent proposed solutions like replay buffer, batch normalization [8] and target neural network, the learning of large non-linear neural network can be more stable and more likely to converge. To be more specific, replay buffer is used to store previously met experience and the neural network is trained by randomly selected experiences so that the correlations between the input data are broken. Batch normalization is used to normalize the input data so that they can have unit mean and variance, making parameters of the network adjusted within a proper magnitude and thus stabilizing the learning process. Target network approach is that we make a copy of the the trained network, force the network to change slowly towards the copy one. This approach can greatly improve the stability of the training process, which can be seen as a version of supervised learning. All these parts constitute the DDPG algorithm.

2.2 Prioritized Experience Replay

The main part of prioritized experience replay is the index used to reflect the importance of each transition. It is natural to select how much an agent can learn from the transition as the criterion, given the current state. However, this criterion is easy to think of but hard to put in practice. A good choice is to use the absolute TD-error value of the transition as the index, which can roughly measure how surprising the transition is to the agent. And this value is easy to get in RL algorithms like Q-learning or SARSA, which already computes and update the policy with the value.

Since TD-error implicitly reflects the amount agent can learn from the experience, it is natural to consider that we only greedily select experiences with the largest TD-error to replay, which is called greedy TD-error prioritization. However, this approach has three shortcomings. First, updates of TD-error only happens in those transitions which are used to replay so that time-costly sweeps through the whole replay buffer are avoided. In this condition, transitions which receive a low magnitude of TD-error on first replay tend not to be replayed for a long time or even never be replayed before being regarded, which apparently wastes a lot of precious transitions. Second, greedy TD-error prioritization only works well in conditions where rewards are deterministic. When the rewards are stochastic and function approximating itself brings noise (i.e. in simulated control task and use DDPG), greedy TD-error prioritization will exhibit bad performance. Third, greedy TD-error prioritization only makes a small part of the experience replayed frequently and the errors reduce slowly when using deep neural network to approximate functions, which means transitions with high TD-error are often replayed. This practice leads to lack of experience diversity, which makes the network over-fitting easily.

Therefore, to alleviate these problems, we introduce some randomness in selecting experience and thus making a trade-off between greedy TD-error prioritization and pure random sampling. This practice is called stochastic prioritization. We guarantee that transitions with lowest magnitude of TD-error have probability to be replayed and transitions' probability of being replayed is in proportion to their priority. To be more specific, in sampling transitions which are used to train the neural network, we make use of the rank-based prioritized experience replay approach [16]. The main difference between the original paper and our paper is that we apply the rank-based prioritized experience replay approach to a new field (i.e. continuous control domains). Concretely, we rank the transitions in the replay buffer according to their absolute value of TD-error. Based on the intuition that transitions with high TD-error should be sampled more frequently for they are more valuable to the learning process, we define the probability of each transition i being sampled to be :$P(i) = \frac{D_i^\alpha}{\sum_k D_k^\alpha}$, where $D_i = \frac{1}{rank(i)} > 0$, rank(i) is the rank of the transition i in the replay buffer with TD-error being the criterion. If α is set to be zero, that belongs to the uniform sampling. When α approaches infinity, it corresponds to greedy prioritization. Therefore, this value controls the extent to which the prioritization is used.

Such randomness has an added advantage. Since pure random sampling minimizes the correlations between the transitions used to train the neural network, selecting those with high magnitude of TD-error will unavoidably add the correlation. This addition is due to the fact that an improper continuous series of actions will have big TD-errors in these transitions, which means the probability of selecting a series of these continuous transitions increases. Randomness added in selecting transitions will correct the problem for it includes those with small TD-errors. Therefore, the drawback brought by biasing towards big TD-errors transitions is alleviated owing to such randomness.

Prioritized experience replay inevitably changes the distribution the expected value will converge to because it tends to sample valuable transitions and thus changing the state visitation frequency. However, such changes may significantly affect the convergence of training process because of the unbiased updates requirement of reinforcement learning. In order to remove the effect this change brings, we add the importance-sampling weights:$W_i = \frac{1}{N^\beta \cdot P(i)^\beta}$ in the computation of weight updates [11]. If β is set to be 1, that means aggressively corrects the prioritized sampling probability $P(i)$.

The unbiased property of policy updates is very important when the training process is approaching the end in many RL fields for the training process is highly dynamic. Moreover, policy, state visitation frequency and target network constantly changes throughout the process, making the training of neural network prone to diverge. Therefore, we intend to use importance-sampling weights to slightly correct the sampling probability in early training stages and increase the correction as training proceeds, which reaches peak at the end of the training process. In other words, in the experiment, we change β linearly from 0.5 to 1.

Importance-sampling weights have an added advantage in our experiment. In training the neural network, in order to make sure that the gradient is accurate, small training steps are necessary. However, in our method, experience prioritization tends to make transitions with high TD-error replayed more frequently while Importance-sampling weights can correct the bias introduced and reduce the change of gradient magnitude, which is equivalent to restrict the range of step size. In this condition, the gradient can be more accurately approximated, stabilizing the training of neural networks.

Therefore, we combine the rank-based prioritized experience replay mechanism with the the state-of-the-art DDPG algorithm. Our main contribution is to replace the uniform sampling approach with prioritized sampling approach (see Algorithm 1).

2.3 Intuitive Explanation of PER

In this section, we presents an intuitive explanation on the characteristics of DDPG with PER. Suppose the experience j is selected to update the network, then the differentiation of the loss function with

Algorithm 1 DDPG with rank-based prioritization

Initialize critic network $Q(s_t, a_t; \theta_1)$ and actor network $\mu(s_t; \theta_2)$ with random weights θ_1, θ_2
Initialize replay buffer B with size V
Set target network $Q'(s_t, a_t; \theta_1) = Q(s_t, a_t; \theta_1)$, $\mu'(s_t; \theta_2) = \mu(s_t; \theta_2)$
Set priority $D_1 = 1$, exponents $\alpha = 0.7$, $\beta = 0.5$, minibatch K=32
for episode = 1, H **do**
  Initialize a random process $\Re$ used to explore actions thoroughly
  Receive initial state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t; \theta_2) + \Re_t$
    Obtain reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer B and set $D_t = \max_{i<t} D_i$
    if t > V **then**
      for j =1, K **do**
        Sample transition j with probability $P(j) = \frac{D_j^\alpha}{\sum_i D_i^\alpha}$
        Compute importance-sampling weight $W_j = \frac{1}{N^\beta \cdot P(j)^\beta \cdot \max_i w_i}$
        Compute TD-error $\delta_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1})) - Q(s_j, a_j)$
        Update the priority of transition j according to absolute TD-error $|\delta_j|$
      end for
      Minimize the loss function to update critic network: $L = \frac{1}{K} \sum_i w_i \delta_i^2$
      Update the actor network:$\nabla_{\theta_2} \mu|_{s_i} \approx \frac{1}{K} \sum_i \nabla_a Q(s_t, a_t; \theta_1)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta_2} \mu(s_t; \theta_2)|_{s_i}$
      Update the target critic network: $Q'(s_t, a_t; \theta_1) = (1-\eta) \cdot Q'(s_t, a_t; \theta_1) + \eta \cdot Q(s_t, a_t; \theta_1)$
      Update the target actor network: $\mu'(s_t; \theta_2) = (1 - \eta) \cdot \mu'(s_t; \theta_2) + \eta \cdot \mu(s_t; \theta_2)$
    end if
  end for
end for

regard to the weight θ_1 can be computed as follows:

$$\begin{aligned} & \nabla_{\theta_1} L(\theta_1) \\ =& \frac{1}{2} \nabla_{\theta_1} (r_t + \gamma Q'(s_{t+1}, a_{t+1}; \theta_1) - Q(s_t, a_t; \theta_1))^2 \\ =& (r_t + \gamma Q'(s_{t+1}, a_{t+1}; \theta_1) - Q(s_t, a_t; \theta_1)) \nabla_{\theta_1} Q(s, a; \theta_1) \\ =& \delta_j \nabla_{\theta_1} Q(s, a; \theta_1). \end{aligned} \tag{5}$$

A prioritized sampling strategy tends to replay experiences with high magnitudes of TD-errors. Thus the TD-error used in Equation (5) has a large magnitude, which leads the parameters of the action-value network to change towards the steepest direction. In this case, the local optimal solution can be avoided in computing the weights of the action-value network. The training process of the network is prone to oscillate, since the change of the weights of the network is enlarged by the high TD-error. Fortunately, the importance-sampling weight reduces the magnitude of the gradient, thus it can stabilize the training process, i.e.,

$$W_j \nabla_{\theta_1} L(\theta_1) = W_j \delta_j \nabla_{\theta_1} Q(s, a; \theta_1). \tag{6}$$

In prioritized sampling, the experiences with high TD-errors δ_j are replayed more frequently, which may lead to low accuracy of the gradient estimation. From Equation (**??**), it is clear that the addition of the IS weight W_j can reduce the magnitude of the gradient change. Thus the gradient change is constrained and the gradient can be more accurately approximated, which will greatly stabilize the training process.

As shown in Fig. 1, a uniform experience replay usually selects the experiences with low magnitudes of TD-errors, and gradually minimize the loss function, which may fall into a local optimal solution. Prioritized experience replay with IS weights enable the replay incline to those experiences with the larger magnitudes of TD-errors. Therefore, it optimizes the loss function in a steeper way, and

its training process is rather stable due to the constrains of IS weights. In contrast, considering prioritized experience replay without IS weights, it may suffer from larger step sizes, and incline to a local optimal solution. Besides, the training process may oscillate. By introducing IS weights, prioritized experience replay can present a more global and stable solution for the reinforcement learning problems.

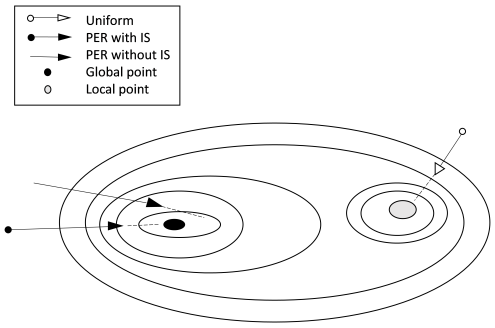

Figure 1: Illustration of the performance of uniform sampling, PER without IS and PER with IS in the training process, respectively. The contour lines constitute the space curve formed by the loss function $L(\theta)$ of all experiences.

3 Experiments

To test the proposed DDPG with PER algorithm, several experiments are conducted. The performances of the proposed algorithm are compared with original DDPG algorithm and related state-of-the-art algorithms.

3.1 Experimental Settings

The network we use is the same as the low-dimensional network introduced in [10], except that the size of the replay buffer is set as 10^4 to fasten the training speed. More specifically, Adam [9] is used to update the parameters of the network. The learning rate is set as 10^{-4} and 10^{-3} for actor and critic network, respectively. To avoid over-fitting problem, a regularization item L_2 is added in the computation of loss function for the critic network. The discount factor γ is set 0.99, and the updating rate of the target networks is set as $\eta = 0.01$. The tanh layer is added in the final layer of the actor network to restrict the actions in an appropriate range. The input of the last hidden layer of critic network consists of actions. The neural network has one input layer, two hidden layers and one output layer, with 400 and 300 units in the two hidden layers, respectively. The parameters of all layers of both critic and actor networks are randomly selected from a uniform distribution with regard to the input of the layers except the final layer. The parameters of the final layer are assigned from the uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$. For all the experiments, Ornstein-Uhlenbeck (OU) process is used as the noise process [21] with $\theta = 0.15$ and $\sigma = 0.2$.

A circular queue is used to store the experiences and a binary heap is utilized as the the data structure of the priority queue to minimize the additional time cost for prioritized sampling. The minibatch size is set as 32. The observations the agents get are low dimensional vectors which are used to describe the agent and the environment (e.g. speed, joint angles and coordinate information). A physical engine called MuJoCo [20] is used to test all the algorithms and the OpenAI Gym platform [4] is exploited to load the MuJoCo model as well as record the average rewards and the training episodes. The model is based on TensorFlow [1]. Four indexes are selected to evaluate the performance of the algorithms, i.e., rewards, learning speed, learning stability and sensitiveness to the hyperparameters. The experiments are carried out for five typical simulated continuous control tasks as shown in Fig. 2.

3.2 Experimental Results

The experimental results of DDPG with PER are shown in Fig. 3 with comparison to the original DDPG with uniform sampling. For the inverted pendulum task, as shown in Fig. 3(1), it is clear

(1) (2) (3) (4) (5)

Figure 2: OpenAI Gym tasks in the experiments. (1)∼(5): inverted pendulum, inverted double pendulum, hopper, halfcheetah and walker.

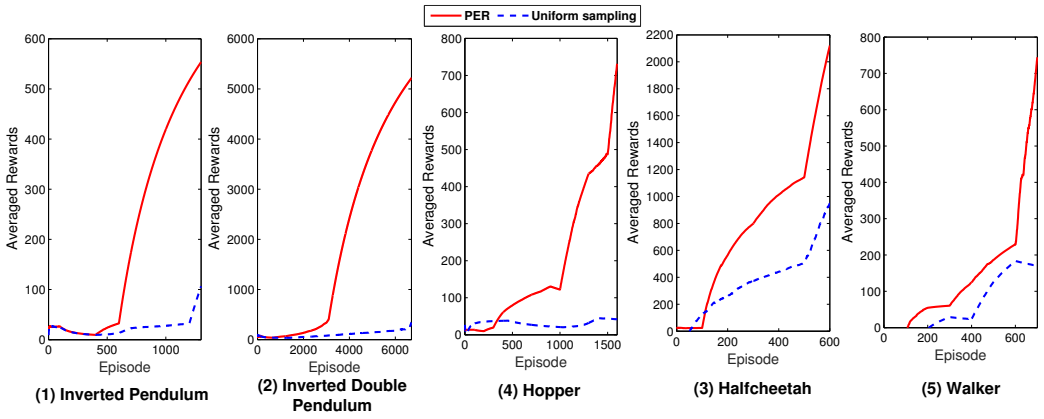

Figure 3: Learning performance regarding averaged rewards for the tasks of (1)∼(5): inverted pendulum, inverted double pendulum, hopper, halfcheetah and walker. Each experiment is run 3 times.

that DDPG with PER outperforms DDPG with uniform sampling regarding the training episodes. DDPG with PER takes about 700 episodes to achieve a total reward of around 600, while DDPG with uniform sampling takes around 1300 episodes to achieve a total reward of 100. In addition, the fewer spines occur in the reward curves of DDPG with PER, which demonstrates more stable performance of DDPG with PER.

For the other four simulated tasks, i.e., inverted double pendulum, halfcheetah, hopper and walker tasks, the experimental settings are the same as the inverted pendulum task except that in halfcheetah the maximum step in an episode is set as 500 in the first 500 episodes and then changed to 2000 to alleviate the harm of the less accurate estimation of the action-value function in early stages. As show in Fig. 3, it is clear that the reward curve of DDPG with PER is almost monotonically increasing in the whole training process while DDPG with uniform sampling is unstable. In addition, DDPG with PER achieves an average reward of 2000 and 800 in the hopper and walker tasks, respectively. It outperforms the original DDPG with uniform sampling in the same training episodes.

3.3 Comparison to More State-of-the-art Algorithms

For continuous control problems, there have been several effective algorithms, such as A3C [12], Q-Prop [5], TRPO [17] and DDPG. Especially according to [5], conservative Q-Prop using trust-region is better than the aggressive version in OpenAI Gym tasks. Hence we further test the proposed DDPG with PER (PER-DDPG) with comparison to TRPO, conservative Q-Prop algorithm using trust-region (TR-c-Q-Prop) and the original DDPG in the halfcheetah, hopper and walker tasks.

The experimental results are shown as in Table 1. The maximum average reward of DDPG with PER is almost twice as that of the other three algorithms in the hopper and walker tasks. Besides, in the

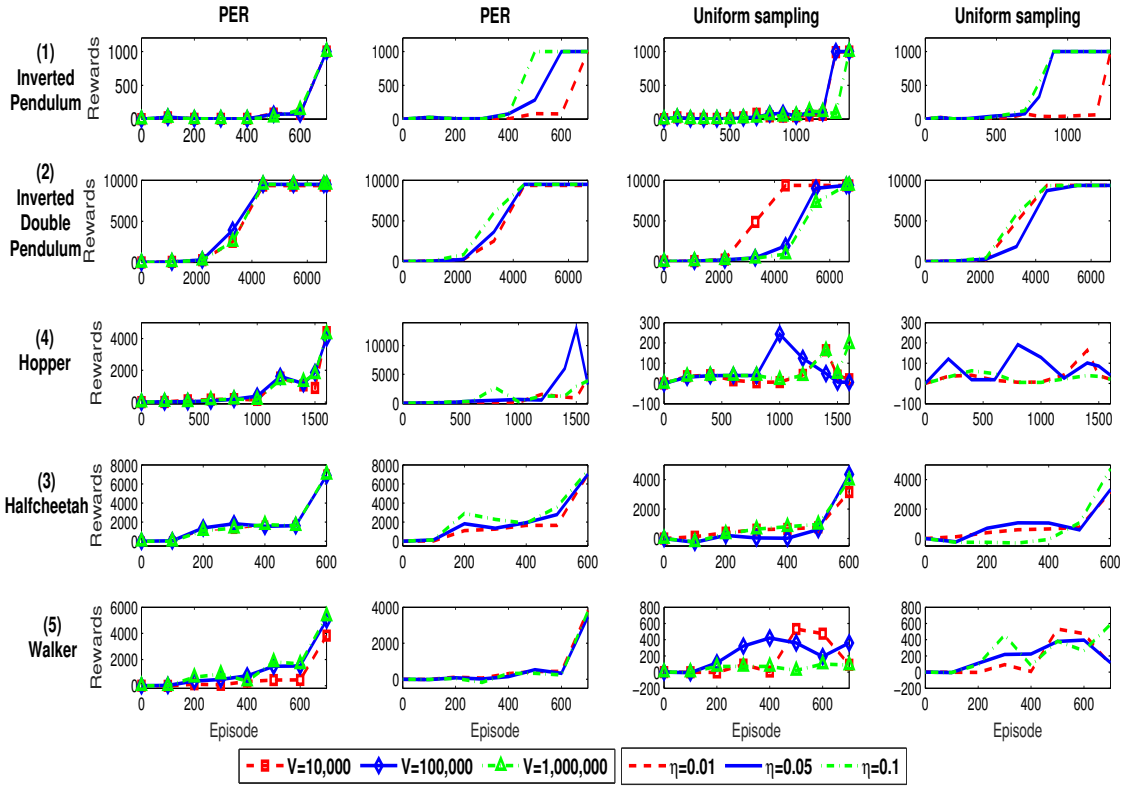

Figure 4: Learning performance regarding different hyperparameters for DDPG with PER and uniform sampling, respectively. (1)∼(5): inverted pendulum, inverted double pendulum, hopper, halfcheetah and walker. Each experiment is run 3 times.

Task	TRPO	PER-DDPG	TR-c-Q-Prop	DDPG
Halfcheetah	4734($> 10^4$)	**21000**(**500**)	4721(4350)	7490(600)
Hopper	2486(5715)	**4400**(1600)	2957(5945)	2604(**965**)
Walker	3567($> 10^4$)	**6900**(4200)	3947(2165)	3626(**2125**)

Table 1: Performance of TRPO, PER-DDPG, TR-c-Q-Prop and DDPG regarding the maximum average reward (the numbers outside the brackets) and episodes (inside the brackets).

halfcheetah task, DDPG with PER obtains far more average rewards $(21,000)$ within fewer training episodes (500) compared with the other three algorithms. All these results further demonstrate the effectiveness of DDPG with PER.

4 Conclusion

A novel DDPG with PER algorithm is proposed for continuous control, where uniform experience replay is replaced by prioritized experience replay with specific algorithms for the DDPG method. By comparison to the original DDPG with uniform sampling and other state-of-the-art methods, the experimental results show that DDPG with PER achieves better performance regarding time efficiency, stability of the learning process and robustness to the hyperparameters. Our future work will focus on prioritized experience replay beyond the criterion of TD-errors, the experience discarding mechanism and more applications on more complicated tasks such as object grasping and humanoid walking.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Michael D Abràmoff, Paulo J Magalhães, and Sunanda J Ram. Image processing with imagej. *Biophotonics international*, 11(7):36–42, 2004.

[3] Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2012.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Shixiang Gu, Timothy P. Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *ICLR*. OpenReview.net, 2017.

[6] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[7] Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, pages 316–321. IEEE, 2017.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

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

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

[15] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

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

[17] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org, 2015.

[18] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[19] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[20] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[21] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[22] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.