AMBER: Adaptive Multi-Batch Experience Replay for Continuous Action Control

Seungyul Han[1] , **Youngchul Sung**[1]

[1]Department of Electrical Engineering,
Korea Advanced Institute of Science and Technology, South Korea

{sy.han, ycsung}@kaist.ac.kr

Abstract

In this paper, a new adaptive multi-batch experience replay scheme is proposed for proximal policy optimization (PPO) for continuous action control. On the contrary to original PPO, the proposed scheme uses the batch samples of past policies as well as the current policy for the update for the next policy, where the number of the used past batches is adaptively determined based on the average importance sampling (IS) weight. The new algorithm constructed by combining PPO with the proposed multi-batch experience replay scheme maintains the advantages of original PPO such as random mini-batch sampling and small bias due to low IS weights by storing the pre-computed advantages and values and adaptively determining the mini-batch size. Numerical results show that the proposed method significantly increases the speed and stability of convergence on various continuous control tasks compared to original PPO.

1 Introduction

Reinforcement learning (RL) aims to optimize the policy for the cumulative reward in a Markov decision process (MDP) environment. SARSA and Q-learning are well-known RL algorithms for learning finite MDP environments, which store all Q values as a table and solve the Bellman equation [Watkins and Dayan, 1992; Rummery and Niranjan, 1994; Sutton and Barto, 1998]. However, if the state space of environment is infinite, all Q values cannot be stored. Deep Q-learning (DQN) [Mnih *et al.*, 2013] solves this problem by using a Q-value neural network to approximate and generalize Q-values from finite experiences, and DQN is shown to outperform the human level in Atari games with discrete action spaces [Mnih *et al.*, 2015]. For discrete action spaces, the policy simply can choose the optimal action that has the maximum Q-value, but this is not possible for continuous action spaces. Thus, policy gradient (PG) methods that parameterize the policy by using a neural network and optimize the parameterized policy to choose optimal action from the given Q-value are considered for continuous action control [Sutton *et al.*, 2000]. Recent PG methods can be classified mainly into two groups: 1) Value-based PG methods that update the policy to choose action by following the maximum distribution or the exponential distribution of Q-value, e.g., deep deterministic policy gradient (DDPG) [Lillicrap *et al.*, 2015], twin-delayed DDPG (TD3) [Fujimoto *et al.*, 2018], and soft-actor critic (SAC) [Haarnoja *et al.*, 2018], and 2) importance sampling(IS)-based PG methods that directly update the policy to maximize the discounted reward sum by using IS, e.g., trust region policy optimization (TRPO) [Schulman *et al.*, 2015a], actor-critic with experience replay (ACER) [Wang *et al.*, 2016], PPO [Schulman *et al.*, 2017]. Both PG methods update the policy parameter by using stochastic gradient descent (SGD), but the convergence speed of SGD is slow since the gradient direction of SGD is unstable. Hence, increasing sample efficiency is important to PG methods for fast convergence. Experience replay (ER), which was first considered in DQN [Mnih *et al.*, 2013], increases sample efficiency by storing old sample from the previous policies and reusing these old samples for current update, and enhances the learning stability by reducing the sample correlation by sampling random mini-batches from a large replay memory. For value-based PG methods, ER can be applied without any modification, so state-of-the-art value-based algorithms (TD3, SAC) use ER. However, applying ER to IS-based PG methods is a challenging problem. For IS-based PG methods, calibration of the statistics between the sample-generating old policies and the policy to update is required through IS weight multiplication [Degris *et al.*, 2012], but using old samples makes large IS weights and this causes large variances in the empirical computation of the loss function. Hence, TRPO and PPO do not consider ER, and ACER uses clipped IS weights with an episodic replay to avoid large variances, and corrects the bias generated from the clipping [Wang *et al.*, 2016].

In this paper, we consider the performance improvement for IS-based PG methods by reusing old samples appropriately based on IS weight analysis and propose a new adaptive multi-batch experience replay (MBER) scheme for PPO, which is currently one of the most popular IS-based PG algorithms. PPO applies clipping but ignores bias, since it uses the sample batch (or horizon) only from the current policy without replay and hence the required IS weight is not so high. In contrast to PPO, the proposed scheme uses the batch samples of past policies as well as the current policy for the update for the next policy, and preserves most advantages of PPO such as random mini-batch sampling and small bias due to

low IS weights. The details of the proposed algorithm will be explained in coming sections.

2 Background

2.1 Reinforcement Learning Problems

In this paper, we assume that the environment is an MDP. $< \mathcal{S},\ \mathcal{A},\ \gamma,\ P,\ r >$ defines a discounted MDP, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, γ is the discount factor, P is the state transition probability, and r is the reward function. For every time step t, the agent chooses an action a_t based on the current state s_t and then the environment gives the next state s_{t+1} according to P and the reward $r_t = r(s_t, a_t)$ to the agent. Reinforcement learning aims to learn the agent's policy $\pi(a_t|s_t)$ that maximizes the average discounted return $J = \mathbb{E}_{\tau_0\sim\pi}[\sum_{t=0}^{\infty}\gamma^t r_t]$, where τ_t denotes a state-action trajectory from time step t: $(s_t,\ a_t,\ s_{t+1},\ a_{t+1}, \cdots)$.

2.2 Policy Gradient Methods

Q-learning is a widely-used reinforcement learning algorithm based on the state-action value function (Q-function). The state-action value function represents the expected return of a state-action pair (s_t, a_t) when a policy π is used, and is denoted by $Q_\pi(s_t, a_t) = \mathbb{E}_{\tau_t\sim\pi}[\sum_{l=t}^{\infty}\gamma^l r_l]$ [Sutton and Barto, 1998]. However, Q-learning is focused on learning a discrete action space. To learn a continuous action environment, PG directly parameterizes the policy by a stochastic policy network $\pi_\theta(a_t|s_t)$ with parameter θ and sets an objective function $L(\theta)$ to optimize the policy based on policy gradient theorem [Sutton *et al.*, 2000]: Value-based PG methods set $L(\theta)$ as the policy follows some distribution of Q-function [Lillicrap *et al.*, 2015; Fujimoto *et al.*, 2018; Haarnoja *et al.*, 2018], and IS-based PG methods set $L(\theta)$ as the discounted return and directly updates the policy to maximize $L(\theta)$ [Schulman *et al.*, 2015a; Schulman *et al.*, 2017].

2.3 IS-based PG and PPO

At each iteration, IS-based PG such as ACER and simple PPO[1] tries to obtain a better policy $\pi_{\tilde{\theta}}$ from the current policy π_θ [Schulman *et al.*, 2015a]:

$$\begin{aligned} L_\theta(\tilde{\theta}) &\triangleq \mathbb{E}_{s_t\sim\rho_{\pi_\theta},\ a_t\sim\pi_{\tilde{\theta}}}\left[A_{\pi_\theta}(s_t, a_t)\right] \\ &= \mathbb{E}_{s_t\sim\rho_{\pi_\theta},\ a_t\sim\pi_\theta}\left[R_t(\tilde{\theta})A_{\pi_\theta}(s_t, a_t)\right], \end{aligned} \tag{1}$$

where $A_{\pi_\theta}(s_t, a_t) = Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t)$ is the advantage function, $V_\pi(s_t) = \mathbb{E}_{a_t,\tau_{t+1}\sim\pi}[\sum_{l=t}^{\infty}\gamma^l r_l]$ is the state-value function, and $R_t(\tilde{\theta}) = \frac{\pi_{\tilde{\theta}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ is the IS weight. Here, the objective function $L_\theta(\tilde{\theta})$ is a function of $\tilde{\theta}$ for given θ, and $\tilde{\theta}$ is the optimization variable. To compute $L_\theta(\tilde{\theta})$ empirically from the samples from the current policy π_θ, the IS weight is multiplied. That is, with $R_t(\tilde{\theta})$ multiplied to $A_{\pi_\theta}(s_t, a_t)$, the second expectation in (1) is over the trajectory generated by the current policy π_θ not by the updated policy $\pi_{\tilde{\theta}}$. Here, large IS weights cause large variances in (1), so ACER and PPO bound or clip the IS weight [Schulman *et al.*, 2017; Wang *et al.*, 2016]. In this paper, we use the clipped important sampling structure of PPO as our baseline. The objective function with clipped IS weights becomes

$$L_{CLIP}(\tilde{\theta}) = \mathbb{E}_{s_t,\ a_t\sim\pi_\theta}\left[\min\{R_t(\tilde{\theta})\hat{A}_t,\ \mathrm{clip}_\epsilon(R_t(\tilde{\theta}))\hat{A}_t\}\right], \tag{2}$$

where $\mathrm{clip}_\epsilon(\cdot) = \max(\min(\cdot, 1+\epsilon), 1-\epsilon)$ with clipping factor ϵ, and $\hat{A}_t$ is the sample advantage function estimated by the generalized advantage estimator (GAE) [Schulman *et al.*, 2015b]:

$$\hat{A}_t = \sum_{l=0}^{N-n-1}(\gamma\lambda)^l\delta_{t+l}, \tag{3}$$

where N is the number of samples in one iteration (horizon), $\delta_t = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$ with the state-value network $V_w(s_t)$ which approximates $V_{\pi_\theta}(s_t)$. Then, PPO updates the state-value network to minimize the square loss:

$$L_V(w) = (V_w(s_t) - \hat{V}_t)^2, \tag{4}$$

where $\hat{V}_t$ is the TD(λ) return [Schulman *et al.*, 2017].

In [Schulman *et al.*, 2017], for continuous action control, a Gaussian policy network is considered, i.e.,

$$a_t \sim \pi_\theta(\cdot|s_t) = \mathcal{N}(\mu(s_t;\phi), \sigma^2), \tag{5}$$

where $\mu(s_t;\phi)$ is the mean neural network whose input is s_t and parameter is ϕ; σ is a standard deviation parameter; and thus $\theta = (\phi, \sigma)$ is the overall policy parameter.

3 Related Work

3.1 Experience Replay on Q-Learning

Q-learning is off-policy learning which only requires sampled tuples to compute the TD error [Sutton and Barto, 1998]. DQN uses the ER technique [Lin, 1993] that stores old sample tuples (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathbf{R}$ and updates the Q-network with the average gradient of the TD error computed from a mini-batch uniformly sampled from $\mathbf{R}$. Value-based PG methods adopt this basic ER of DQN. As an extension of this basic ER, [Schaul *et al.*, 2015] considered prioritized ER to give a sampling distribution on the replay instead of uniform random sampling so that samples with higher TD errors are used more frequently to obtain the optimal Q faster than DQN. [Liu and Zou, 2017] analyzed the effect of the replay memory size on DQN and proposed an adaptive replay memory scheme based on the TD error to find a proper replay size for each discrete task. It is shown that this adaptive replay size for DQN enhances the overall performance.

3.2 Experience Replay on IS-based PG

ER can be applied to IS-based PG for continuous action control to increase sample efficiency. As seen in (1), the multiplication of the IS weight $R_t(\tilde{\theta}) = \frac{\pi_{\tilde{\theta}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ is required to use the samples from old policies for current policy update. In case that ER uses samples from many previous policies, the required IS weight is very large and this induces bias even

[1] We only consider simple PPO without adaptive KL penalty since simple PPO has the best performance on continuous action control tasks [Schulman *et al.*, 2017].

though clipping is applied. The induced bias makes the learning process unstable and disturbs the computation of the expected loss function. Thus, ACER uses ER with bias correction, and proposes an episodic ER scheme that samples and stores on the basis of episodes because it computes an off-policy correction Q-function estimator which requires whole samples in a trajectory as Algorithm 3 in [Wang *et al.*, 2016].

4 Multi-Batch Experience Replay

4.1 Batch Structures of ACER and PPO

Before introducing our new replay scheme, we compare the batch description of ACER and PPO for updating the policy, as shown in Fig. 1. ACER uses an episodic ER to increase sample efficiency. In the continuous action case, ACER stores trajectories from $V = 100$ previous policies and each policy generates a trajectory of $M = 50$ time steps in replay memory $\mathbf{R}$. For each update period, ACER chooses $W \sim Poisson(4)$ random episodes from $\mathbf{R}$ to update the policy. Then, different statistics among the samples in the replay causes bias, and the episodic sample mini-batch is highly correlated. On the other hand, PPO does not use ER but collects a single batch of size $N = 2048$ time steps from the current policy. Then, PPO draws a mini-batch of size $M = 64$ randomly and uniformly from the single batch; updates the policy to the direction of the gradient of the empirical loss computed from the drawn mini-batch:

$$\hat{L}_{CLIP}(\tilde{\theta}) = \frac{1}{M}\sum_{m=0}^{M-1} \min\{R_m(\tilde{\theta})\hat{A}_m,\ \mathrm{clip}_\epsilon(R_m(\tilde{\theta}))\hat{A}_m\}; \tag{6}$$

and updates the value network to the direction of the negative gradient of

$$\hat{L}_V(w) = \frac{1}{M}\sum_{m=0}^{M-1}(V_w(s_m) - \hat{V}_m)^2, \tag{7}$$

where $\hat{A}_m$, $R_m(\tilde{\theta})$, $V_w(s_m)$, and $\hat{V}_m$ are values corresponding to the m-th sample in the mini-batch [Schulman *et al.*, 2017]. This procedure is repeated for 10 epochs for a single batch of size N. Here, 1 epoch means that we use every samples in the batch to update it once. In other words, PPO updates the policy by drawing $10 \cdot N/M$ mini-batches. Note that PPO uses current samples only, so it can ignore bias because the corresponding IS weights do not exceed the clipping factor mostly. Furthermore, the samples in a mini-batch drawn uniformly from the total batch of size N in PPO have little sample correlation because they are scattered over the total batch. However, PPO discards all samples from all the past policies except the current policy for the next policy update and this reduces sample efficiency.

4.2 The Proposed Multi-Batch Experience Replay Scheme

We now present our MBER scheme suitable to PPO-style IS-based PG, which increases sample efficiency, maintains random mini-batch sampling to diminish the sample correlation, and reduces the IS weight to avoid bias. We apply our MBER scheme to PPO to construct an enhanced algorithm named PPO-MBER, which includes PPO as a special case.

In order to obtain the next policy, the proposed scheme uses the batch samples of $L-1$ past policies and the current policy, whereas original PPO uses the batch samples from only the current policy, as illustrated in Fig. 1. In Fig. 1, the different colors indicate samples from the different sample batches. The stored information for MBER in the replay memory $\mathbf{R}$ is as follows. To compute the required IS weight $R_t(\tilde{\theta}) = \frac{\pi_{\tilde{\theta}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ for each sample in a random mini-batch, MBER stores the statistical information of every sample in $\mathbf{R}$. Under the assumption of a Gaussian policy network (5), the required statistical information for each sample is the mean $\mu_t := \mu(s_t;\phi)$ and the standard deviation σ. Furthermore, MBER stores the pre-calculated estimated advantage $\hat{A}_t$ and target value $\hat{V}_t$ of every sample in $\mathbf{R}$. Thus, MBER stores the overall sample information $(s_t,\ a_t,\ \hat{A}_t,\ \hat{V}_t, \mu_t, \sigma)$ regarding the batch samples from the most recent L policies, as described in Fig. 1. The storage of the statistical information (μ_t, σ) and the values $(\hat{A}_t, \hat{V}_t)$ in addition to (s_t, a_t) for every sample in the replay memory makes it possible to draw a random mini-batch from $\mathbf{R}$ not a trajectory like in ACER. Since the policy at the i-th iteration generates a batch of N samples, we can rewrite the stored information by using two indices $i = 0, 1, \cdots$ and $n = 0, 1, \cdots, N-1$ (such that time step $t = iN + n$) as $\{B_{i-L+1}, \cdots, B_i\}$ from the most recent L policies $i, i-1, \cdots, i-L+1$, where

$$B_i = (s_{i,n},\ a_{i,n},\ \hat{A}_{i,n},\ \hat{V}_{i,n}, \mu_{i,n}, \sigma_i), \quad n = 0, \cdots, N-1. \tag{8}$$

In addition to using the batch samples from most recent L policies, MBER enlarges the mini-batch size by L times compared to that of original PPO, to reduce the average IS weight. If we set the mini-batch size of MBER to be the same as that of PPO with the same epoch, then the number of updates of PPO-MBER is L times larger than that of PPO. Then, the updated policy statistic is too much different from the current policy statistic and thus the average IS weight becomes large as L increases. This causes bias and is detrimental to the performance. To avoid this, we enlarge the mini-batch size of MBER by L times and this reduces the average IS weight by making the number of updates the same as that of PPO with the same epoch. In this way, MBER can ignore bias without much concern, because its IS weight is similar to that of PPO. Fig. 2 shows the average IS weight[2] of all sampled mini-batches at each iteration when $M = 64$ and $M = 64 \times L$. It is seen that PPO-MBER maintains the same level of the important sampling weight as original PPO.

5 Adaptive Batch Drop

PPO-MBER can significantly enhance the overall performance compared to PPO by using the MBER scheme, as seen in Section 7. However, we observe that the PPO-MBER performance for each task depends on the replay length L, and hence the choice of L is crucial to PPO-MBER. For the two

[2]Actually, we averaged $\mathrm{abs}(1 - R_m(\tilde{\theta})) + 1$ instead of $R_m(\tilde{\theta})$ to see the degree of deviation from 1.

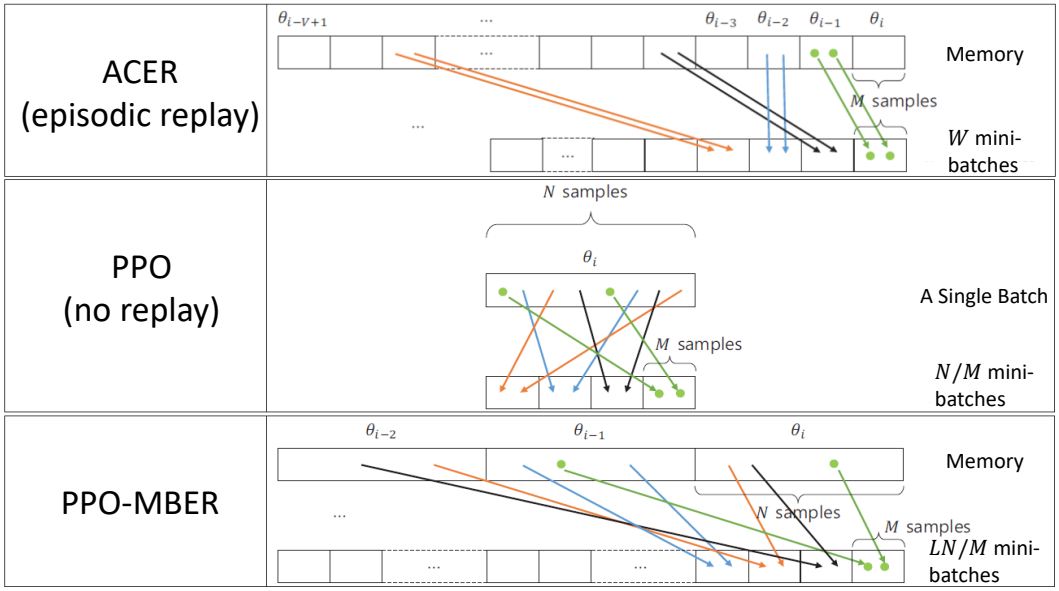

Figure 1: Batch construction of ACER, PPO and PPO with the proposed MBER (PPO-MBER): $N = 8$ and $M = 2$.

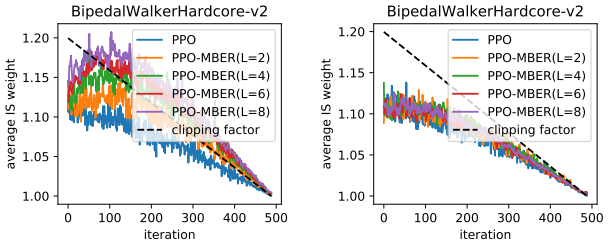

Figure 2: Average IS weight of BipedalWalkerHardcore for PPO-MBER: (Left) $M = 64,\ \epsilon = 0.2$ and (Right) $M = 64L,\ \epsilon = 0.2$

extreme examples, Pendulum and Humanoid whose dimensions are 1 and 17 respectively, note that the performance of Pendulum is proportional to the replay length but the performance of Humanoid is inversely proportional to the replay length as in Fig. 5. To analyze the cause of this phenomenon, we define the batch average IS weight between the old policy θ_{i-l} and the current policy θ_i as

$$R'_{i,l} = \frac{1}{N}\sum_{n=1}^{N-1} 1 + \text{abs}\left(1 - \frac{\pi_{\theta_i}(a_{i-l,n}|s_{i-l,n})}{\pi_{\theta_{i-l}}(a_{i-l,n}|s_{i-l,n})}\right), \quad (9)$$

where $a_{i-l,n},\ s_{i-l,n} \in B_{i-l}$. Note that this is different from the average of $1 + \text{abs}(1 - R_m(\tilde{\theta}))$ which depends on the updating policy $\pi_{\tilde{\theta}}$ not the current policy π_{θ_i}. Fig. 3 shows $R'_{i,l}$ of PPO-MBER for Pendulum and Humanoid tasks. It is seen that $R'_{i,l}$ increases as l increases, because the batch statistic is updated as iteration goes on. It is also seen that Humanoid has "large" $R'_{i,l}$ and Pendulum has "small" $R'_{i,l}$. From the two examples, it can be inferred that the batch samples B_{i-l} with large $R'_{i,l}$ are too old for updating $\tilde{\theta}$ at the current policy parameter θ_i and can harm the performance, as in the case of Humanoid. On the other hand, if $R'_{i,l}$ is small, more old samples can be used for update and this is beneficial to the performance. Therefore, it is observed in Table 1 that original PPO, i.e., $L = 1$ is best for Humanoid and PPO-MBER with $L = 8$ is best for Pendulum (In Table 1, we only consider L up to 8). Exploiting this fact, we propose an adaptive MBER (AMBER) scheme which adaptively chooses the batches to use for update from the replay memory. In the proposed AMBER, we store the batch samples from policies $\theta_i, \theta_{i-1}, \cdots, \theta_{i-L+1}$, but use only the batches B_{i-l}'s whose $R'_{i,l}$ is smaller than the batch drop factor ϵ_b. Since $R'_{i,l}$ increases as time goes, AMBER uses the most recent L' sample batches whose $R'_{i,l}$ is less than ϵ_b. It is seen in Table 1 that PPO-AMBER well selects the proper replay length.

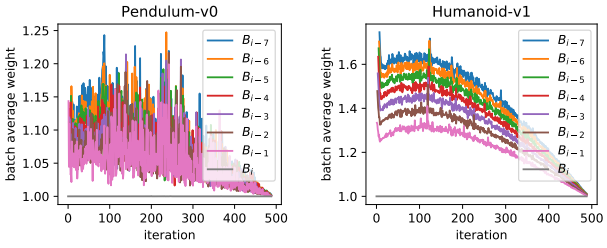

Figure 3: Batch average weight $R'_{i,l}$ of PPO-MBER ($L = 8,\ \epsilon = 0.4$): (Left) Pendulum and (Right) Humanoid

6 The Algorithm

Now, we present our proposed algorithm PPO-(A)MBER that maximizes the objective function $\hat{L}_{CLIP}(\tilde{\theta})$ in (6) for continuous action control. We assume the Gaussian policy network $\pi_{\tilde{\theta}}$ in (5) and the value network V_w. (They do not share pa-

rameters.) We define the overall parameter $\tilde{\theta}_{ALL}$ combining the policy parameter $\tilde{\theta}$ and the value parameter w. The objective function is given by [Schulman *et al.*, 2017]

$$\hat{L}(\tilde{\theta}_{ALL}) = \hat{L}_{CLIP}(\tilde{\theta}) - c_v \hat{L}_V(w), \tag{10}$$

where $\hat{L}_{CLIP}(\tilde{\theta})$ is in (6), $\hat{L}_V(w)$ is in (7), and c_v is a constant (we use $c_v = 1$ in the paper). The detailed algorithm is summarized as A.1. in supplementary material.

7 Experiments

7.1 Environment Description and Parameter Setup

To evaluate our ER scheme, we conducted numerical experiments on OpenAI GYM environments [Brockman *et al.*, 2016]. We selected continuous action control environments of GYM: Mujoco physics engines [Todorov *et al.*, 2012] (HalfCheetah, Huamnoid, InvertedDobulePendulum, Pendulum, Swimmer, Walker2d), classical control (Pendulum), and Box2D (BipedalWalker, BipedalWalkerHardcore)[Catto, 2011]. The dimensions of state and action for each task are described in Section A.2. of supplementary material. We provide the supplementary material for source code, various descriptions and additional experimental results which is available at: https://www.dropbox.com/sh/bvvyvosebfs96to/AAB0wK9qm3yNC7OAri01s76oa?dl=0.

We used PPO baselines of OpenAI [Dhariwal *et al.*, 2017] and compared the performance of PPO-(A)MBER with various replay lengths $L = 1$ (PPO), $2, 4, 6, 8$ on various Mujoco continuous action control tasks. The detailed parameters of PPO/PPO-MBER are described in Section A.2. of supplementary material. Adam step size β and clipping factor ϵ decay linearly as time-step goes on from the initial values to 0. The Gaussian mean network and the value network are feed-forward neural networks that have 2 hidden layers of size 64 like in [Schulman *et al.*, 2017]. For all the performance plots/tables in this paper, we averaged 10 simulations per each task with random seeds. For each performance plot, the X-axis is time step, the Y-axis is the average return of the lastest 100 episodes at each time step, and the line in the plot is the mean score of 10 random seeds. For each performance table, results are described as the mean $\pm$ one standard deviation of 10 seeds and the best scores are in boldface.

7.2 Performance and Ablation Study of PPO-MBER

In [Schulman *et al.*, 2017], the optimal clipping factor is 0.2 for PPO. However, it depends on the task set. Since our task set is a bit different from that in [Schulman *et al.*, 2017], we first evaluated the performance of PPO and PPO-MBER by sweeping the clipping factor from $\epsilon = 0.2$ to $\epsilon = 0.7$ whose results are summarized in Section A.3. of supplementary material. We observe that loosening the clipping factor a bit is beneficial for both PPO and PPO-MBER in the considered set of tasks, especially PPO-MBER with larger replay lengths. This is because loosening the clipping factor reduces the bias and increases the variance of the loss expectation, but a larger mini-batch of PPO-MBER reduces the variance by offsetting[3]. However, too large a clipping factor harms the performance. As a results, the best clipping factor is $\epsilon = 0.3$ for PPO and $\epsilon = 0.4$ for PPO-MBER. The detailed final score for each task for PPO and PPO-MBER with the best clipping factors is given in Table. 1. PPO sometimes fails to perfectly learn the environment as the number of random seeds increases from 3 of [Schulman *et al.*, 2017] to 10 of ours, but PPO-MBER learns all environments more stably since it averages more samples based on enlarged mini-batches, so there is a large performance gap in this case. It is observed that in most environments, PPO-MBER with proper L significantly enhances both the final performance results as compared to PPO.

We then investigated the performance of random mini-batches versus episodic mini-batches for PPO-MBER with $L = 2$, $\epsilon = 0.4$. In the episodic case, we draw each mini-batch by picking a consequent trajectory of size M like ACER. Fig. 4 shows the results under HalfCheetah and Swimmer environments. It is seen that there is a notable performance gap between the two cases. This means that random mini-batch drawing from the replay memory storing precomputed advantages and values in MBER has the advantage of reducing the sample correlation in a mini-batch and this is beneficial to the performance.

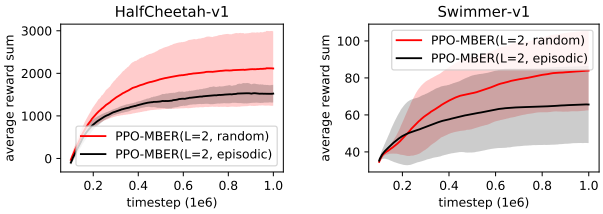

Figure 4: Performance comparison of HalfCheetah and Swimmer for PPO-MBER ($L = 2$, $\epsilon = 0.4$) with uniformly random mini-batch and episodic mini-batch

7.3 Performance of PPO-AMBER

From the ablation study in the previous subsection, we set $\epsilon = 0.4$, which is good for a wide range of tasks, and set $L = 8$ as the maximum replay size for PPO-AMBER. PPO-AMBER shrinks the mini-batch size as $M = 64 \times$ # of active batches as described in previous section. The batch drop factor ϵ_b is linearly annihilated from the initial value to zero as time step goes on. To search for optimal batch drop factor, we again sweep the initial value of the batch drop factor from 0.1 to 0.3 and conclude that $\epsilon_b = 0.25$ is the best. In addition, Fig. 6 shows the number of active batches of PPO-AMBER for various batch drop factors for Pendulum and BipedalWalkerHardcore tasks, and $\epsilon_b = 0.25$ can select appropriate number of batches for both tasks. Fig. 5 show the performance of PPO ($\epsilon = 0.3$), PPO-MBER ($\epsilon = 0.4$), and PPO-AMBER

[3]One may think that PPO with a large mini-batch size also has the same effect, but enlarging the mini-batch size without increasing the replay memory size increases the sample correlation and reduces the number of updates too much, so it is not helpful for PPO.

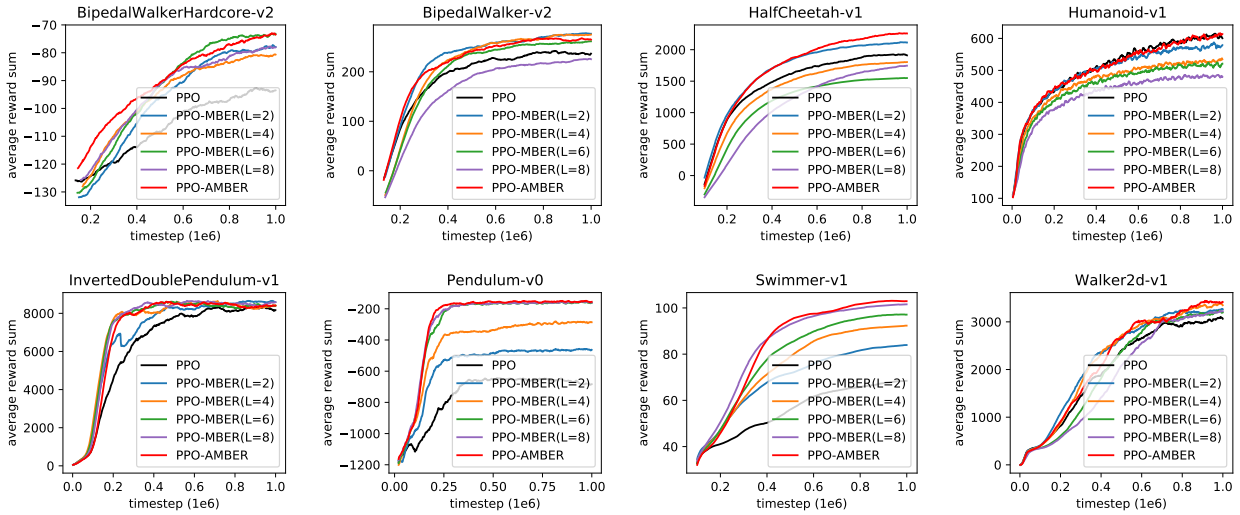

Figure 5: Performance comparison on continuous control tasks for parameter-tuned setup

Table 1: Average return of final 100 episodes of parameter-tuned PPO, PPO-MBER and PPO-AMBER

	PPO	PPO-MBER ($L=2$)	PPO-MBER ($L=4$)	PPO-MBER ($L=6$)	PPO-MBER ($L=8$)	PPO-AMBER
BipedalWalker	236 ± 26	$\mathbf{276 \pm 16}$	275 ± 19	261 ± 13	225 ± 75	265 ± 16
BipedalWalkerHardcore	-93.5 ± 10.8	-77.9 ± 20.8	-80.7 ± 18.4	$\mathbf{-73.2 \pm 18.8}$	-78.1 ± 15.8	-73.5 ± 10.7
HalfCheetah	1910 ± 778	2113 ± 874	1803 ± 611	1549 ± 562	1745 ± 529	$\mathbf{2258 \pm 1039}$
Humanoid	600 ± 43	578 ± 41	534 ± 34	521 ± 24	480 ± 14	$\mathbf{613 \pm 67}$
InvertedDoublePendulum	8167 ± 630	$\mathbf{8588 \pm 271}$	8407 ± 305	8402 ± 209	8587 ± 208	8406 ± 363
Pendulum	-683 ± 494	-463 ± 391	-286 ± 306	-161 ± 7	-160 ± 8	$\mathbf{-155 \pm 12}$
Swimmer	68.4 ± 19.0	83.9 ± 21.5	92.3 ± 21.5	97.1 ± 24.5	101.6 ± 26.2	$\mathbf{102.9 \pm 26.4}$
Walker2d	3065 ± 532	3264 ± 577	3348 ± 498	3196 ± 429	3223 ± 445	$\mathbf{3415 \pm 416}$

($L = 8$, $\epsilon = 0.4$, $\epsilon_b = 0.25$) on various tasks. Additional performance scores of other Mujoco environments and various parameter setup are given by A.3. in supplementary material. It is seen that PPO-AMBER with $\epsilon_b = 0.25$ automatically selects almost optimal replay size from $L = 1$ to $L = 8$. So, with PPO-AMBER one need not be concerned about designing the replay memory size for the proposed ER scheme, and it significantly enhances the overall performance. We additionally compared the performance of PPO-AMBER with other PG methods (TRPO, ACER) in Section A.4. of supplementary material, and it is observed that PPO-AMBER outperforms both TRPO and ACER.

7.4 Further Discussion

It is observed that PPO-AMBER enhances the performance of tasks with low action dimensions compared to PPO by using old sample batches, but it is hard to improve tasks with high action dimensions. This is because higher action dimensions yields larger IS weights. Hence, we provide an additional IS analysis for those environments in Section A.5. of supplementary material. The analysis suggests that AMBER fits to low action dimensional tasks or sufficiently small learning rates to prevent the IS weights from becoming too large.

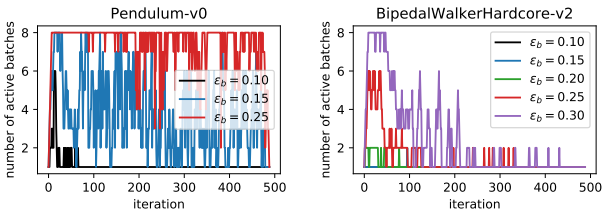

Figure 6: The number of active batches of Pendulum and BipedalWalkerHardcore for PPO-AMBER with various ϵ_b

8 Conclusion

In this paper, we have proposed a MBER scheme for PPO-style IS-based PG, which significantly enhances the speed and stability of convergence on various continuous control tasks (Mujoco tasks, classic control, and Box2d on OpenAI GYM) by 1) increasing the sample efficiency without causing much bias by fixing the number of updates and reducing the IS weight, 2) reducing the sample correlation by drawing random mini-batches with the pre-computed and stored advantages and values, and 3) dropping too old samples in the replay memory adaptively. Numerical results show that PPO-AMBER significantly improves original PPO.

Acknowledgements

This work was supported in part by the ICT R&D program of MSIP/IITP (2016-0-00563, Research on Adaptive Machine Learning Technology Development for Intelligent Autonomous Digital Companion) and in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Science and ICT) (NRF-2017R1E1A1A03070788).

References

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

[Catto, 2011] Erin Catto. Box2d: A 2d physics engine for games, 2011.

[Degris *et al.*, 2012] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[Dhariwal *et al.*, 2017] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

[Fujimoto *et al.*, 2018] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[Lillicrap *et al.*, 2015] 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.

[Lin, 1993] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[Liu and Zou, 2017] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. *arXiv preprint arXiv:1710.06574*, 2017.

[Mnih *et al.*, 2013] 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.

[Mnih *et al.*, 2015] 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):529–533, 2015.

[Rummery and Niranjan, 1994] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.

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

[Schulman *et al.*, 2015a] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[Schulman *et al.*, 2015b] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, Cambridge, MA, 1998.

[Sutton *et al.*, 2000] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

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

[Wang *et al.*, 2016] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.