An Evaluation of DDPG, TD3, SAC, and PPO: Deep Reinforcement Learning Algorithms for Controlling Continuous System

Shijie Liu

Department of Electrical and Electronics Engineering, The Hong Kong Polytechnic University, Hong Kong SAR, 100872, China
22100838d@connect.polyu.hk

Abstract. Continuous systems are physical systems that can be stimulated by continuous and analog variables. The parameters or variables are within a range of values. An excellent continuous controlling policy enables the system to act appropriately and smoothly without much intervention, which can be useful in robotics, self-driving, industries, etc. The DRL algorithm has extensive applications in continuous systems control. This essay will explore the performance of four DRL algorithms, that is the Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO) by using environment from the four of environments in Mujoco in Gym. Comparative experiments are done, and the highest rewards and the required number of iterations to converge are compared. The result of comparative experiments illustrates that these DRL algorithms can learn relatively appropriate policies in continuous controlling tasks. In particular, TD3 and SAC were found to be able to learn the controlling policy more effectively. Further research is needed to find better ways to adjust hyperparameters.

Keywords: Reinforcement Learning, Deep Learning, Continuous System Controlling.

1 Introduction

Deep Reinforcement Learning (DRL) is a powerful tool for training agents to implement complex tasks. It can be applied in many robotic systems. Deep reinforcement learning allows robotic systems to adapt to different environmental changes as well as learn the best behavior to take in different situations, enhancing its independence and reducing human interventions. DRL has been proven to be able to solve challenges such as noise, and variability in tasks [1].

Robotic hardware is becoming more and more complex nowadays, which is calling for better stimulation tools. Multi-Joint dynamics with Contact (Mujoco) as a physics engine that can successfully imitate contacts and illustrate the state in joint coordinates was developed [2]. Fast and precise imitation can be achieved through

B. H. Ahmad (ed.), *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, Advances in Intelligent Systems Research 180,
https://doi.org/10.2991/978-94-6463-370-2_3

Mujoco. In the Mujoco part of the Gym documentation, several high-dimensional continuous robotic systems are included, such as an ant-like robot, Half-Cheetah (cat-like robot), and humanoid robot [3]. Gaussian Noise was added to improve the stochasticity. Thus, these environments enable us to better stimulate real robotic scenarios and test the robustness of the DRL algorithms [3].

This paper explores how the different Mujoco environments have an impact on the performance of Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Soft Actor Critic (SAC), and Proximal Policy Optimization (PPO). How these four algorithms perform in specific environments will be discussed. What's more, potential drawback that may affect the performance of PPO is also speculated. Half-Cheetah-v4, Swimmer-v4, Ant-v4, Hopper-v3 are selected. Table 1 shows the details of these four environments.

Table 1. Descriptions of the four environments.

	Size of observation Space	Size of action Space	Rewards
Half-Cheetah	17	6	Forward reward; Too large action penalty
Swimmer	8	2	Forward reward; Too large action penalty
Hopper	11	3	Healthy reward, forward; Too large action penalty
Ant	27	8	Healthy reward, forward reward, too large action penalty, contact cost

2 Literature Review

The action space of many tasks especially those physical control tasks are high dimensional as well as continuous [3]. By absorbing advantages from the Actor-Critic [4] and Deterministic Policy Gradient (DPG) as well as the DQN, the authors created the Deep Deterministic Policy Gradient (DDPG) [3, 5]. The DDPG shows a good performance in dealing with continuous controlling. However, DDPG suffers from the overestimation of Q values [6]. Therefore, Fujimoto, etc. use the clipped double Q learning methods so that minimum Q value can be adopted and so that it tends to underestimate [7]. Another algorithm that is similar to the TD3 is the Soft Actor-Critic. It adopts a maximum entropy framework, which can encourage exploration and improve robustness [8]. The objective of maximum entropy reinforcement learning is to achieve the highest possible cumulative reward while also ensuring that the policy's entropy is maximized. The aforementioned algorithms are all off-policy algorithms. As for on-policy algorithms, PPO is also suitable for continuous system-controlling tasks. As an on-policy algorithm, it is aimed at finding the best improvement step without stepping back so previously which may cause the

performance collapse. It avoids the over-complicated implementation in TRPO while can perform not worse than it [9].

There are several efforts to compare the performance of algorithms in continuous system-controlling tasks. Existing work compares the performance of DDPG, SAC, TRPO, PPO, and TD3 by using the v1 version of Mujoco in Gym, such as Walker2d-v1, and Ant-v1 [8]. The author experimented to show the outperformance of SAC compared to other algorithms. Besides, experiments are conducted involving the DDPG, TD3, and TAD3 without clarifying which specific environments were used [6]. Previous work compares DDPG, SAC, TD3, and DARC in OpenAI Gym and PyBullet Gym [10].

Overall, this research shows that the DDPG, TD3, SAC, and PPO have a strong ability to learn policy to control the continuous system. However, there are still several challenges in this area. For example, it is not clear which algorithm is best suited for which task and how different environments can affect the performances of different algorithms. Further research is needed to tackle these issues and enable us to obtain a better understanding of RL in controlling continuous systems in Mujoco.

3 Method

The continuous controlling task in the Mujoco is a Markov Decision Process. That is both stochasticity and determinism are contained in continuous controlling tasks in Mujoco. The observation of states, transition after actions, and rewards include randomness due to the noise and other settings of the environments but is partly controllable since the actions can be selected by agent itself [11]. The interaction between agents and the environment can be denoted as a Markov chain, which contains the state, action, transition, and rewards. The Markov chain is the experience that enables the deep reinforcement learning algorithms to learn the optimal policy. Deep reinforcement learning algorithms automatically extract the best policy from Markov chain to maximize the accumulated rewards [12]. Details of four deep reinforcement learning algorithms used in this paper will be included below.

3.1 Deep Deterministic Policy Gradient (DDPG)

DDPG has an actor-critic architecture that includes a Q-function (critic) and a policy (actor). The actor's role is to choose actions based on the current policy, while the critic assesses the actions taken by the actor and provides feedback to update the policy. The reward is the feedback from the environment to evaluate how well the agent performs in a specific task according to the specific action and state, while the critic network calculates the Q value based on the action selected by the policy. As an off-policy algorithm, DDPG can sample the data and learn from its experiences from its replay buffer.

The Bellman equation can describe the best Q function, which is written as Q*(s, a).

$$Q^*(s,a) = E_{s'\sim P}\left[r(s,a) + \gamma \max_{a'} Q^*\left(s', a'\right)\right] \quad (1)$$

where $r(s,a)$ means the reward obtained after implementing action a in state s, γ represents the discount factor to determine the weight of future reward. The higher value of γ, the more emphasis that future reward is given. Vice versa. *s* and *a* is the current state and action, *s'* is obtained from the distribution *P(·|s, a)* predicted by the policy which shows the probability of entering *s'* if execute *a* in *s*. *a'* is the action chosen in state *s'*. To determine how close the current Q-function is to the best Q-function, the TD error is calculated. TD error is to compute the difference between the target Q value and the current Q value.

$$TDerror = R + \gamma Q\left(s', a'\right) - Q(s,a) \quad (2)$$

The target Q value here is $R + \gamma Q\left(s', a'\right)$ and current Q value denote as $Q(s,a)$. And the loss function will be defined as the square of TD error: Mean Squared Bellman Error (MSBE).

The above is similar to DQN. However, tricks are applied. Target Q value can be denoted as:

$$r + \gamma(1-d)\max\left(Q_\phi\left(s', a'\right)\right) \quad (3)$$

The d here denotes whether the agent arrives at the end state. If it does, d will be 1, otherwise, it will be 0. Thus *a'* is selected by the target policy network and Q here should be the target Q function.

The target network is updated to follow the current network, which is called a soft update.

$$\phi' \leftarrow \rho\phi + (1-\rho)\phi'(critic) \text{ or } \theta' \leftarrow \rho\theta + (1-\rho)\theta'(actor) \quad (4)$$

where ϕ or θ is the parameter of the target network and ϕ' or θ' is the parameter of the current network. Besides, ρ is a hyperparameter that controls how fast the target network updates. The smaller ρ is, the slower it updates. If there's no target network, that is a current network used when computing Q values, the minimization of MSBE will be erratic. During training, the aim is to make the target Q neural network closer to the current network with delay to improve the stability.

In addition, the *a'* is not from the replay buffer but is predicted by the target policy network instead, which can be denoted as $\pi'_\theta(s')$. Therefore, the MSBE for DDPG is

$$J_Q(\phi) = E_{(s,a,r,s',d)\sim D}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q'_\phi\left(s', \pi'_\theta(s')\right)\right)^2\right)\right] \quad (3)$$

The D is the dataset of transition and ϕ is the parameters of Q function, d is a Boolean value that describes whether this episode is ended.

In terms of policy learning, the Q-function is used to evaluate the action chosen by the policy. As a result, the objective function can be expressed as follows:

$$J(\theta) = \max_{\theta} E\left[Q_{\phi}\left(s, \pi_{\theta}(s)\right)\right] \tag{4}$$

Gradient descent will be applied on the policy network and Q function network while two target networks only apply the soft update, no gradient descent.

3.2 Twin Delayed DDPG (TD3)

TD3 can solve several disadvantages of DDPG. The first is the problem that the policy is very likely to exploit a wrongly approximated sharp peak for some actions by the Q function. A target policy smoothing can address it. Its formula is given by:

$$a^{'}\left(s^{'}\right) = \text{clip}\left(\mu_{\theta_{\text{targ}}}\left(s^{'}\right) + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \epsilon \sim \mathcal{N}(0, \sigma) \tag{5}$$

A limited amount of noise ϵ which satisfies the normal distribution is added to each dimension of action predicted by the target policy. And the whole action is limited to a_{Low} and a_{High}.

Another issue of DDPG is the overestimation of the target Q value. Therefore, clipped double Q learning is adopted. Two different target Q functions and Q functions are defined. When computing the target Q values, the minimum result is used. The MSBE for two Q functions both use the minimum target Q values. As for the policy learning side, one Q function is randomly selected from two Q functions. And similar to DDPG, Q function is used to evaluate the policy. The soft update is also applied to update two target Q functions and target policy, which have the same principle as DDPG.

3.3 Soft Actor-Critic (SAC)

SAC has a similar structure compared to the TD3. But there are some differences. The first is that TD3 uses a deterministic policy. The output of the policy network will be a certain vector which is the action. SAC uses a stochastic policy. Standard deviation and the mean of a distribution (often a normal distribution) can be obtained from the output of the policy network. Then the action can be taken by sampling from the distribution.

The most important characteristic of SAC is its entropy regularization. The entropy can encourage exploration. The agent will consider not only the accumulated future reward but also the entropy. Thus, the policy can be written as:

$$\pi^{*} = \text{argmax}_{\pi} \mathrm{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^{t}\left(\mathrm{r}(\mathrm{s}_{t}, \mathrm{a}_{t}, \mathrm{s}_{t+1}) + \alpha \mathrm{H}\left(\pi(\cdot|\mathrm{s}_{t})\right)\right)\right] \tag{6}$$

Here the entropy $\mathrm{H}\left(\pi(\cdot|\mathrm{s}_{t})\right) = -\log \pi\left(a_{t}|s_{t}\right)$, t is the time step, which means how many actions does the agents take. τ represents the trajectory, which records the sequence of states, actions, rewards and it is also sampled from the policy. α can control how the entropy is emphasized. Therefore, the Bellman equation will be changed to

$$Q^{\pi}(s,a) = E_{s' \sim P\, a' \sim \pi}\left[R(s,a,s') + \gamma\left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \quad (7)$$

Where s' is sampled from the transition probability P, that is probability of transitioning from one state to another. And a' is sampled from the policy. SAC also has two different target Q function, which has a similar structure to TD3. Thus, the MSBE loss is

$$L(\theta_i, \mathcal{D}) = E_{(s,a)\sim\mathbb{D}}\left[\left(Q_{\theta_i}(s,a) - (r + \gamma(1-d)\left(\min_{i=1,2} Q_{\theta_i^{target}}(s',a') - \alpha \log \pi_{\phi}(a'|s')\right)\right)^2\right] \quad (8)$$

The a' here is the selected by the policy network. There's no target policy network in SAC. As for the policy learning side, entropy also needs to be considered. It is given that $V^{\pi}(s) = E_{a\sim\pi}[Q^{\pi}(s,a) - \alpha \log \pi(a|s)]$.

3.4 Proximal Policy Optimization (PPO)

PPO has an advantage actor-critic structure (A2C). The objective function of policy learning sides is as below:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}A(s,a), \text{clip}\left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A(s,a)\right) \quad (9)$$

$\pi_{\theta}(a|s)$ and the $\pi_{\theta_k}(a|s)$ is the probability of choosing action a in state s in current policy and the old policy. This fraction is to evaluate the disparity of the old and new policy. When the value of the advantage function is positive, the objective function will increase if the probability of selecting an action increases. But if the fraction is bigger than $1+\epsilon$, the objective function can only use $1+\epsilon$. The objective function won't increase because of the higher $\pi_{\theta}(a|s)$ which is over $1+\epsilon$. Vice versa. This can control the differences between old policies and new policies.

The advantage function is defined to be:

$$A_t = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}, \text{ where } \delta_t = R(s,a,s') + \gamma v_{\theta}(s') - v_{\theta}(s) \quad (10)$$

λ can adjust the variance and bias in the advantage estimate. The smaller the λ, the lower the variance but the higher bias. Vise versa. t is the time step at that moment, and k represents the number of steps to be taken. δ_t is the Temporal Differences (TD) error, which shows the differences between the approximated value of taking an action and the value of current states. The advantage function enables the advantage of action to be estimated, that is, how good an action is compared to the average value of taking all kinds of action. Besides, to update the value function, the squared of disparity between estimated state value and accumulated discounted future reward ($\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$) is defined as the loss function and gradient descent will be applied.

4 Result

All the algorithms use a similar or the same neural network, with 256 hidden nodes in both the first layer and second layer, and the same activation function to make the comparison as fair as possible. The same random seeds are used. One episode of training refers to from start to terminate in one round. There's a difference when training between the PPO and the other three algorithms. When training the other three algorithms, the one-time loss backward follows one action, while the PPO trains one time in one episode. If the number of training of PPO agents equals the number of actions taken, the performance of it is somehow much worse. If only train once in one episode in the other three algorithms, it will take a very long time but the rewards may still not increase significantly. Therefore, the number of episodes needed to achieve relatively high rewards between PPO and the other three algorithms can have a big gap. The plot of rewards for each episode for PPO is separated from the others. The value of the y-axis is the average of rewards in ten episodes. The results are shown below.

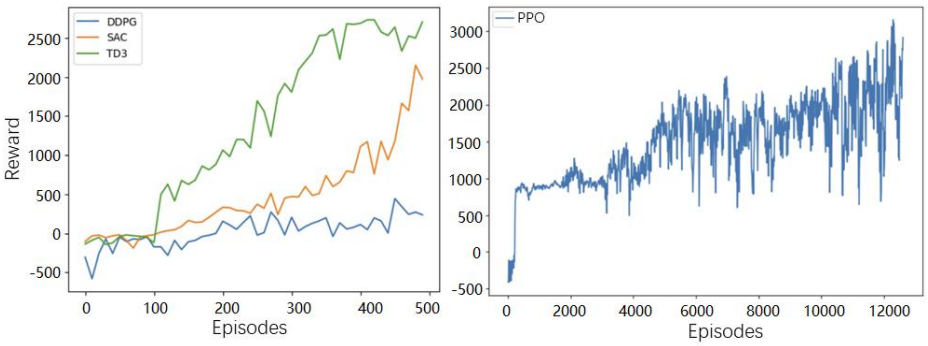

Fig. 1. Results on Ant-v4 (Figure credit: Original).

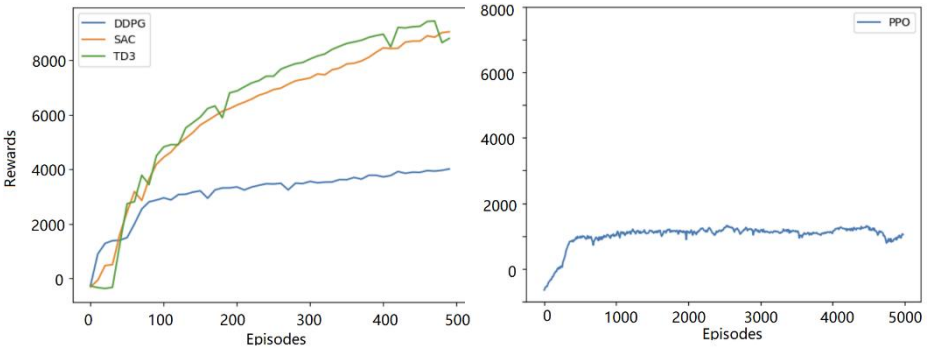

Fig. 2. Results on Half Cheetah-v4 (Figure credit: Original).

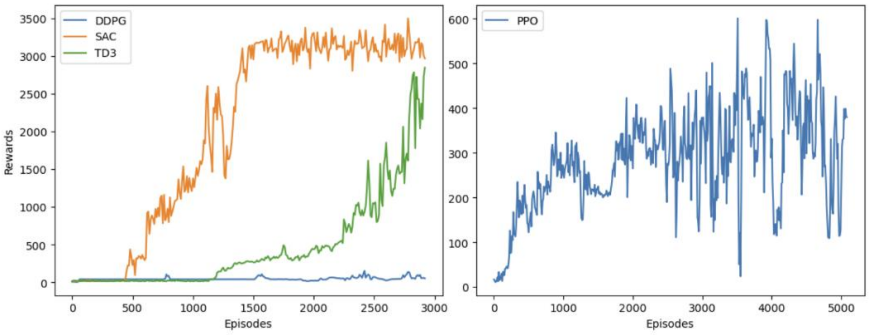

Fig. 3. Results on Half Hopper-v3 (Figure credit: Original).

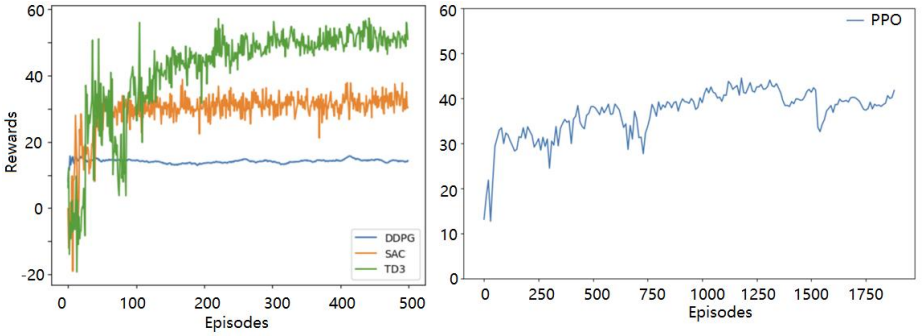

Fig. 4. Results on Swimmer-v4 (Figure credit: Original).

In Ant-v4 (Fig. 1), The PPO can achieve the highest reward finally but seems more unstable. The SAC and TD3 can achieve rewards that are over 2000, but SAC will take more time, which may as a result of the SAC explored for a long time. This may be attributed to its entropy bonus mechanism. The DDPG performs the worst. This may be because of the overestimation of the target Q value.

As for the Half-cheetah (Fig. 2), TD3 performs best again, SAC next, then DDPG, and PPO worst. And in Hopper v3 (Fig. 3), SAC performs best, TD3 next then PPO and DDPG worst. In this task, the TD3 takes a long time to achieve a result which is more than 1000. The DDPG somehow only increased a little at the beginning but converges very fast. In the swimmer-v4 (Fig. 4), the descending rank of the highest reward when converging is TD3, SAC, PPO, and DDPG. The number of iterations needed to converge is smaller than the previous task.

One of the potential drawbacks of the implementation of PPO is the influence of batch size. According to previous work, the stability of PPO can be affected by batch size [13]. But in many environments, such as Ant, Hopper, and Swimmer, the number of iterations needed to finish a game in one episode can vary, sometimes it can reach 1000, while sometimes it is smaller than 10. Thus, if the batch size is too high, data from some episodes may not be trained, if the batch size is low, the stability will be negatively impacted. But the other 3 algorithms are off-policy algorithms, the batch

size can be larger because data could be directly sampled from the replay buffer. This situation may cause potential instability of training or worse performance in PPO.

In previous research, the Soft Actor Critic can achieve the highest rewards at last in most environments, while PPO can achieve the best reward in this paper [8]. Besides, DDPG in result in this paper and result in previous research is similar, it doesn't perform very well.

In previous work, in the Half Cheetah, the DDPG can perform better than the SAC [10]. And the DDPG performs much better in Hopper compared to the results in this paper. Besides, the overall result in Ant and the performance of TD3 and SAC in Hopper are similar.

Overall, the results from this paper and other papers have some similarities and differences. But it can be concluded that in most cases, the TD3 or SAC performs best in Ant, Hopper, Half Cheetah, and Swimmer. The DDPG didn't perform well in most tasks, but whether the PPO can do well is still debatable. This difference may be caused by different hyperparameters.

5 Conclusion

This paper gave a basic introduction to applying deep reinforcement learning in controlling continuous systems. Also, this paper introduces the significance of DRL in the continuous system controlling, Mujoco, and include a literature review of DDPG, SAC, TD3, and PPO as well as some related comparative experiment that has already been done. The key principles of these four algorithms are also discussed. Then comparative experiments of these four algorithms are also conducted in Ant-v4, Half Cheetah-v4, Swimmer-v4, and Hopper-v3. Finally, a comparison between the results and results from another research is illustrated. A conclusion can be drawn from these results that SAC or TD3 is the most likely to perform best in Mujoco environments tried in this paper, while the performance of PPO is still unsure and the DDPG cannot perform as well as TD3 and SAC in most tasks. To improve, more environments can be tried. Besides, the hyperparameters should be adjusted to enhance the performance of models. Also, some methods should be applied to solve the issues in PPO related to the batch size and its potential threat to its stability.

References

1. Morales, E. F., Murrieta-Cid, R., Becerra, I., & Esquivel-Basaldua, M. A. A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning. Intelligent Service Robotics, 14(5), 773-805 (2021).
2. Todorov, E., Erez, T., & Tassa, Y. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems, 5026-5033 (2012).
3. Mujoco. MuJoCo-Gym Documentation, URL: https://www.gymlibrary.dev/environments/mujoco/. Last accessed: Aug. 11, 2023.

4. Konda, V., & Tsitsiklis, J. Actor-critic algorithms. Advances in neural information processing systems, 12, 1008-1014 (1999).
5. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. Deterministic policy gradient algorithms. In International conference on machine learning. 387-395 (2014).
6. Tiong, T., Saad, I., Teo, K. T. K., & bin Lago, H. Deep reinforcement learning with robust deep deterministic policy gradient. arXiv:2011.00001 (2020).
7. Fujimoto, S., van Hoof, H., & Meger, D. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477 (2018).
8. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018).
9. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017).
10. Lyu, J., Ma, X., Yan, J., & Li, X. Efficient continuous control with double actors and regularized critics. In Proceedings of the AAAI Conference on Artificial Intelligence. 36(7), 7655-7663 (2022).
11. Uther, W. Markov Decision Processes. arXiv:1701.00001 (2017).
12. Van Otterlo, M., & Wiering, M. A. Reinforcement learning and Markov decision processes. arXiv:1201.00001 (2012).
13. Hilton, J., Cobbe, K., & Schulman, J. Batch size-invariance for policy optimization. Advances in Neural Information Processing Systems, 35, 17086-17098 (2022).