Deep Reinforcement Learning for an Anthropomorphic Robotic Arm Under Sparse Reward Tasks

Hao Cheng, Feng Duan(✉), and Haosi Zheng

Department of Artificial Intelligence, Nankai University, Tianjin 300350, China
duanf@nankai.edu.cn

Abstract. Training operation skills of a robotic arm using Reinforcement Learning algorithms requires a process of reward shaping, which needs considerable time to adjust. Instead, the setting of sparse rewards makes the tasks clear and easy to modify when the task changes. However, it is a challenge for the agent to learn directly from sparse reward signals because of the lack of reward guidance. To solve this problem, we propose an algorithm based on the DDPG algorithm and add three techniques: Hindsight Experience Replay for improving sample efficiency, Expert Data Initialization for accelerating learning speed in the early stage, Action Clip Scaling for acting stable. For validating our algorithm, we built a simulation environment of an anthropomorphic robotic arm based on the Pybullet module and set up the training interface. There are two tasks trained under sparse reward signals: *Push* task, *Pick and Place* task. The experimental results show that the agent can quickly improve the operation skill level in the early stage. It also has a good convergence effect in the later stage, which effectively solving the sparse rewards problem.

Keywords: Reinforcement learning · Sparse rewards · Anthropomorphic arm

1 Introduction

Reinforcement Learning (RL) combined with Deep Neural Network is currently a research highlight in recent years. Deep Reinforcement Learning (DRL) has achieved many breakthroughs in video games, robotics, and other fields due to its strong versatility. For example, the proposal of the Deep Q Network (DQN) algorithm in 2013 made the agent comparable to human-level operation in video games for the first time [1]. High-dimensional continuous control methods like PPO, TRPO have been successfully applied in the field of robotics [2,3]. Robots are trained to complete operation skills [4–6], and humanoid robots can also be controlled by the RL algorithm [7,8].

For robots with RL algorithms, solving different tasks requires different reward shaping, which means that once the task targets change, the rewards need to be reset and adjusted. It will be a tedious process. In addition, a simple

© Springer Nature Switzerland AG 2021
X.-J. Liu et al. (Eds.): ICIRA 2021, LNAI 13014, pp. 79–89, 2021.
https://doi.org/10.1007/978-3-030-89098-8_8

linear transformation of the set rewards will produce a significant effect on the training result and cause unacceptable fluctuations. When the rewards is inappropriate, it may cause the agent to converge to a local optimum. If the reward function is too complicated, the agent's behavior after training may deviate from the experimenter's intention. As a result, reward shaping engineering seriously damages the versatility of RL, making it hard to transplanted fast.

As an alternative, sparse rewards can overcome the above problems. The sparse rewards we set are binary signals. i.e., 1 when completing the task, otherwise 0. The advantage of sparse rewards is that the task goals are clear and unambiguous. Moreover, when the task goal changes, only a tiny amount of the code needs to be modified. Nevertheless, the ensuing problem is that it is challenging for the agent to gain successful experience without dense rewards guiding, which requires us to design an effective algorithm to ensure the agent can obtain good training results.

In this paper, we adopted the Deep Deterministic Policy Gradient (DDPG) algorithm to deal with the robot physical problem of the high-dimensional continuous action space. Based on DDPG, we added three techniques: Hindsight Experience Replay (HER), Expert Data Initialization, and Action Clip Scaling. We built an anthropomorphic robotic arm simulation environment based on the Pybullet module and set up the training interface for RL. There are two tasks set under the sparse rewards: *Push* task, *Pick and Place* task. The experimental results show that the agent can learn successful experience in the early stage due to the initialization of the expert data. With the utilize of the HER algorithm, the sample efficiency is greatly improved. In the later stage, due to the Action Clip technique, the agent's actions change to be stable, and finally it have a good convergence effect.

2 Related Work

In order to solve the challenge of sparse rewards, there are mainly three types of methods in the literature: Curiosity Module, Curriculum Learning, and Hierarchical RL. These methods will be introduced respectively below.

2.1 Curiosity Module

Curiosity is a characteristic of human agents. When facing sparse rewards in the real world, humans can still learn a skill because of their intrinsic curiosity motivation. Therefore, we can apply the idea of curiosity to the RL algorithm. The pivotal idea is to treat the rewards from the environment as external rewards and then set additional intrinsic rewards to encourage the agent to explore. The sum of external rewards and internal rewards is the training goal of the agent.

The difference among curiosity methods in the literature mainly lies in how to set intrinsic rewards. In [9], Intrinsic Curiosity Module is proposed to generate intrinsic rewards. It is mainly divided into two sub-modules: forward and inverse

dynamics models. The loss of the inverse dynamics model is defined as the difference between the real action and the predicted action, and the forward model is the difference between the prediction of the next state and the real state. Combine the loss of the two, the self-supervised method is used to update the network and prediction error is treated as the intrinsic rewards. Similar to the above method, in [10], a method named Random Network Distillation (RND) is proposed. The RND input the observed state to a randomly initialized fixed target network as a prediction task $f(x)$, and then send the predicted value after the state is input from the predictor network to $f\hat{(}x)$, regard the prediction error as an inherent reward. Finally, it uses MSE to update the network.

2.2 Curriculum Learning

Curriculum Learning (CL) is a methodology of optimizing the order in which the agent acquires experience so as to accelerate training convergence on the final target [11]. A curriculum can be thought of as an ordering over experience samples. CL can first train on one or more source tasks and then transfer the acquired knowledge to solve the target task. The agent can learn prior knowledge from a simple, intermediate task and gradually transit to the difficulty of the final target task.

2.3 Hierarchy Reinforcement Learning

Hierarchical reinforcement learning (HRL) is a methodology of multiple layers of policy training [12]. In HRL, goals are generated in a top-down fashion [13]. The upper-level agent conducts long-term planning and decision-making over a more extended period (lower time resolution) to form a high-level reasoning policy. By setting intrinsic rewards, It converts the goals of the high-level planning policy into a low-level action policy under the higher time resolution. In a nutshell, the manager (high-level agent) is responsible for planning, and the work (low-level agent) is responsible for making the actions.

3 Background

3.1 Reinforcement Learning

Markov decision process (MDP) is a universal framework to model the problem in RL. The agent continuously interacts with the environment to obtain data, i.e., The agent makes actions, and the environment responds to these actions and presenting new situations to the agent [14].

MDP can be presented as a tuple (S, A, P, R, γ), which consists of a set of states S, a set of actions A, a transition function $P(s'|s, a)$, a reward function R and a discount factor γ. The goal of RL is to find an optimal policy $\pi(a|s)$, and maximize the discounted return G_t. It can be described as (1),

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, 0 < \gamma < 1 \tag{1}$$

3.2 Policy Gradient Optimization

As mentioned above, The goal of RL is to maximize the discounted return. In RL, there are two types of methods to solve this problem, i.e., the value-based method and the policy-based method. However, the policy obtained from the value-based method is usually a finite set of mappings from state space to action space. As a comparison, a policy-based method generates a parameterized policy target function described as (2),

$$U(\theta) = \sum_{\tau} P(\tau|\theta)R(\tau) \tag{2}$$

where θ is the parameters of the policy, τ is a sample trajectory. The parameterized policy makes the problem easier to better convergence and is suitable for solving problems with a large or continuous action space.

The policy-based method optimizes the parameters through the gradient ascent algorithm (also called the policy gradient) and obtains the final parameters through iterative calculations.

$$\theta_{new} = \theta_{old} + \alpha \nabla_{\theta} U(\theta) \tag{3}$$

By computing the derivative of $U(\theta)$ with respect to θ, we can get the policy gradient (4),

$$\nabla_{\theta} U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} log P(\tau; \theta) R(\tau) \tag{4}$$

where m is the number of the trajectories. However, It still a problem of complex computing, which needs further simplification and estimation.

4 Method

We use Deep Deterministic Policy Gradient (DDPG) algorithm [15] of the Actor-Critic (AC) framework as the basis and add three techniques to it: Hindsight Experience Replay (HER), Expert Data Initialization, and Action Clip Scaling. The schematic view of the algorithm is shown in Fig. 1.

First, we use expert data to initialize the Replay Buffer and then start training. We collect k episodes of data and put it into the replay buffer. When updating the network, we take out N mini-batches of data from the Replay Buffer and replace goals through the HER module. The replaced data are used to update the Actor and the Critic network. Finally, the updated Actor output the action in the following sample episodes with the utilization of the Action Clip module to limit the range of the actions.

4.1 Base Algorithm: Deep Deterministic Policy Gradient

DDPG is a model-free RL algorithm that can effectively deal with continuous control problems. It is also an Actor-Critic framework based on the Deterministic

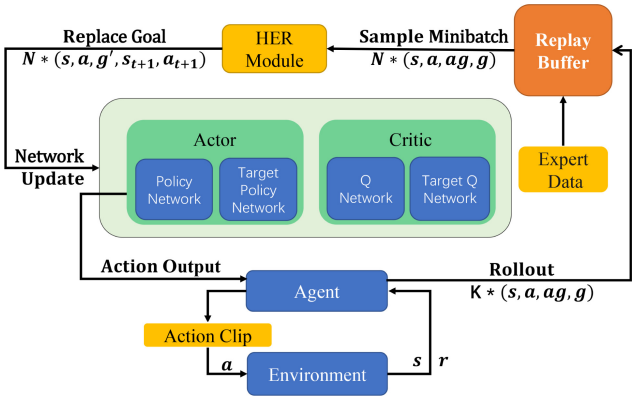

Fig. 1. Schematic view of the algorithm.

Policy Gradient (DPG) algorithm [16]. Different from the output of a probability distribution of actions in a stochastic policy algorithm, in DDPG, the output is a deterministic action in the state s, which has better stability in performance. The algorithm maintain two networks: the Actor Network $\mu(s|\theta^{\mu}) : S \rightarrow A$ and the Critic Network $Q(s, a|\theta^{Q}) : S \times A \rightarrow R$. The actor as the target policy outputs the action a in the state s, and the critic evaluates the actions. The actions will be added an exploration noise, the expression as shown in (5),

$$a_t = \mu(s_t|\theta^{\mu}) + N_t \tag{5}$$

where N_t is a noise distribution. The Critic updates the network by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2$, and the Actor updates the network through computing the gradient of the expected return on the trajectories:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{si} \tag{6}$$

In addition, DDPG incorporated the successful experience of DQN and introduces two techniques: Replay Buffer and Target Network. The Replay Buffer is used to store the sample data and execute random uniform sampling from it when the network updates, so as to break the correlation between the sample data and improve the sample efficiency. The target network is an independently set up network used to estimate the TD error when the network updates, which can effectively avoid the fluctuation caused by the action-value update to be brought into the next round of iterative calculations.

4.2 Techniques for Sparse Rewards

Hindsight Experience Replay. Hindsight Experience Replay (HER) [17] is motivated by the experience of human agents. Human agents can learn from

failures, i.e., learn from unexpected results. First of all, the HER algorithm can be combined with any off-policy algorithm. The pivotal idea of the HER algorithm is to re-examine the trajectory with different goals. For example, in a episode, $S = s_1, s_2, ..., s_T$, but the desired goal $g \neq s_1, s_2, ...s_T$. In a standard RL algorithm, this is a failed sample experience. RL cannot get an effective policy update from this episode. By using the HER algorithm, the desired target G is replaced with some state in S (for example, the final state s_T). The agent can finally learn from a reconstructed success.

The HER algorithm can be regarded as implicit Curriculum Learning. By replacing the desired goal, we can set an intermediate task for the agent to help it gradually achieving from a sub-target to the final target. In this paper, we use the *future* strategy to sample experience data from the Replay Buffer. i.e., when the sampling moment is i, we select an achieved goal from the subsequent time step as the sampling goal. The expression is as (7):

$$g_i \leftarrow ag_i, j \in [i+1, T] \tag{7}$$

Through *future* sampling, we can construct more successful transitions than the *final* (S_t as the sampling goal) or *random* (sample from all transition) sampling goal strategy.

The adoption of the HER algorithm greatly improves the sampling efficiency and provides a huge effect for solving the sparse rewards problems.

Expert Data Initialization. In the early stage of the training, it takes considerable time for the agent to have a fundamental breakthrough before the HER algorithm can highlight its role. For example, if the agent has not touched the object in the *Push* task or the object has not been picked up by hand in the *pick and place* task, no successful experience will be obtained or reconstructed.

In order to accelerate the learning speed of the early stage and make the agent can directly learn successful experience, we use the program to generate trajectories that try to complete the task and record the transitions of the episodes when the result is a success. At the beginning of the training, the collected expert data are poured into the Replay Buffer as the initialization process, effectively solving the problem of large fluctuation and slow learning efficiency in the early training stage.

Action Clip Scaling. The actions output by the agent needs to be restricted. On the one hand, unlimited action space will affect the training effect. On the other hand, a motion of large range is also prohibited in real robots for damaging the motors. We can limit the range of the actions by clipping it.

$$\begin{aligned} action &\leftarrow clip(action, lim_l, lim_h), \\ clip(x, x_1, x_2) &= \begin{cases} if\ x < x_1 & x \leftarrow x_1 \\ if\ x_1 < x < x_2 & x \leftarrow x \\ if\ x_2 < x & x \leftarrow x_2 \end{cases} \end{aligned} \tag{8}$$

At the early period of training, the agent is allowed to take larger actions to explore. As the training progresses, the agent has learned a part of the experience. We narrow down the range of the output actions (Scale down lim_l and lim_h). There is no requirement for force output in the task. Moreover, in MDP, interactive data satisfies the Markov property, i.e., the next state of the system is only related to the current state. As a result, such a change will not excessively affect the training effect and improve the accuracy and precision of motion control.

5 Experiments and Results

5.1 Environments

The RL environment built in this paper is based on the Pybullet module [18]. We built a 7 DOF anthropomorphic arm [19,20] simulation environment to test our algorithm.

First, we consider two different tasks:

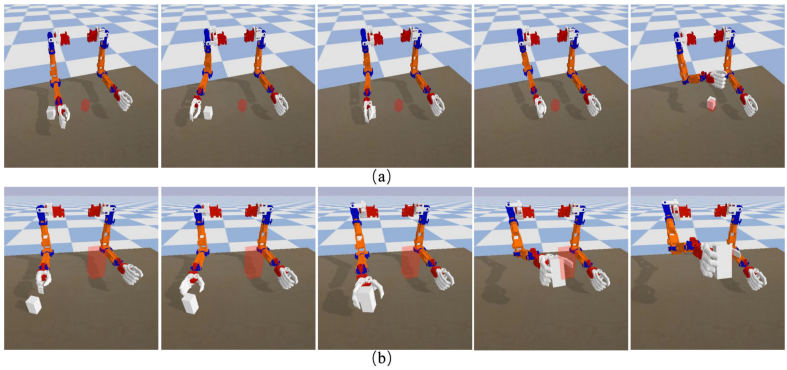

Fig. 2. (a) *Push* task (b) *Pick and Place* task

Push. The agent pushes the object placed on the table to the target position. The end-effector of the arm is blocked.

Pick and Place. The agent picks up the object on the table and places it at the target position in the air.

The details of the RL interface settings are as follows:

Observations. Observation is the state vector returned after each time the agent interacts with the environment. It will be input into the policy for gradient optimization. It includes the position of the end-effector (3), the orientation of the end-effector (3), the linear velocity of the end-effector(3), the angular velocity of the end-effector (3), the position of the object (3), the orientation of the object (3), the linear velocity of the object (3), the angular velocity of the object (3), the position of the target (3), the relative position of the object and the end-effector (3). The angle-related elements are represented by Euler angles. The total dimension of the observation vector is 30.

Actions. The actions is represented by a four-dimensional vector $[d_x, d_y, d_z, d_{grip}]$. In the *Push* task, the hand is blocked and the value of the d_{grip} is 0. The position of the hand performs the output action based on the current position. i.e., in the Cartesian coordinate, the next position reached by the hand is:

$$P_{next} = [\ p_{curx} + d_x,\ p_{cury} + d_y,\ p_{curz} + d_z,\ p_{curgrip} + d_{grip}\ ] \tag{9}$$

We then use inverse kinematics to convert the hand position into the joint angles in the joint space, completing the computation of the output action.

Rewards. As mentioned above, we use sparse rewards instead of dense rewards to guide the agent to complete training. The sparse rewards is defined as:

$$\begin{aligned} r(s, a, g) &= -[\ f_g(s') = 0\ ], \\ f_g(s') &= [\ |g - s_{obj}| \leq \delta\ ] \end{aligned} \tag{10}$$

where s' is the next state after the agent take the action a in the state s, g is the goal position, s_{obj} is the object position in the state s, δ is the tolerance distance between the object and the goal ($\delta = 5$ cm is set in the two tasks). The value of the expression in brackets means that if the expression is true, it is 1, otherwise it is 0. In a nutshell, when the object reaches the goal position and the error is below δ, the reward is 0, otherwise is -1.

Initial States. The object is randomly set to a position where the arm can reach, and the initial position and posture of the robotic arm is fixed.

Goal. The goal position is set at a position reachable by the end of the arm. In the *Push* task, the goal position is set on the table, while in the task of *Pick and Place*, the goal position is set in the air.

5.2 Experimental Details and Results

Before the training, we first collected 1000 episodes of expert data for *Push* task and *Pick and Place* task separately by using program. Before the training, we put the expert data into the Replay Buffer.

There are 100 episodes of samples in an epoch, and the agent is allowed to perform actions for 150 timesteps in each episode. After the agent performs an action, it needs to wait for 20 sub-steps (1/240 s per sub-step), i.e., the agent's action frequency 12 Hz. We update the network every two episodes sampling. After 100 epochs, we change action clip limits from $(-0.5\,\mathrm{m}, 0.5\,\mathrm{m})$ to $(-0.1\,\mathrm{m}, 0.1\,\mathrm{m})$ in *Push* task, $(-0.5\,\mathrm{m}, 0.5\,\mathrm{m})$ to $(-0.15\,\mathrm{m}, 0.15\,\mathrm{m})$ in *Pick and Place* task. The experiment was conducted on 3 random seeds. After each epoch training, we evaluate the each epoch trained model for 50 episodes, and finally the success rate curves was generated. As a comparison, we add a set of experiments without expert data.

The result is shown as Fig. 3. Due to the role of Expert Data Initialization, the agent can improve quickly in the early stage. Note that the training effect of the *pick and place* task is worse than that of the *push* task. This is because the *pick and place* task requires the agent to use the hand to pick up the object and put it in the air, which is much more difficult than the *push* task. This leads to the fact that in the *Push* task, the agent achieved a success rate of about 90% in 75 epochs, and the fluctuation is comparable small. While in the *Pick and Place* task, the success rate improve relatively slower and exist fluctuation. But it converges in the later period at a success rate of 90%. However, in the comparison group without expert data, it takes a long time for the agent to learn successful experience in the push task, and there are large fluctuations in three random seed experiments. In the Pick and Place task, the agent lacking expert data cannot learn successful experience at all.

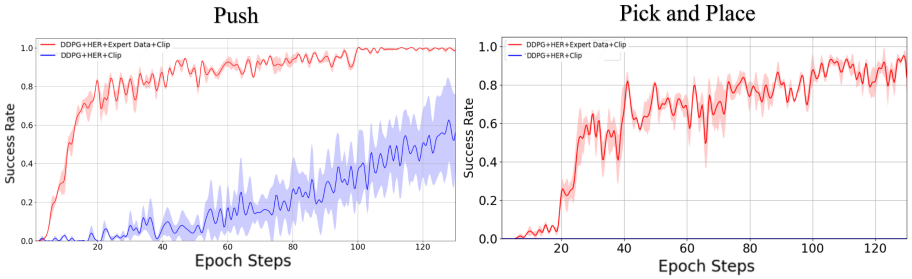

Fig. 3. Success rate

After 100 epoch, we change action clip limits. We can observe that the success rate after 100 epoch obviously fluctuates less, which means the agent's actions more stable. We drew the action curves using models of 99 epoch and 101 epoch in the *Push* task, as shown in Fig. 4. In the two curves, the initial positions, orientation of the object and the target are the same. The blue curve is when the agent's action is limited to (−0.5, 0.5), while the light blue is (−0.1, 0.1). The two curves record the hand positions of the agent from the beginning to the completion of the task. Although the blue curve takes less time to complete the task, the acceleration during the movement is very large, and there are some

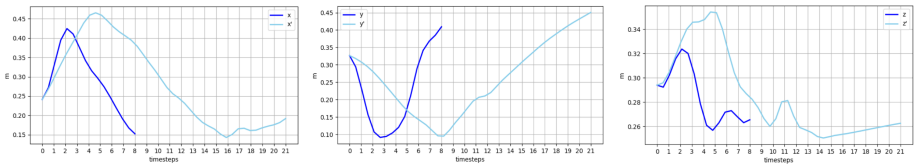

Fig. 4. The use effect of Action Clip

peaks, which are not allowed on a real robot because it will damage the motor. As a comparison, in the light blue curve, the action to complete the task is more stable and gentle.

6 Conclusion

Aiming at the sparse rewards problem, we propose an algorithm based on DDPG and add three techniques: HER, Expert Data Initialization, Action Clip Scaling. We built a simulation environment for anthropomorphic robotic arms based on the Pybullet module and set two tasks under sparse rewards: *Push* task, *Pick and Place* task. The experiments show that the agent has a better convergence effect in the *Push* task. For the *Pick and Place* task has more complex actions, it is more challenging to learn. The learning process slow and fluctuating in the early stage, but in the end, the agent learns good results. Finally, we solved the two sparse rewards problems, which proved the effectiveness of our algorithm, and the action of the robotic arm becomes stable due to the utilization of Action Clip Scaling.

In future work, we will further improve the algorithm and realize the cooperation of the two arms through the multi-agent algorithm to further enhance the intelligence of the robot arms, and it is expected that the algorithm will be transferred to the real robot arms.

Acknowledgements. This work was supported by the National Key R&D Program of China (No. 2017YFE0129700), the National Natural Science Foundation of China (Key Program) (No. 11932013), the Tianjin Natural Science Foundation for Distinguished Young Scholars (No. 18JCJQJC46100), and the Tianjin Science and Technology Plan Project (No. 18ZXJMTG00260).

References

1. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
2. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897. PMLR (2015)

3. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
4. Niroui, F., Zhang, K., Kashino, Z., Nejat, G.: Deep reinforcement learning robot for search and rescue applications: exploration in unknown cluttered environments. IEEE Robot. Autom. Lett. **4**(2), 610–617 (2019)
5. Nguyen, H., La, H.: Review of deep reinforcement learning for robot manipulation. In: 2019 Third IEEE International Conference on Robotic Computing (IRC), pp. 590–595. IEEE (2019)
6. Li, F., Jiang, Q., Zhang, S., Wei, M., Song, R.: Robot skill acquisition in assembly process using deep reinforcement learning. Neurocomputing **345**, 92–102 (2019)
7. Garcia, J., Shafie, D.: Teaching a humanoid robot to walk faster through Safe Reinforcement Learning. Eng. Appl. Artif. Intell. **88**, 103360 (2020)
8. Abreu, M., Lau, N., Sousa, A., Reis, L.P.: Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning. In: 2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 1–8. IEEE (2019)
9. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning, pp. 2778–2787. PMLR (2017)
10. Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by random network distillation. arXiv preprint arXiv:1810.12894 (2018)
11. Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E., Stone, P.: Curriculum learning for reinforcement learning domains: a framework and survey. J. Mach. Learn. Res. **21**(181), 1–50 (2020)
12. Vezhnevets, A.S., et al.: Feudal networks for hierarchical reinforcement learning. In: International Conference on Machine Learning, pp. 3540–3549. PMLR (2017)
13. Nachum, O., Gu, S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. arXiv preprint arXiv:1805.08296 (2018)
14. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
15. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
16. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: International Conference on Machine Learning, pp. 387–395. PMLR (2014)
17. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., et al.: Hindsight experience replay. arXiv preprint arXiv:1707.01495 (2017)
18. Pybullet. https://pybullet.org/wordpress/. Accessed 29 April 2021
19. Li, W., et al.: Design of a 2 motor 2 degrees-of-freedom coupled tendon-driven joint module. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 943–948. IEEE (2018)
20. Li, W., et al.: Modularization of 2-and 3-DoF coupled tendon-driven joints. IEEE Trans. Robot. (2020)