EasyChair Preprint

№ 8873

Learning Robot Arm Controls Using Augmented Random Search in a Simulated Environment

Somnuk Phon-Amnuaisuk, Peter David Shannon and Saiful Omar

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

September 24, 2022

Learning Robot Arm Controls using Augmented Random Search in A Simulated Environment

No Author Given

No Institute Given

Abstract. We investigate the learning of continuous action policy for controlling a six-axes robot arm. Traditional tabular Q-Learning can handle discrete actions well but less so for continuous actions since the tabular approach is constrained by the size of the state-value table. Recent advances in deep Reinforcement Learning (deep RL) and Policy Gradient (PG) learning abstract the look-up table using function approximators such as artificial neural networks (ANNs). ANNs abstract loop-up policy tables as policy networks that can predict discrete actions as well as continuous actions. However, deep RL and PG learning were criticized for their complexity. It was reported in recent works that Augmented Random Search (ARS) has a better sample efficiency and a simpler hyper-parameter tuning. This motivates us to apply the technique to our robot-arm reaching tasks. We constructed a custom simulated robot arm environment using *Unity Machine Learning Agents* game engine, then designed three robot-arm reaching tasks. Twelve models were trained using ARS techniques. Another four models were trained using the state-of-the-art PG learning technique i.e., proximal policy optimization (PPO). Results from models trained using PPO provide a baseline from the PG technique. Empirical results of models trained using ARS and PPO were analyzed and discussed.

Keywords: Augmented Random Search · Robot arm controls · Reinforcement Learning

1 Introduction

Learning a stochastic policy function $\pi_\theta : \mathcal{S} \mapsto \mathcal{A}$ (that maps states $s \in \mathcal{S}$ to actions $a \in \mathcal{A}$) using parameters θ may be achieved using *policy gradient* techniques. In the past decades, the policy gradient approach has received a lot of attention from the community and many tactics have been devised to improve learning effectiveness, for examples, *experience replay* in *deep Q-Learning* [1], trust-region, and importance sampling in *Trust Region Policy Optimization (TRPO)* [2]. These tactics enhance performances and improve the sample efficiency issue. However, these tactics also introduce many hyper-parameters into the learning process and result in hyper-parameter sensitivity [3].

A recent study using *Augmented Random Search (ARS)* by [4] demonstrated improved performance over policy gradient techniques in terms of *sample efficiency* and simple *hyper-parameters* tuning. Sample efficiency implies fewer

training samples. The linear policy search using ARS also has fewer hyper-parameters. These plus points motivate us to investigate the ARS technique in the robot arm controlling tasks.

Our contributions in this paper are in the following activities: (i) constructing a custom simulated robot arm environment using Unity ML-Agents game engine [5], and (ii) investigating ARS in the learning of continuous control policy in the simulated environment as well as benchmarking the results with the *Proximal Policy Optimization (PPO)* [6] which is a state-of-the-art policy gradient technique. We report the empirical results from using ARS to approximate a stochastic policy in controlling a 6DoF robot arm, and a comparison with the PPO technique. The rest of the paper is organized into the following sections: Section 2 discusses the background of policy gradient-based optimization and policy gradient search; Section 3 discusses our approach and gives the details of the experimental setup; Section 4 provides the experimental results and provides a critical discussion of the output; and finally, the conclusion and further research are presented in Section 5.

2 Estimating Policy using Random Search

Follow [7], *Markov Decision Process (MDP)* is a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_s)$. Let $t \in \{0, 1, 2, ...\}$ denote a discrete time step, $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ denote a state and an action at time step t respectively. The state transition probabilities $\mathcal{P}^a_{ss'}$ represent the action policy of an agent. $\mathcal{P}^a_{ss'}$ can be approximated through a value function, or directly using an independent function approximator with its own parameters θ, such as neural network weights.

The Policy Gradient (PG) approach has been explored since the early stage of reinforcement learning research [8]. In recent years, many variations of PG approach have re-emerged as a popular gradient-based RL techniques. The basics of PG technique can be summarized as follow: let θ denote a vector representing policy parameters and ρ representing the performance of the policy. A policy gradient method approximates the near optimal policy by adjusting the policy based on gradient computing from $\Delta\theta \approx \alpha\frac{\partial\rho}{\partial\theta}$, where α is the step size and $\frac{\partial\rho}{\partial\theta}$ can be expressed as:

$$\frac{\partial\rho}{\partial\theta} = \sum_s d^\pi(s) \sum_a \frac{\partial\pi(s,a)}{\partial\theta} Q^\pi(s,a) \tag{1}$$

where $d^\pi(s)$ is the stationary distribution of states, and $Q^\pi(s,a)$ is the value of a state-action pair under π.

Many improvement of the standard policy gradient given in Eq. 1 have been reported in recent years. Deterministic Policy Gradient (DPG) [9] and Deep Deterministic Policy Gradient (DDPG) [10] extend a discrete control to a continuous control. Trust Region Policy Optimization (TRPO) improves training stability by curbing the update of new untrusted regions [2]. Proximal Policy

Optimization (PPO) further simplifies TPRO by using a clipped surrogate objective while retaining similar performance, further improving sample efficiency [6].

As the policy gradient methods become sophisticated, their hyper-parameters become complex and their performance too becomes sensitive to the hyper-parameters tuning [3]. The authors of [4] employed ARS to train linear policies and able to achieve state-of-the-art sample efficiency on the *Multi-Joint dynamics with Contact (MuJoCo)* which is a proprietary physics engine for locomotion tasks. ARS performs *policy space search* based on the *finite-difference* technique which will be explained in the next section.

2.1 Policy Space Search using Augmented Random Search

Population-based search techniques exploit information that implicitly captures gradient information. This is commonly obtained by keeping tracked of individuals with good performance. For example, the concepts of *Global-best particle* and *Local-best particle* in the *Particle Swarm Optimization (PSO)* could be seen as providing a means to calculate gradient to other particles in the swarm.

A random policy decides its next action from all possible actions. A pure random policy does not have a clear notion of gradient and does not use gradient information to guide the search. However, a random search could explore various information to guide the search, such as keeping track of its fruitful actions, and inspecting the local landscape before deciding on its next action. Exploiting extra information often improves its performance.

A Basic Random Search (BRS) In [4], the authors exploit gradient information in the policy parameter space θ by perturbing θ in the positive and negative directions i.e., $\theta + \nu\delta$ and $\theta - \nu\delta$ where $\nu < 1$ is the noise, δ is random numbers sampled from a normal distribution, and δ has the same dimension as the policy parameter θ.

This postive-negative perturbation approach is known as *finite-diferrences* technique. In brief, let $r(\theta + \nu\delta, \mathbf{x})$ and $r(\theta - \nu\delta, \mathbf{x})$ be the rewards obtained from input $\mathbf{x}$ following the perturbed policy parameters θ in positive and negative directions. If N random moves are sampled by perturbing the policy parameters θ_t (at step t) N times, then a local gradient can be expressed as:

$$\Delta_t = \frac{1}{N} \sum_{n=1}^{N} \delta_n [r(\theta - \nu\delta, \mathbf{x}) - r(\theta - \nu\delta, \mathbf{x})]_n \tag{2}$$

and the policy parameters θ_t can be updated

$$\theta_{t+1} = \theta_t + \alpha \Delta_t \tag{3}$$

where α is the learning rate.

Augmented Random Search (ARS) In brief, ARS augments BRS in the following areas: (i) normalize the states, $x = \frac{x-\mu}{\sigma_x}$, (ii) scaling by the standard deviation σ_R, where σ_R is computed from rewards of all perturbations and (iii) using top performing b directions (see the update section in the Alg. 1). ARS algorithm is included below for readers' convenience. Readers are invited to read detailed discussion in the original paper by [4].

Algorithm 1 Augmented Random Search (ARS)

Let ν a positive constant < 1, let α be the learning rate > 0, let N the number of perturbations, and b be the number of top-performing perturbations, $b < N$.
Let $|A| = p$ be the desired number of actions, let θ and be a $p \times n$ matrix representing the parameters of the policy π, and let δ_k be a $p \times n$ matrix representing the kth perturbation. The ARS can be expressed as follow:

initialize: $\theta_0 = 0 \in R^{p\times n}, \mu_0 = 0 \in R^{p\times n}, \Sigma_0 = I \in R^{p\times n}$, and $j = 0$.
while end condition not satisfied **do**:
  Generate N perturbations δ_k from a normal distribution.
  Generate 2N episodes and their 2N rewards using π_{k+} and π_{k-}
    Normalize $\pi_{j,k,+} = (\theta_j + \nu\delta_k)\text{diag}(\Sigma_j)^{-1/2}(\mathbf{x} - \mu_j)$ and
    Normalize $\pi_{j,k,-} = (\theta_j - \nu\delta_k)\text{diag}(\Sigma_j)^{-1/2}(\mathbf{x} - \mu_j)$
  Collect the rewards $r(\pi_{j,k,+})$ and $r(\pi_{j,k,+})$ for each perturbation.
  Sort all δ_k by $max(r(\pi_{j,k,+}), r(\pi_{j,k,+}))$
  Update
    $\theta_{j+1} = \theta_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} [r(\pi_{j,k,+}) - r(\pi_{j,k,+})]\delta_k$
  $j \leftarrow j + 1$
end while

---

3 Empirical Set Up using Robot Arm Domain

In our implementation, we use a free robot arm asset for Unity shared by Lukasz-Konopacki[1]. We construct a custom task domain for evaluation using the Unity Machine Learning Agents Toolkit (Unity ML-Agents)[2]. The task domain here is to train a six degree of freedom robot arm (6DoF) to reach a cube randomly spawned on a bench.

3.1 Designing Robot Arm-Reaching Tasks

Figure 1 shows the three tasks carried out in this report. The robot learns to reach the cube on the left hand side for the task 1 (top-left), on the right hand side for the task 2 (middle-left) and on both sides for the task 3 (bottom-left).

[1] https://unitylist.com/p/w03/Robot-Simulator-Unity
[2] https://github.com/Unity-Technologies/ml-agents

From the set up, the complexity of the task 1 and task 2 should be at the same level and the complexity of task 3 should be higher than the task 1 and 2.

It is our intention to design the three tasks to be closely related in order to see the behaviors of linear policy learning using ARS. Upon inspecting the task domain, a human would quickly realize that the main distinction between task 1 and task 2 is the rotation of the base axis, while skills learned for moving other arm joints are the same. However, the ARS (as well as other contemporary policy gradient techniques) will not be able to infer this since it involves meta-level reasoning about the domain.

3.2 State Representations

The robot arm has six axes. These six axes are controlled by stepping motors in the actual physical robot. These axes serve the following purposes: (i) turning the base (1st axis) between [-180, 180] degrees, (ii) rotating the 2nd arm axis between [-60, 140] degrees, (iii) rotating the 3rd arm axis between [-72, 185] degrees, (iv) turning the 4th arm axis between [-180, 180] degrees, (v) rotating the effector (5th axis) between [-120, 120] degrees, and (vi) turning the effector (6th axis) between [-360, 360] degrees.

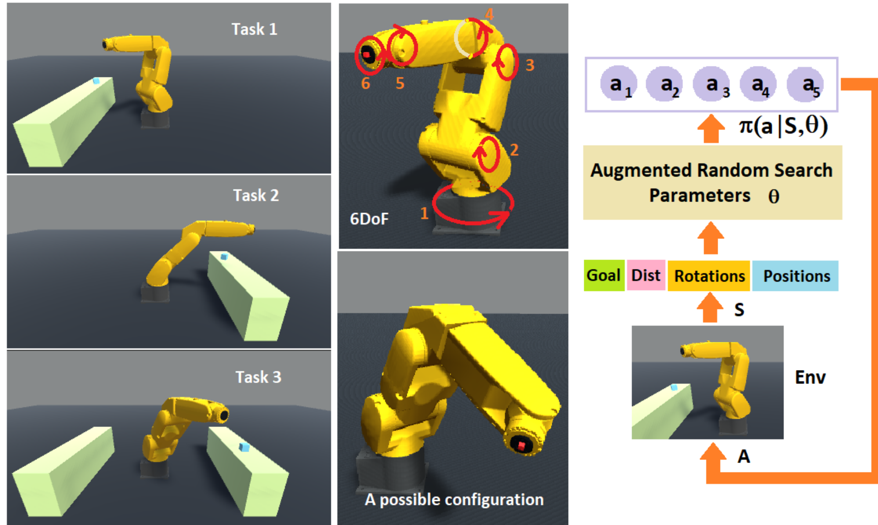

Fig. 1. This figure illustrates the three tasks carried out in our experiments, task 1, task 2, and task 3 (from top-left to bottom-left, respectively). The robot learns to move its effector to the cube with minimum time steps by moving its joints. The middle pane shows information reagarding the 6DoF axes. The right pane presents a graphical summary of the environment

Observations and Actions In our experiment, it was decided to articulate the robot arm with five actions, each for controlling one axis. This was because turning an effector (the 6th axis) would not affect the distance between the effector to the goal. Hence, articulating five axes are sufficient for our robot-arm reaching tasks. Therefore, the output vector has the space size of 5 values.

Let $\mathbf{x}$ denote an observation at any time step, the vector $\mathbf{x}$ captures the following information: (i) the position of the goal (i.e., (x, y, z) coordinate of the cube), (ii) the distance (dx, dy, dz) between the effector and the goal, (iii) the rotations of all arm axes in quaternion $(a + bi + cj + dk)$, and (iv) the position of all arm axes (x, y, z). Hence the observation vector has the space size of 48 values which should be sufficient and appropriate to represent the state space of our task domain.

3.3 Training A Robot Arm using ARS

The training process was given an initial parameter θ. The Unity ML-Agents environment randomly spawned the goal on the bench and randomly configured the robots starting position. The benches were placed at the following coordinates. The bench on the right: depth was 0.55 to 0.75, width was -0.5 to 0.5 and heightwais 0.3. The bench on the left: depth was -0.55 to -0.75, width was -0.5 to 0.5 and height was 0.3.

Observation and Action The robot-arm agent collected observations and corresponding actions were generated using the policy parameter θ. In our setup, each episode is limited to 64 steps. The agent received a positive reward signal if it moved closer to the goal and received a negative reward signal if it moved further from the goal. The episode is terminated and the negative reward signal is fed-back to the learning algorithm if the robot arm moved into an impossible physical configuration, such as: moving into the ground, hitting the bench, or crashing its joints. This is achieved by implementing collision detection for the robot arms. The learning of parameters θ has been described in the Section 2. In the training phase, rewards are logged and used as the learning indicator.

Evaluation Criteria The performance of the trained model was measured using the final distance between the effector and the goal (in this implementation, the episode is terminated if the distance was less than or equal to 0.05 unit). In this report, after the models were trained, they were tested using randomly generated goals for 200 trials, each trial was limited with 64 steps. For each trial, the final distance between the effector and the goal was recorded. Comparing histogram plots among different models revealed their performances.

Parameters Setting We started our experiments by running task 1 nine times based on combinations of the following learning rate, $lr \in \{0.05, 0.01, 0.005\}$ and perturbation $N \in \{16, 24, 32\}$. Each run was limited to only 300 time steps. We

Table 1. Parameters employed in our ARS experiments

	Learning rate α	Constant ν	Perturb N	Top performing	Episode steps	Training steps	Evaluation episodes
All tasks	0.02	0.03	24	12	64	800	200

inspected the results and found that too low a learning rate and too many perturbations may not be desirable. Table 1 summarizes the parameters employed in our experiments.

4 Empirical Results and Discussion

A total of twelve trained models were created for the evaluation of ARS, four models for each task. Figure 2 (left pane) summarises the rewards obtained during the training phase. We limited the training to 800 time-steps, then saved the models.

Each model was then tested with 200 random goal positions (the choice of 200 is arbitrary). The final distances between the effector and the goal were recorded and their histograms are plotted on the right pane of Figure 2. It is observed that ARS could learn task 1 and task 2 better than task 3.

For task 1 and 2, the robot learned to locate the goal successfully. It could, most of the time, reach the goal within 0.2 units within 64 steps. Task 3 is, however, less successful. It is observed that the robot performed well on one of the benches while performing poorly on the other bench. This could be contributed to the linear policy parameters θ.

Comparing with Results from PPO *Proximal Policy Optimization (PPO)* is the current state of the art policy gradient method. In comparison to ARS, PPO requires more training samples. Since PPO employs neural network which gives non-linear policy parameters. We decided to train the three tasks using PPO for 1.2 million steps and compared results from both techniques.

The density plots of final distances obtained from PPO are superimposed with results from ARS (see Figure 3). The performance from ARS and PPO are comparable. We would comment that PPO offers a slightly better output quality as the density plots are slightly shifted to the left. This implies that the distance to the goal is slightly nearer than ARS method. PPO appears to deliver a better performance in task 3 as well which could be from its non-linear policy. Finally, ARS technique possesses a much better sample efficiency than PPO.

5 Conclusion

We investigated the application of augmented random search (ARS) in training robot-arm reaching tasks. Three tasks were set up for the 6DoF robot arm

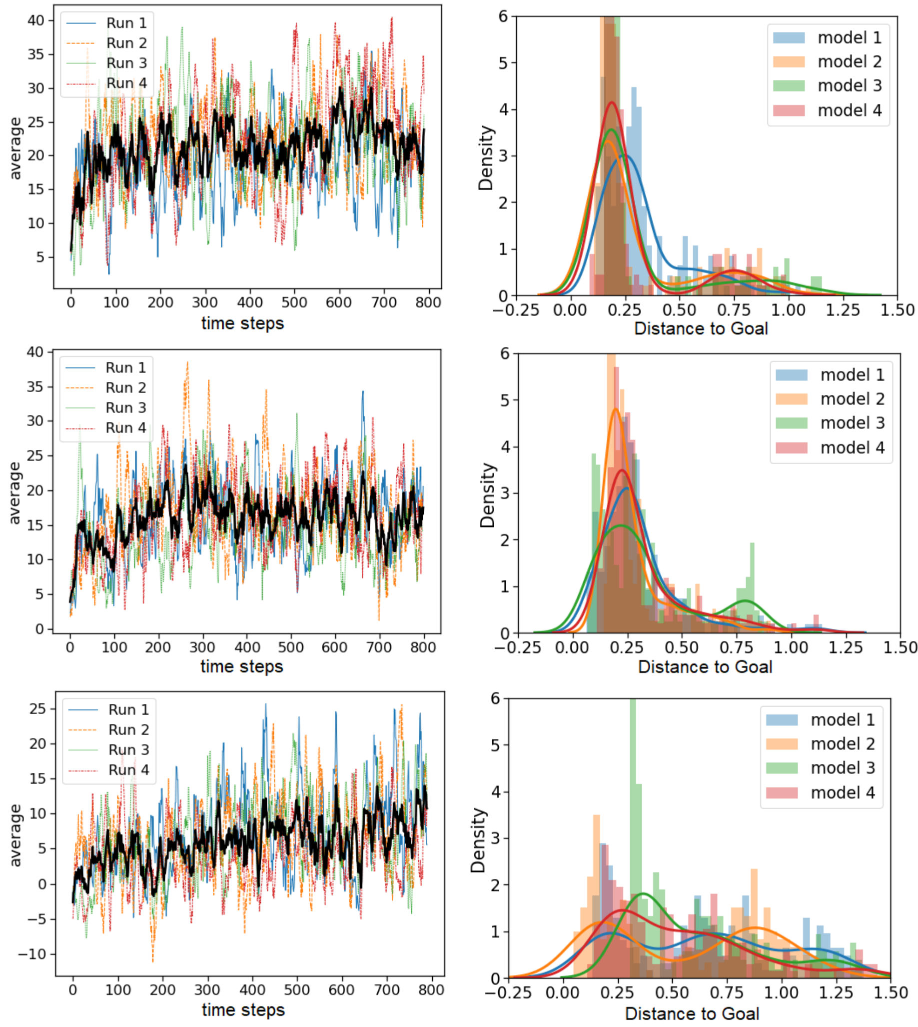

Fig. 2. Top row: The training rewards from task 1 (left). The black line is the average reward from the four runs. The evaluation of the four models based on the final distance between the effector to the goal is shown on the right pane. Second and third rows: Rewards and evaluations of task 2 and task 3, respectively.

to reach a randomly spawned cube on a bench. The empirical results confirm previous results by [4] that ARS requires fewer samples in the training process (i.e., sample efficiency). We believe that policy learning based on a linear policy may be one of the reasons behind its sample efficiency property. We speculate that the performance of a linear policy approach should be inferior to a non-linear policy approach. Hence, the three tasks were also benchmarked with the current state-of-the-art proximal policy optimization (PPO) which employed non-linear

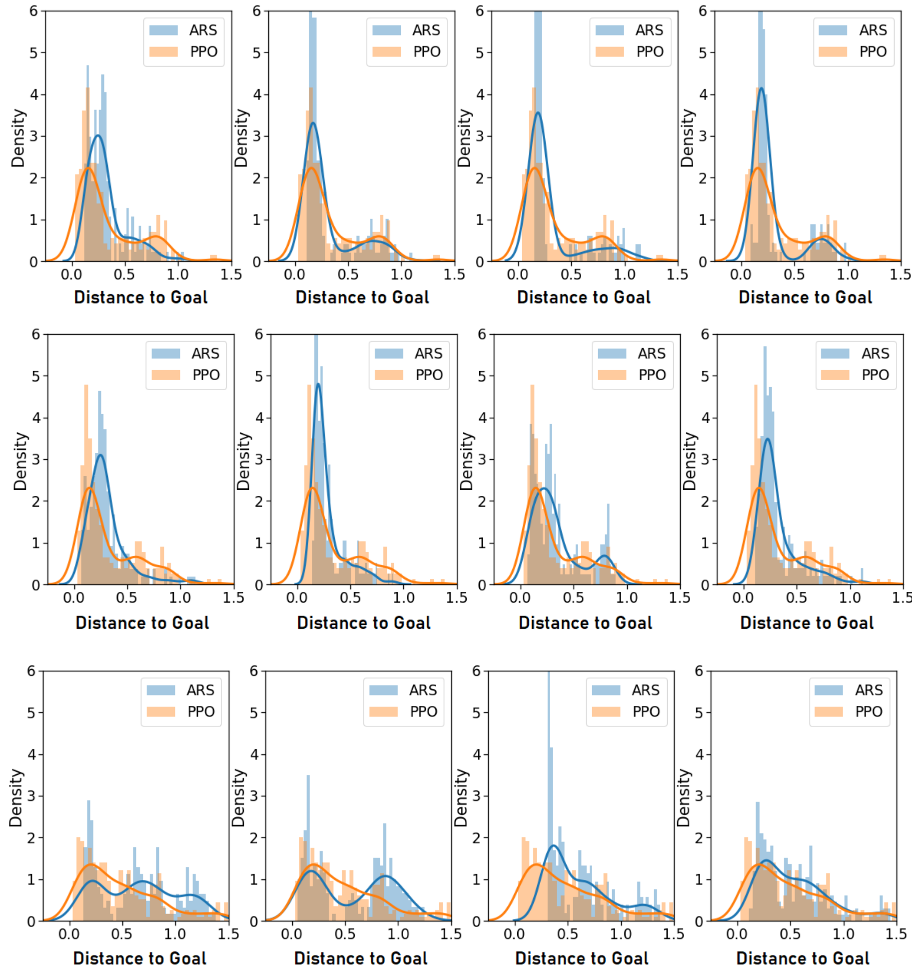

Fig. 3. This figure provides graphical comparison between ARS annd PPO. The density plot of task 1, 2 and 3 from PPO are superimposed on the density plots from ARS. Top row. second row and thgird rows show the output of task 1, task 2 and task 3, respectively.

policy parameters θ. From the empirical results, PPO exhibits a slightly better performance than ARS from all test runs.

In future works, we are interested in exploring the skill transfer between tasks. Transferring skills learned in performing one task to another relevant task without retraining (or with a short training session) will mitigate issues related to training effectiveness in deep RL such as the sample efficiency issue, the long training time issue.

Acknowledgments We would like to thank ... for their financial support given to this research.

References

1. Mnih, V., Kavukcuoglu,K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature, 518(7540):529 (2015)
2. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization. In: arXiv:1502.05477 (2015)
3. Islam, R., Henderson, P., Gomrokchi, M., and Precup, D.: Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. arXiv preprint arXiv:1708.04133, (2017)
4. Mania, H., Guy, A., Recht, B.: Simple random search of static linear policies is competitive for reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 1800–1809 (2018)
5. Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D.: Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents. (2020)
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. In: arXiv:1707.06347 (2017)
7. Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems, pp. 1057–1063 (1999)
8. Tesaruo, G.: Neurogammon wins computer olympiad. Neural Computation **1**:321-323 (1989)
9. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., et al.: Deterministic Policy Gradient Algorithms. In: proceedings of the 31st International Conference on Machine Learning (ICML), Jun 2014, Beijing, China. [hal-00938992] (2014)
10. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: arXiv:1509.02971 (2019)