EFFICIENT EXPLORATION VIA ACTOR-CRITIC ENSEMBLE

Sihao Chen
wingnutbc@berkeley.edu

Rahul Kumar
rahulkumar@berkeley.edu

December 19, 2020

ABSTRACT

Off-policy actor-critic Reinforcement Learning (RL) algorithms like Deep Deterministic Policy Gradient (DDPG) suffer from instability and dependence on careful hyperparameter tuning. We propose an off-policy actor-critic RL algorithm, Full Ensemble Deep Deterministic Policy Gradient (FEDDPG), that uses two ensemble functions to combine multiple actor networks and critic networks for exploration. In addition to using an ensemble, our algorithm improves the stability and robustness of DDPG by incorporating multi-step learning and prioritized experience replay. The performance of the agent outperforms state-of-the-art off-policy methods on multiple MuJoCo continuous control environments. Finally, we describe an ensemble formulation of SAC, indicating that some of our approaches can also be applied to algorithms that learn a non-deterministic policy.

***K*eywords** Actor-Critic · Ensemble · Multi-Step Learning · Prioritized Experience Replay

1 Introduction

The deep deterministic policy gradient (DDPG) algorithm is an off policy, actor-critic method that trains a Q function and a policy over a continuous action space [1]. One of the primary issues with DDPG is that it can be unstable and requires very careful hyperparameter tuning. Twin Delayed DDPG improves upon the original DDPG by maintaining two Q functions, adding clipped noise to actions during training, and performing delayed policy updates [2]. The Soft Actor Critic (SAC) algorithm also improves the stability of DDPG by incorporating the double Q trick, and by adding a maximum entropy term to the objective [3]. An earlier result by Hessel et al. proposed Rainbow, an agent incorporating double Q-learning, prioritized replay, and other methods for increasing the performance of DQN.

Some previous works use ensemble methods to improve the performance of RL agents. Wiering et al. [4] designed four ensemble methods combining five RL algorithms based on their value functions. Hans et al. [5] used an ensemble of networks to improve the performance of Fitted Q Iteration [6][7]. Bootstrapped DQN [8] uses a Q ensemble to approximate Thompson sampling, resulting in improved exploration and a performance boost in challenging video games. Zhang et al. [9] uses an actor ensemble and performs look-ahead tree search with those actors in continuous control problems. However, none of these works combine the actor ensemble and the critic ensemble, nor do they compare their performance with the state-of-the-art off-policy reinforcement learning algorithms, namely TD3 [2] and SAC [3].

Considering the techniques described above for improving the stability and efficiency of DDPG, we propose an improvement to DDPG that utilizes an ensemble of actor networks and critic networks. We call a function that reduces the outputs of the ensemble to one element an ensemble function. For example, an ensemble function could, given the outputs of each actor and critic network, select the single action with the highest average Q value. Previous results have used Q-Ensembles to improve exploration rates by considering upper confidence bounds (UCB) [10]. That is, an agent selects the action which maximizes the expected Q value plus a weight times the standard deviation of all the Q values for that action [10].

Additionally, we show that the ensemble method also works for stochastic policies by formulating a simplified ensemble version of SAC. Experiments show that the performance of this ensemble SAC method exceeds the existing version.

2 Background

We provide a very brief summary of Markov decision processes (MDPs) and some established algorithms for reinforcement learning. A Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, consists of a state space $\mathcal{S}$, a set of actions $\mathcal{A}$,

transition probabilities $T(s, a, s') = P(s'|s, a)$, a reward function $R(s, a)$, and a discount factor γ. The goal is to find a parameterized policy $\pi_\phi(s)$ that maximizes the expected infinite horizon reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$ under that policy. The Deep Deterministic Policy Gradient (DDPG) algorithm simultaneously trains a policy network π with parameter vector ϕ and a Q function with parameter vector θ [1]. For stability, target Q and policy networks are maintained with parameters θ' and ϕ'. These parameters are updated by Polyak averaging with the parameters θ and ϕ. Starting from some state s, DDPG performs the action $\pi_\phi(s)$, possibly with some added noise, collects a sample (s, a, r, s'), and stores that sample in a replay buffer $\mathcal{B}$. Periodically, a batch $\mathcal{D}$ of samples is drawn uniformly at random from $\mathcal{B}$, and the actor and critic functions are updated by taking steps on the following loss functions:

$$J_Q(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s')) - Q_\theta(s, a) \right)^2 \right]$$
$$J_\pi(\phi) = - \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[Q_\theta(s, \pi_\phi(s)) \right]$$

Twin Delayed DDPG (TD3) modifies DDPG to use two Q functions Q_{θ_1} and Q_{θ_2}, each with a corresponding target function [2]. When calculating backups, the target Q function with a smaller output is used:

$$y = r + \gamma \min_i Q_{\theta'_i}(s', a'),$$

where $a' = \pi_{\phi'}(s') + \epsilon$, and ϵ is clipped noise. The Q networks are then updated by taking steps on the losses

$$J_{Q_i}(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(Q_{\theta_i}(s, a) - y \right)^2 \right].$$

The policy is updated according to

$$J_\pi(\phi) = - \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[Q_{\theta_1}(s, \pi_\phi(s)) \right].$$

In our experiments, we found that modifying this update to

$$J_\pi(\phi) = - \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[\min Q_{\theta_i}(s, \pi_\phi(s)) \right], i = 1, 2$$

improves the training speed and final performance (see TD3 and 2Q1Pi in Table 2). Hence, we propose a new ensemble method that choose an appropriate value from a critic ensemble.

3 Extensions to DDPG

3.1 Multi-Step Learning

Q-learning often tries to optimize a mean square Bellman error, with the Bellman backup calculated as

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_{\phi'}(s_{t+1}, a_{t+1}) \tag{1}$$

This target can be biased due to an incorrect critic network. An n-step return estimator [11] tries to reduce target value bias by calculating targets using an n-step lookahead:

$$y_t = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{t'} + \gamma^N \max_{a_{t+N}} Q_{\phi'}(s_{t+N}, a_{t+N}) \tag{2}$$

With appropriate choices of n, multi-step return typically leads to faster learning, especially in early steps. However, it suffers from high bias when used in off-policy methods. This problem can be fixed by importance sampling [12], which is also used by Prioritized Experience Replay (3.2).

3.2 Prioritized Experience Replay

In deep reinforcement learning, experience replay [13] allows agents to learn from their interactions in a different order from the one in which they collected those interactions during exploration. While traditional methods draw samples uniformly randomly from all saved interactions, agents with Prioritized Experience Replay [14] sample transactions according to their importance. D4PG [15] uses the absolute TD-error $|\delta|$ of the experiences to evaluate the value of the experiences, where the TD-error of a Q-network Q_ϕ is given as:

$$\delta = r(s_t, a_t) + \gamma Q'_\phi(s_{t+1}, a_{t+1}) - Q_\phi(s_t, a_t). \tag{3}$$

However, TD-error alone does not take into account the importance of the examples to the actor networks. DDPGfD [16] uses a modified priority function:

$$p = \delta^2 + \lambda|\nabla_a Q_\phi(s_t, a_t)|^2 + \epsilon + \epsilon_D, \tag{4}$$

The second term represents the loss applied to the actor, ϵ is a small positive constant to make sure all samples have a positive priority, and ϵ_D is a positive constant to increase the probabilities the demonstration transactions get sampled. We don't have demonstration transactions, so we remove ϵ_D. In addition, we found the square in (4) leads to unstable updates in our experiments, so we modified the priority to:

$$p = \delta + \lambda\|\nabla_a Q_\phi(s_t, a_t)\| + \epsilon. \tag{5}$$

similar to DDPGfD [16], we use $\beta = 1$ as we want to learn the correct distribution from the beginning. We selected $\alpha = 0.5$ after testing different values of α in multiple environments.

4 Efficient Exploration via Actor-Critic Ensemble

Empirically, multiple policy networks can learn the optimal policy in complex environments where the best strategy cannot be well represented by a single policy network [9]. In addition, multiple Q networks attenuate the overestimation bias of Q-learning [2]. Based on these two observations, we propose a method called Full Ensemble DDPG (FEDDPG). During training, the agent picks an action according to the actor ensemble function f, with additive clipped noise. The resulting interaction with the environment is stored in a prioritized buffer with a priority calculated according to (5). To update our networks, we draw N samples from our replay buffer based on their priorities. When updating networks for a sample (s, a, r, s'), our agent picks an action $\tilde{a}$ using the actor ensemble function f again. The critic ensemble function g is used to compute an estimate of the reward our agent expects to accrue starting from the state s'. We compute target Q values as $y = r + \gamma g(s', \tilde{a})$, and then update each Q function independently towards the targets. The policy is updated in a manner similar to DDPG, except that for each sample, we only consider the Q function that produces the smallest value. This is further described in section 5.1. Pseudocode for our algorithm is shown below.

Algorithm 1 Full Ensemble DDPG

Input: number of policy networks K_1, number of Q networks K_2, total steps T, batch size N, discount γ, policy delay d, polyak update factor τ, policy ensemble function f, critic ensemble function g

Initialize K_1 copies of independently initialized policy networks $\{\pi_{\phi_i}\}_{i=1}^{K_1}$
Initialize K_2 copies of independently initialized Q networks $\{Q_{\theta_j}\}_{j=1}^{K_2}$
Define $\Phi := \{\phi_i\}_{i=1}^{K_1}$, $\Theta := \{\theta_j\}_{j=1}^{K_2}$
Initialize target networks $\Phi' \leftarrow \Phi, \Theta' \leftarrow \Theta$
Initialize replay buffer $\mathcal{B}$
for step $t = 1, \cdots, T$ **do**
    Pick an action with random noise $a_t \leftarrow f(s_t|\Phi, \xi_t), \xi_t \sim \mathcal{N}(0, 1)$
    Take action a_t, receive state s_{t+1} and reward r_t from environment
    Add (s_t, a_t, r_t, s_{t+1}) to replay buffer $\mathcal{B}$
    Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
    Pick an action using actor ensemble $\tilde{a} \leftarrow f(s'|\Phi')$
    Calculate Bellman backup using critic ensemble $y \leftarrow r + \gamma g(s', \tilde{a}|\Theta')$
    Update critics $\theta_j \leftarrow \text{argmin}_{\theta_j} N^{-1} \sum (y - Q_{\theta_j}(s, a))^2, j = 1, \cdots, K_2$
    if t mod d **then**
        Update policies by the deterministic policy gradient:
        $\nabla_{\phi_i} J(\phi_i) = N^{-1} \sum \nabla_a \min_j Q_{\theta_j}(s, a)|_{a=\pi_{\phi_i}(s)} \nabla_{\phi_i} \pi_{\phi_i}(s), i = 1, \cdots, K_1$
        Update target networks:
        $\Theta' \leftarrow \tau\Theta + (1 - \tau)\Theta'$
        $\Phi' \leftarrow \tau\Phi + (1 - \tau)\Phi'$
    end if
end for

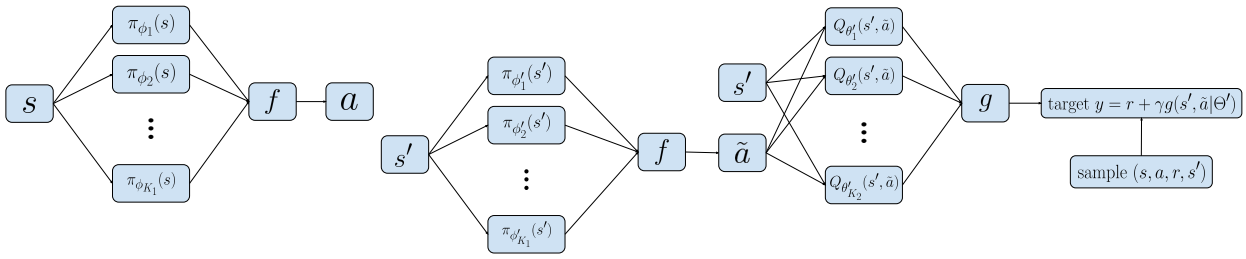

Figure 1: A schematic representation of the networks our algorithm uses. The ensemble function f selects an action to use given the output of each policy. The function g is used for calculating regression targets given the outputs of the target Q networks.

Table 1: Hyperparameters used for Figure 2.

Method	HalfCheetah-v2	Walker2d-v2	Ant-v2	Swimmer-v2
Prioritized Experience Replay	True	False	True	False
Target Steps	1	1	1	5
K_1	1	2	1	1
K_2	2	3	2	2

5 Experiments

5.1 Ensemble Functions

We propose two ways to choose the best action network for exploration and calculating target values: UCB Ensemble and Min-Q Ensemble. For the UCB ensemble, we construct a UCB by adding the empirical standard deviation $\sigma(s_t, \pi_{\phi_i}(s_t))$ of $\{Q_j(s_t, \pi_{\phi_i}(s_t)\}_{j=1}^{K_2}$ to the empirical mean value $\mu(s_t, \pi_{\phi_i}(s_t))$ of $\{Q_j(s_t, \pi_{\phi_i}(s_t)\}_{j=1}^{K_2}$. The agent chooses the action that maximizes the UCB, where the standard deviation is weighted by some hyperparameter λ_s:

$$\begin{aligned} i_t &= \underset{i}{\operatorname{argmax}} \{\mu(s_t, \pi_{\phi_i}(s_t)) + \lambda_s \sigma(s_t, \pi_{\phi_i}(s_t))\} \\ a_t &= \pi_{\phi_{i_t}}(s_t) \end{aligned} \tag{6}$$

The mean and standard deviation in (6) are computed over the Q values $Q_{\theta_j}(s_t, \pi_{\phi_i}(s_t))$. Intuitively, an agent acting using a Min-Q ensemble prefers the action that maximizes the most pessimistic expectation of reward, and therefore simply chooses the action that has the highest minimum Q value over all Q-networks:

$$i_t = \underset{i}{\operatorname{argmax}} \min_j Q_j(s_t, \pi_{\phi_i}(s_t))_{j=1}^{K_2}. \tag{7}$$

5.2 Evaluation

To evaluate our algorithm, we compare the performance of our method to previous techniques, in particular DDPG [1], TD3 [2], and SAC [3] on multiple MuJoCo continuous control tasks [17], interfaced through OpenAI Gym [18]. We used the original evironment settings proposed by Brockman et al. without modification of the environment rewards.
For all the algorithms we implemented, we use the same two-layer perceptron (MLP) of 400 and 300 hidden nodes respectively. We use rectified linear units (ReLU) between each linear layer as the activation function for both the actor and critic, and a final tanh unit following the output of the actor. We then scale the action by multiplying it with the upper bound of the action value. Following the configuration in the TD3 paper [2], the critic receives both the state and action as input to the first layer. We use a replay buffer with a capacity of 10^6 to store the transactions. Replay buffers with and without prioritized replay were tested. We show the learning curves of the best configurations for comparison with other algorithms. For multi-step returns, we choose different target steps for each environment. The hyperparameters used to generate the plots in Figure 2 are shown in Table 1.
All network parameters are updated using Adam [19] with a learning rate of 10^{-3}. We set the maximum episode length to 1000. Each time the maximum episode length is reached or the environment returns a termination signal, the networks are updated using the steps in the current episode with a mini-batch of 100, sampled from a replay buffer containing the entire history of the agent. We also adopt clipped Gaussian noise and delayed policy updates from TD3

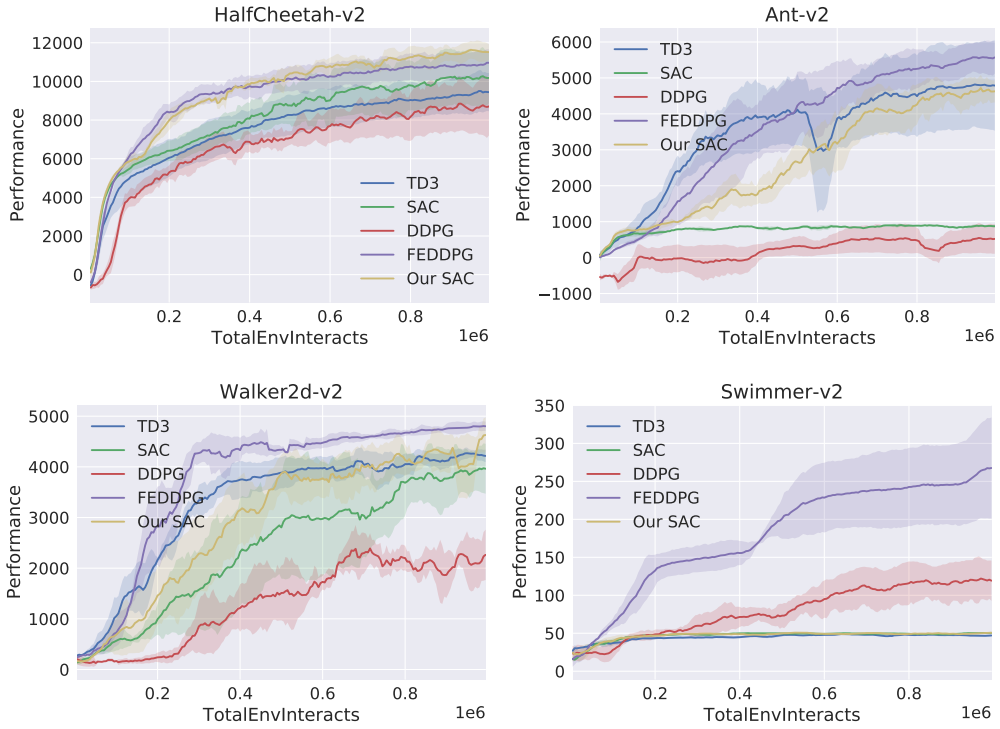

Figure 2: Learning curves for the OpenAI gym continuous control tasks. Full Ensemble DDPG (purple) outperforms other techniques in all the environments.

[2]. In the calculation of the Bellman backup, a random noise $\epsilon \sim \mathcal{N}(0, 0.2)$ is added to the target policy, clipped to $(-0.5, 0.5)$. The delayed policy updates consist of only updating the actor and target critic network every d iterations, with $d = 2$. Both target networks are updated by Polyak averaging using $\tau = 0.005$. As in TD3 [2], we also use a purely exploratory (randomized) policy for the first 10000 time steps of stable length environments (HalfCheetah-v2 and Ant-v2) and the first 1000 time steps for the remaining environments. Afterwards, we use an off-policy exploration strategy, adding Gaussian noise $\mathcal{N}(0, 0.1)$ to each action.

To show that our method also works for stochastic policies, we implemented an ensemble version of SAC by simply replacing the Q-value used to calculate loss from the first Q-network to the Min-Q ensemble of the Q-networks. Only 1 actor network is used for all tasks.

The learning curves are presented in Figure 2. FEDDPG matches or outperforms all other algorithms in final performance across all tasks and has a faster learning speed in most tasks. The final performance and learning speed of our ensemble SAC algorithm surpasses the original SAC implementation in all tasks tested.

5.3 Ablation Study

We perform ablation studies to understand the contribution of each individual component: multi-step return, prioritized experience replay, actor ensemble, and critic ensemble. We present our results in Table 2, in which we compare the performance of various ablations of FEDDPG, along with our modifications to the architecture and hyper-parameters. The significance of each component varies from task to task. While one policy network and two critic networks are enough for many environments, increasing the number of networks improves the performance of the agent for some tasks. Multi-step return and prioritized experience replay improve the agent's performance significantly in some tasks while worsening it in other tasks.

6 Discussion

The ensemble method can be seen as a preliminary high-level policy in hierarchical reinforcement learning [20]. The Option framework [21][22] is a hierarchical reinforcement learning framework that uses high-level policies to control lower-level policies (options). The Option-Critic Architecture [23] introduces a method to learn options end-to-end using policy gradients, which can work on both discrete and continuous state and action spaces. A newer paper [9]

Table 2: Average return over the last 10 evaluations over 10 trials of 1 million time steps. Maximum value for each task is bolded. mQnP means $K_1 = n, K_2 = m$.

Method	HalfCheetah-v2	Walker2d-v2	Ant-v2	Swimmer-v2
DDPG	8416.36	2180.22	380.33	122.16
TD3	8671.86	4138.32	4421.38	45.73
2Q1Pi	9876.06	**5003.46**	5085.48	47.91
3Q2Pi	8916.56	4744.50	5277.33	54.19
2Q1Pi+5steps	8671.86	2246.41	1362.74	136.58
3Q2Pi+5steps	8362.57	2895.26	1476.20	160.07
2Q1Pi+PER	9101.12	4137.24	**5425.78**	46.67
3Q2Pi+PER	**9889.42**	3931.52	5132.02	52.96
2Q1Pi+PER+5steps	4790.63	3654.46	1209.88	48.65
3Q2Pi+PER+5steps	4680.68	1972.01	1574.09	**209.55**

relates actor ensembles with options and performs a look-ahead tree search with those actors in continuous control problems.

DDPG [1] adds noise sampled from a noise process $\mathcal{N}$ to the actor policy. An Ornstein-Uhlenbeck process [24] can be used to generate temporally correlated noise for exploration in physical control problems, although Gaussian noise is often used in practice. TD3 [2] adds a clipped Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma)$ to each dimension of the action. Additionally, NoisyNets [25] are neural networks whose weights and biases are perturbed by a parametric function of the noise. A linear layer represented by $y = wx + b$ can be converted to a corresponding noisy linear layer defined by $y = (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b$. NoisyNets have been shown to outperform ϵ-greedy exploration in discrete action spaces [25]. Future works can use NoisyNets to improve the exploration further.

References

[1] 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:1509.02971 [cs, stat]*, Sep 2015. arXiv: 1509.02971.

[2] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv:1802.09477 [cs, stat]*, Feb 2018. arXiv: 1802.09477.

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

[4] M. A. Wiering and H. van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):930–936, Aug 2008.

[5] Alexander Hans and Steffen Udluft. Ensembles of neural networks for robust reinforcement learning. In *2010 Ninth International Conference on Machine Learning and Applications*, page 401–406, Dec 2010.

[6] Martin Riedmiller. *Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method*, volume 3720, page 317–328. Springer Berlin Heidelberg, 2005.

[7] András Antos, Csaba Szepesvári, and Rémi Munos. Fitted q-iteration in continuous action-space mdps. page 9–16, 2008.

[8] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv:1602.04621 [cs, stat]*, Jul 2016. arXiv: 1602.04621.

[9] Shangtong Zhang and Hengshuai Yao. Ace: An actor ensemble algorithm for continuous control with tree search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5789–5796, Jul 2019.

[10] Richard Y. Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. Ucb exploration via q-ensembles. *arXiv:1706.01502 [cs, stat]*, Nov 2017. arXiv: 1706.01502.

[11] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[12] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. June 2016.

[13] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

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

[15] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. Apr 2018.

[16] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv:1707.08817 [cs]*, Oct 2018. arXiv: 1707.08817.

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

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

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. Dec 2014.

[20] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1–2):41–77, Jan 2003.

[21] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, Aug 1999.

[22] Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

[23] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, Feb 2017.

[24] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical Review*, 36(5):823, Sep 1930.

[25] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, and et al. Noisy networks for exploration. *arXiv:1706.10295 [cs, stat]*, Jul 2019. arXiv: 1706.10295.