Implementation

REDQ - Randomized Ensembled Double Q-Learning

Alberto Maté
Lukas Meggle

ATCI
MSc in Artificial Intelligence - UPC, UB & URV

UNIVERSITAT DE BARCELONA

UNIVERSITAT ROVIRA i VIRGILI

Contents

1 Introduction

This report presents the implementation and testing of the REDQ (Randomized Ensembled Double Q-Learning algorithm) [1], along with an analysis of Soft Actor-Critic (SAC) [2], in the context of the Advanced Topics in Computational Intelligence subject. Since REDQ is considered a generalization of SAC, analyzing both algorithms will provide a deeper understanding of REDQ's functionalities and its improvements over SAC. The focus of the experimentation will be on two environments: LunarLander-v2 and BipedalWalker-v3, both from the OpenAI Gym library. These environments are chosen for their diverse state and action spaces, which provide a comprehensive testing ground for the REDQ algorithm and its comparison to SAC.

2 Environments

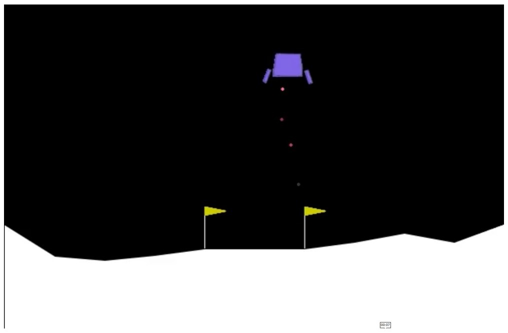
This section details the two environments used to evaluate the performance of REDQ and SAC. Both environments are provided by OpenAI Gym [3], a popular toolkit for reinforcement learning research.

2.1 LunarLander-v2

The LunarLander-v2 environment is a landing simulation where a lander must land on a pad on the moon's surface. The lander has primary and secondary engines, as well as two legs that should touch the ground when landing. The state consists of six real values and two binary values, while the action is composed of two real values. The environment gives different rewards based on various conditions, such as landing, crashing, using engines, and stepping away from the goal.

2.2 BipedalWalker-v3

The BipedalWalker-v3 environment is a simulation of a bipedal robot walking through a slightly uneven terrain. The robot has two legs, each with two joints, and the state is composed of 24 real values, including hull angle speed, angular velocity, horizontal and vertical speed, angular position and speed of the joints, binary values for leg contact, and 10 lidar measurements. The action is a vector of four real values representing the torque to apply to each joint. The environment rewards the robot for moving forward and penalizes it for falling or applying motor torque.


(a) Lunar Lander Environment

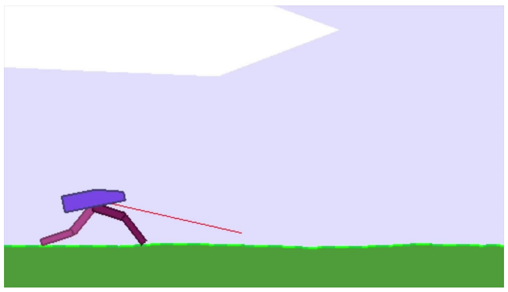

(b) Bipedal Walker Environment

Figure 1: Environments used for experimentation.

3 REDQ

REDQ [1] is a model-free reinforcement learning algorithm that achieves high sample efficiency in continuous action space domains. The algorithm employs a high Update-To-Data (UTD) ratio, which is the number of updates taken by the agent compared to the number of actual interactions with the environment. This high UTD ratio contributes to the algorithm's high sample efficiency. The key components of REDQ include:

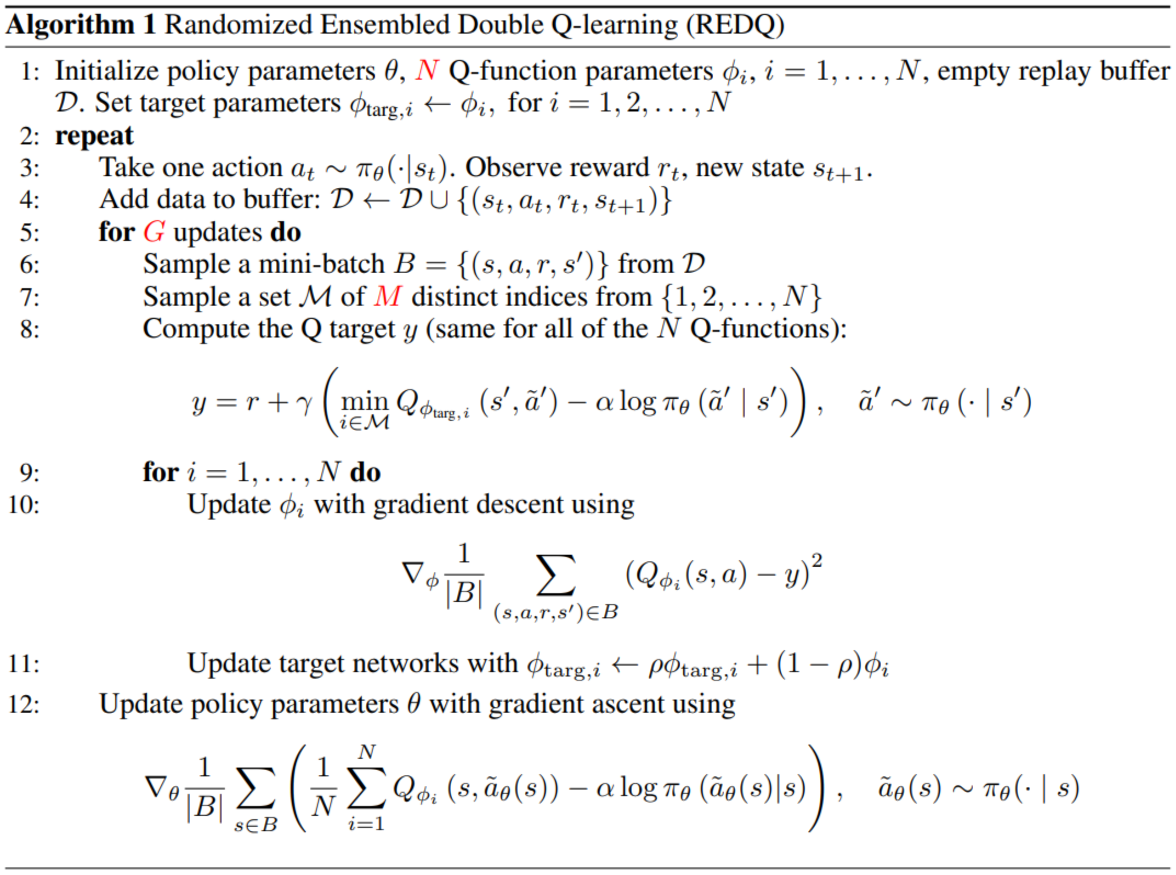
- UTD ratio > 1: REDQ uses a UTD ratio greater than 1, which enables the algorithm to achieve high sample efficiency.
- Ensemble of Q-functions: REDQ utilizes an ensemble of Q-functions, where each Q-function is randomly and independently initialized but updated with the same target. This reduces the variance in the Q-function estimate.
- In-target minimization: The target for the Q-function includes a minimization over a random subset of the Q-functions from the ensemble. This reduces over-estimation bias and controls the average Q-function bias.

REDQ shares similarities with Maxmin Q-learning [4], which also uses ensembles and minimizes over multiple Q-functions in the target. However, REDQ and Maxmin Q-learning have significant differences, such as the size of the subset minimized in the target and the control of over-estimation bias and variance of the Q estimate.

REDQ has three key hyperparameters: G, N, and M.

- G: Number of gradient steps per interaction
- N: Number of critic networks
- M: Size of the random subset over N critics

When $N = M = 2$ and $G = 1$, REDQ simply becomes the underlying off-policy algorithm, such as SAC. When $N = M > 2$ and $G = 1$, REDQ is similar to, but not equivalent to, Maxmin Q-learning. In practice, a wide range of values around $N = 10$ and $G = 20$ work well for REDQ. REDQ has been shown to be the first successful model-free DRL algorithm for continuous-action spaces using a UTD ratio $G >> 1$, achieving high sample efficiency and performance comparable to or better than state-of-the-art model-based methods.



Algorithm 1 Randomized Ensembled Double Q-learning (REDQ)

```
1: Initialize policy parameters $\theta$, $N$ Q-function parameters $\phi_i$, $i = 1, \ldots, N$, empty replay buffer
   $\mathcal{D}$. Set target parameters $\phi_{\text{targ},i} \leftarrow \phi_i$, for $i = 1, 2, \ldots, N$
2: repeat
3:     Take one action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward $r_t$, new state $s_{t+1}$.
4:     Add data to buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
5:     for $G$ updates do
6:         Sample a mini-batch $B = \{(s, a, r, s')\}$ from $\mathcal{D}$
7:         Sample a set $\mathcal{M}$ of $M$ distinct indices from $\{1, 2, \ldots, N\}$
8:         Compute the Q target $y$ (same for all of the $N$ Q-functions):
               $y = r + \gamma \left( \min_{i \in \mathcal{M}} Q_{\phi_{\text{targ},i}} (s', \tilde{a}') - \alpha \log \pi_\theta (\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta (\cdot \mid s')$
9:         for $i = 1, \ldots, N$ do
10:            Update $\phi_i$ with gradient descent using
                   $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s') \in B} (Q_{\phi_i}(s, a) - y)^2$
11:            Update target networks with $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i$
12:    Update policy parameters $\theta$ with gradient ascent using
           $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left( \frac{1}{N} \sum_{i=1}^{N} Q_{\phi_i} (s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta (\tilde{a}_\theta(s)|s) \right), \quad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot \mid s)$
```

Figure 2: Pseudocode of REDQ built over Soft Actor-Critic

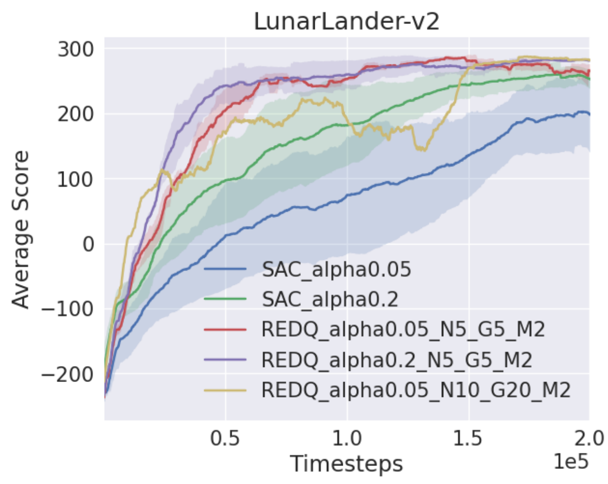
4 Results

As explained in Section 3, REDQ has 3 main hyperparameters (G, N, M). Moreover, since we have built REDQ over the off-policy algorithm SAC, we have an extra hyperparameter (α). In the following experiments we have run several combinations to analyse the effect over the learning procedure. We used two different environments (see Section 2) and different random seeds for the same configuration for a more reliable comparison.

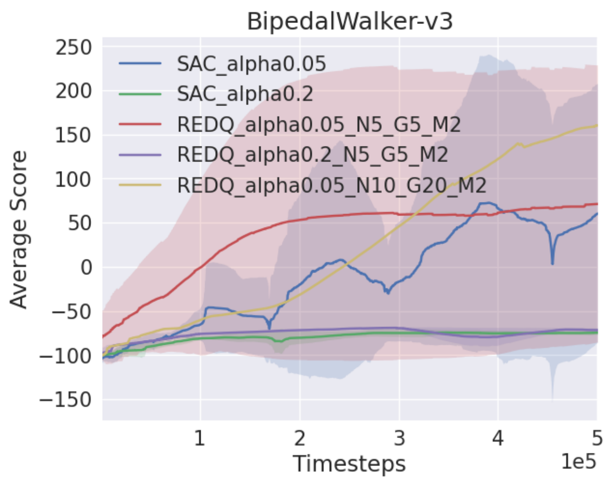
ID	α	N	G	M	Runs
SAC_alpha0.05	0.05				4
SAC_alpha0.2	0.2				4
REDQ_alpha0.05_N5_G5_M2	0.05	5	5	2	2
REDQ_alpha0.2_N5_G5_M2	0.2	5	5	2	2
REDQ_alpha0.05_N10_G20_M2	0.05	10	20	2	1

Table 1: Description of the configuration of the experiments. Runs represents the number of random seed initialization for each configuration

Table 1 provides a description of each experiment configuration. The "Runs" column denotes the number of random seed initializations used for each specific configuration. The number of runs is decreased as the parameter G increased (due to augmenting the gradient steps increase notably the training wall-clock time). However, if would like to do a better comparison of the algorithms we should run all the algorithms with an equal and higher number of initilizations. In our experiments, we varied α for both the SAC and REDQ algorithms while keeping N, G, and M constant in certain cases and varying them in others. We tried with the recommended configuration of the original REDQ paper [1] ($N = 10, G = 20, M = 2$) and with a lower complexity configuration ($N = 5, G = 5, M = 2$) since our environments are not as complex as the one used in the original paper. The number of training steps for LunarLander-v2 and BipedalWalker-v3 are 200.000 and 500.000 respectively.



(a) LunarLander-v2 (200.000 steps)



(b) BipedalWalker-v3 (500.000 steps)

Figure 3: Average reward score smoothed over 100 episodes in training for the different experiments configurations. The shade area represents the confidence interval of the reward since the same configuration has been run several times with different seeds. Note that the x-axis is the number of timesteps and not the number of episodes. The figure plotting code is based on [5].

Figure 3 shows the results for all the configurations. Now, we will analyzed in detail each of the insights that we can get:

- The α depends on the environment. For example using $\alpha = 0.2$ helps the agent to learn faster in LunarLander-v2 while the same value has detrimental effects in the BipedalWalker-v3. This is an expected outcome since in the original SAC paper [2] they claim that α should be tuned for each environment. In [6] they proposed a way to tune this parameter using an heuristic. However, this has not been implemented in this work.

- REDQ outperforms SAC in the learning speed. It can be seen that for both environments and different α, using REDQ accelerate the training phase being more sample efficient than SAC. This outcome was the main motivation of the authors of REDQ. Moreover, in the case of LunarLander-v2, REDQ seems to be more stable and less dependent on the random seed because the shade area is smaller compared with SAC configurations. However, this fact can't be affirmed for the BipedalWalker-v3 environment. For example, one of the runs REDQ_alpha0.05_N5_G5_M2 learns the task in less than 200.000 steps whereas the other run does not learning nothing.

- Using higher complexity REDQ (increasing the number of Q-function network N and the number of gradient steps G) seem to not be extremely useful. Is it true that in LunarLander-v2, REDQ_alpha0.05_N10_G20_M2 is the faster to reach the 100 average reward, although later the learning is less stable. We hypothesize that the main reason is that the environments are not complex enough.

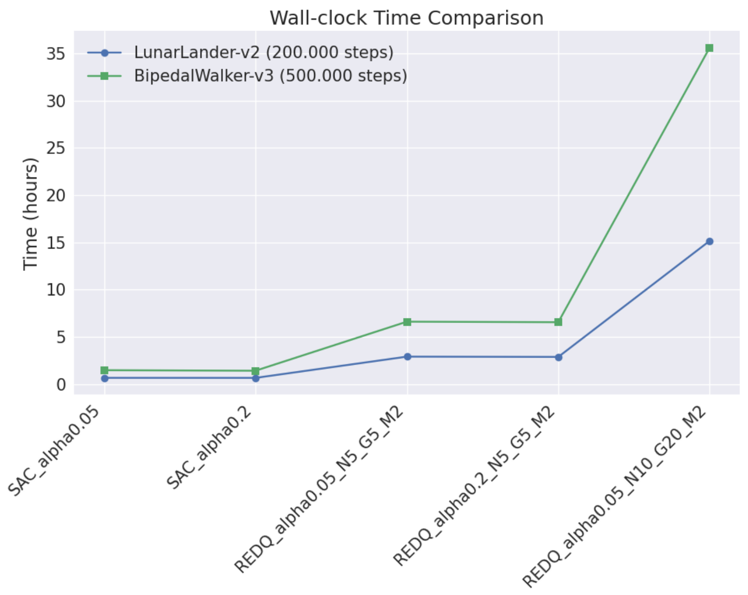


Figure 4: Training time (hours) for each configuration. Experiments have been made in a NVIDIA GeForce RTX 3090 GPU

While REDQ demonstrates greater sample efficiency than SAC, requiring fewer interactions with the environment, it comes at the cost of wall-clock time efficiency, which refers to the total training time. As shown in Figure 4, REDQ is significantly slower than standard SAC. This is primarily due to two factors: first, REDQ optimizes a much larger set of networks (N compared to 2 in SAC) and second, it performs multiple gradient steps per interaction, whereas SAC utilizes only one. These combined factors can lead to REDQ being up to 23 times slower than SAC.

Finally, Figure 6 and 5 shows qualitative analysis for the learned policies for the agents using REDQ.

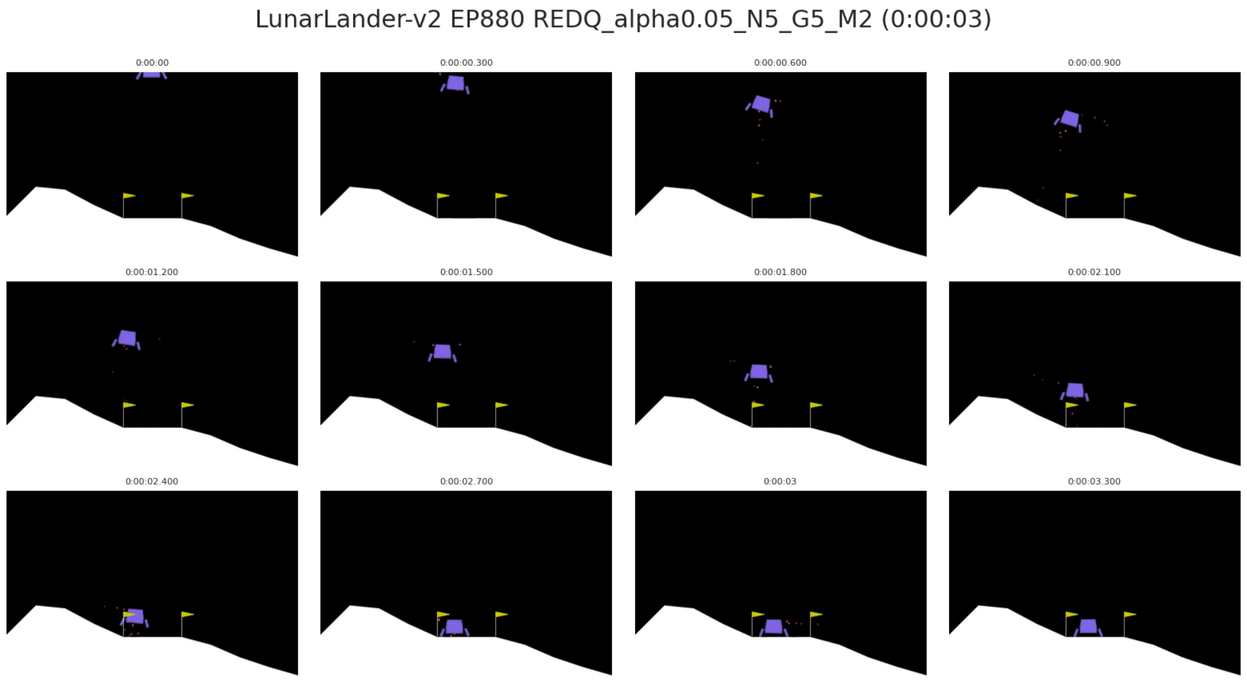


Figure 5: Frames sequence from an agent with a policy in the last training episode from REDQ_alpha0.05_N5_G5_M2 in LunarLander-v2. The spaceship is capable to land successfully in just 3 seconds

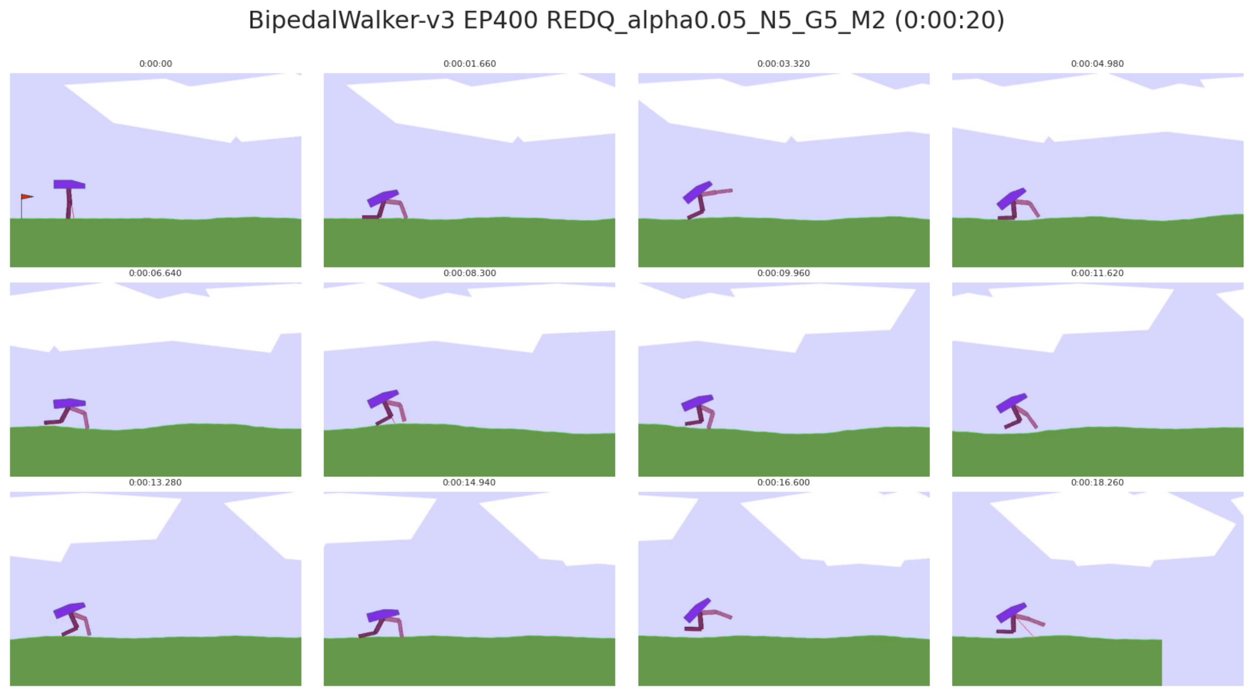


Figure 6: Frames sequence from an agent with a policy in the last training episode from REDQ_alpha0.05_N5_G5_M2 in BipedalWalker-v3. The agent learns to walk without falling reaching to the end of the path

5 Conclusion

This work investigated the performance of REDQ, a model-free reinforcement learning algorithm, and compared it to SAC in two environments: LunarLander-v2 and BipedalWalker-v3. The experiments confirmed REDQ's key strength: achieving faster learning with fewer interactions with the environment (higher sample efficiency) compared to SAC. This aligns with the theoretical foundation of REDQ's high Update-To-Data (UTD) ratio.

However, the increased sample efficiency comes at a cost. REDQ requires significantly more training time (lower wall-clock time efficiency) compared to SAC. This is primarily because REDQ optimizes a larger ensemble of Q-functions and performs multiple gradient steps per interaction. While REDQ demonstrates impressive sample efficiency, algorithms like DropQ [7] aim to achieve similar efficiency with potentially lower computational cost.

It is important to note that for a more comprehensive comparison, future experiments should involve a broader range of environments and random initilizations. This would provide a more robust understanding of how both REDQ and SAC perform across different learning tasks.

References

[1] Xinyue Chen et al. “Randomized ensembled double q-learning: Learning fast without a model”. In: *arXiv preprint arXiv:2101.05982* (2021).

[2] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[3] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[4] Qingfeng Lan et al. “Maxmin q-learning: Controlling the estimation bias of q-learning”. In: *arXiv preprint arXiv:2002.06487* (2020).

[5] Joshua Achiam. “Spinning Up in Deep Reinforcement Learning”. In: (2018).

[6] Tuomas Haarnoja et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).

[7] Takuya Hiraoka et al. *Dropout Q-Functions for Doubly Efficient Reinforcement Learning*. 2022. arXiv: 2110.02034 [cs.LG].