Monte Carlo Bias Correction in Q-Learning

Dimitris Papadimitriou(✉)

University of California, Berkeley, USA
dimitri@berkeley.edu

Abstract. The Q-learning algorithm suffers from overestimation bias due to the maximum operator appearing in its update rule. Other popular variants of Q-learning, like double Q-learning, can on the other hand cause underestimation of the action values. In many stochastic environments both underestimation and overestimation can lead to sub-optimal strategies. In this paper, we present a variation of Q-learning that uses elements from Monte-Carlo Reinforcement Learning to correct for the overestimation bias. Our method *1)* makes no assumptions on the distributions of the action values or the rewards, *2)* does not require extensive hyperparameter tuning unlike other popular variants proposed to deal with the overestimation bias and *3)* requires storing only two estimators, similar to double Q-learning, along with the most recent episode. Our method is shown to effectively control for the overestimation bias in a number of simulated stochastic environments leading to better policies with higher cumulative rewards and action values that are closer to the optimal ones, as compared to a number of well-established approaches.

Keywords: Q-learning · Overestimation · Bias

1 Introduction

Reinforcement Learning (RL) is a control technique that enables an agent to make informative decisions in unknown environments by interacting with them in time [10]. The RL algorithms can be generally categorized in model-based and model-free methods. Model-based methods learn the underlying dynamics of the system and incorporate that information in the decision process. One of the benefits of this line of work over model-free methods is the smaller amount of data required to train the agents. On the other hand, model-free methods directly estimate value functions or policies from interactions with the environment. Model-free methods do not suffer from model bias, which arises due to the insufficient estimation of the underlying dynamics, as model-based methods frequently do.

In this work, we focus on model-free methods and more specifically on variants of the celebrated Q-learning algorithm [12]. The wide popularity of Q-learning can be attributed to the simplicity of the algorithm as it follows a simple update rule that uses the current estimate of the action values of a state,

B. Goertzel et al. (Eds.): AGI 2022, LNAI 13539, pp. 343–352, 2023.
https://doi.org/10.1007/978-3-031-19907-3_33

the reward observed while transitioning from that state to the next and the maximum over all possible actions estimate of the action values at the next state. Using the latter maximum operator however has been proven to cause Q-learning to overestimate the action values. This overestimation is attributed to the uncertainty in the estimates and the possible stochasticity of the environment. This phenomenon frequently leads to poor policies [11].

To deal with the overestimation bias a number of variants of Q-learning have been proposed in the literature. Double-Q learning [6] uses a double estimator approach, where one estimator determines the maximizing action and the other independently determines that action's value. Despite the fact that double Q-learning tends to underestimate the actual value functions, it is shown to outperform Q-learning by leading to better policies in many environments. To balance between over- and under-estimation [13] extend double Q-learning by using an appropriately weighted sum of the single and double estimators in the update rule. Weighted Q-learning [1] uses a weighted estimate among the action values in the place of the maximum in the update rule. Furthermore, theoretically backed bias correction techniques have been proposed in the asymptotic regime for normally distributed rewards [8]. Finally, Maxmin Q-learning [7] uses a number of independent action value estimators and selects the maximizing action by considering the minimum of these action values for each action. It should be noted that overestimation bias has also been shown to affect policy gradient algorithms which are predominantly used in continuous control applications [3].

Another branch of model-free learning algorithms, that is broadly related to our work, is that of Monte Carlo (MC) based RL methods. The Monte Carlo Exploring Starts (MCES) algorithm uses simulations to obtain estimates of the cumulative discounted reward from a state-action pair to the end of the task and uses those estimates to update the action value of that state-action pair [10]. Although MC methods do not lead to overestimated action values they tend to have high variance in their estimates compared to Q-learning [5].

In this paper we combine the Q-learning algorithm with MC techniques to reduce the overestimation bias of the former in applications with finite state-action spaces and episodic tasks. In summary, at the end of each episode of the algorithm we compute the realized discounted cumulative reward from each visited state-action pair. Given that and the current action value estimates we can obtain an estimate for the bias incurred in each visited state-action pair, which we then use to update a running bias estimator in our problem. That bias estimate is subsequently subtracted from the action value estimates in the Q-learning update rule to correct for the overestimation bias. We show in a number of benchmark environments that our method consistently returns action value estimates with low bias while producing policies that most of the times outperform those from other approaches.

The rest of the paper in organized in the following way. Section 2 introduces basic concepts of Q-learning and tries to shed some light in the reasoning behind the overestimation bias while Sect. 3 summarizes already established techniques designed to tackle overestimation bias. Section 4 introduces in detail our bias correction method. Finally, Sect. 5 compares the performance of our method with the alternatives in a number of simulated stochastic environments.

2 Preliminaries

Reinforcement Learning applies to problems in which an agent interacts with an environment and uses that information for informative decision making with the ultimate goal being utility maximization over a finite horizon T. We model such environments with Markov Decision Processes (MDP) which are defined by tuples $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, P the transition probabilities in the environment, R the reward function and $\gamma \in (0, 1)$ is the discount factor. We assume the state and action spaces are finite and their cardinalities are denoted with $|S|$ and $|A|$ respectively and we use $\mathcal{A}(s)$ for the set of available actions in state s. The system transition probabilities $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ are denoted with $P(s_{t+1}|s_t, a_t)$. After each transition the agent observes a reward $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ obtained at state s_t by applying input a_t and transitioning to state s_{t+1} denoted with $R_{a_t}(s_t, s_{t+1})$. The discount factor γ controls for the significance of short versus long term rewards. The goal of the agent is to find an optimal policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ that maximizes the expected discounted sum of rewards $\mathbb{E}_\pi \left[\sum_{i=0}^{T} \gamma^i R_{a_t}(s_t, s_{t+1})\right]$ starting from an initial state s_0.

Such a policy can be found via Q-learning in which state-action dependent Q-functions $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ are estimated. $Q_\pi(s_t, a_t)$ quantifies the cumulative discounted reward from state s_t when applying action a_t and then following policy π for the duration of the task. At iteration $t+1$ the Q-learning algorithm with a learning rate α, which frequently is a function of the state-action pair, uses the following update rule

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{a_t}(s_t, s_{t+1}) + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right). \tag{1}$$

The set of available actions $\mathcal{A}(s_{t+1})$ at state s_{t+1} and the policy π from the action values will be dropped in subsequent expressions for notational conciseness. The optimal policy can be derived by greedily choosing the action that corresponds to the highest action value from each state s, $\pi^* = \arg\max_a Q^*(s, a)$.

The term $\max_a Q_t(s_{t+1}, a)$ appearing in (1) causes the action value estimates to be positively biased. Ideally, the update rule would choose at time step $t+1$ the action value with the highest expected value $max_a \mathbb{E}[Q(s_{t+1}, a)]$. However, the true underlying action values and consequently their expected values are unknown and instead are substituted by the sample estimates $max_a Q(s_{t+1}, a)$. The action value samples though are polluted with noise, attributed to the stochasticity of the environment and the estimation uncertainty of the action values themselves. Even if the noise has zero mean the max operator will likely still positively bias the action value estimates. For detailed proofs of the overestimation bias in Q-learning we refer the reader to [6,9]. To obtain some intuition

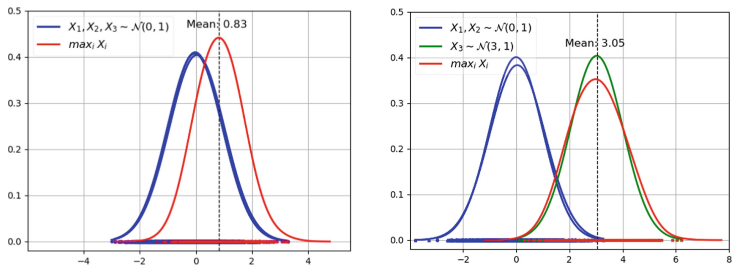

Fig. 1. (*left*) 100 simulations, each with three samples from the three normal distributions (blue dots) and the maximum of those three (red dots) and the empirical distributions of $X_i, i = 1, \ldots, 3$ and $\max_i X_i$. The mean of the empirical distribution of the maximum is 0.83. (*right*) Similar to the left but one of the normal distributions has now mean 3. (Color figure online)

on the overestimation bias we include a simulation that showcases that effect. Assume that we have N alternatives $X_1, X_2, \ldots, X_N$ each associated with realized rewards $x_i,\ i = 1, \ldots, N$ coming from a normal distribution $\mathcal{N}_i(0, 1)$. We repeatedly observe the realizations of all N rewards and we select the maximum one $\max_i x_i$. It is known from extreme value theory that the maximum of these standard normally distributed random variables will asymptotically converge to a Gumbel distribution with positive mean [4]. Intuitively, since the distributions have the same zero mean, it is highly likely that at least one of them in a realization will be positive and since we are picking the maximum of the N samples, $\max_i x_i$ most of the times will be positive as well, thus biasing the estimator. Figure 1 shows the effect of the discrepancy among the alternatives in the bias of the maximum operator for $N = 3$. When one of the alternatives is distinctively better - it comes from a distribution with higher mean - then we observe minimal overestimation, since the max function systematically selects the sample that comes from the distribution with the higher mean. On the other hand, if all distributions generate similar rewards as in the left figure, the effect of overestimation becomes significant. Frequently in applications the different action alternatives of the action values are similar to each other leading to positive bias in the estimates.

3 Related Works

The aforementioned max-operator bias frequently leads the Q-learning algorithm to overestimate action values. A number of methods have been proposed to alleviate this problem most of which resolve to using more than one estimator for the action values to tackle overestimation.

Double Q-learning [6] uses two independent estimators for the action values, one to choose the optimal action and another to extract the action value. It requires storing two tables Q^A and Q^B and the agent follows an ϵ-greedy policy in which actions can be selected using both tables, for instance

$a^* = \arg\max_a Q^A(s, a) + Q^B(s, a)$. At each iteration one of the Q-tables is randomly chosen to be updated. When table $i \in \{A, B\}$ is updated then the target in expression (1) is equal to $R_{a_t}(s_t, s_{t+1}) + \gamma Q_t^{-i}(s_{t+1}, a^*)$, where $a^* = \arg\max_a Q^i(s_{t+1}, a)$. The notation $-i$ denotes the alternative choice to i.

The structure of the weighted double Q-learning algorithm [13] is similar to that of double Q-learning as it also employs two Q-tables A and B. The major difference is that it uses a weight that is a function of both Q-tables at each iteration to update a randomly chosen Q-table with the weighted average of the two tables. When table i is updated the target value in the update rule becomes $R_{a_t}(s_t, s_{t+1}) + \gamma(\beta^i Q_t^i(s_{t+1}, a^*) + (1 - \beta^i)Q_t^{-i}(s_{t+1}, a^*))$, where β^i is a weight parameter depending on the Q-values evaluated at $a^* = \arg\max_a Q^i(s_{t+1}, a)$, $a_L = \arg\min_a Q^i(s_{t+1}, a)$ and c is a user specified hyperparameter.

Maxmin Q-learning [7] employs m independent Q-tables to balance between over- and under-estimation bias, where the number of Q-tables is another hyperparameter of the algorithm. Both the optimal action choice of the agent as well as the Q-table update rule uses the Q_{min} table which for a state s is constructed as $Q_{min}(s, a) = \min_{i \in \{1,...,m\}} Q^i(s, a), \ \forall a$. The Maxmin Q-learning target is $R_{a_t}(s_t, s_{t+1}) + \gamma \max_a Q_{min}(s_{t+1}, a))$. For some value $m > 0$ the Maxmin Q-learning estimates switch from being overestimated to being underestimated and depending on which of the two, if any, is beneficial for a particular environment the user can tune m appropriately.

4 Monte Carlo Bias Correction

Most of the aforementioned methods utilize a multiple estimator scheme to tackle the overestimation bias. In the process of creating estimates with reduced bias the double weighted and Maxmin variants of Q-learning need extra hyperparameter tuning. Our method exploits already available information from the realized trajectories in order to reduce bias without requiring significantly more tuning than the original Q-learning algorithm. The intuition behind the algorithm is to use information obtained during an episode to construct bias estimates of the action values and use those to correct for the bias. More specifically, at the end of each episode we compute the realized discounted cumulative reward from each state-action pair visited to the end of that episode. Given the current action value estimates we compute an estimate for the bias at state-action pair s, a by subtracting the realized action values from the current action value estimates (2). That estimate is then used to update a bias table, which is similar in dimensions to the Q-table, for that particular pair (3). At each iteration of the Q-learning algorithm the bias term is subtracted from the max action value to compensate for the overestimation bias. Our algorithm can be broadly seen as a combination of the Q-learning algorithm with elements from a variant of the Monte Carlo Exploring Starts (MCES) algorithm [10].

In more detail, during each episode at each time step the agent acts in the environment using an ϵ-greedy policy. During the episode, the sequence of state-action pairs visited and rewards observed is saved. Once the episode is done after T_{ep} steps, for each state-action pair in the trajectory $s_i, a_i, i = T_{ep}, \dots, 1$ we compute the realized cumulative discounted reward sequence $\hat{Q}(s_i, a_i)$ from time step i until the end of that episode. This realized $\hat{Q}$ value is an estimate of the action value that is not biased by the max operator and can be used to obtain a sample estimate for the bias incurred at state-action pair s_i, a_i as follows

$$\hat{B}(s_i, a_i) = Q(s_i, a_i) - \hat{Q}(s_i, a_i). \tag{2}$$

The samples in (2) are then used to update a running average of the bias for each s_i, a_i similar to the update rule in (1)

$$B_{t+1}(s_t, a_t) = B_t(s_t, a_t) + \alpha' \left(\hat{B}(s_t, a_t) - B_t(s_t, a_t)\right), \tag{3}$$

where we allow for the use of a different learning rate α' from the one used in (1). The target value in the Q-learning update rule of our method is

$$Q_{t+1}(s_t, a_t) = \\ Q_t(s_t, a_t) + \alpha \left(R_{a_t}(s_t, s_{t+1}) + \gamma(Q_t(s_{t+1}, a^*) - B_t(s_{t+1}, a^*)) - Q_t(s_t, a_t)\right), \tag{4}$$

Algorithm 1: Monte-Carlo Bias Corrected Q-learning (MBCQ)

```
Choose learning rates $\alpha, \alpha'$
Initialize $Q(s, a)$ randomly, $B(s, a) = 0 \; \forall s, a$
Repeat until convergence
    $T_{ep} \leftarrow 0$
    While episode not over:
        Choose action $a$ in state $s$ using policy derived from $Q$ (e.g. $\epsilon$-greedy)
        Move to state $s'$ and observe reward $R_a(s, s')$
        $a^* \leftarrow \arg\max_a Q(s', a)$
        $\delta \leftarrow R_a(s, s') + \gamma(Q(s', a^*) - B(s', a^*))$
        $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\delta$
        Store $s, a, R_a$
        $T_{ep} \leftarrow T_{ep} + 1$
        $s \leftarrow s'$
    $\hat{Q} \leftarrow R_{a_{T_{ep}}}(s_{T_{ep}})$
    For each uniquely visited tuple $(s_i, a_i, R_{a_i}), i = T_{ep} - 1, \dots, 1$:
        $\hat{Q} \leftarrow R_{a_i}(s_i, s_{i+1}) + \gamma\hat{Q}$
        $\hat{B} \leftarrow Q(s_i, a_i) - \hat{Q}$
        $B(s_i, a_i) \leftarrow (1 - \alpha')B(s_i, a_i) + \alpha'\hat{B}$
```

where $a^* = \arg\max_a Q(s_{t+1}, a)$. More details on our approach can be seen in Algorithm 1. It should be noted that our method requires the storage of the most recent trajectory along with the bias table B. For tasks in which episodes have a large duration and $\gamma < 1$ a more computationally efficient rolling window scheme can be used to approximate the total reward from a particular state. In the following section we compare the effectiveness of our method with that of other popular methods designed to tackle overestimation bias in Q-learning.

5 Experiments

This section quantifies the performance of our method, abbreviated to MBCQ to denote the Monte Carlo-like bias correction, in a number of stochastic benchmark environments both with discrete and continuous state spaces. We compare the performance of our method with that of single Q-learning (Q), Double Q-learning (DQ), Weighted Double Q-learning (WDQ) and MaxMin Q-learning (MMQ). The last three of the aforementioned methods are some of the primary approaches in literature that deal with the overestimation bias.

In all experiments we choose a polynomial learning rate for the Q-learning update $\alpha(s,a) = 1/n(s,a)^{0.8}$, where $n(s,a)$ denotes the number of times the state-action pair s,a has been visited. For algorithms that require more than one Q-table we use distinct learning rates for each table i, $\alpha_i(s,a) = 1/n_i(s,a)^{0.8}$. The probability of randomly choosing an action in the ϵ-greedy policy also diminishes with a polynomial rate, $\epsilon = 1/\sqrt{n(s,a)}$. For the learning rate of the bias update we select a constant rate $\alpha' = 0.01$. In all the experiments the discount factor γ is set to 0.95.

5.1 Roulette

Roulette is a stochastic environment consisting of a single state and 171 actions. The 170 betting actions include betting on individual numbers, red or black color, odd or even numbers, etc. The 171th action is a termination action with zero reward after which the agent walks away from the table. The agent bets each time 1\$, with no budget constraints, and the average expected reward from each betting action is 0.947\$, leading to an expected loss of 0.053\$ for each bet. The optimal strategy for the agent is to clearly walk away. All actions return to the same state except for when the agent decides to walk away.

We sequentially update the action values for each action for 10^5 trials. We repeat this experiment 10 times. We compute and report the mean of the action values over all actions, averaged over the 10 experiments for all five methods as shown in Table 1. All the methods overestimate the maximum action value as the optimal one is equal to zero. Q-learning suffers from excessive overestimation while weighted double Q-learning has diminishing bias for growing c. Double and Maxmin Q-learning show significantly lower overestimation compared to single Q-learning. On the other hand our method obtains estimates that approach the actual action values.

Table 1. Average over all non-terminating actions of the action values.

Q	DQ	WDQ ($c = 10$)	WDQ ($c = 100$)	MMQ	MBCQ
9.70\$	0.02\$	5.57\$	0.025\$	0.15\$	$\mathbf{6.3 \cdot 10^{-4}}$\$

5.2 Grid World

The grid world environment (see Fig. 2 in [6]) consists of a $n \times n$ dimensional grid in which the agent starts from the bottom left cell s_0 and tries to navigate towards the top right cell, which is the terminal state s_g. At each cell the agent can choose one of the possible actions $\{north, east, south, west\}$. If any of the actions taken leads the agent off the grid then the agent remains in the same cell. The agent receives equally likely a reward of -4 or 2 for any action leading to a non-terminal state and a reward of 15 or -5 for successfully reaching the goal state. The optimal strategy of the agent is to follow the shortest path towards the terminal state. During the experiments we record the value function of the starting state $V(s_0) = max_a Q(s_0, a)$ and the average reward per time step. The duration of the task is 10^4 iterations. The plotted curves have been smoothed with an exponential kernel with parameter 0.1 for better exposition.

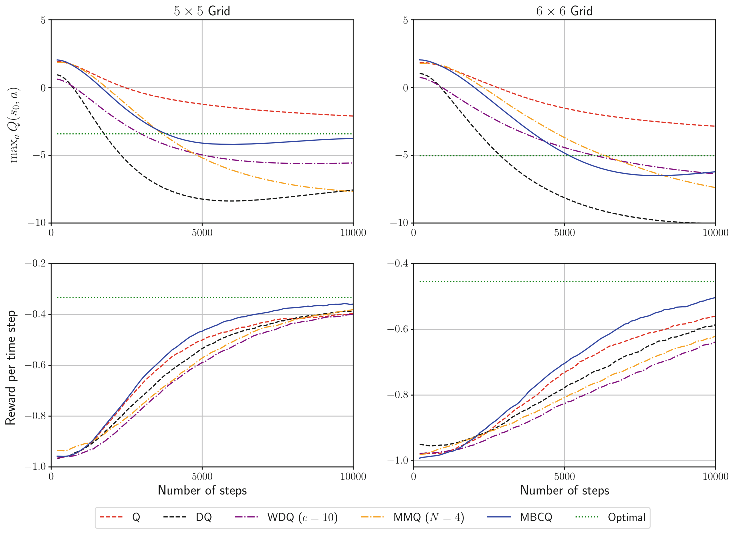

Fig. 2. Maximum action value for s_0 and average reward per step for two different grid sizes averaged over 1000 runs. The green dotted lines correspond to the optimal maximum action values and average rewards. (Color figure online)

Q-learning and double Q-learning consistently over- and under-estimate the action values respectively, as expected from their theoretical analysis. On the other hand weighted double Q-learning tends to suffer less from overestimation. Our method consistently approaches the optimal action values in all the experiments and is never far off. The average reward obtained using our method is consistently higher compared to all others. We also report in Table 2 the values of $V(s_0)$ that all five algorithms converged to after 10^5 iterations for grid sizes $n = 3, 4, 5$ and 6. The results were averaged over 1000 runs. The optimal value

Table 2. Maximum action values $V(s_0)$ after 10^5 iterations.

n	$V^*(s_0)$	Q	DQ	WDQ ($c = 10$)	MMQ ($N = 4$)	MBCQ
3	0.36	0.37	−0.65	−0.18	−0.09	**0.36**
4	−1.62	**−1.58**	−2.79	−2.25	−2.43	**−1.66**
5	−3.41	−3.30	−4.69	−4.15	−4.75	**−3.46**
6	−5.03	−4.76	−6.53	−5.82	−7.17	**−5.09**

function as a function of the grid size is given by $V^*(s_0) = 5\gamma^{2(N-1)} - \sum_{i=0}^{2N-3} \gamma^i$. Our method consistently manages to approach the optimal values of $V^*(s_0)$ closer than any of the other methods.

5.3 Taxi

The Taxi environment [2] is the last domain we will test our algorithm. In this environment the agent can pick up a passenger who is in any of the R,G,Y,B locations and transport the passenger to one of the remaining locations. For each allowed move on the map that does not deliver a passenger to the specified location the agent incurs a cost of 1. Regarding the rest of the rewards we will be studying two cases, one with deterministic and one with stochastic rewards. In the deterministic case, for illegal actions like trying to drop a passenger on the wrong location, the agent incurs a reward of −4 while the reward for correctly delivering the passenger to the specified location is 8. In the stochastic case, for misplaced passenger deliveries the agent incurs a reward of −10 or 2 and for correct deliveries the agent obtains a reward of 20 or −4, where all alternatives have equal probability of occurring.

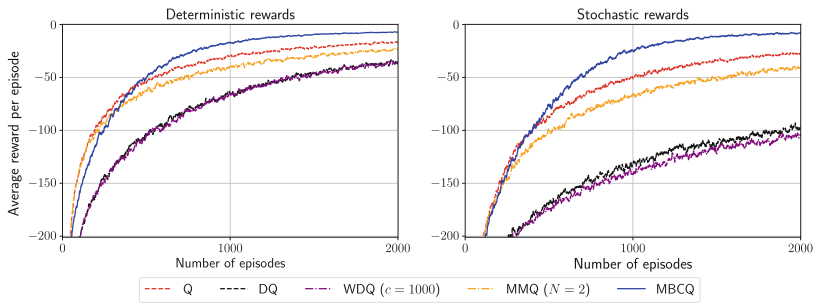

Fig. 3. Average reward per episode in Taxi environment with deterministic and stochastic rewards. Results averaged over 100 runs.

We report the average reward per iteration for 2000 iterations averaged over 100 runs. The resulting curves have also been smoothed with an exponential kernel with parameter 0.1. Figure 3 shows the average reward plots for the deterministic and stochastic cases. MBCQ clearly outperforms all the other methods on both cases by leading to policies that obtain higher rewards.

6 Summary

Overestimation bias is a flaw of the Q-learning algorithm that usually leads to overestimated action values and frequently to suboptimal policies. We presented a novel algorithm that combines the Q-learning algorithm with Monte Carlo methods to obtain action value estimates with reduced bias. Our method utilizes information already gathered through past trajectories to construct estimates of the bias and uses these estimates in the Q-learning update rule to compensate for the maximization bias. Our method consistently outperforms the current state of the art approaches in a number of stochastic environments with discrete state spaces without requiring extensive hyperparameter tuning. It should be noted that most of the methods presented in this work assume fully observable states. We leave possible extensions to POMDPs for future work.

References

1. D'Eramo, C., Restelli, M., Nuara, A.: Estimating maximum expected value through gaussian approximation. In: International Conference on Machine Learning, pp. 1032–1040 (2016)
2. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. **13**, 227–303 (2000)
3. S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In International Conference on Machine Learning, pages 1587–1596. PMLR, 2018
4. Gumbel, E.J.: Statistics of Extremes. Columbia University Press, New York (1958)
5. Hanna, J.P., Niekum, S., Stone, P.: Importance sampling in reinforcement learning with an estimated behavior policy. Mach. Learn. **110**(6), 1267–1317 (2021). https://doi.org/10.1007/s10994-020-05938-9
6. Hasselt, H.V.: Double Q-learning. In: Advances in Neural Information Processing Systems, pp. 2613–2621 (2010)
7. Lan, Q., Pan, Y., Fyshe, A., White, M.: Maxmin Q-learning: controlling the estimation bias of Q-learning. arXiv preprint arXiv:2002.06487 (2020)
8. Lee, D., Defourny, B., Powell, W.B.: Bias-corrected q-learning to control max-operator bias in q-learning. In: 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), pp. 93–99. IEEE (2013)
9. Smith, J.E., Winkler, R.L.: The optimizer's curse: skepticism and postdecision surprise in decision analysis. Manage. Sci. **52**(3), 311–322 (2006)
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
11. Thrun, S., Schwartz, A.: Issues in using function approximation for reinforcement learning. In: Proceedings of the Fourth Connectionist Models Summer School, pp. 255–263. Hillsdale, NJ (1993)
12. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
13. Zhang, Z., Pan, Z., Kochenderfer, M.J.: Weighted double q-learning. In IJCAI, pp. 3455–3461 (2017)