# RandMin Q-learning: Controlling Overestimation Bias

**Yunshu Ouyang**
ouyangy@student.ethz.ch

## Abstract

Q-learning is known to suffer from overestimation bias as it approximates the maximum action value using the maximum estimated action value. Several algorithms have been proposed to control overestimation bias but they either introduce underestimation bias or make use of multiple action value estimates which is computationally not efficient. In this paper, we propose one generalization of Q-learning called *RandMin Q-learning* which maintains two estimates of the action value function and provides a parameter to control the estimation bias. We show the convergence of our algorithm in the tabular case and generalize its idea to the function approximation setting: *RandMin DQN*. We empirically verify that our algorithm achieves superior performance on several highly stochastic problems in the tabular case but also in environments with a continuous state space.

## 1 Introduction

Q-learning algorithm is one of the most popular reinforcement learning algorithm proposed by Watkins [1] which can be used to optimally solve Markov Decision Processes. The update rule is based on the Bellman optimality equation [2]. At every step the agent uses the observed reward and the estimated value of the maximal action in the next state to make the update of the current action-value estimate. The simplicity of this update rule led Q-learning to become one of the most widespread algorithms in reinforcement learning.

However, this simple update rule also has its flaws. It has been shown to suffer from the problem of overestimation bias (Thrun and Schwartz [3], Hasselt [4]). The reason is due to the agent using the maximum over action values to make the updates. The overestimation bias can significantly impede the quality of the learned policy or even lead to failures of Q-learning [3]. The problem also persists in the function approximation setting [4].

In this work, we first review existing approaches for reducing estimation bias and compare the pros and cons of the existing methods. We then use this to motivate a new variant of Q-learning called RandMin Q-learning. The algorithm maintains two action-value estimates where only the second estimate is used to compute the target action-value. This second estimate is updated using a subset of all available experiences which is chosen based on the first action-value estimate and one input parameter. This parameter is able to control the estimation bias varying from positive to negative.

Our main contributions are summarized as follows:

1. We propose a simple algorithm called RandMin Q-learning which is a variant of Q-learning. We show our algorithm converges in the tabular setting and empirically verify that our algorithm is able to control the bias and outperform other Q-learning variants in highly stochastic environments.
2. We extend our algorithm to the function approximation setting: RandMin DQN. Experiments show our algorithm performs as least as good as DQN and its variants in benchmark problems.

## 2 Background and Related Work

**Problem description.** We formalize the problem as a Markov Decision Process (MDP), $(\mathcal{S}, \mathcal{A}, \mathrm{P}, r, \mu, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathrm{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition model, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the expected reward function, $\mu \in \Delta(\mathcal{S})$ the initial state distribution and $\gamma \in [0, 1]$ is the discount factor.

At each time step $t$, the agent observes a state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$ and then receives a reward $r(s_t, a_t)$ and transitions to a new state $s_{t+1} \in \mathcal{S}$ with probability $\mathrm{P}(s_t, a_t, s_{t+1})$.

The goal of the agent is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximize the expected return starting from some initial state:

$$\mathbb{E}_{s\sim\mu}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, \pi\right]$$

The state-action value function $\mathcal{Q}^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ under policy $\pi$ is defined as

$$\mathcal{Q}^\pi(s, a) := \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a\right]$$

The optimal state-value function $\mathcal{Q}^*$ satisfies the Bellman optimality equation ([2]):

$$\mathcal{Q}^*(s, a) = r(s, a) + \gamma \sum_{s'\in\mathcal{S}} \mathrm{P}(s'|s, a) \max_{a'\in\mathcal{A}} Q^*(s', a') \tag{1}$$

Using $\mathcal{Q}^*$ we can define the optimal policy $\pi^*(s) \in \arg\max_{a\in\mathcal{A}} \mathcal{Q}^*(s, a)$.

**Q-learning.** Q-learning is an off-policy algorithm which attempts to learn the state-action values $\mathcal{Q}$ for the optimal policy. Its update rule directly follows from Equation (1):

$$\mathcal{Q}(s_t, a_t) \leftarrow \mathcal{Q}(s_t, a_t) + \alpha\Big(r_t + \gamma \max_{a'\in\mathcal{A}} \mathcal{Q}(s_{t+1}, a') - \mathcal{Q}(s_t, a_t)\Big)$$

where $\alpha$ is the learning rate. This algorithm is known to converge in the tabular setting (Tsitsiklis [5]), with some limited results for the function approximation setting (Melo and Ribeiro [6]). Q-learning can be interpreted as using the single estimator method to estimate the maximum expected value of $\mathcal{Q}(s', a')$, $\max_{a'} \mathbb{E}[\mathcal{Q}(s', a')]$. Since $\max_{a'} \mathcal{Q}(s', a')$ is an unbiased sample drawn from a distribution with mean $\mathbb{E}[\max_{a'} \mathcal{Q}(s', a')]$ and $\mathbb{E}[\max_{a'} \mathcal{Q}(s', a')] \geq \max_{a'} \mathbb{E}[\mathcal{Q}(s', a')]$, the estimator $\max_{a'} \mathcal{Q}(s', a')$ has a positive bias which leads Q-learning to suffer from overestimating its action values.

**Variants of Q-learning.** Various techniques have been proposed to solve this overestimation bias problem. We describe some of them below and briefly discuss the pros and cons of each algorithm.

Double Q-learning [4] makes use of two estimators $\mathcal{Q}^A$ and $\mathcal{Q}^B$ instead of a single one. Each estimator uses the value from the other estimator when computing the target for the update. Since $\mathcal{Q}^B$ was updated on the same problem, but with a different set of experience samples, $\mathcal{Q}^B(s_{t+1}, a')$ can be considered an unbiased estimate for $\mathbb{E}[\mathcal{Q}(s_{t+1}, a')]$. However, since $\mathbb{E}[\mathcal{Q}^B(s_{t+1}, a')] \leq \max_a \mathbb{E}[\mathcal{Q}^B(s_{t+1}, a)] = \max_a \mathbb{E}[\mathcal{Q}(s_{t+1}, a)]$, both estimators $\mathcal{Q}^B(s_{t+1}, a')$ and $\mathcal{Q}^A(s_{t+1}, a')$ have a negative bias which leads double Q-learning to suffer from underestimating its action values in some stochastic environments.

Weighted double Q-learning [7] combines a single estimator and the double estimator in order to balance between overestimation and underestimation. This estimate represents a trade-off between the overestimation of Q-learning and the underestimation of double Q-learning. It can be shown that there always exists a weight parameter such that the estimator becomes an unbiased estimator of $\max_a \mathbb{E}[Q(s, a)]$. However, there is no standard method for determining the correct weight in practice. Furthermore, this algorithm cannot be easily generalized to the function approximation setting unlike the other variants of Q-learning.

Ensemble Q-learning [8] and Averaged Q-learning [9] take averages over multiple action values in order to reduce both the overestimation bias and the estimation variance. However, the average operation will never completely remove the overestimation bias as the average of several overestimation biases is always positive. Therefore, both algorithms still suffer from the overestimation bias issue.

Clipped Double Q-learning [10] takes the maximum over the minimum of two action-value estimates. This idea is further generalized by Lan et al. [11] with Maxmin Q-learning which mitigates the overestimation bias using a minimization over multiple action-value estimates. Maxmin Q-learning maintains $N$ estimates of the action values, $\mathcal{Q}^i$ and use the minimum of these estimates in the Q-learning target: $\max_{a'} \min_{i \in \{1,\ldots,N\}} \mathcal{Q}^i(s', a')$. Lan et al. [11] showed that with an appropriate number $N$ of action-value functions, the algorithm produces an unbiased estimate of the target state action value. However, the number $N$ of required action-value functions for an unbiased estimate increases with the size of the action space. Furthermore, Maxmin Q-learning requires to store $N$ different action value functions which may be impractical for large state and action spaces and the algorithm converges more slowly than all previous variants since only one among the $N$ Q-functions gets updated at each step.

## 3 RandMin Q-learning

In this section, we develop a new algorithm, RandMin Q-learning, which is a variant of Q-learning that aims to produce an unbiased estimate as well as reduce the estimation variance of action values. Furthermore, we present their generalization to the function approximation setting called RandMin DQN.

**Intuition.** One drawback of Maxmin Q-learning is the fact that the number of Q-function estimates required in order to obtain an unbiased estimate might be large. Having multiple estimates is impractical and more time and space consuming. Therefore, we seek to improve on Maxmin Q-learning by designing an algorithm that is able to control the estimation bias while maintaining only two estimates of the action values instead of multiple ones.

The idea is to maintain in addition to the action values $\mathcal{Q}$, another estimate of the action values $\mathcal{Q}'$. The latter is used for calculating the Q-learning target and should ideally approximate the minimum over all $N$ estimates in the Maxmin Q-learning update rule. To do so, our algorithm updates its estimate on a subset of all experiences that is chosen based on the temporal difference and a probability $p$. $p$ is an input parameter that is used to control the estimation bias.

**Algorithm 1:** RandMin Q-learning

```
Input: step-size $\alpha$, exploration parameter $\epsilon > 0$, probability $p$
Initialize the value functions $\mathcal{Q}$, $\mathcal{Q}'$ randomly
Observe initial state $s$
while Agent is interacting with the Environment do
    Choose action $a$ by $\epsilon$-greedy based on $\mathcal{Q}'$
    Take action $a$, observe $r$, $s'$
    Get temporal difference $\delta \leftarrow r + \gamma \max_{a' \in \mathcal{A}} \mathcal{Q}'(s', a') - \mathcal{Q}(s, a)$
    Update action-value $\mathcal{Q}$: $\mathcal{Q}(s, a) \leftarrow \mathcal{Q}(s, a) + \alpha\delta$
    Sample $x \in_{u.a.r.} [0, 1)$
    if $\delta < 0$ or $x < p$ then
        Update action-value $\mathcal{Q}'$: $\mathcal{Q}'(s, a) \leftarrow \mathcal{Q}'(s, a) + \alpha\delta$
    end
    $s \leftarrow s'$
end
```

**Description of the algorithm.** RandMin Q-learning makes use of two estimates of the action-value function, $\mathcal{Q}$ and $\mathcal{Q}'$, and an additional parameter $p \in [0, 1]$ to control the bias. The first estimate $\mathcal{Q}$ updates its value by using the second estimate $\mathcal{Q}'$ to compute the target action-value.

The second estimate $\mathcal{Q}'$ uses the usual Q-learning update rule but makes the additional decision whether or not to make the update. To be precise, $\mathcal{Q}'$ is updated in the following two cases:

1. if the temporal difference $\delta$ is negative, then the second estimate is updated (with probability 1).
2. otherwise, the value is updated with some probability $p$.

The full algorithm is summarized in Algorithm 1.

Notice that for $p = 0$, $\mathcal{Q}'$ only gets updated when the temporal difference is negative and hence, its value keeps decreasing. Therefore, in this case, RandMin clearly produces an underestimation bias. For $p = 1$, $\mathcal{Q}'$ equals $\mathcal{Q}$ and therefore, the algorithm is the same as Q-learning, which overestimates the target value function.

**Convergence of the algorithm.** Lan et al. [11] presents a generalized Q-learning framework for proving the convergence of variants of the Q-learning algorithm.

Generalized Q-learning considers $N$ different action-value estimates $\mathcal{Q}^1, \dots, \mathcal{Q}^N$ and which all use the following update rule at time step $t$:

$$\mathcal{Q}^i_{sa}(t) \leftarrow \mathcal{Q}^i_{sa}(t-1) + \alpha^i_{sa}(t-1)(Y^{GQ} - \mathcal{Q}^i_{sa}(t-1))$$

where $Y^{GQ}$ is the target action-value of Generalized Q-learning and is defined based on all $N$ action-value estimates within the last $K$ steps:

$$Y^{GQ} = r + \gamma \mathcal{Q}^{GQ}_{s'}(t-1)$$

where $\mathcal{Q}^{GQ}_{s'}(t)$ is a function of $Q^i_s(t-j)\ \forall i \in \{1, \dots, N\}\ \forall j \in \{1, \dots, K\}$:

$$\mathcal{Q}^{GQ}_s(t) = G \begin{pmatrix} \mathcal{Q}^1_s(t-K) & \dots & \mathcal{Q}^1_s(t-1) \\ \mathcal{Q}^2_s(t-K) & \dots & \mathcal{Q}^2_s(t-1) \\ \vdots & \ddots & \vdots \\ \mathcal{Q}^N_s(t-K) & \dots & \mathcal{Q}^N_s(t-1) \end{pmatrix}$$

For simplicity, the vector $(\mathcal{Q}^{GQ}_{sa}(t))_{a \in \mathcal{A}}$ is denoted as $\mathcal{Q}^{GQ}_s(t)$, same for $\mathcal{Q}^i_s(t)$.

Lan et al. [11] showed Generalized Q-learning is guaranteed to converge to the optimal action-value function under the following conditions:

1. Conditions on $G$. Let $G : \mathbb{R}^{nNK} \to \mathbb{R}$ and $\mathcal{Q} = (\mathcal{Q}^{ij}_a) \in \mathbb{R}^{nNK}, a \in \mathcal{A}$ and $|\mathcal{A}| = n$, $i \in \{1, \dots, N\}, j \in \{1, \dots, K\}$.
   (a) If $\mathcal{Q}^{ij}_a = \mathcal{Q}^{kl}_a,\ \forall i, k, j, l, a$ then, $G(\mathcal{Q}) = \max_a \mathcal{Q}^{ij}_a$
   (b) $\forall \mathcal{Q}, \mathcal{Q}' \in \mathbb{R}^{nNK},\ |G(\mathcal{Q}) - G(\mathcal{Q}')| \le \max_{a,i,j} |\mathcal{Q}^{ij}_a - \mathcal{Q}'^{ij}_a|$
2. Conditions on $\alpha^i_{sa}(t)$. There exists some constant $C$ such that for every $(s, a) \in \mathcal{S} \times \mathcal{A}, i \in \{1, \dots, N\}, 0 \le \alpha^i_{sa}(t) \le 1$ and

$$\sum_{t=0}^{\infty} (\alpha^i_{sa}(t))^2 \le C, \qquad \sum_{t=0}^{\infty} \alpha^i_{sa}(t) = \infty \tag{2}$$

**Lemma 1.** *RandMin Q-learning converges to the optimal action-value function.*

*Proof.* RandMin Q-learning can be rewritten to fit the generalized Q-learning framework as follows. Namely, for RandMin Q-learning $N = 2$ and $K = 1$. Furthermore, the function $G$ is defined as

$$\mathcal{Q}^{GQ}_s(t) = G \begin{pmatrix} \mathcal{Q}^1_s(t-1) \\ \mathcal{Q}^2_s(t-1) \end{pmatrix} = \max_{a \in \mathcal{A}} \mathcal{Q}^2_s(t-1)$$

Both conditions on $G$ are then satisfied trivially.

$$G(\mathcal{Q}) = \max_a \mathcal{Q}^{21}_a = \max_a \mathcal{Q}^{11}_a = \max_a \mathcal{Q}^{i1}_a\ \forall i$$

$$|G(\mathcal{Q}) - G(\mathcal{Q}')| = |\max_a \mathcal{Q}^{21}_a - \max_a \mathcal{Q}'^{21}_a| \le \max_a |\mathcal{Q}^{21}_a - \mathcal{Q}'^{21}_a| \le \max_{a,i,j} |\mathcal{Q}^{ij}_a - \mathcal{Q}'^{ij}_a|$$

Notice that the condition on the step size is the usual condition that ensures every state-action pair gets visited infinitely often. The step sizes for the first estimate $\mathcal{Q}$ can be chosen as to satisfy condition 2. As for the second estimate $\mathcal{Q}'$, notice that it does not get updated at every step meaning that some of the $\alpha^i_{sa}(t)$ are set to 0. To compensate for this, RandMin Q-learning could use the following learning rates

$$\alpha'^i_{sa}(t) := \begin{cases} \alpha^i_{sa}(k) & \text{if the } k\text{-th update of } \mathcal{Q}' \text{ happens at step } t \\ 0 & \text{otherwise} \end{cases}$$

Then, the first inequality of Equation (2) is trivially satisfied. For the second inequality, notice that for any $M \in \mathbb{N}$

$$\mathbb{P}\Big[\sum_{t=1}^{2Mp^{-1}} \alpha_i'(t) \leq \sum_{t=1}^{M} \alpha_i(t)\Big] \leq \mathbb{P}\Big[X \sim \text{Bin}(2Mp^{-1}, p) \leq M\Big]$$
$$\leq \exp\Big(-\frac{1}{2}\frac{1}{2^2}2M\Big)$$

by Chernoff's inequality. As $M \to \infty$, this probability tends to 0 and hence, the second inequality is also satisfied.

Since both assumptions are satisfied, we can use the Theorem from Lan et al. [11] to conclude RandMin Q-learning also converges to the optimal policy. □

**Algorithm 2:** RandMin DQN

```
Input: step-size $\alpha$, exploration parameter $\epsilon > 0$, update parameter $M$
Initialize the action-value functions $\mathcal{Q}$, $\mathcal{Q}'$ with random weights $\theta$, resp. $\theta'$
Initialize empty replay buffer $D$ Observe initial state $s$
while Agent is interacting with the Environment do
    Choose action $a$ by $\epsilon$-greedy based on $\mathcal{Q}'$
    Take action $a$, observe $r, s'$
    Store transition $(s, a, r, s')$ in $D$

    Sample random mini-batch $\mathcal{B}$ of transition $(s, a, r, s')$ from $D$
    Get update target: $y_{s,a} \leftarrow r + \gamma \max_{a' \in \mathcal{A}} Q'(s', a')$
    Get temporal difference $\delta_{s,a} \leftarrow Y_{s,a} - Q(s, a)$
    Update action-value $\mathcal{Q}$: $\theta = \arg\min_\theta \sum_{(s,a,r,s') \in \mathcal{B}} \delta_{s,a}^2$

    Sample random mini-batch $\mathcal{B}^1$ from $\mathcal{B}$ of size $M$
    Get set of interesting transitions: $\mathcal{B}^2 := \{(s, a, r, s') \in \mathcal{B} | \delta_{s,a} < 0\}$
    Get update temporal difference: $\delta'_{s,a} \leftarrow Y_{s,a} - Q'(s, a)$
    Update action-value $\mathcal{Q}'$: $\theta' = \arg\min_\theta \sum_{(s,a,r,s') \in \mathcal{B}^1 \cup \mathcal{B}^2} {\delta'_{s,a}}^2$
end
```

**Extension to the function approximation setting.** We provide an extension of the RandMin Q-learning algorithm to the function approximation setting: RandMin DQN. The algorithm takes an integer $M$ as input which corresponds to the probability $p$ in RandMin Q-learning and also maintains two Q-networks, one for each estimator of the action values. The second estimator $\mathcal{Q}'$ is used for calculating the Q-learning target and is updated using a subset of the random mini-batch of transitions used to update $\mathcal{Q}$.

Precisely, at each step a random mini-batch of transitions is sampled from the replay buffer. $\mathcal{Q}$ is updated on this mini-batch using the update target computed using $\mathcal{Q}'$. Then, $\mathcal{Q}'$ is updated on a subset of the same mini-batch where this subset is chosen as follows: we first sample a subset of the mini-batch of size $M$ uniformly at random, we then select all transitions whose target value is higher than the current action-value estimate of $\mathcal{Q}$. The mini-batch used to update $\mathcal{Q}'$ is then the union of the two subsets. The algorithm is summarized in Algorithm 2.

The algorithm can also easily incorporate the standard tricks on DQN such as using target networks or using a prioritized experience replay buffer to achieve better performance.

## 4 Experiments

In this section, we first present empirical comparisons of Q-learning and its variants in terms of the action-value estimate and policy quality on the game of roulette and four MDP problems modified from a $3 \times 3$ grid world problem [4]. In the second part, we analyze the performance of `RandMin DQN` in several benchmark environments.

**Tabular setting.** For the tabular setting, we considered two environments: Roulette and Grid World. Roulette is a bandit problem with 171 arms, consisting of 170 betting actions with an expected loss of $\$-0.053$ per play and one walk-away action yielding $\$0$. Grid World [4] consists of a $3 \times 3$ grid where the agent tries to learn to reach the top right cell starting form the bottom left cell. In order to check for robustness of the algorithms in highly stochastic environments, we modified the rewards of each state-action pair for both environments. For Grid World, the agent receives a random reward drawn from $\mathcal{N}(0, \sigma_0)$ for any action ending in the goal state and a random reward drawn from $\mathcal{N}(-1, \sigma_1)$ for any action at a non-goal state. We conducted experiments for various values of $\sigma_0, \sigma_1 \in \{1, 10\}$.

The experimental setup is as follows. We trained each algorithm with 150 episodes for Roulette and 500 episodes for Grid World. The number of steps to find the optimal action and the reward were used as the performance measure. All experimental results were averaged over 100 runs. The key algorithm settings included the step-sizes, exploration parameter. All algorithms used $\epsilon$-greedy with $\epsilon = 0.1$. For each algorithm, the best step-size was chosen from $\{0.005, 0.01, 0.02, 0.04, 0.08\}$. For weighted double Q-learning, the parameter $c$ was chosen from $\{1, 10, 100\}$. For Maxmin Q-learning, the parameter $N$ was chosen from $\{2, 4, 6, 8\}$. For RandMin Q-learning, the parameter $p$ was chosen from $\{0.1, 0.2, \ldots, 0.9\}$. We also provide an open-source implementation of our method available at `https://github.com/yooyoo9/randmin`.

Figure 1 shows the number of steps each algorithm uses before reaching the end state as a function of the number of episodes. The optimal policy is to go directly towards the top right cell and this can be done within 4 steps. As shown in Figure 1, RandMin uses the minimal number of steps compared to all the other algorithms meaning that they succeed in learning the optimal policy. Q seems to perform best when the variance of the non-goal states is small but achieves less good results when the latter becomes large.

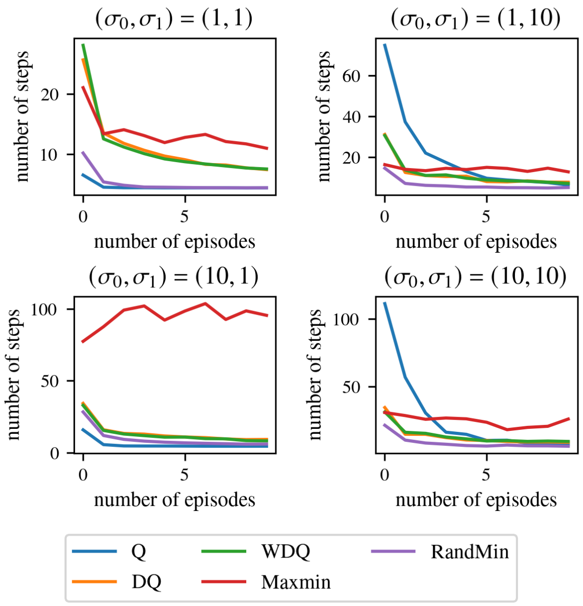

Figure 1: Experimental results for Gridworld with different variance parameters. Only WDQ and RandMin are able to converge to the optimal policy among all five algorithms. Maxmin actually performs worse than Q and DQ for certain values of the variance.

For Roulette, we compared the action values over all betting actions as found by Q-leaning (Q), double Q-learning (DQ), weighted double Q-learning (WDQ), Maxmin Q-learning (Maxmin) and RandMin Q-learning (RandMin) with the true action-value function. The mean squared error between the two can be found in Table 1. In addition, we recorded the mean reward of the last 50 episodes for each algorithms together with their run-time. Notice that in this stochastic environment, Q and DQ fail to converge to the optimal policy. WQD, Maxmin and RandMin all perform similarly with WQD having the best approximation of the state-action values and RandMin being the faster than the other two variants.

| | Q | DQ | WQD | Maxmin | RandMin |
|---|---|---|---|---|---|
| MSE to optimal $\mathcal{Q}^*$ | 59.69 $\pm$ 0.00 | 35.22 $\pm$ 0.00 | **0.06 $\pm$ 0.00** | 0.25 $\pm$ 0.00 | 0.27 $\pm$ 0.00 |
| Mean reward of last 50 episodes | -15.91 $\pm$ 11.65 | -7.99 $\pm$ 8.21 | **-0.06 $\pm$ 0.57** | **-0.10 $\pm$ 0.25** | **-0.10 $\pm$ 0.40** |
| Average learning time | 21.35 $\pm$ 12.44 | 12.50 $\pm$ 7.14 | **2.81 $\pm$ 1.53** | 3.87 $\pm$ 2.41 | **1.96 $\pm$ 1.06** |

Table 1: Results for all algorithms. Q and DQ fail to converge and incur large losses in terms of reward. WDQ has the lowest MSE to the optimal Q function. All three algorithms WDQ, Maxmin and RandMin achieve optimal reward of 0. Maxmin is the slowest among the three.

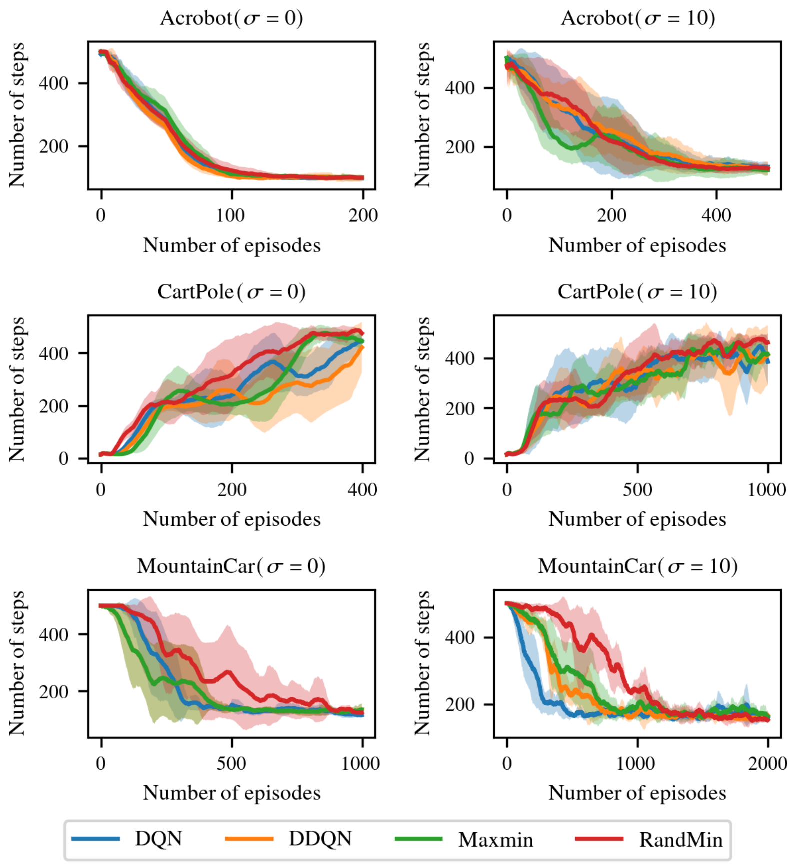

Figure 2: Experimental results for Acrobot, Cart Pole and Mountain Car with $\sigma = \{0, 10\}$. Very little difference between the performance of the four algorithms can be seen in the Acrobot environment as all four are able to converge to the optimal policy within the same number of episodes. However, RandMin clearly outperforms the other three algorithms for CartPole. The number of steps for which the agent stays in the game is the highest and its convergence is also faster in the non-stochastic setting. All four algorithms achieve similar performance in the Mountain Car environment with RandMin converging the slowest. DQN seems to be the best for this environment.

**Benchmark environments with continuous state space.** We here provide the results of RandMin DQN (RandMin) compared to DQN, DDQN and Maxmin DQN (Maxmin) for the following environments: Acrobot Sutton [12], Cart Pole Barto et al. [13] and Mountain Car Moore [14]. In our experiments, we modify the rewards to be stochastic with same mean value: the reward signal is sampled from a Gaussian distribution $\mathcal{N}(-1, \sigma^2)$ on each time step. All three environments are modified from the Gym [15] implementation.

The step-size was chosen from $[5 * 10^{-3}, 10^{-3}, 5 * 10^{-4}, 10^{-4}, 5 * 10^{-5}]$. The neural network is a multi-layer perceptron with hidden layers fixed to $[64]$. The discount factor was 0.99. The size of the replay buffer was 10,000. The weights of the neural networks were optimized by Adam. The batch size is 64. The target network was updated at the end of each episode. $\epsilon$-greedy was applied as the exploration strategy with $\epsilon$ decreasing from 1.0 to 0.01 with a decay of 0.995 at each step. The number of target networks $N$ for Maxmin DQN was chosen from $[2, 4, 6, 8]$ and the batch-size for Randmin DQN was chosen from $[8, 16, 32, 64$. Results were averaged over 5 runs for each algorithm. All the experiments were conducted using PyTorch [16]. They were run on an Intel(R) Xeon(R) CPU E5-2697 v4 2.30GHz machine and an NVIDIA GeForce GTX 1080 Ti GPU.

Table 2 shows the number of steps required for each algorithm in each environment to reach the end state in the last episode. For Acrobot and Mountain Car, the smaller the number of steps the better it is. For Cart Pole, the agent should try to stay in the game for as many steps as possible. However, we set a maximal number of steps per episode to 500 so the nearer the number of steps is to 500, the better the algorithm performs. We see that RandMin DQN outperforms Maxmin DQN in most environments and also achieves the best result for Cart Pole even in the non-stochastic setting.

In addition to Table 2, Figure 2 provides the learning curves of all algorithms in the six different environments. It is particularly noticeable that RandMin outperforms all other algorithms for the Cart Pole environment.

| Data Set | DQN | DDQN | Maxmin DQN | Randmin DQN |
|---|---|---|---|---|
| Acrobot($\sigma = 0$) | **88.34 ± 0.73** | 95.12 ± 4.55 | 91.04 ± 3.28 | 91.73 ± 2.34 |
| Acrobot($\sigma = 10$) | 117.91 ± 8.59 | 110.32 ± 9.54 | **107.71 ± 10.27** | 124.04 ± 18.38 |
| Cart Pole($\sigma = 0$) | 445.61 ± 31.50 | 421.50 ± 95.44 | 445.54 ± 31.27 | **475.22 ± 12.67** |
| Cart Pole($\sigma = 10$) | 384.99 ± 90.24 | 412.07 ± 118.79 | 415.78 ± 44.78 | **462.19 ± 35.48** |
| Mountain Car($\sigma = 0$) | **116.64 ± 8.14** | 135.94 ± 20.09 | 135.94 ± 20.09 | 125.36 ± 6.58 |
| Mountain Car($\sigma = 10$) | 158.48 ± 14.06 | 167.04 ± 12.90 | 164.26 ± 17.48 | **152.69 ± 7.90** |

Table 2: Average number of steps in last 50 episodes in all three environments with different variance parameters. RandMin DQN achieves the best performance in half of the cases. Its performance is suboptimal for Acrobot but does not differ from the best result by much. DQN is able to achieve results similar to Randmin DQN in stochastic environments for Acrobot and Mountain Car but fails to do so for Cart Pole. Maxmin DQN achieves best performance for Acrobot($\sigma = 10$) but fails to do so in all other settings.

## 5 Conclusion/Future Work

Inspired by Maxmin Q-learning, we designed a new variant of Q-learning called RandMin Q-learning which improves upon Maxmin Q-learning by maintaining two estimates of the action-value function while being able to control the estimation bias. From the experiments, we see that RandMin perform at least as good as the state-of-the-art variants of Q-learning in non-stochastic environments. While other algorithms such as Q-learning and Double Q-learning tend to fail in certain highly stochastic environments, our algorithm is still able to converge to the optimal action value function. Furthermore, it achieves similar performance to Maxmin Q-learning while using much less memory and computational power. Our generalization of RandMin Q-learning to the approximation setting also empirically showed itself to perform as least as good as the other variants.

One possible direction for future work would be to dive deeper into the theoretical guarantees of RandMin Q-learning and the tuning of the parameter $p$. Indeed, Lan et al. [11] showed that Maxmin Q-learning is able to produce an unbiased estimate of the target function for a given number $N$ of estimates used. One question would be whether this statement also holds for RandMin Q-learning and how to find the parameter $p$ that produces such an unbiased estimate.

## References

[1] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[2] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[3] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, pages 255–263. Hillsdale, NJ, 1993.

[4] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23: 2613–2621, 2010.

[5] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

[6] Francisco S Melo and M Isabel Ribeiro. Q-learning with linear function approximation. In *International Conference on Computational Learning Theory*, pages 308–322. Springer, 2007.

[7] Zongzhang Zhang, Zhiyuan Pan, and Mykel J Kochenderfer. Weighted double q-learning. In *IJCAI*, pages 3455–3461, 2017.

[8] Oren Peer, Chen Tessler, Nadav Merlis, and Ron Meir. Ensemble bootstrapping for q-learning. *arXiv preprint arXiv:2103.00445*, 2021.

[9] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pages 176–185. PMLR, 2017.

[10] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[11] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

[12] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

[13] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[14] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

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

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.