FAKULTÄT FÜR WIRTSCHAFTSINFORMATIK
UND WIRTSCHAFTSMATHEMATIK

UNIVERSITÄT MANNHEIM

Prof. Dr. Leif Döring
Sara Klein, Benedikt Wille

Reinforcement Learning

10. Exercise Sheet

1. Convergence of Q-Learning

The assumptions and definitions of Theorem 4.5.4 (Convergence of Q-Learning) are given. Moreover, let

$$F(Q)(s,a) \coloneqq \mathbb{E}_s^{\pi^a}\big[R(s,a) + \gamma \max_{a' \in \mathcal{A}_{S_1}} Q(S_1, a')\big]$$

and

$$\varepsilon_n(s,a) \coloneqq R(s,a) + \gamma \max_{a' \in \mathcal{A}_{s'}} Q_n(s', a') - F(Q_n)(s,a)$$

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ and $n \in \mathbb{N}$. Show that the sequence

$$Q_{n+1}(s,a) \coloneqq Q_n(s,a) + \alpha_n(s,a)\big(F(Q_n)(s,a) - Q_n(s,a) + \varepsilon_n(s,a)\big), n \in \mathbb{N}$$

almost surely converges to Q^*.

2. SARSA

Rewrite a k-armed Bandit as an MDP in such a way that SARSA (Algorithm 20 with ϵ_n-greedy policy updates and $\alpha(s,a) = \frac{1}{N(s,a)+1}$) corresponds to the ϵ_n-greedy algorithm introduced in Chapter 1. Check that both algorithms are equivalent.

3. n-step TD

a) Write pseudocode for n-step TD algorithms for evaluation of V^π and Q^π and prove the convergence by checking the n-step Bellman expectation equations

$$T^\pi V(s) = \mathbb{E}_s^\pi\Big[R(s, A_0) + \sum_{t=1}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n V(S_n)\Big]$$

and

$$T^\pi Q(s,a) = \mathbb{E}_s^{\pi^a}\Big[R(s,a) + \sum_{t=1}^{n-1} \gamma^t R(S_t, A_t) + \gamma^n Q(S_n, A_n)\Big]$$

and the conditions of Theorem 4.3.8 on the error term. Note that the algorithm only starts to update after the MDP ran for n steps.

b) Write pseudocode for an n-step SARSA control algorithm.

4. Programming Task: Q-Learning vs. SARSA on cliffwalk

We want to use cliff walk, a special type of grid world, to illustrate some differences in the behaviour of Q-learning and SARSA regarding avoiding states with very low rewards. In cliff walk, as in grid world, the player moves along a grid world one step at a time starting in the top left corner. However, the goal is located in the top right corner and there is a cliff spanning from the second to second-last grid of the top row, from which the player falls upon entering the cliff square and after which the player is transferred directly to the starting square. In addition, there is a windy version, where a small probability of walking up is added in each step, regardless of the chosen action. For the implementation you can orient yourself on the code example given on the lecture's web page.

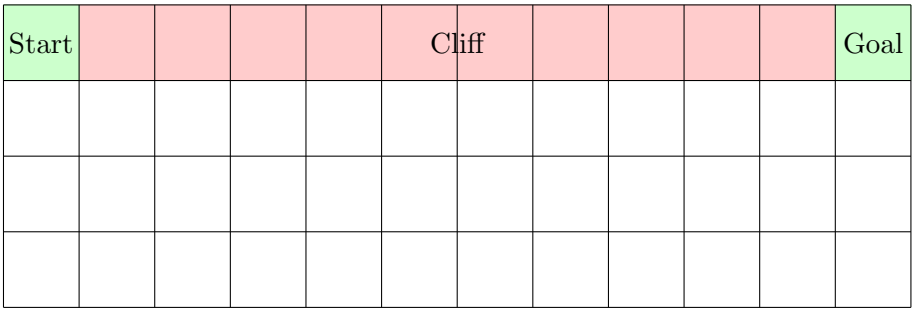

Abbildung 1: Visualization of the cliff walk game.

a) Implement cliff walk and windy cliff walk.

b) Implement the Algorithms 19 and 20, Q-Learning and SARSA, with ϵ-greedy policy updates each.

c) Compare the strategies that each algorithm learns for different amounts of iteration steps in both versions of cliff walk. Set the reward for the goal as 100, for the cliff states as -100 and in all other cases as -1. Can you think of a reason why different policies may turn up in the optimization steps based on the algorithm update rule? Can you deduce a hypothesis on the general behaviour and comparability of the algorithms in the face of states with very low rewards?

5. Programming Task*: Maximization Bias Example

In this task we want to implement a Markov Decision process, which will show properties and difficulties of Q-Learning. The modeling of the constructed example comes from Example 6.7. The state space $\mathcal{S}$ is given by $\mathcal{S} = \{C, D, T\}$, where T is a terminal state. Furthermore, the possible actions in state C are given by $\mathcal{A}_C = \{\text{left}, \text{right}\}$ and $\mathcal{A}_D = \{0, 1, \ldots, n\}$, where $n \in \mathbb{N}$. The transition probabilities are given by $p(T; C, \text{right}) = 1, p(D; C, \text{right}) = 1$ and $p(D; T, a) = 1$ for all $a \in \mathcal{A}_D$. Moreover, the distribution of rewards is given by $R(S, A) \sim \mathcal{N}(-0.1, 1)$ if $S(\omega) = C, A(\omega) \in A_D, \omega \in \Omega$ and $R(S, A) \equiv 0$ otherwise.

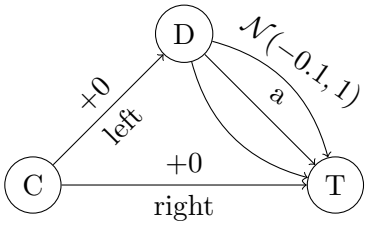

Abbildung 2: Visualization of the game from task 4. Action a is any action which can be played in state D. So $a \in A_D$.

6. Programming Task*: One Step Approximate Dynamic Programming

In this task we want to implement the algorithms from chapter 4.5

(a) Implement the one step approximate dynamic programming Algorithms 18-23 (policy evaluation for Q^π, Q-learning, SARSA, (Truncated/Clipped) Double Q-learning. Furthermore, implement a modification of Algorithm 18, the policy evaluation for Q^π replacing the one-step update with a Monte Carlo estimator update.

(b) Apply the algorithms to one of the examples so far from the lecture and determine the optimal ϵ-greedy policy. Comparte the algorithms, applying similar metrics as in the lecture to the example 4.5.8.