Machine Learning - Exercise 9

Reinforcement Learning: The recycling robot

Goal of the exercise

- The goal of this exercise is to find the optimal policy for the recycling robot seen in the lecture.
- In this problem, a recycling robot has to search for empty cans to collect (each can defines a "reward" given to the robot). It can also decide to stay where it is to save its battery and wait that somebody brings it a can (which gives less cans in average than actively searching for them).
- The robot has two battery levels, *high* and *low*.
- In the *high* level, the robot can either search or wait. Searching can bring the battery level to the *low* state with a probability $1 - \alpha$, while waiting always leaves the robot in the *high* state. Searching brings an expected reward of $\mathcal{R}^\text{search}$. Waiting leaves in the *high* state and brings an expected reward of $\mathcal{R}^\text{wait}$.
- In the *low* state, three actions are possible. Searching brings an expected reward of $\mathcal{R}^\text{search}$ and leads to the same state *low* with a probability of β. However, it can also completely empty the battery (probability 1-β), which leads to the intervention of a human operator and a negative reward of -3. Waiting leaves in the *low* state and brings an expected reward of $\mathcal{R}^\text{wait}$. Recharging the battery leads to the *high* state but brings no reward.

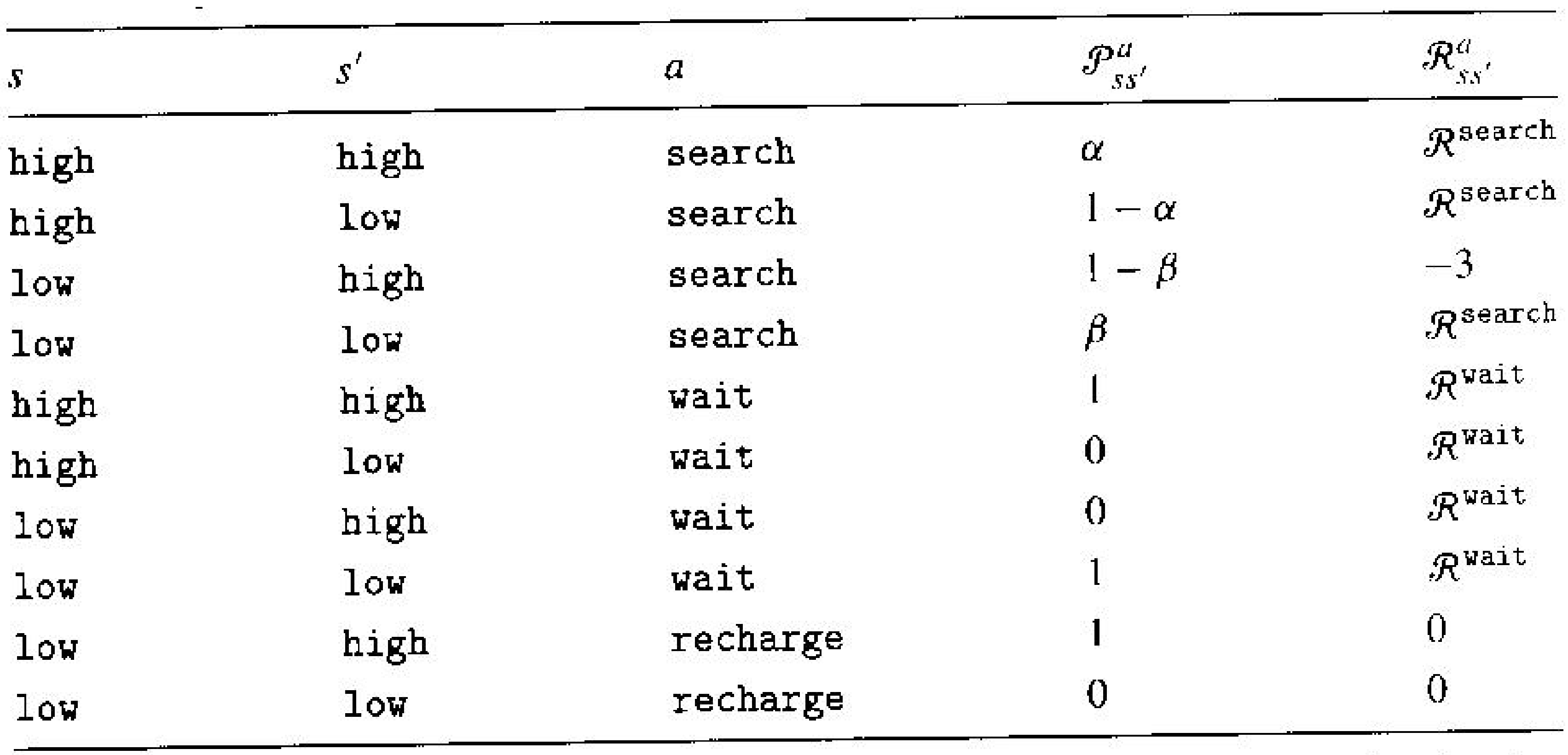
MDP for the recycling robot

This problem defines a finite MDP, with two states *high* and *low* corresponding to the battery level. The actions *search* and *wait* are possible in the *high* and *low* states, while the action *recharge* is only possible in the *low* state.

$$\begin{aligned} \mathcal{S} &= \{\text{high}, \text{low}\} \\ \mathcal{A}(\text{high}) &= \{\text{search}, \text{wait}\} \\ \mathcal{A}(\text{low}) &= \{\text{search}, \text{wait}, \text{recharge}\} \end{aligned}$$

The transition probabilities $\mathcal{P}_{ss'}^{a}$ and expected rewards associated to these transitions $\mathcal{R}_{ss'}^{a}$ of this MDP are summarized in the table 1 and figure 1.

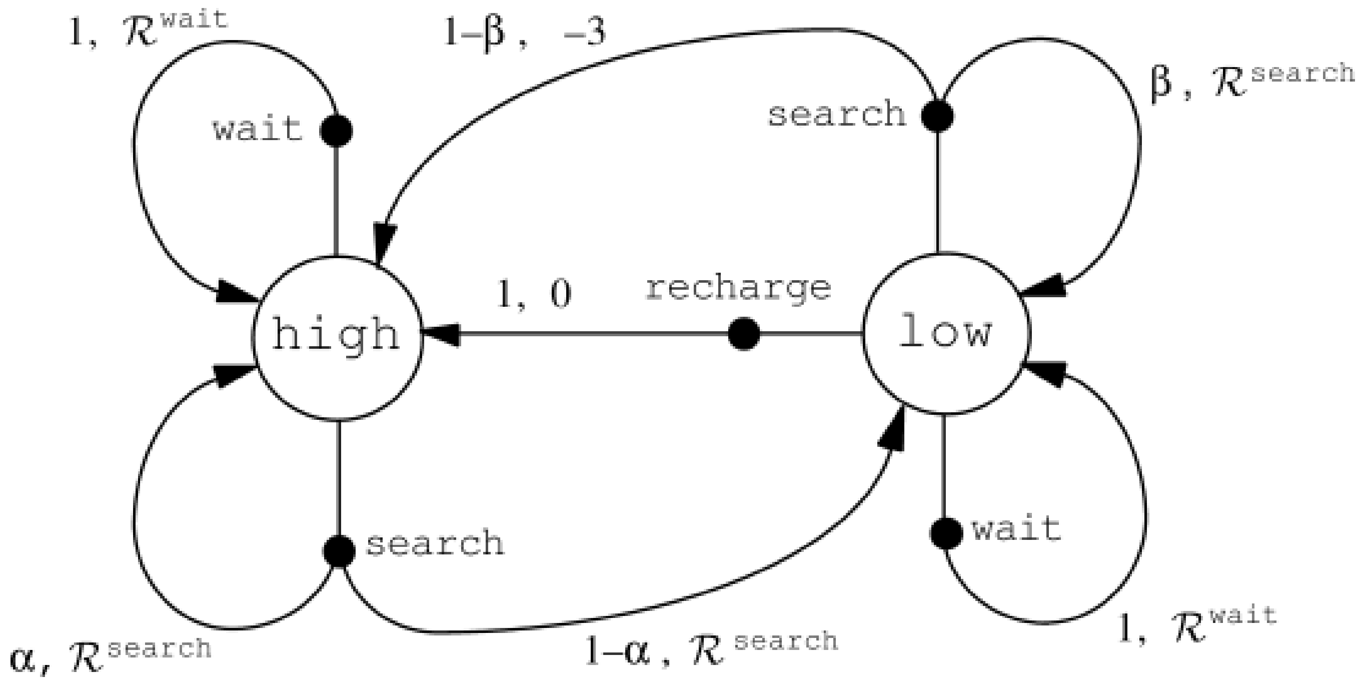
MDP as a table



s	s'	a	$\mathcal{P}^{a}_{ss'}$	$\mathcal{R}^{a}_{ss'}$
high	high	search	α	$\mathcal{R}^{\texttt{search}}$
high	low	search	$1-\alpha$	$\mathcal{R}^{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$\mathcal{R}^{\texttt{search}}$
high	high	wait	1	$\mathcal{R}^{\texttt{wait}}$
high	low	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	high	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	low	wait	1	$\mathcal{R}^{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0

Table 1: Transition probabilities and expected rewards of the recycling robot.

MDP as a graph



1, $\mathcal{R}^{\text{wait}}$
wait
1–β, –3
search
β, $\mathcal{R}^{\text{search}}$
high
1, 0
recharge
low
search
wait
α, $\mathcal{R}^{\text{search}}$
1–α, $\mathcal{R}^{\text{search}}$
1, $\mathcal{R}^{\text{wait}}$

Figure 1: Transition graph of the recycling robot.

Dynamic programming

- The goal of this exercise is to find the optimal policy π^* of the robot, i.e to find for each state the action that should be performed systematically in order to gather the maximum of reward on the long term.
- One simple solution (called *value iteration*) to this problem is to compute the optimal value of each state, which represents the cumulated return expected to be obtained after being in this state and following thereafter the optimal policy π^*.
- This optimal policy is defined by:

$$V^*(s) = E_{\pi^*}(\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} | s_t = s) \quad \forall s \in \mathcal{S}$$

Dynamic programming

- The Bellman optimality equation defines the optimal value of a state as a function of the value of all other states that can be reached from this state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^*(s, a) \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s \in \mathcal{S}} \mathcal{P}_{ss'}^a \cdot [\mathcal{R}_{ss'}^a + \gamma \cdot V^*(s')]
\end{aligned}
$$

where $Q^*(s, a)$ is the optimal value of the action a taken in the state s (also defined in terms of expected return after taking the action a in the state s and thereafter following the optimal policy π^*).

Questions

1. Adapt the Bellman optimality equation to the problem. First, for every state s and possible action a, find the optimal value of the action with the form:

$$Q^*(s, a) = f(V^*(\text{high}), V^*(\text{low}), \alpha, \beta, \gamma, \mathcal{R}^{\text{search}}, \mathcal{R}^{\text{wait}})$$

Deduce the Bellman optimality equation for the two states $V^*(\text{high}), V^*(\text{low})$.

2. Solve using Python these two equations. As the two equations are interdependent ($V^*(\text{high})$ and $V^*(\text{low})$ are both on the left- and right-sides of the equations), you will have to solve them incrementally. Take 0.0 for the initial values of $V^*(\text{high})$ and $V^*(\text{low})$, and update these values with the obtained equations until they have converged.

The convergence criterium can for example be when the difference between two successive values of $V^*(\text{high})$ (or $V^*(\text{low})$) falls below 0.001 % of the value. You will take the following values for the parameters:

$$\alpha = 0.3 \quad \beta = 0.2 \quad \gamma = 0.7 \quad \mathcal{R}^{\text{search}} = 6 \quad \mathcal{R}^{\text{wait}} = 2$$

Questions

3. Based on the found Q^* values of the actions, determine the optimal policy of the agent: the agent should systematically take the action that has the maximal Q^* value in the current state. Does this strategy make sense?

4. Change the value of discounting parameter $\gamma = 0.3$ so that the agent becomes short-sighted: it only takes into account the immediate rewards, but forgets about the long-term. Does it change the strategy? Explain why.

5. Change the parameters to:
$$\alpha = 0.01 \quad \beta = 0.2 \quad \gamma = 0.7 \quad \mathcal{R}^{\text{search}} = 6 \quad \mathcal{R}^{\text{wait}} = 5$$
Find the optimal policy. What is the optimal action to be taken in the state *high*, although the probability to stay in this state is very small? Why?

6. Try to find a set of parameters where it would be optimal to wait while in the *high* state.

7. Try to find a set of parameters where it would be optimal to search while in the *low* state.