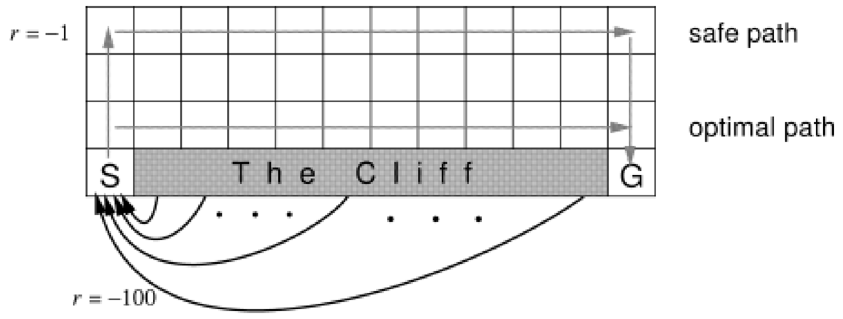
Artificial Intelligence
Exercise 4

Marc Toussaint
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

10. Dezember 2018

1 Programmieaufgabe: Sarsa vs Q-Learning (vs your Agent)

Consider the following *Cliff Walking problem.*



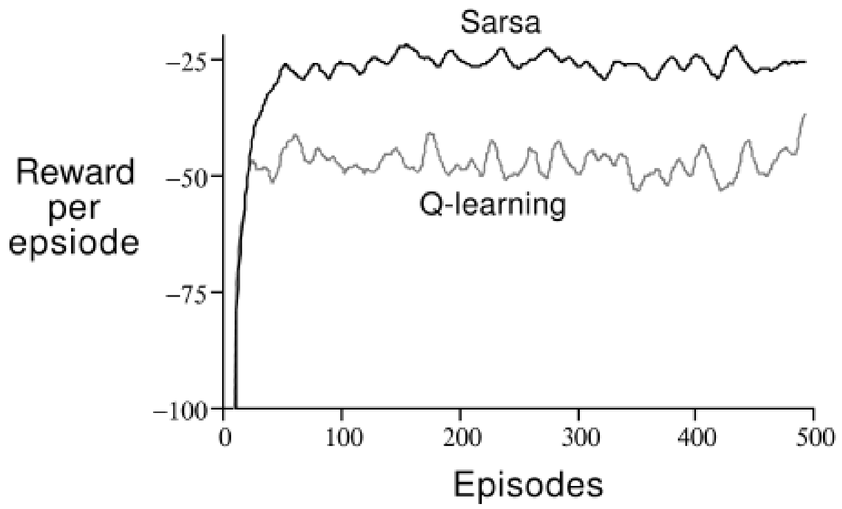


Abbildung 1: Cliffwalk environment and reference plot (smoothed)

In your git repo you find an implementation of the *Cliff Walking* environment. This is a standard undiscounted ($\gamma = 1$), episodic task, with start (S) and goal (G) states, and the actions `up, down, left, right` causing deterministic movement. The reward is -1 for all transitions except into the region marked *The Cliff.* Stepping into this region incurs a reward of -100 and sends the agent instantly back to the start. An episode ends when reaching the goal state and NOT when falling down the cliff.

Recall: See slide 06:12 for pseudo code of Q-Learning; it updates the Q-function using

$$Q(s,a) \leftarrow Q(s,a) + \alpha\ [r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)]$$

. SARSA is exactly the same algorithm, except that it updates the Q-function with

$$Q(s,a) \leftarrow Q(s,a) + \alpha\, [r + \gamma Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)]$$

To implement this SARSA update, you need to sample the next action a' before updating the Q-function.

Exercises:

(a) Implement the SARSA and Q-learning methods using the ϵ-greedy action selection strategy with a fixed $\epsilon = 0.1$. Choose a small learning rate $\alpha = 0.1$. Compare the resulting policies when greedily selecting actions based on the learned Q-tables.

To compare the agents' online performance run them for at least 500 episodes and log the *reward per episode* in a numpy array. Plot your logged data with *episodes* as x-axis and *reward per episode* as y-axis. Export the logged reward array for Q-Leaning and Sarsa to *R_ql.csv* and *R_sa.csv respectively.* See below on how to plot using python.

(b) Propose a schedule that gradually reduces ϵ, starting from $\epsilon = 1$. Then redo question 1 using your schedule for ϵ instead of the fixed value. Again plot the learning graph and export your logged rewards per episode to *R_ql_sched.csv* and *R_sa_sched.csv* respectively.

(c) In the lectures we introduced Rmax, which is a *model-based* approach. Here we consider a simplified model-free Rmax-variant working with Q-Learning: Implement standard Q-learning using greedy action selection but a modified reward function. The modified reward function assigns $r_{max} = 0$ to unknown states $(\#(s,a) < 100)$ and the original reward for known states. Plot and export your rewards per episode to *R_rmax.csv.*

Be sure to upload your code and all your csv files containing the rewards per episode. The evaluation is based on these files.

For each exercise get an understanding for the agent's online behavior and their learned policies. Be ready to explain in class!

The following example shows how to plot and export data contained in a numpy array. This is all you need to create the plots and generate the csv files for the above exercises.

```
import numpy as np
import matplotlib.pyplot as plt

Y = np.array([5, 8, 1, 4])

# Export to CSV
np.savetxt(Y, 'Y.csv')

# Plot and display
plt.plot(Y)
plt.show()
```

Note: The graph may be very spiky so it might be a good idea to smooth your plot before comparing it with the graph given above, e.g. by using the simple box filter provided in the code. However you have to export and hand in the un-smoothed version.

2 Votieraufgabe: Eligibilities in TD-learning

Consider TD-learning in the same maze as in the previous exercise 1 (Value Iteration), where the agent starts in state 4, and action outcomes are stochastic. Describe at what events plain TD-learning will update the value function, how it will update it. Assuming the agent always takes clock-wise actions, guess roughly how many steps the agent will have taken when for the first time $V(s_4)$ becomes non-zero. How would this be different for eligibility traces?