Exercise 4
Temporal Difference Learning

In last weeks exercise you implemented a simple gridworld MDP that adheres to the OpenAI Gym interface. This week we will put your implementation into practice and try to solve it using SARSA and Q-Learning.

1 SARSA and Q-Learning

You can find the code skeleton inside the `sarsa_q_learning.py` file. We provide to you a sample solution of the last exercise inside `gym_gridworld.py`. Visualization helpers can find inside `helper.py`. There you find three classes:

- `SARSAQBaseAgent`: The base class, which has the `__init__` constructor and a `action` function, which returns the ϵ-greedy action for a state s. The SARSA and Q-Learning agents are extensions of this class.
- `SARSAAgent` and `QLearningAgent`: The SARSA and Q-Learning agent classes with methods `learn` and `update_Q`.

Remark: You are free to implement the inner workings of your agents as you wish. For the visualization tools to work you will however have to work with a Q-value member variable Q, which is a numpy array of shape $[grid\_height, grid\_width, num\_actions]$. Alternatively, it should be easy to adjust the visualization helper functions as needed.

Programming Tasks:

1. **ϵ-greedy actions**: Implement the `action` function which should return a random action with a probability of ϵ and the greedy action w.r.t the current Q-value estimates of state s with a probability of $1 - \epsilon$.
2. **SARSA**:
   - **Q-value update**: Implement the Q-value update rule of SARSA for a tuple (s, a, r, s', a') inside `update_Q` d.
   - **Learning loop**: Implement the training loop of the SARSA agent, which should step through the environment for `n_timesteps` steps.
3. **Q-Learning**: Repeat the same steps for the Q-Learning agents.

Test your implementation and verify that it's working correctly. To make things easier you can start testing on a simpler gridworld environment by replacing the map of your gridworld with a simpler one. You can see a sample results in Figure 1.

2 Cliffwalking

In order to highlight the difference between SARSA and Q-Learning, try to replicate the famous *Cliff Walking* environment using your (or our) gridworld implementation (Figure 2). As our implementation only supports square grid shapes, we replicated this environment inside a 4×4 grid with only two cliff cells between start and finish in our reference implementation. This turned out to work well enough.

Train a SARSA and Q-Learning agent on this environment. What is the explanation for the difference in learnt policies and Q-functions? How does this relate to SARSA being considered an on-policy method and Q-Learning being an off-policy method?

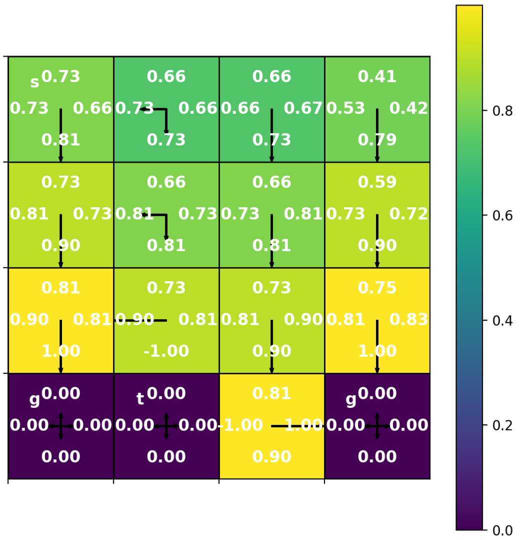

Figure 1: Sample output for a correctly implemented Q-Learning agent on the gridworld environment ($200k$ training steps, $\epsilon = 0.4$, $\gamma = 0.9$). States $(1, 2)$ and $(2, 2)$ should actually have a $50 - 50$ policy for actions down and right, which they don't have due to rounding error. Results can also be quite sensitive to hyper-parameters and a correct implementation can still lead to confusing results sometimes.

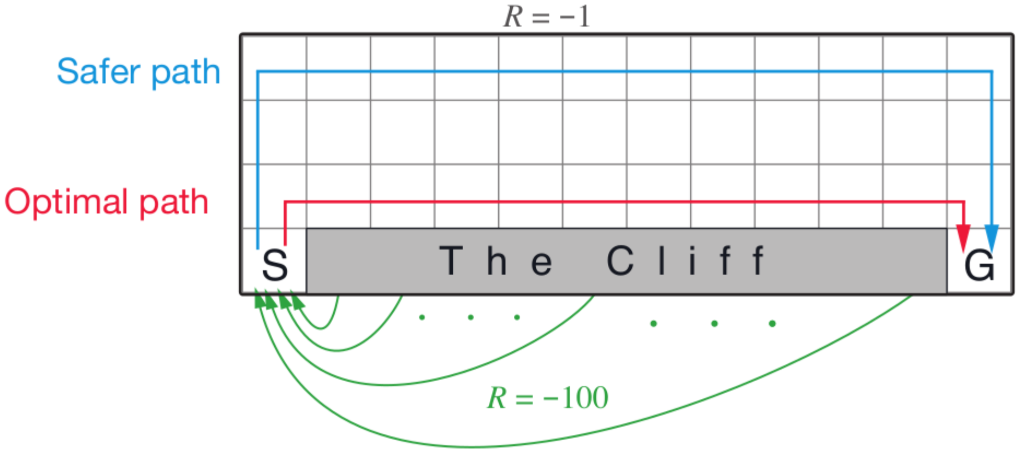

Figure 2: The Cliff Walking environment.