Reinforcement Learning WS 2024
TAs: Pavel Kolev and Pierre Schumacher
Due-date: **12.11.2024, 14:00** (upload to ILIAS as one file)
Filename: homework4-NAME1-NAME2-...zip
Homework Sheet 4
November 5, 2024

Note: The tutorial sessions on Tuesday will be in different rooms from 12.11 onwards. Please check the Ilias course website.
This time we will consider the case where the MDP of the system is unknown and we want to control it.

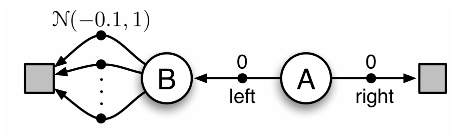
1 Q-Learning and SARSA

First some more theoretical questions:

(a) Why is Q-learning considered an off-policy control method?

(b) Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as SARSA? Will they make exactly the same action selections and weight updates?

(c) Consider the small MDP shown below. The MDP has two non-terminal states A and B. Episodes always start in A with a choice between two actions, left and right. The right action transitions immediately to the terminal state with a reward and return of zero. The left action transitions to B, also with a reward of zero, from which there are many possible actions all of which cause immediate termination with a reward drawn from a normal distribution with mean 0.1 and variance 1.0.



(a) Which action should be taken?

(b) Think about the Q estimation with Q-learning. Which behavior do you expect?

2 Hands-on the Gridworld

You will implement Q-learning and test it on a simple Gridworld domain.
The code for this exercise contains needs two more files than the previous exercise. You can get all of them from in `4_gridworld_qlearning.zip` file:

Files:

agent.py The file in which you will write your agents (with small modifications)

mdp.py Abstract class for general MDPs.

environment.py Abstract class for general reinforcement learning environments (compare to mdp.py)

gridworld.py The Gridworld code and test harness. (with small modifications)

gridworldClass.py Definition of Gridworld class.

crawler.py The crawler robot simulation code (not needed this time)

utils.py some utility code, see below.

The remaining files graphicsGridworldDisplay.py, graphicsCrawlerDisplay.py (not required), graphicsUtils.py, and textGridworldDisplay.py can be ignored entirely.
You will need to fill in portions of `agent.py`.

Gridworld: Consult the previous exercise sheets for the general usage of the gridworld simulator.

2.1 Q-Learning

You will write a Q-learning agent, which does very little on construction, but then learns by trial and error interactions with the environment through its update(state, action, nextState, reward) method. A stub of a Q-learner is specified in QLearningAgent in agent.py, and you can select it with the option '-a q'. You should first run your Q-learner through several episodes under manual control without noise (e.g. '-k 5 -n 0 -m'), for example on the MazeGrid. Watch how the agent learns about the state it was just in. Your actual agent should be an epsilon-greedy learner, meaning it chooses random actions *epsilon*-faction of the time, and follows its current best q-values otherwise.

(a) Train your Q-learner on the MazeGrid for 100 episodes.

```
python3 gridworld.py -g MazeGrid -a q -k 100 -q
```

How are the learned values different from those learned by value iteration (previous exercise) and shown in Fig. 1, and why?

How can you make them closer to the optimal values?

(b) Train your Q-learner on the BridgeGrid with no noise (-n 0.0) for 100 episodes. How do the learned q-values compare to those of the value iteration agent (see Fig. 2)?

Why will your agent usually not learn the optimal policy?

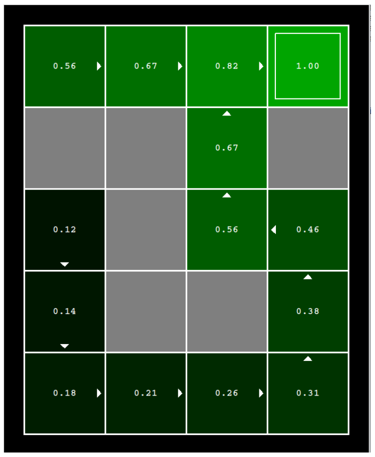


Figure 1: Value function obtained with value iteration on MazeGrid with default parameters.

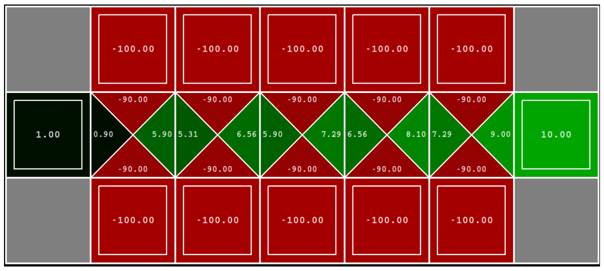


Figure 2: Q-function obtained with value iteration on BridgeGrid without noise.

(c) Train your Q-learner on the CliffGrid for 300 episodes. Compare the value it learns for the start state with the average returns from the training episodes (printed out automatically). Why are they so different?

(d) Feel free to play around with your own Grid world or one of the existing ones.