Assignment #4: Simple Robot Navigation (Q-Learning)

CSCI 364 Spring 2017 Oberlin College

Due: Wednesday April 12 at 11:00 AM

Background

In class, we looked at a grid world example where a robot agent moves around from a start state S (in the bottom left corner) and attempts to reach a goal state G (in the top right corner). The agent earns a small cost for each movement, a positive reward (+1) for reaching the goal G, and a negative reward (-1) for accidentally falling into a pit right below the goal G. Unfortunately, the robot's motors are not perfect, meaning that sometimes it ends up in a different location from where it intended to go. Thus, the next state transitions are stochastic, meaning we cannot simply solve this problem using Search algorithms (e.g., A* Search). A summary description of the grid world is given in Chapter 17 of the class textbook.

														G
														P
		X	**X**			**X**								
		X	**X**			**X**								
S														

Figure 1: The Robot's Grid World

S = Start Location G = Goal Location P = Pit Location X = Obstacle

Assignment

Your assignment is to write a program in **Python or Java** that uses Q-Learning to enable the robot to learn how to navigate from its starting location in the bottom left corner to the goal location in the top right corner.

You can get started on the assignment by following this link:
https://classroom.github.com/assignment-invitations/8f4a8e8bf610abff39bba864a503149d

I've given you a Grid class that represents the grid on the previous page. The only methods you should need to use from this class are:

1) `generateStartState()`, which returns the starting state for the robot
2) `generateNextState(state, action)`, which takes in the current `state` and a chosen `action` as parameters and returns a randomly chosen next state for the agent
3) `generateReward(state, action)`, which also takes in the current `state` and a chosen `action` as parameters and returns a reward for the agent (to use for reinforcing that action choice)

Also, you will want to use the `ABSORBING_STATE` in Grid to know when your agent has finished an episode – once this is the ***current* state**, your agent is done!

You should run your agent for 100 episodes (each starting at the start state and ending at the `ABSORBING_STATE`) and track the discounted, cumulative reward earned by the agent for that episode:

$$\sum_{t=0}^{H} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^H r_H$$

where it takes *H* steps to reach `ABSORBING_STATE` (note *H* might vary from episode to episode). For this assignment, set $\gamma = 0.9$. You should not reset the Q-Table between these 100 episodes so that the agent can remember what it has learned in previous episodes to help it do better in the current episode.

After you have the results from the 100 episodes, you should plot the results in a line chart where the x-axis is the episode number and the y-axis is the cumulative, discounted reward for each episode. Hopefully you should see the agent's performance generally increase over episodes.

For the line chart, you should plot three lines: one for the results of an agent using Softmax to decide what action to take, and two for the results of agents uses ϵ-greedy action selection (where you pick two values for ϵ). You will want to reset your Q-Table between evaluating the three different action selection approaches. Save your resulting line chart as an image and commit it to your Github repository.

Within a README file, you should include:

1. A short paragraph (3-5 sentences) comparing the results in your line chart between the three different action selection approaches.
2. A short paragraph describing your experience during the assignment (what did you enjoy, what was difficult, etc.)
3. An estimation of how much time you spent on the assignment, and
4. An affirmation that you adhered to the honor code

Please remember to commit your solution to your repository on GitHub. You do not need to wait until you are done with the assignment; it is good practice to do so not only after each coding session, but maybe after hitting important milestones or solving bugs during a coding session. ***Make sure to document your code***, explaining how you implemented the data structures and algorithms for Q-Learning.

Honor Code

Each student is to complete this assignment individually. However, students are encouraged to collaborate with one another to discuss the abstract design and processes of their implementations. For example, please feel free to discuss the pseudocode for the Q-Learning,

Softmax, and ϵ-greedy algorithms to help each other work through issues understanding exactly how the algorithms work. At the same, since this is an individual assignment, no code can be shared between students, nor can students look at each other's code.