CS 7642: Reinforcement Learning and Decision Making **Homework #3**

Sarsa

1 Problem

1.1 Description

For this assignment, you will build a Sarsa agent which will learn policies in the OpenAI Gym Frozen Lake environment. OpenAI Gym is a platform where users can test their RL algorithms on a selection of carefully crafted environments. As we will continue to use OpenAI Gym through Project 2, this assignment also provides an opportunity to familiarize yourself with its interface.

Frozen Lake is a grid world environment that is highly stochastic, where the agent must cross a slippery frozen lake which has deadly holes to fall through. The agent begins in the starting state S and is given a reward of 1 if it reaches the goal state G. A reward of 0 is given for all other transitions.

The agent can take one of four possible moves at each state (left, down, right, or up). The frozen cells F are slippery, so the agent's actions succeed only $\frac{1}{3}$ of the time, while the other $\frac{2}{3}$ are split evenly between the two directions orthogonal to the intended direction. If the agent lands in a hole H, then the episode terminates.

You will be given a randomized Frozen Lake map with a corresponding set of parameters to train your Sarsa agent with. If your agent is implemented correctly, then after training it for the specified number of episodes, your agent will produce the same policy (not necessarily an optimal policy) as the automatic grader.

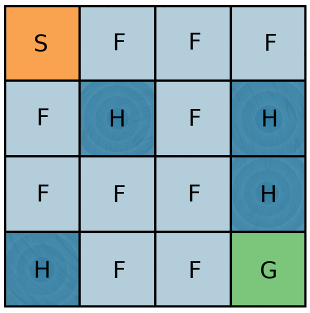

Figure 1: Example Frozen Lake Map

1.2 Sarsa $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

Sarsa uses temporal-difference learning to form a model-free on-policy reinforcement-learning algorithm that solves the *control* problem. It is model-free because it does not need and does not use a model of the environment, namely neither a transition nor reward function; instead, Sarsa samples transitions and rewards online.

It is on-policy because it learns about the same policy that generates its behaviors (this is in contrast to Q-learning, which you'll examine in your next homework). That is, Sarsa estimates the action-value function of its behavior policy. In this homework, you will not be training a Sarsa agent to approximate the *optimal* action-value function; instead, the hyperparameters of both the exploration strategy and the algorithm will be given to you as input—the goal being to verify that your SARSA agent is correctly implemented.

1.3 Procedure

Since this homework requires you to match a non-deterministic output between your agent and the autograder's agent, attention to detail to each of the following points is required:

- You must use Python and the library NumPy for this homework
- Install OpenAI Gym (e.g. `pip install gym`).

- Instantiate the Frozen Lake environment with `gym.make('FrozenLake-v0', desc=desc).unwrapped`, where the attribute `desc` is a square version of the string input **amap**. This means you must resize **amap** so that it looks like a square grid, where the input characters fill in the grid row by row. Make sure you use the `unwrapped` method as indicated above.
- The input **seed** should be used to seed the random number generators for both Gym and NumPy. Do *not* use the Python standard library's `random` library.
- Implement your Sarsa agent using an ϵ-greedy behavioral policy. Specifically, you must use `numpy.random.random` to choose whether or not the action is greedy, and `numpy.random.randint` to select the random action.
- Initialize the agent's Q-table to zeros.
- Train your agent using the given input parameters. The input `amap` is the Frozen Lake map that you need to resize and provide to the `desc` attribute when you instantiate your environment. The input `gamma` is the discount rate. The input `alpha` is the learning rate. The input `epsilon` is the parameter for the ϵ-greedy behavior strategy your Sarsa agent will use. Specifically, an action should be selected uniformly at random if a random number drawn uniformly between 0 and 1 is less than ϵ. If the greedy action is selected, the action with lowest index should be selected in case of ties. The input `n_episodes` is the number of episodes to train your agent. Finally, `seed` is the number used to seed both Gym's random number generator and NumPy's random number generator.
- To sync with the autograder, your Sarsa implementation should select the action corresponding to the next state the agent will visit *even when* that next state is a terminal state (this action will never be executed by the agent).
- The answer you provide should be the greedy policy with respect to the Q-function obtained by your Sarsa agent after the completion of the final episode. Specifically, the policy should be expressed as a string of characters: `<, v, >, ^`, representing left, down, right, and up, respectively. The ordering of the actions in the output should reflect the ordering of states in **amap**. Provide answers for the specific problems you are given on Canvas.

2 Examples

The following examples can be used to verify that your agent is implemented correctly.

- Input: amap="SFFG"; gamma=1.0; alpha=0.24;
  epsilon=0.09; n_episodes=49553; seed=202404

  Output: `<<v<`
- Input: amap="SFFFHFFFFFFFFFFG"; gamma=1.0; alpha=0.25;
  epsilon=0.29; n_episodes=14697; seed=741684

  Output: `^vv><>>vvv>v>>><`
- Input: amap="SFFFFHFFFFFFFFFFFFFFFFFFG"; gamma=0.91; alpha=0.12;
  epsilon=0.13; n_episodes=42271; seed=983459

  Output: `^>>>><>>>vvv>>vv>>>>v>>^<`

3 Resources

3.1 Lectures

- Lesson 4: Convergence

3.2 Readings

- Chapter 6 (6.4 Sarsa: On-policy TD Control) of Sutton and Barto 2020

4 Submission Details

The due date is indicated on the Canvas page for this assignment.
Make sure you have set your timezone in Canvas to ensure the deadline is accurate.
Submit your answers on Canvas, as outlined in section 1.3. You will have a total of 10 submission attempts - only the highest score is kept.

References

[SB20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2nd Ed. MIT press, 2020. URL: http://incompleteideas.net/book/the-book-2nd.html.