Assignment 4 - Tabular methods

Reinforcement Learning
MMAI 845, Spring, 2024
Due date – See syllabus

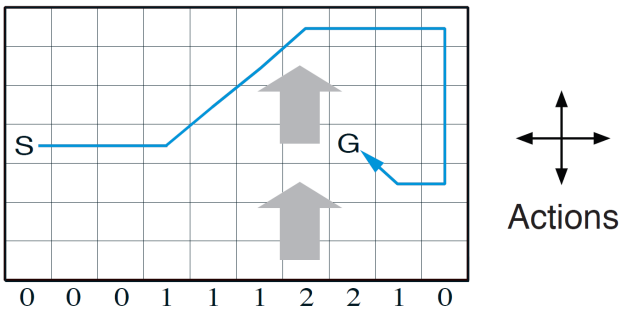

Figure 1: Gridworld with a crosswind running upward through the middle of the grid. The strength of the wind is given below each column.

We consider the Windy Gridworld (Figure 1) provided in Example 6.5 of the textbook. The cells of the grid correspond to the states of the environment. The state of the agent is represented by the pair (# of column or x position,# of row or y position). For example, the start position S is represented by the pair $(0, 3)$ and the goal position G is represented by the pair $(7, 3)$. Your goal is to use twp different reinforcement learning algorithms to learn solutions that are close to the optimal, which is shown in Figure 1. These are the rules in the Windy Gridworld environment:

- At each cell, four actions are possible: up, down, right, and left, which deterministically cause the agent to move one cell in the respective direction on the grid, but in the middle region, the resultant next states are shifted upward by a "wind". For example, if you are one cell to the right of the goal, then the action left takes you to the cell just above the goal.
- This is an undiscounted episodic task, i.e., $\gamma = 1$.
- The agent always receive -1 as reward for any action.

- The episode ends when the goal is reached.

Q1 – SARSA (10 pts): Use the pseudocode shown in page 130 of the textbook to complete the SARSA algorithm and tune the parameters for the algorithm to converge to the optimal policy.

Task 1.1 (4 pts): Implement the expression that updates the Q-table for the SARSA algorithm in tabular.py.

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]. \tag{1}$$

Task 1.2 (6 pts): The initial parameters we selected for this algorithm will not find a solution. You have to select the step size α, probability of selecting a random action ϵ, and number of episodes so that the algorithm finds a solution that is close to the optimal in the main.py. The total reward of the optimal solution is -14, your solution should be return a reward that is bigger or equal to -16.

Try to understand the effect of increasing or decreasing the parameters and make decisions based on that, rather than changing them arbitrarily. Keep the learning rate (step_size) bigger than 0 and smaller than 1. Keep the probability of selection a random action ϵ small, less than 10%, and make sure to run as many episodes as you need to stabilize the rewards. When the episodic rewards flatten, the learning process is usually finished or the algorithm is stuck. You can judge by looking at the solution.

Q2 – Q-learning (10 pts): Use the pseudocode shown in page 130 of the textbook to complete the Q-learning algorithm and tune the parameters for the algorithm to converge to the optimal policy.

Task 2.1 and 2.2 (4 pts): Implement the expression that updates the Q-table for the Q-learning algorithm in tabular.py.

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)], \tag{2}$$

This will be done in two steps: computing the action that maximizes Q for the next state S', and updating the table $Q(S, A)$.

Task 2.3 (6 pts): The initial parameters we selected for this algorithm will not find a solution. You have to select the step size α, probability of selecting a random action ϵ, and number of episodes so that the algorithm finds a solution that is close to the optimal in the main.py. The total reward of the optimal solution is -14, your solution should be return a reward that is bigger or equal to -16.

Try to understand the effect of increasing or decreasing the parameters and make decisions based on that, rather than changing them arbitrarily. Keep the learning rate (step_size) bigger than 0 and smaller than 1. Keep the probability of selection a random action ϵ small, less than 10%, and make sure to run as many episodes as you need to stabilize the rewards. When the episodic rewards

flatten, the learning process is usually finished or the algorithm is stuck. You can judge by looking at the solution.

Running The Code: The assignment comes with three files: **main.py** - This is the code that you have to run, and it is where Q-learning and SARSA are called; **tabular.py** - This is the code that implements SARSA and Q-learning; and **windy_gridworld.py** - This is the code that implements the environment. You do not need to modify it. There are two flags in the main code that you can set to False or True to run SARSA and Q-learning.

When you run SARSA or Q-learning, the algorithm is going to plot the accumulated rewards for all episodes. When you are done checking the rewards, you must close the window. Then, it will plot the learned policy, which you must also close, and it will finally run a simulation of the agent moving with the learned policy.

Formatting: Please upload all **three** completed .py file named in the format

$$tabular\_\{FNAME\}\_\{LNAME\}.py$$

where {FNAME} is your first name and {LNAME} is your last name.