Lab 3: Classical Reinforcement Learning

1 Introduction

In this lab we will attempt to construct an agent capable of managing the race simulator from the previous lab. So far, we have tackled building agents in a stationary, contextual bandit setting in which actions in the current turn affect the state in the next turn only through the information the agent has to learn from. The race environment is not stationary; actions in the current turn affect the location on the track and the velocities available to the agent in the next turn. The reinforcement learning algorithms we've encountered that can be used to build an agent capable of dealing with a non-stationary environment are Monte Carlo algorithms and the SARSA TD(0) algorithm; we will use these to train our agent here, including an explicit strategy for ensuring continual exploration of the possible actions.

2 Set up the racetrack environment

Begin by slightly modifying the racetrack environment you developed for Lab 2. In particular, change the environment so that if the car crashes it is sent back to the start line and the episode continues. You may work with the code you developed or my version of the racetrack that I will provide. The full description of the modified environment follows.

The modified racetrack. Consider driving a race car around a turn. You want to go as fast as possible, but not so fast as to run off the track. The racetrack is a grid – it begins with a vertical section 10 cells wide and 30 cells high, followed by a right turn into a horizontal section 15 cells wide and 10 cells tall. The starting line is the row of cells at the bottom of the first section. The finish line is the column of cells at the right of the horizontal section. The car begins at any cell on the starting line and each turn it is at one of the grid positions. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in each step, for a total of nine (3×3) actions. The vertical velocity component is restricted to be nonnegative. Both velocity components are restricted to be less than 5, and they cannot both be zero except at the starting line. Each episode begins in one of the randomly selected start states with both velocity components zero and ends when the car crosses the finish line. If the car attempts to go through the track boundary (anywhere but the finish line) it is automatically returned to a randomly selected position on the starting line, its velocity is set to 0, and the episode continues.

3 Learn about the environment

1. Set up the environment class by specifying the appropriate dictionary. In my version of the racetrack the config has the following structure:

```
height = 30    #set track height
width = 25    #set track width
v_track_width = 10    #set road width for the vert section
h_track_width = 10    #set road width for the horiz section
config = {'track_shape':{'height':height
                              ,'width':width
                              ,'v_track_width':v_track_width
                              ,'h_track_width':h_track_width}
          ,'starting_position':None
         ,'starting_velocity':None}    #set up the config
```

2. Experiment with this modified racetrack by running the step function. Simulate 100 races with randomly generated values for a (this will be different from the last lab because the environment has been modified).

4 Solving Markov Decision Problems via Monte Carlo & SARSA

1. Decide how you will represent the state information. Remember – this will be the only information that the agent has available to make decisions.

2. Decide how you will reward the agent. The goal of the agent is to reach to finish line without crashing and you need a reward function, $r(s', a, s)$, to incentivize the agent to do this.

3. Implement an agent capable of taking actions in this environment by writing python code to execute the following algorithms. Train the agent by running these algorithms for an "appropriate" number of episodes (likely no more than 10,000).

Algorithm 1 On-policy, first visit Monte Carlo Control with ε-soft policies

```
Input: $\gamma, \varepsilon \in (0,1)$ and $N \in \mathbb{N}_+$    ▷ Specify hyperparameters
 1: Init: $Q_\pi(s,a) \in \mathbb{R}$, $Z(s,a) = \emptyset$ for all $s, a$    ▷ Initialize $Q$-values/return history
 2: for $n = 1, 2, 3, \ldots, N$ do
 3:     Sample $a$ according to $\pi(s)$ to generate an episode:
            $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, \ldots, r_{T-1}, s_{T-1}, a_{T-1}, r_T$
 4:     Update $G = 0$    ▷ Initialize the return estimate
 5:     for $t = T-1, T-2, \ldots, 3, 2, 1$ do
 6:         Set $G \leftarrow r_{t+1} + \gamma G$    ▷ Update return
 7:         if for all $t' < t$ we have $(s_{t'}, a_{t'}) \neq (s_t, a_t)$ then
 8:             Add $G$ as a new element of $Z(s_t, a_t)$    ▷ Update return history
 9:             Update $Q_\pi(s_t, a_t) \leftarrow$ average of $Z(s_t, a_t)$    ▷ Update $Q$-value
10:             Set $a^* = \text{argmax}_a\{Q_\pi(s_t, a)\}$    ▷ Find best action for $s_t$
11:             Update $\pi(a|s_t) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{if } a = a^* \\ \frac{\varepsilon}{|A|} & \text{else.} \end{cases}$    ▷ Improve the policy
12:         end if
13:     end for
14: end for
```

Algorithm 2 SARSA with ε-greedy policies

```
Input: $\varepsilon, \gamma, \alpha \in (0,1)$, and $N \in \mathbb{N}_+$    ▷ Specify policy and hyperparameters
 1: Init: $Q_\pi(s,a) \in \mathbb{R}$ for all $s, a$    ▷ Initialize $Q$-values
 2: for $n = 1, 2, 3, \ldots, N$ do    ▷ Loop over episodes
 3:     Sample $s \in S$ and $a$ according to:    ▷ Choose initial state/action
            $\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{if } a = \text{argmax}_a\{Q(s,a)\} \\ \frac{\varepsilon}{|A|} & \text{else.} \end{cases}$
 4:     for $t = 1, 2, 3, \ldots, T$ do    ▷ Loop over turns
 5:         Implement $a$ to generate $r$ and $s'$    ▷ Get new reward/state
 6:         Sample $a'$ according to $\pi(s')$    ▷ Choose new action
 7:         Update $Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha\left[r + \gamma Q_\pi(s',a') - Q_\pi(s,a)\right]$
 8:         Update $s \leftarrow s'$ and $a \leftarrow a'$
 9:     end for
10: end for
```

Lab Assignment Report

Write up your analysis in a separate report of no more than 2 pages. Your report should answer the following questions:

1. Describe how you rewarded the agent and how you conceptualized the state.
2. In your simulations in 3.2. how many turns does it take on average for the agent to cross the finish line with random actions? How often did the agent cross the finish line with the unmodified racetrack? How will the modification we made to the racetrack interact with the reward function to enhance or impede the agent's ability to learn?
3. Consider the process of training your agent in 4.3. For each of the two algorithms how many episodes did you train your agent for? How many turns does each episode take as training progresses? Why do you think there might be a relationship between training time and episode length? Provide an visualization that shows relationship between episode and episode length for the two algorithms. Which one is more sample efficient?
4. What does the learned policy of the agent look like? Specifically, create and provide a visualization of what the agent chooses to do at each location of the track.
5. How do the initial Q-values compare to the learned Q-values for the two algorithms?
6. What effect (if any) do γ and ε have on the performance of the racetrack agent?

5 Submission Instructions

- Write up the answers to the questions in a short word document; aim for no more than 2 pages of text and include all graphics generated. Add footnotes identifying which sentence addresses which questions. Write in complete sentences organized into paragraphs – your goal is to explain what you've done and what you've learned to your audience (me!). Include the appropriate plots you've generated as mentioned above. Convert this to pdf and submit it. Submit your .ipynb file and an .html of it as well.
- The grading rubric for this assignment will be available in Canvas.
- NO OTHER SUBMISSION TYPES WILL BE ACCEPTED.
- You are welcome to use generative AI as you code up your solution. If you do this you must include all prompts used as an appendix in your written report or provide a shareable link to them. Any uncited use of generative AI (i.e. prompts not provided) will be considered plagiarism. You may not use generative AI to write your report.