CS 138: Homework 2

Lawrence Qiu

October 12, 2025

1 Introduction

Exercise 5.12 of Sutton and Barto asks us to apply a Monte Carlo control method towards a racetrack-like reinforcement learning environment. Monte Carlo methods utilize trajectories and rewards generated from interactions with the environment to inform q or value functions, rather than directly accessing the state transition function like in dynamic programming methods. Specifically, on the completion of the episode, the discounted cumulative reward is calculated for each state or state-action pair and the value or q function is updated. This is in contrast to temporal difference methods in which the previous step's value or q function is updated relative to the next.

In the racetrack environment, the agent controls a vehicle attempting to navigate a right turn. On each time step, the agent can increase or decrease the x and y velocity of the vehicle, limited to positive values less than 5. Additionally, the vehicle cannot be stationary. The map is a tile grid, with a horizontal line denoting the starting tiles and a vertical line denoting finish tiles. The vehicle starts at a random start tiles and must reach the finish tiles without hitting any boundary tiles. If it does, the agent restarts at a starting tile and the velocity is reset. To motivate the agent to reach the end as fast as possible, the agent is given -1 reward until reaching the end. There is a 10% chance on each time step that the acceleration requested by the agent is zeroed out.

2 Part 1: Implementing the exercise

For the Monte Carlo method, I implemented an on-policy every-visit incremental MC algorithm. The Q function and a count C function are represented as arrays, both initialized to zero. Then, after each episode, the cumulative reward G is set to zero and each step is visited backwards. G is then updated with the discounted reward

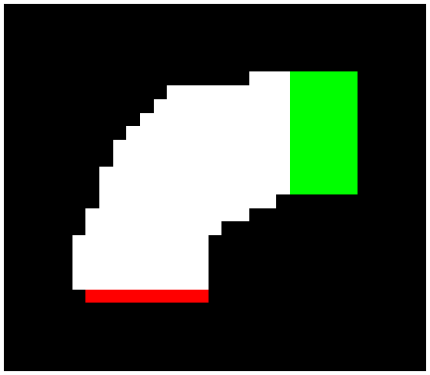
$$G \leftarrow \gamma G + R$$

and Q and C are updated incrementally

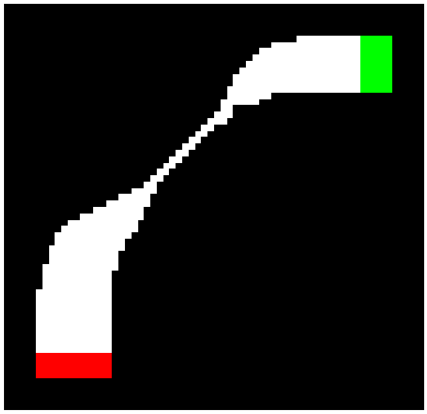
$$C \leftarrow C + 1, \qquad Q \leftarrow Q + \frac{1}{C}(G - Q).$$

For control, I simply used ϵ-greedy where there is an ϵ probability of picking a random action and a $1 - \epsilon$ probability of picking the maximum value action with ties broken randomly.

To implement the environment, I first created a way of representing the map as a file. I decided to use an image so that it can be edited using conventional image editors. I also decided to represent the start and end lines as tiles so that they can all be incorporated into a single file. The following images represent the *easy* and *hard* map. To calculate the next position, I decided to just increment the position by velocity. This means it is necessary to extend the end tiles so that the vehicle cannot jump past the finish line. The rest of the implementation was straightforward based on the problem description.



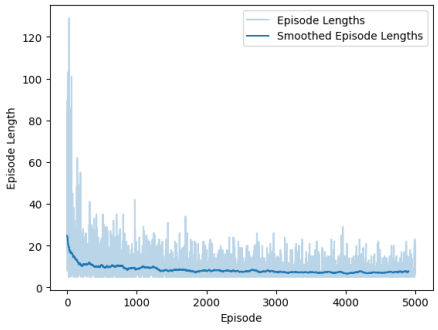
(a) Easy map. Red tiles represent starting points and green tiles represent finish points. The path is wide and short.



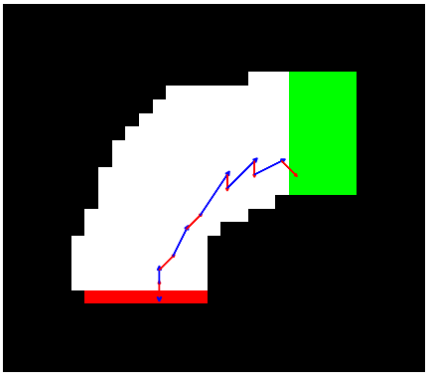
(b) Hard map. Red tiles represent starting points and green tiles represent finish points. The path is longer with a thinned bottleneck in the middle.

Figure 1: Maps used in the racetrack environment.

For the following tests, I used the following parameters: # episodes = 5000, $\epsilon = 0.1$, $\gamma = 0.9$. The following figures illustrate how the agent converged for both.

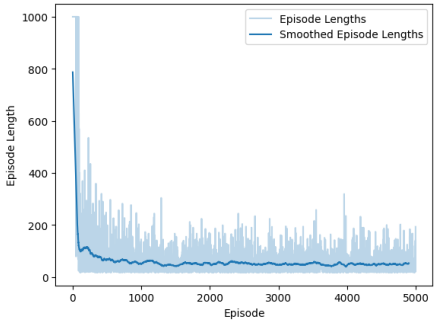


(a) Episode length by episode (easy map). The agent converges quickly, averaging to 8 steps per episode by the end of training.

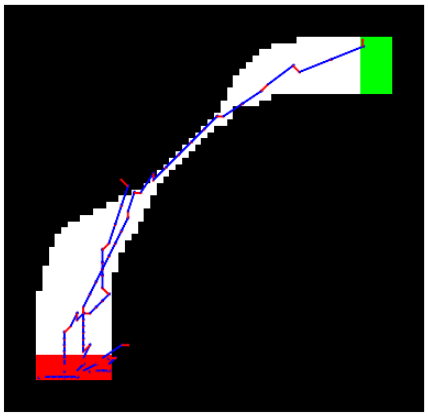


(b) Sample trajectory (easy map). Blue lines denote existing velocity while red lines denote acceleration.

Figure 2: Training results on the easy map.



(a) Episode length by episode (hard map). The agent converges slightly slower but still quickly with average of 50 steps per episode at end of training. Several resets likely occur due to zeroed out acceleration.



(b) Sample trajectory (hard map).

Figure 3: Training results on the hard map.

To see the effects of the hyperparameters ϵ and γ, I performed a grid search with $10^{-5} < \epsilon < 10^{0}$ and $10^{-10} < \gamma < 10^{0}$. The results are shown in Figure 4. The plot shows the total number of steps over 5000 episodes—this value reflects both the converged path as well as the speed of convergence. Each set is averaged over 10 runs.

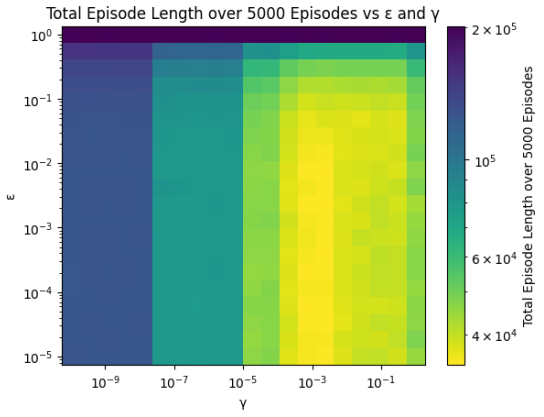


Figure 4: Total number of steps vs. ϵ and γ. ϵ does not seem to have a significant effect under 10^{-2}, indicating that exploratory steps are mostly harmful. This is likely because explicit exploration is not necessary as the initialization of q to 0 would be considered optimistic as rewards are negative so exploration is already driven. Interestingly, γ does have an effect with an optimal value of around 10^{-3}. The minimal total number of steps is 34686.

Due to the significance of the initial value of q, I also performed a hyper-parameter search over $-20 < q < 0$ and $10^{-5} < \epsilon < 10^{0}$. This was done with $\gamma = 0.95$ so the minimal possible discounted cumulative reward is -20.

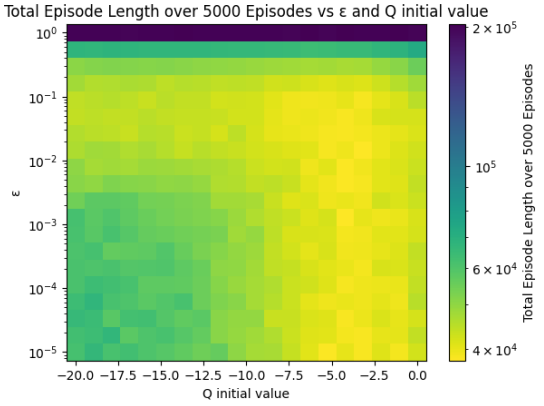


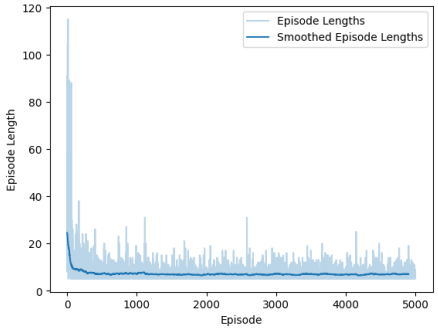
Figure 5: Total number of steps vs. ϵ and initial q value. For high ϵ values ($\epsilon > 10^{-2}$), the initial q value doesn't matter as much as ϵ guides exploration. For lower ϵ values, initial q value does matter, with optimal value around -4 and performance decaying below this. This is because with low ϵ and q exploration is inhibited. The minimal total number of steps is 37830.

3 Part 2: Comparison with Upper-Confidence-Bound selection

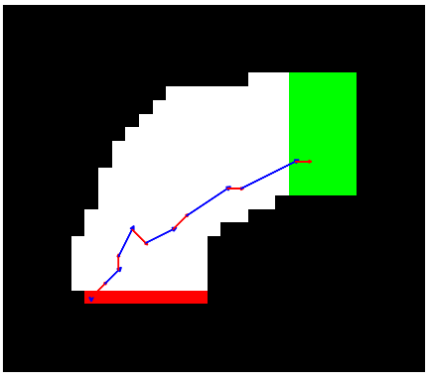
An alternative to ϵ-greedy exploration is upper confidence bound exploration. In UCB, the highest-value action is always selected, but the q value is offset with an uncertainty / exploration term. This term depends on both the number of times the state/action pair has been explored before as well as the current total number of steps, as to increase the total number of exploration steps as time passes. Specifically, the value of each action is calculated as follow:

$$A_t = \arg\max_a \left[Q_t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}} \right].$$

The coefficient c denotes how much weight the exploration term has. I performed the same experiment on the easy map with $c = 0.1$. Figure 6 shows the number of steps per episode over time and a sample trajectory.



(a) Episode length by episode (UCB on easy map). The policy quickly converges to about 8 steps per episode, similarly to ϵ-greedy. The variance is lower likely because UCB is not stochastic (besides tie breaking).



(b) Sample trajectory (UCB on easy map), similar to those under ϵ-greedy.

Figure 6: UCB on the easy map: learning curve and sample trajectory.

I performed a hyperparameter search with $10^{-5} < c < 10^{1}$ and $10^{-10} < \gamma < 10^{0}$. The results are shown in Figure 7.

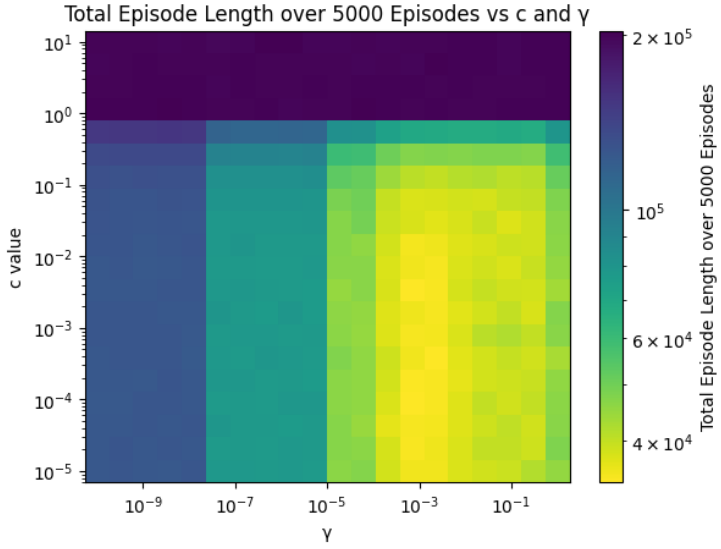


Figure 7: Total number of steps vs. c and γ. High c values ($> 10^0$) result in poor performance due to excess exploration. Below that the plot is very similar to the results for ϵ-greedy with optimal γ value of 10^{-3}. The minimal total step count is 33901.

4 Summary and Conclusion

I implemented a Monte Carlo control algorithm and applied it to the racecar environment. I tested two exploration methods: ϵ-greedy and upper confidence bound. For both methods, low exploration rates performed best. This was likely because explicit exploration was not necessary because the initial q values are optimistic: q values can only be negative as rewards are negative so when they are initialized to zero, the exploration algorithm is incentivized to pick less explored options.