Racecar with Monte Carlo Control

CS138 HW2

Joseph Egan

October 2022

Contents

Introduction

Monte Carlo control allows an agent to estimate the value of state-action pairs in a Markov Decision Process (MDP) without already having a model of the environment. The agent generates a series of episodes by taking actions that move the agent from state to state until either it is successful, or the episode terminates for some other reason. At the end of the episode, the final state-action pair is assigned the final reward as its value, and then, working backwards, each state-action pair visited is assigned a value based on the reward received at that pair plus some discounted sum of all rewards received afterwards. Applying an epsilon-greedy choice selector then allows the agent to find optimal pathways.

Goal

The goal of this project was to create a racetrack "track" environment with a "car" agent that utilizes MC methods to successfully navigate two tracks. An episode consists of a random starting spot, a trajectory determined by an epsilon-greedy selection method from a simple average Q-table, and either termination via the finish line or going out of bounds of the racetrack. We were able to demonstrate

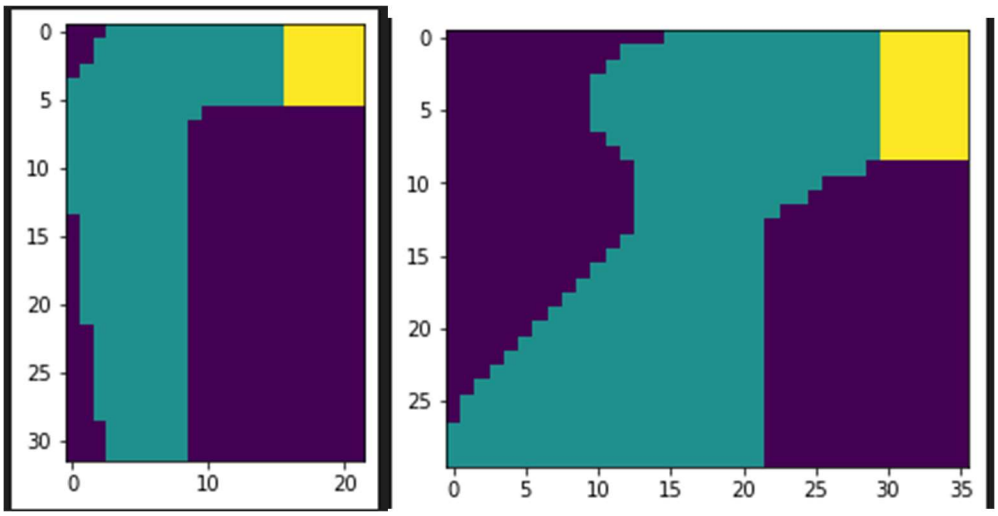
that this method allowed the car to build an accurate trajectory for each possible starting point. Finally, an experiment to speed up learning is proposed and tested.

Methodology

Environment Setup

The environment was a class that took a numpy matrix and reward parameters as inputs. The numpy matrix provided the structure of the racetrack where 0 was out of bounds, 1 was on the track, and 2 was the finish zone. We created two tracks – small and large, which refer to the general width of the tracks:

Small is on the left, and Large is on the right. Here is what they look like in our trajectory charts (without any trajectories plotted):



For rewards, completing the track successfully yielded 20, traveling out of bounds or off the grid yielded -5, and each step yielded -1.

Agent Setup

The agent was a class that took the following inputs:

Epsilon	Epsilon for epsilon-greedy exploration, 0.1 for all cars
Gamma	Discount rate for returns, 0.5 for all cars
Noise_Ratio	Proportion of time when action selection would be overridden to [0,0], 0.1 used
Allow_Noise	Boolean to toggle allowing noise
Episode_max	Max number of episodes the car would perform, to prevent endless/excessive looping
Q_frame	Multi-dimentional array for state-action pairs specific to each track
Avail_actions	List of possible [X,Y] velocity increments
Vel_max	Max speed
Vel_min	Minimum speed

Action Selection & Value Estimation

The car would start at a random spot on the starting line (bottom row of 1's) with (x,y) velocities of [0,0]. In each position, the car chose an acceleration value of +1, -1, or 0 for both the X and Y direction. Velocities for both directions were bounded at 0 and 4 inclusively. At each step, a method would generate a list of potential [X,Y] acceleration pairs based on current velocity and position (the car could not select [0,0] on the starting line). The car then chose the action that had the highest estimated value according to the Q-table initiated by the q_frame input (initially all 0) that was available based on the actions allowed. Exploration (random selection of an action) was toggled to occur 10% of the time. With the allow_noise toggled to True, despite what the car selected, there was a 10% chance that the action would be [0,0].

At the end of an episode (either finishing or going out of bounds), values were assigned to each state-action pair visited in that episode in reverse order following this algorithm:

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  $G \leftarrow \gamma G + R_{t+1}$
  Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
    Append G to $Returns(S_t)$
    $V(S_t) \leftarrow \text{average}(Returns(S_t))$

The "Unless…" line was irrelevant in this case since velocity could not be [0,0] except at the first step, and the car could only move up and to the right.

Where the V(St) update occurred incrementally (historical returns for each state-action pair were not stored). Since the policy being evaluated and updated was also the policy being used for exploration, this was an on-policy method.

Termination

Several methods for determining the terminus of running episodes were explored. Initially, a hard cap of episodes was used. This method was useful in comparing average rewards between noise and no noise, and later, our experiment vs base models. However, we also explored using a proportion of average reward over the last X episodes vs average reward over the last Y episodes where Y >> X. The idea would be that when these two averages became very similar, significant improvements were no longer being made to the policy. However, since overall rewards can vary drastically between starting spots, this often ended with the cars stopping much earlier than we wanted. Eventually we settled on using the ratio of the number of finished races (episodes) over total episodes run. When this reached our

threshold, the car was done. This allowed us to compare which cars could converge faster. Our threshold was set somewhat arbitrarily as (1-epsilon) since this value positively correlated with higher victory percentages. In other words, after learning the best paths, a lower epsilon car should complete the race more often.

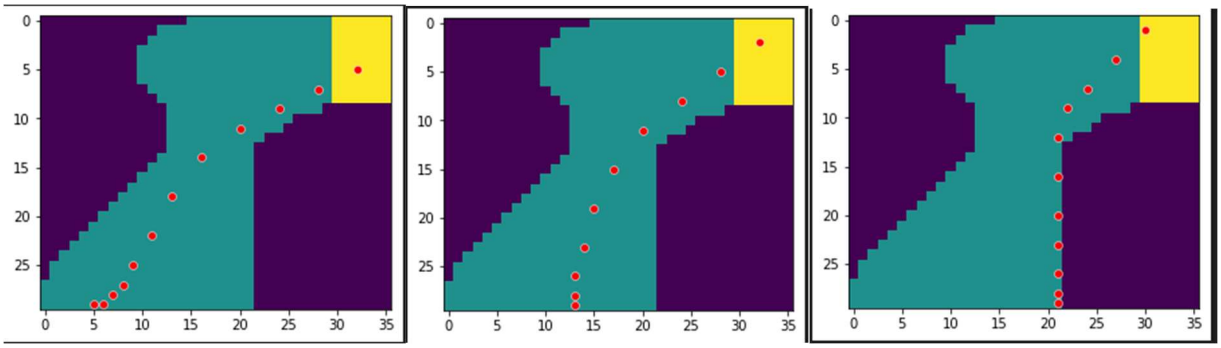
Analysis of Results

Under these parameters, it was found that cars were able to converge on the large track both with and without the noise. However, on the small track, the cars were not able to converge in less than 700,000 episodes (we did not test past that number), and we ultimately capped them at 400,000 episodes:

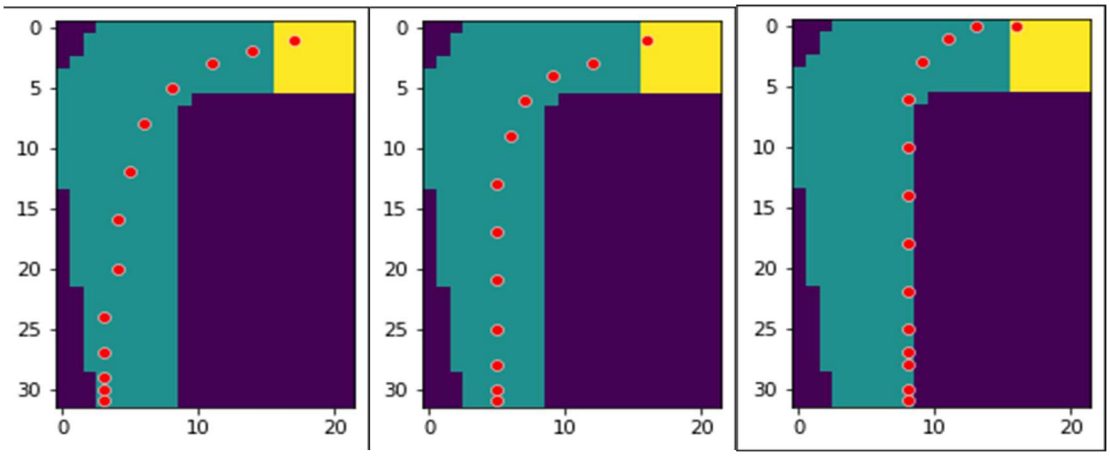
Car_Type	Total_Episodes	Total_Finishes	Final_Victory_Pct	Last_50_Avg_Reward
LT Racecar with Noise	140734	125394	0.891000	8.113333
LT Racecar with No Noise	68643	61161	0.891001	9.080000

Car_Type	Total_Episodes	Total_Finishes	Final_Victory_Pct	Last_50_Avg_Reward
ST Racecar with Noise	400000	280344	0.70086	0.136667
ST Racecar with No Noise	400000	302484	0.75621	3.203333

It is clear from the above that the small track was much more difficult. This makes sense – the sharp turn took more time to figure out as the car often needed to slow down vertical velocity as it was speeding up horizontal velocity, whereas the large track allowed for a smoother diagonal trajectory. This is confirmed when looking at a sampling of final trajectories calculated:



Final greedy trajectories for starting on the large track at 5, 13, and 21.



Final greedy trajectories for starting on the small track at 3, 5, and 8.

It is also clear from the results that the noise had a large impact on the learning rate of the car. For the large track, the car converged more than twice as quickly when it did not have noise. For the small track, the car was able to achieve a much higher average reward after 400,000 episodes.

Experimentation

An additional method was added to the car class that allowed a human to drive the track (without noise). Our experiment was to manually drive the track successfully for 20 episodes for both tracks. The q values generated from this were then loaded into a fresh car. To bootstrap the value of the human laps, the counts for each state-action pair visited by the human were multiplied by 10. Therefore, a 'hybrid' car started with 200 episodes pre-loaded with successful completions. The hybrid cars were then let loose on the large and small tracks similar to the non-hybrids tested previously:

Car_Type	Total_Episodes	Total_Finishes	Final_Victory_Pct	Last_50_Avg_Reward
LT Racecar with Noise	140734	125394	0.891000	8.113333
LT Racecar with Noise and Human Boost	172505	153702	0.891000	8.560000
LT Racecar with No Noise	68643	61161	0.891001	9.080000
LT Racecar with No Noise and Human Boost	70257	62599	0.891000	9.526667

While the human boost yielded higher rewards in both scenarios, when noise was included, it actually caused the car to need more episodes to converge to the 90% victory level. It is unknown why this happened, but it is possible that the pre-loaded q values hindered the car when it started experiencing noise, since the human had no noise when training.

Car_Type	Total_Episodes	Total_Finishes	Final_Victory_Pct	Last_50_Avg_Reward
ST Racecar with Noise	400000	280344	0.700860	0.136667
ST Racecar with Noise and Human Boost	400000	267393	0.668482	0.903333
ST Racecar with No Noise	400000	302484	0.756210	3.203333
ST Racecar with No Noise and Human Boost	400000	313303	0.783258	5.083333

Similarly, the benefit of the human boost is clearer without noise than with noise on the small track. Average reward is higher in both scenarios; however, victory percentage is lower with the human boost

with noise than without. Again, this could be because the human q values were generated without noise, but other explanations are possible.

Conclusions

In this project we tested MC control to train an agent to navigate two racetracks successfully from random starting points. Human interaction was added with the pre-loaded q values, and it was shown that without the noise component, it was beneficial to the car to have the human help. When noise was included, it became less clear whether the human boost was helping. Ultimately, if the goal of adding human involvement was to reduce the amount of time spent learning for the car, this human boost is not actually very helpful, since it took around 10 minutes to run the 20 episodes manually on each track, whereas the car could do 300,000 episodes in that time. However, under other constraints like compute time or limited episodes, human interaction could be helpful.

Future Improvements/Questions

The biggest improvement would be running multiple simulations at different seeds for each car type instead of the one run for each car that we did here due to computing and time limitations. Our results would be much more conclusive with multiple samples. Additionally, different values for epsilon, gamma, and noise_ratio could be sampled to tune the performance of the car.

Another experiment that would have been interesting would be changing the step penalty (-1) to be a function of distance from the finish line. We would expect that the car would learn faster since it would greedily move through regions rather than towards a fixed area.

Bonus Questions

Exercise 5.2

Suppose every-visit MC was used instead of first-visit MC on the blackjack task. Would you expect the results to be very different? Why or why not?

Answer:

The results should not be different. In blackjack, with each hand played being an episode, there is no possibility of reaching the same state-action twice since the cards are sampled without replacement. Therefore, first-visit would be the same as every-visit.

Exercise 5.4

The pseudocode for Monte Carlo ES is inefficient because, for each state–action pair, it maintains a list of all returns and repeatedly calculates their mean. It would be more efficient to use techniques similar to those explained in Section 2.4 to maintain just the mean and a count (for each state–action pair) and update them incrementally. Describe how the pseudocode would be altered to achieve this.

Answer:

I actually already did this in my code, it's just $Q \leftarrow Q + (1/n)*(R-Q)$.