CS0138 Assignment 2: Monte Carlo Simulation

Peter Nadel

October 2025

1 Introduction

For this assignment, we were asked to simulate the movement of a car through a racetrack using a Monte Carlo control algorithm. Specifically, the agent in our simulation must learn how to control its velocity in order to move through the racetrack. In both the X and Y directions, it is able to increment its current velocity by -1, 0, or 1, with a total set of 9 actions. Additionally, the agent may *not* exceed a speed of 5 in either direction or fall below a speed of zero in either direction.

The various constraints of this problem make it difficult to implement, but Monte Carlo methods excel at this task, so long as the track is not too complex. The goal of Monte Carlo policy estimation and modeling is to simulate a variety of actions for many episode so as to determine the probability distribution over the entire decision space. Take a simpler example of the racetrack problem. Let's say our track is 5 squares by 5. That would give us 25 unique positions that the car could be in. Likewise, we must have a separate representation for each of the possible velocity values our car could have in each of these positions. Added to these, we must also account for the three possible actions in both the X and Y directions. This huge set of possible states and actions (5 X positions x 5 Y positions x 5 possible X velocities x 5 possible Y velocities x 3 possible X actions x 3 possible Y actions = 5,625 possible configurations) is very unwieldy to handle for any deterministic methods. Instead, Monte Carlo samples a set of random trajectories in our decision space and updates our policy accordingly at the end of each episode. Though we are unable to grasp the entire space, given enough episodes, Monte Carlo will converge to a solution which maximizes the rewards in an episode.

As a result, Monte Carlo works well for the racetrack problem. Because the decision space is so huge, but the number of actions is in comparison small, Monte Carlo is able to converge to a solution very quickly.

2 Methods

2.1 Standard racetrack problem

Sutton and Barto provide two 36x36 tracks to learn through the Monte Carlo method. Their array representation is shown below, with the starting positions marked in green and the finish line marked in yellow. The blue track is surrounded with purple off-track zones. Per the assignment, if the car touches these areas, the episode continues but the car is randomly replaced at a starting position.

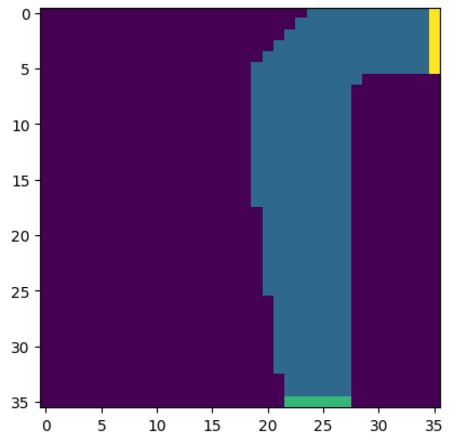

Figure 1: Track 1 from Sutton and Barto

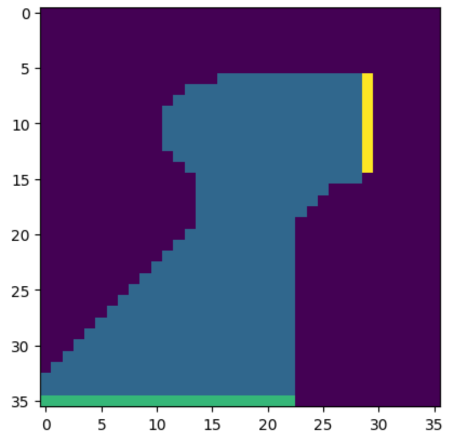

Figure 2: Track 2 from Sutton and Barto

So that the agent is able to learn how to navigate any track, we applied the

On-policy first-visit Monte Carlo control algorithm as described in Sutton and Barto:

Algorithm 1 On-policy first-visit Monte Carlo control algorithm

```
Initialize the following:
$\pi \leftarrow$ an arbitrary epsilon-soft policy
$Q(s, a) \in \mathbb{R}$ (as zeros)
$C(s, a) \in \mathbb{R}$ (as zeros)
loop
    Generate an episode following $\pi$
    $G \leftarrow 0$
    loop
        $G \leftarrow \gamma G + R_{t+1}$
        $C(s, a) \leftarrow 1$
        $Q(s, a) \leftarrow G - Q(s, a)$
        $A^* \leftarrow \arg\max_a Q(s, a)$ (with ties broken arbitrarily)
        $\pi(a|S_t) \leftarrow A^*$
    end loop
end loop
```

When implementing this approach in Python, we needed to add a few features due to some ambiguity in the problem. First, we added a number of maximum steps which an episode can last. If an episode hits this limit, it stops and the Q table is updated based on those steps. In early episodes, the agent's decision are entirely random, meaning that it often fails to reach the finish line or even get close to it. A key disadvantage of Monte Carlo methods is that they requires the episode to complete to update the Q table. As a result, long episodes which do not complete can hold up training, not letting the agent learn from its continual mistakes.

Second, because the starting line of both tacks are at the bottom and the finish lines is at the top, increments to the Y position must be subtracted rather than added to the current Y position. Though counter-intuitive, this detail was critical to teaching the agent.

Last, when using our trained Q table to develop completed trajectories, we needed to turn off all stochasticity. This includes both the noise factor mentioned in the problem description and the epsilon-soft policy.

2.2 Extra experiment

For our extra experiment, we chose to explore the effect of varying the γ parameter. This value represents the discount of the end of episode reward in the following formula from Sutton and Barto:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

This parameter, as a result, controls how much confidence the agent has in a previous reward. The higher the γ the more the agent learns from previous results, while the lower it gets, the less faith the agent has in these previous results. It helps to vary this parameter for both theoretical and practical reasons. First, by varying this parameter, we can see exactly the effect that it has on the agent during training and during the update step. We hypothesize that not only will the agent perform worse at lower values of γ but also that its results will be more varied and less controlled, perhaps failing to converge at all in the allotted number of episodes. Second, this experiment will help us pinpoint what exact value of γ is best for training an agent for this task.

3 Results

3.1 Standard racetrack problem

After training an agent for each of the above tracks, we report the following optimal trajectories:

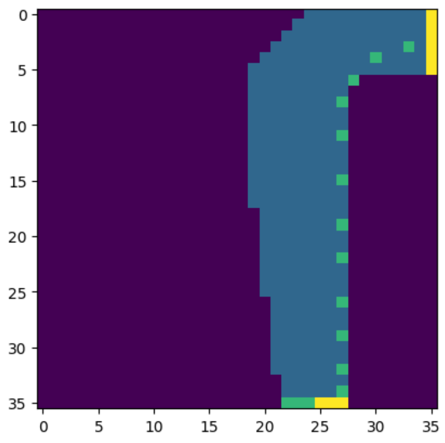

Figure 3: Learned path of track 1 from Sutton and Barto

Both trajectories show that, through the Monte Carlo method alone, both agents were able to learn the right turn behavior needed to complete the tracks.

At first glance, these trajectories look very similar, but they show a distinct difference in how each track is constructed and thus how each agent learned to complete the track. In figure 3, we see that the agent takes very consistently sized steps to reach its goal, leaning to modulate its speed so that it doesn't fly off the track once it must turn. On the other hand, figure 4 shows an agent which as learned that it can get to the finish line with larger steps, that is larger increments to its velocity components. Indeed, where this agent must be more careful is the beginning as there are many configurations in which it can find itself on the side of the track, not on course to get to the end.

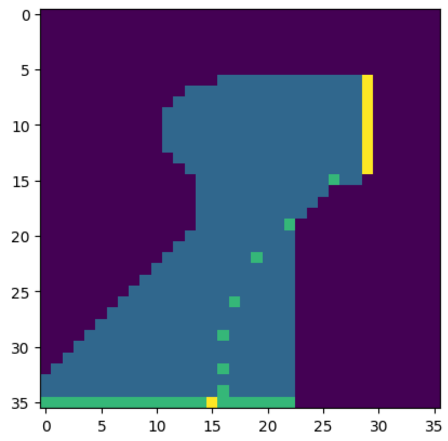

Figure 4: Learned path of track 2 from Sutton and Barto

3.2 Extra experiment

As mentioned above, we chose to vary our γ value over several training runs to see the effect that this parameter has over training. Below is plotted a rolling average of the amount of steps needed to complete the course against the index of that episode during the training process.

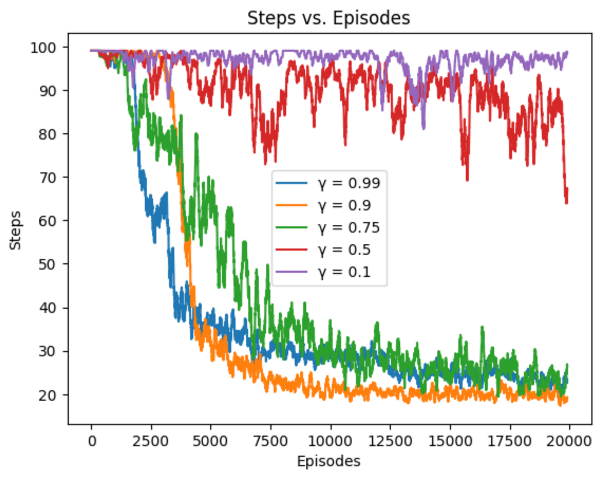

Figure 5: Steps vs. Episodes on track 1 from Sutton and Barto

Very clearly, we see that γ values of 0.5 and below far under-perform policies trained with higher values of γ. However, this is not the only thing that we can learn from this figures. We also observe that higher γ values result in much more variation in episode length from episode to episode. This is a key feature

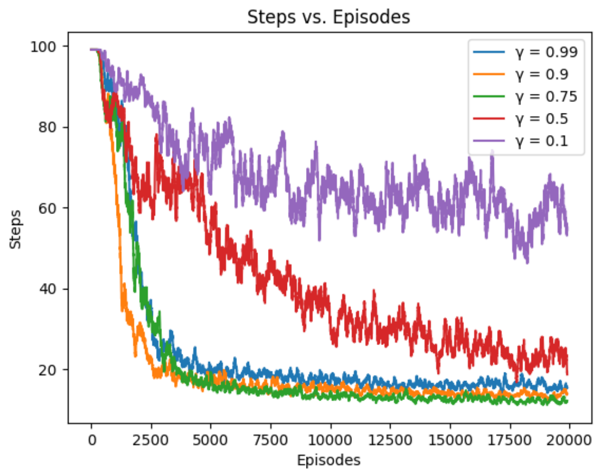

Figure 6: Steps vs. Episodes on Sutton and Barto

as the more variation in our training the less optimal our policy becomes over the whole training run. As a result, we can see that the γ parameter plays a key role in controlling how consistent training is over all episodes. This can be a detriment though too. When $\gamma = .99$, we see that episode length follows the same same pattern as when $\gamma = .9$, but in both cases, this training process converges to a higher value than when $\gamma = .9$. When γ is very close to 1, past rewards are not discounted enough, while when γ is closer to 0, past rewards are discounted too much. Importantly, both γ values give the agent a skewed view of the track and inhibiting learning.