CS 138 RL: Programming Assignment 2

Robert Pitkin

October 12, 2022

1 Background

Monte Carlo methods are one of the fundamental learning methods in reinforcement learning for approximating value functions and optimal policies. What differentiates Monte Carlo (MC) methods from previous tabular solution methods we've discussed in class, such as dynamic programming and K-armed bandits, is that MC methods don't require full knowledge of the environment, only experience. This experience comes in the form of simulated or pre-generated sequences of states, actions, and rewards known as roll-outs. By learning only from experience, MC methods are freed from the need for known probability distributions for state transitions, which are hard to come by in real-world or dynamic applications. As a result, the way MC methods solve the reinforcement learning problem is by averaging sampled returns from roll-outs in order to approximate the optimal value function or policy. While initial approximations are typically poor representations of the optimal values, MC methods are guaranteed to converge to the true optimal value and policy functions as the number of roll-outs approaches infinity. In practice, however, a satisfactory result can usually be achieved with a large, but efficient number of roll-outs.

Similar to dynamic programming methods, MC methods also follow the idea of generalized policy iteration. That is, alternating iterations of policy evaluation and policy improvement are carried out in order to approach optimality for both the policy and value functions. These two processes seem to compete with each other, as the policy evaluation step optimizes the current value function by following the policy and the policy improvement step optimizes the policy by acting greedily with respect to the value function, but at each iteration both functions gradually improve until optimal values are achieved. The combination of policy evaluation and improvement for MC methods is known as Monte Carlo Control, analogous to policy iteration for dynamic programming methods, and comes in two forms: on-policy control and off-policy control.

On-policy MC methods utilize a single policy that is used for both generating roll-outs and evaluation and improvement. Off-policy MC methods, on the other hand, use two policies, a behavior policy, which is used for episode generation and usually contains elements of randomness (such as ϵ-greedy action selection or random action selection), and a target policy, which is initialized arbitrarily, but is greedily improved using generated roll-outs from the behavior policy and contains no element of randomness. The benefit of on-policy MC methods is that only a single policy has to be maintained and usually results in a simplified implementation. However, because on-policy methods are required to have exploration built in, such as ϵ-greedy action selection, the generalized policy iteration process only results in a near-optimal policy since there's always a probability for exploration. This is where off-policy MC methods succeed. Because the target policy used in off-policy methods always acts greedily and is optimized according to the constantly exploratory roll-outs generated from the behavior policy, it can guarantee both optimality and efficient exploration throughout the entire policy iteration process. Although off-policy MC methods are generally harder to implement and are slower to converge, they are overall more powerful in their capabilities than their on-policy counterparts.

2 Problem Statement

For this assignment I have created a racetrack simulation, described by exercise 5.12 on page 111 of [1], and employed on-policy first visit Monte Carlo Control in order to demonstrate the capability of MC methods and exhibit a near-optimal policy for the posed problem. I start by arbitrarily initializing the policy and value functions and generating roll-outs as the problem describes. Then I extend the assignment with an investigation into what the impact of human-generated episodes for the initial steps of policy iteration has on the speed of convergence to the near-optimal policy.

2.1 Exercise 5.12 and the Racetrack Environment

Exercise 5.12 from [1] tasks us with creating a racetrack environment with a reasonably sharp right turn. The racetrack itself is made up of discrete grid cells with color-indicated sections for the start states and terminating states, each of which represents a possible position of the racecar. The velocity in the problem is also discrete, described by the number of grid cells to move vertically and the number of cells to move horizontally at each time step. However, the problem restricts both components to being non-negative and no larger than five. This restricts our agent to only moving forwards or right (or both), which stops the agent from being able to loop around in circles. The actions for this environment are then the potential changes that could be made to each velocity component at each time step and are restricted to (-1, 0, 1) for both components resulting in nine total possible actions. In my experiment, I made the assumption that the changes in velocity were instantaneous, thereby adjusting the velocity before the agent moves. As for the rewards, the problem originally states that the agent is given a reward of -1 for every time step and +1 if it reaches a terminal state and to simply reset to the starting line if the agent goes off the edge of the track. I significantly changed the rewards to be +100 when the terminal state is reached and -100 if the agent goes off the track, both of which end the episode. This way, the agent is more dissuaded to drive off the edge of the track and crossing the finish line has a large enough positive reward to make significant changes to the values of the previous states visited. Lastly, the problem also states that the agent also selects the action (0, 0) with a 10% probability at each time step, which was implemented in addition to whatever exploration is done by the agent. The purpose of this exercise is to demonstrate how MC control methods compute optimal policies and to display learned optimal trajectories as our results.

3 Methods

The implemented racetrack and on-policy first visit monte carlo control method were made in Python 3 using the pseudo-code and descriptions on page 101 and 111-112 of [1] as inspiration and then was modified to create the "more challenging" environment described in exercise 5.12. The cited pseudo-code is shown below:

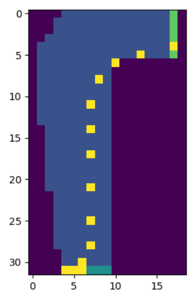
Algorithm 1 On-policy first-visit MC control

```
Algorithm parameter: small $\epsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\epsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in S, a \in A(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in S, a \in A(s)$
Repeat forever (for each episode):
    Generate an episode following $\pi : S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, ..., 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, ..., S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)
            $A^* \leftarrow \text{argmax}_a Q(S_t, A_t)$
            For all $a \in A(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|A(S_t)| \text{ if } a = A^* \\ \epsilon/|A(S_t)| \text{ if } a \neq A^* \end{cases}$
```

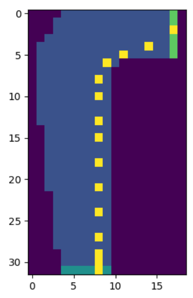
The racetrack grid itself was created as a two-dimensional list, which had a different number at each index representing "out of bounds" (0), "racetrack" (1), "starting line" (2), and "finish line" (3). The NumPy library was utilized to select random actions or generate random numbers and the MatPlotLib library was used to display trajectories. Additionally, the random seed was set to be the instance number for each episode to ensure repeatability. In other words, episode 0 had a seed of 0, episode 1 had a seed of 1, and so on.

4 Results

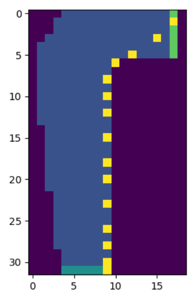
For each of the racetracks shown on page 112 of [1], the on-policy first visit MC control agent simulated episodes with an ϵ value of 0.1 and a γ value of 0.9 until convergence. Convergence in this experiment was defined to be when the difference between the weighted average of the previous ten episodes and the previous hundred episodes was less than $1e^{-6}$. It took around 100,000 episodes for the first racetrack to converge and around 200,000 episodes for the second to converge. At the end of the episodes, five additional random episodes were generated using the learned policies with random action selection and added noise removed. The trajectories are displayed below.



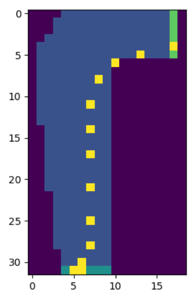
(a) Track 1 - Optimal Trajectory 1



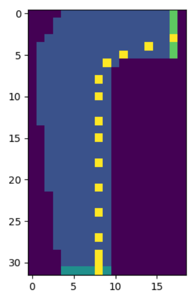
(b) Track 1 - Optimal Trajectory 2



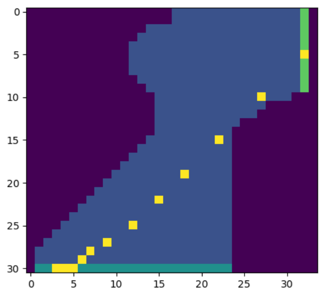
(c) Track 1 - Optimal Trajectory 3



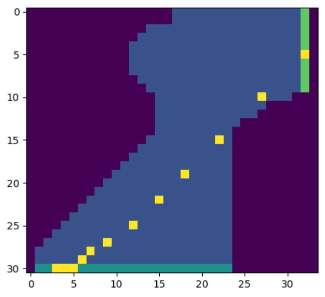
(d) Track 1 - Optimal Trajectory 4



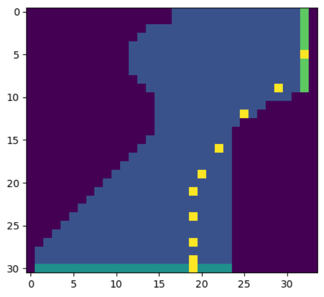
(e) Track 1 - Optimal Trajectory 5



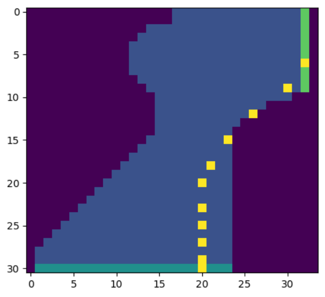
(a) Track 2 - Optimal Trajectory 1



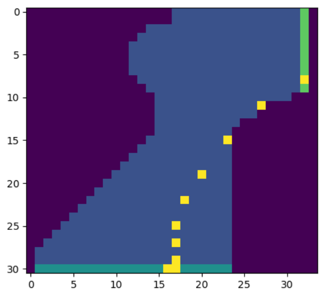
(b) Track 2 - Optimal Trajectory 2



(c) Track 2 - Optimal Trajectory 3



(d) Track 2 - Optimal Trajectory 4



(e) Track 2 - Optimal Trajectory 5

5 Discussion

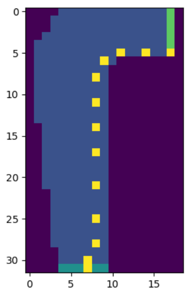
There two interesting patterns that cropped up across both racetrack policies. The first of these patterns was the agent learning to cut corners. Because of the nature of the velocity components as well as the instantaneous movements described earlier, the agent learns to take seemingly diagonal moves that cross over out of bounds grid cells but end up on another cell on the track. Such behavior can be found at the sharp turn for both tracks, but particularly in the last four racecar positions of optimal trajectory 4 for track 2. This behavior is clearly optimal since it covers the most distance without having to significantly change the velocity. However, it would be reasonably hard for a human to input such moves without stopping to calculate out where the racecar would be after each move. Yet, from the agent's point of view, it has no sense of a "corner", it only knows that given a particular cell near the corner, it is advantageous to move diagonally (i.e. add positive values to both velocity components). Such comparisons will be further explored in the extension.

The second interesting pattern that can be seen in the results is the horizontal movement on the starting track. Given the premise of the environment, one would expect the optimal move would always be to move forwards. However, if the agent already discovers the optimal policy for a given starting state, an exploratory agent could just as easily explore the other starting states until it found one that it had already seen and suddenly act greedily from then on. I suspect such events occurred occasionally for both tracks, as evidenced by optimal trajectory 1 for track 1 and track 2. Such behavior could by addressed by adding many more episodes to learn from, so that there's sufficient exploration at each starting state, or by raising the ϵ value for the agent.

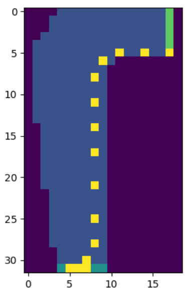
6 Extension: Human Generated Episodes

For the extension section of my homework, I wanted to explore how passing human-generated episodes to the agent would affect the speed of convergence. In order to do so, I generated ten episodes on the first track (see Appendix) and then passed them to the agent as the first ten episodes to process. The results are displayed below:

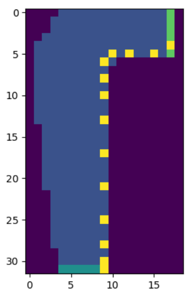
	Without human generated episodes	With human generated episodes
Episodes until convergence	103,594	38,798
Average timesteps for optimal trajectories	14.6	13.8



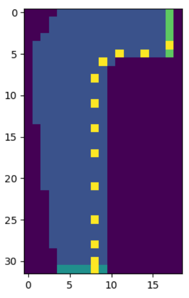
(a) Optimal Trajectory 1



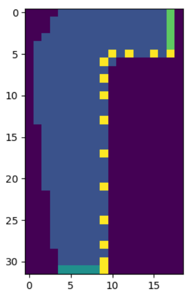
(b) Optimal Trajectory 2



(c) Optimal Trajectory 3



(d) Optimal Trajectory 4



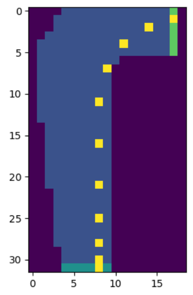
(e) Optimal Trajectory 5

As one can see, the generated optimal trajectories are almost identical to those found in the results section for track 1, but it took 73% less episodes to converge. The most significant differences between these generated trajectories and the ones learned from scratch are a much more consistent starting pattern of constantly increasing vertical velocity and then constantly decreasing it near the corner and the much sharper/more linear behavior after the corner. The first behavior can be seen very consistently across all ten generated human trajectories in the Appendix, but the second behavior appears to be novel. It is neither in the originally learned trajectories nor the human trajectories as both sets tended to use larger arcs around the corner attempting to conserve momentum. Furthermore, including the human generated episodes lowered the average timesteps required to complete the track by about 5.5% from 14.6 to 13.8, which is closer to the average of the purely human generated episodes of 11.5. This gain in performance can be attributed to the agent's learned tendency to take larger steps initially, a pattern consistently displayed in the human trials, but not in the original trajectories shown earlier. Overall, the addition of human-generated episodes to initialize the agent's learning made a significant improvement on the agent's ability to learn a successful policy.

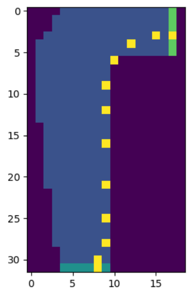
7 Conclusion

Monte Carlo methods are an effective learning method for the racetrack problem posed by exercise 5.12 of [1]. Although convergence is a difficult measure for an on-policy Monte Carlo control method, my experiments showed that even an approximate convergence metric can result in near-optimal behavior and successful completion of the task. The two ways to further improve the performance displayed in my experiments is to either increase the probability of exploration ϵ or to further refine the test for convergence, such as using a weighted average of the previous 1,000 episodes. Both methods would increase the amount of exploration done, which would further optimize the learned policy. Lastly, the addition of even ten human-generated episodes to initialize the on-policy first-visit Monte Carlo control method significantly reduced the amount of episodes until convergence, which is an important consideration for real-world applications and should actively be explored.

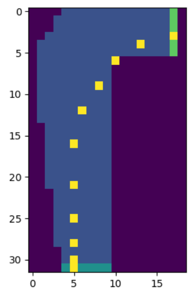
8 Appendix



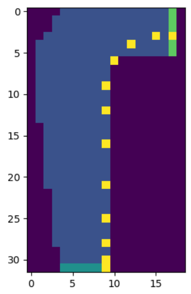
(a) Human Trajectory 1



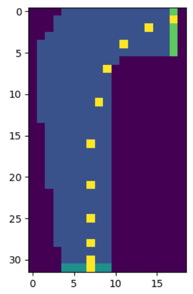
(b) Human Trajectory 2



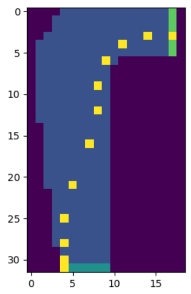
(c) Human Trajectory 3



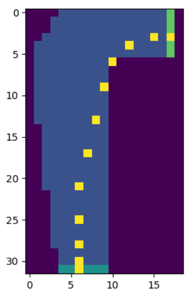
(d) Human Trajectory 4



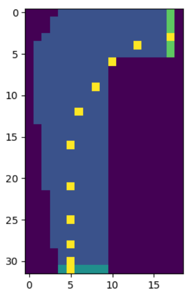
(e) Human Trajectory 5



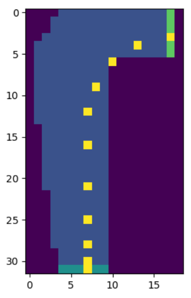
(f) Human Trajectory 6



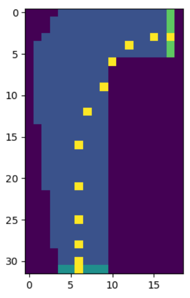
(a) Human Trajectory 7



(b) Human Trajectory 8



(c) Human Trajectory 9



(d) Human Trajectory 10

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.