COMP 138 RL: Homework 2

Mattia Danese

October 17, 2022

1 Motivation

The core of reinforcement learning is learning through interaction. That is, an agent learns something, or to do something, by observing its environment, interacting with its environment, and observing the feedback it receives from its environment. For this to work, though, the agent must have a complete model of the environment. This entails the agent knowing about the complete probability distributions of all possible transitions within the environment. So how can reinforcement learning be achieved when the agent does not have a complete model of the environment (which is fairly common for more complex environments)? This is where Monte Carlo methods come in. The basis of Monte Carlo methods is running many simulations of the agent in the environment with the purpose of accumulating a lot of data. With this observed *real* data, the agent can then estimate (and update its estimates of) the values of states and actions in the environment and formulate (and improve) some policy. After some time, these estimates will converge to their true values and thus the policy of the agent will converge to the optimal policy. In this report, the *Off-Policy Monte Carlo Control* method is used to learn how to navigate a race track and find the optimal way to get from the start-line to the finish-line.

2 Introduction

2.1 On Policy vs. Off Policy Monte Carlo

The first step in understanding the *Off-Policy Monte Carlo Control* method is knowing the important difference between on-policy and off-policy Monte Carlo methods. Agents that employ on-policy Monte Carlo methods learn and interact with the environment by using the same policy that they are trying to learn and improve. Agents that employ off-policy Monte Carlo methods, on the other hand, learn and interact with the environment using a *different* policy than the one they are trying to learn and improve. In other words, off-policy Monte Carlo methods have a *behavior* policy, the policy used to pick the agent's actions, and a *target* policy, the policy that is evaluated and improved. As such, the *behavior* policy must be a soft policy and adhere to the assumption of *coverage*: every

action possible under the *target* policy has a non-zero probability of being chosen by the *behavior* policy.

2.2 Off Policy Monte Carlo Control

The figure below showcases pseudocode for the *Off-Policy Monte Carlo Control* algorithm.

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

This method can be broken down into four simple steps:

- Initialization
- Generate an *episode* (i.e. a list of state,action,reward tuples) by following the *behavior* policy
- Once an *episode* is generated, update *target* policy by updating state-action weights and state-action values using discounted future returns
- Repeat steps 2 and 3 until policy has converged

2.3 Agent Parameters

In this experiment, the *behavior* policy of the agent will be the $\epsilon-greedy$ policy, where $\epsilon = 0.1$, and its *target* policy will be the *greedy* policy. The discounting factor γ, used in policy improvement, is set to 0.95. Additionally, the agent will generate and evaluate 1000 episodes.

2.4 Environment Parameters

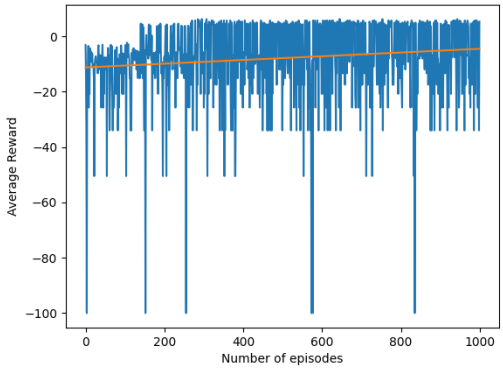
In this environment, the agent will receive a reward of -1 for every time step; this will encourage the agent to get to the finish-line as fast as possible. A *crash*

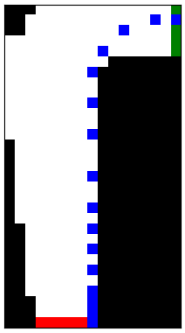
is defined as going off the racetrack entirely or going into a barrier. In such an event, the agent will receive a reward of -100 (to greatly discourage it from choosing the same trajectory) and the episode will end. Otherwise, if the agent gets to (or goes passed) the finish line, it will receive a reward of +100 and the episode will end. Additionally, the state space is defined as every combination of location and velocity, $(pos_x, pos_y, vel_x, vel_y)$ tuples, and the action space is defined as it is in the textbook,

3 Results

The figures below showcase the results of this experiment for the two racetracks mentioned in the textbook and will be further discussed and analyzed.

3.1 First Racetrack



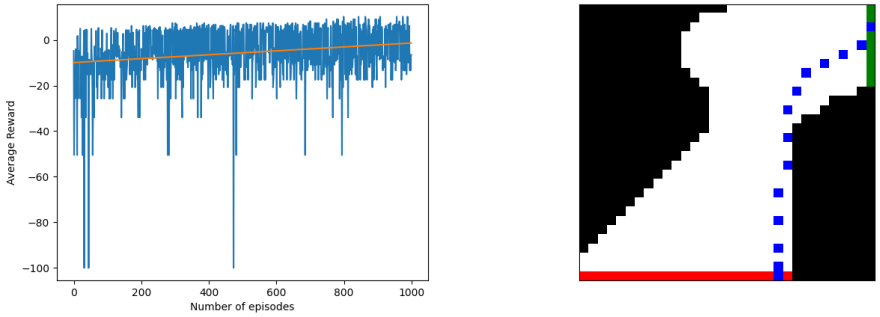


The figure on the left shows the average reward per episode as the agent generated more episodes and consequently improved its *target* policy. It is clear that the agent learned how to navigate the racetrack the more episodes it generated and evaluated. The evident lines that go down to -100 and -50 represent the agent crashing fairly early in the episode. One indication that the agent learned over time is that these very negative rewards occur less and less as the agent evaluates more and more episodes. Additionally, the orange line is the trend line for average episode reward and it clearly increases as the number of episodes increases, once again pointing to the agent improving its policy. That being said, there is still somewhat significant negative reward towards the tail-end of episodes, though this could be attributed to random chance (the agent still explores 10% of the time even in later episodes) or to the possibility that some states were not visited enough and thus their value has not converged yet.

The figure on the right shows the optimal path found by the agent, given a random starting point on the start-line. Once again, it is clear that the agent

did in fact learn which actions to take in which states in order to get to the finish line the quickest. In this optimal path, the agent picked actions that increased its vertical velocity the most, which would bring it near the finish line the fastest. Then, as soon as it could turn to the right, the agent immediately picked actions that would increase its horizontal velocity and also decrease its vertical velocity (so it would not go off the top of the racetrack). It is also important to note how the agent seems to be slowing down as it gets closer to the point where it can finally turn right. This could just be due to chance, though it can also be argued that the *bootstrapping* attribute of the Monte Carlo algorithm allowed for the agent to "know" that it would soon have to make a sharp right-turn; as such, it decided to preemptively slow down much like one would when driving a car.

3.2 Second Racetrack



The figure on the left shows the average reward per episode as the agent generated more episodes and consequently improved its *target* policy. It is clear that the agent learned how to navigate the racetrack the more episodes it generated and evaluated. Much of the same analysis of the agent's performance in the first racetrack can be said about its performance in this racetrack. The very negative rewards of -100 and -50, which once again resemble early crashes in an episode, are more frequent in earlier episodes rather than in later ones. Additionally, the orange line is again the trend line of average reward per episode and it is clearly increasing as the number of episodes increase.

The figure on the right shows the optimal path found by the agent, given a random starting point on the start-line. Once again, it is clear that the agent did in fact learn which actions to take in which states in order to get to the finish line the quickest. Similar to the first racetrack, the agent starts off by solely increasing its vertical velocity as much as it can, then slowing down as it reaches the point where it can turn right, and finally increasing its horizontal velocity and, at the same time, decreasing its vertical velocity. The reasons for this behavior are the same as those mentioned for the first racetrack. Additionally, one interesting thing to note is that the agent does not hug the black edge after

it reaches the point where it can turn right, which intuitively seems like the correct thing to do. This could be attributed to convergence not being reached, though I think a more probable explanation is that it simply is not the most optimal path. That is, for the agent to adjust its velocity in order to hug that black edge, a low vertical velocity would be needed. Such a velocity might be more costly (take more time steps) to reach than just increasing vertical velocity in order to get to the point where the agent can turn right the fastest possible and then *simultaneously* increasing horizontal velocity and decreasing vertical velocity (to make what we can observe is a 45 degree right turn).

4 Conclusion

In this report, the experiment that was explored was evaluating how an agent can learn the optimal path of a racetrack through repeated simulation and policy improvement. By using the *Off-Policy Monte Carlo Control* method, the agent was able to clearly improve its *target* policy even though it used its *behavior* policy to determine its actions. This experiment shows that it is not always necessary to have a complete model of the environment and that, given appropriate time and memory, generating organic data through real simulations of the agent in the environment may suffice for the agent to learn its goal. Special attention needs to be given, however, to the assumption of "appropriate time and memory". Some scenarios, where a complete model does not exist, are very complex with humongous state and action spaces. Given the complexity of these environments, using Monte Carlo methods may yield usable results but it could very well be impractical, from a time and memory perspective, for the agent to actually converge to the optimal policy.

5 Secondary Experiment

5.1 Introduction

One of the possible drawbacks of Monte Carlo methods may be the cut-throat nature of episodic learning. In the previous experiment conducted (and many other experiments dealing with navigation), if the agent crashes, then the episode ends and the agent gets immediately brought back to the starting position. It goes without saying that this is a fairly unforgiving procedure; the agent must not choose an action that will cause it to crash otherwise it will get reset and lose all of the progress it made thus far in the episode. Another way to deal with the agent choosing an action that will cause it to crash is to simply reset the agent back to the position it was in before choosing that "bad" action and have it pick an action again. This allows the agent to potentially have multiple attempts at picking actions for every state that it reaches and furthering its progress in the episode. However, potential drawbacks of this method is that it may severely slow down learning since, with Monte Carlo methods, the episode must end before the agent can make any action-value updates. This experiment

attempts to combine these two approaches by giving the agent one *retry* if it picks an action that will lead it to crash per episode. The goal of this experiment is to see if faster learning can be achieved than solely using the first method of dealing with agent crashes.

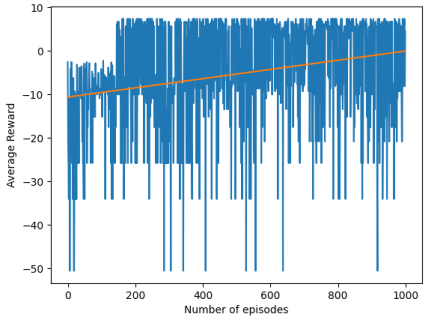
5.2 Agent and Environment Parameters

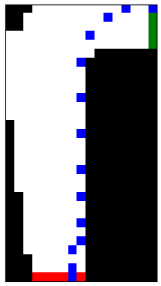
All agent and environment parameters will remain consistent with those specified in the previous experiment.

5.3 Results

The figures below showcase the results of this experiment for the two racetracks mentioned in the textbook and will be further discussed and analyzed.

5.3.1 First Racetrack with One Retry



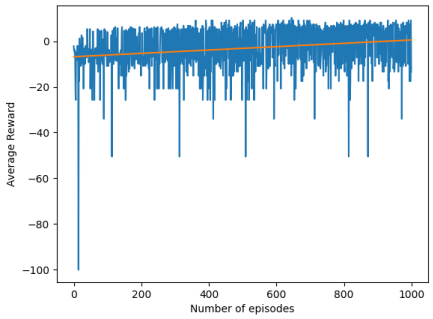


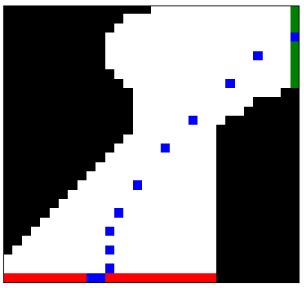
The figures above showcase the average reward per episode and optimal path found by the agents on the first race track when they were able to have one *retry* (i.e. the ability to pick another action upon picking an action leading to a crash) per episode. The majority of the average rewards in the average reward per episode graph is bounded between (roughly) 10 and -30, which is very similar to the corresponding graph in the initial experiment. However, a key difference is that the trend line is steeper for this experiment, indicating more aggressive and faster learning. Additionally, the lowest average reward (i.e the extreme) for this experiment is -50 instead of -100. This is most likely directly attributed to the agent "saving" itself from a crash by having the ability to re-pick an action. That being said, a reward of -50 has still a fairly big magnitude, so the agent most likely still crashed shortly after re-picking. Thus, it can be inferred that the agent receiving extreme negative reward and/or crashing may be mitigated if it is given more *retries* per episode.

Moving on, the optimal path found by the agents is almost identical to that found by the agents of the previous experiment. In both cases, the agents got

to and stayed to the right of the track and solely accelerated forwards until they could turn to the right in which they did so. The agents of this experiment did take one less step to reach the goal than the agents of the initial experiment, though this difference is too minimal to truly be significant. Having almost identical optimal paths is not surprising, however, since having the ability to re-pick an action intuitively would only speed up learning but would not necessarily make it *better*.

5.3.2 Second Racetrack with One Retry





The figures above showcase the average reward per episode and optimal path found by the agents on the second race track when they were able to have one *retry* (i.e. the ability to pick another action upon picking an action leading to a crash) per episode. The analysis for these results follows the same train of thought and reasoning as that for the first race track. It is clear that the trend line of the average reward per episode is shifted up relative to that of the initial experiment. This signifies that the agents most likely crashed less frequently as more crashes would bring the average of rewards down seeing as agent would receive a very negative reward. Similar to the agents' performance on the first race track, the agents garnered an average reward of -100 a lot less frequently than in the initial experiment. This most likely means the agents were able to avoid some crashes through their ability of re-picking an action.

Similar to the first race track, the optimal path found by the agents this time is very similar to that of the first experiment. In both experiments, the agents learned that the optimal behavior is to accelerate as much up and to the right as possible and then solely turn right near the top-right area of the track. The optimal path found by the agents in this experiment again took less steps than the one found in the initial experiment, 12 and 14 respectively. However, the difference in steps is again to small to be significant and may just be attributed to the difference in starting positions (starting in the middle of the starting line is ideal for accelerating in both directions at the same time).

5.4 Conclusion

It is clear that having the ability to re-pick an episode-ending action once per episode definitely helped the agent learn the optimality of actions faster than not having this ability. Being able to re-pick an action resulted in much higher overall reward averages across all episodes for both racetracks. Additionally, the agent was able to reach the goal in less steps, for both racetracks, when it had the ability to re-pick an action; though the difference in steps was fairly minimal. As such, it can be concluded that re-picking an episode-ending action once per episode aided in the *speed* in which the agent learned and not necessarily in the *quality* of learning. It will remain to be seen what the effect of adding more *retires* per episode will have, though it may be assumed it will only help with agent learning until the length of episodes begin to hinder the learning rate. All in all, this experiment has shown that allowing an agent to re-pick an action greatly improves learning.