Comp 138 RL: Homework 2

Matt McDermott

October 10, 2020

Background

Monte Carlo methods provide an attractive option for RL applications due to the fact that they do not require full knowledge of the environment in which an agent is acting. MC methods work by calculating weighted average sample returns for various state-action combinations obtained through "experiences" or sample sequences of states, actions, and rewards.

Problem Statement

The goal of this exercise was to determine the optimal policy for a "car" driving around a turn on an arbitrarily drawn racetrack. This meant that the car should move around the turn to reach a goal state as quickly as possible without going off the track. The available actions at any position and velocity state were to accelerate by (-1,0,1) unit/step in the x and y directions, meaning that there were 9 possible actions for each position-velocity combination. At each step, there was a chance that the action taken by the policy would be ignored and the car would not accelerate with probability 0.1, in the context of this problem this was referred to as a "windy" state.

Methods

The simulation was broken up into episodes, each starting with the car at zero velocity at a random point on the starting line and finishing as soon as the car reached any point on the finish zone. There was a constant reward of -1 for every timestep that the car did not reach the finish. If the car left the boundary of the track it would have both velocity components reset to zero and would be placed at a random point on the start again. For the first part of this exercise, velocity was saturated at (0,4) for x and (-4,0) for y meaning that the car could only go in some combination of up and right. The second part of this exercise removes that constraint and allows the car to achieve both positive and negative velocity in both directions.

Maps were generated through input images of size 1000 by 1000 pixels made in MS Paint. The image is discretized into a 30x30 gridworld and rgb values at each point are used to determine if that grid space is on the road, off the road, on the start or on the finish. Each simulation begins with assigning all state-action combinations with equal highly negative values. In situations such as this where two or more state-action pairs have equal value scores, the respective policy was chosen at random from the highest valued options. In the case of the beginning of the simulation, this effectively creates a fully random policy. The algorithm then follows an epsilon-greedy off-policy approach with probability ε that a random action will be chosen and probability $(1-\varepsilon)$ that the agent will follow the target policy. The simulation updates velocity states according to the acceleration from the policy, with the exception of "windy" timesteps in which the acceleration is set to zero, regardless of what the intended policy values were. Velocities are saturated if outside the (min,max) range for the simulation being run and positions are updated accordingly. If the updated position falls outside the boundaries of the road, the car is teleported back to the start and the trial continues. At the end of each timestep, state and action values are recorded

in a history array. Once the car reaches a point in the finish zone the evaluation step of the trial stops and the improvement step begins. The algorithm uses ordinary importance sampling in order to weigh how much each of the actions taken by the policy led to reaching the goal state. Starting with the first element of the history (the last state before finishing) the values of each state are updated recursively according to:

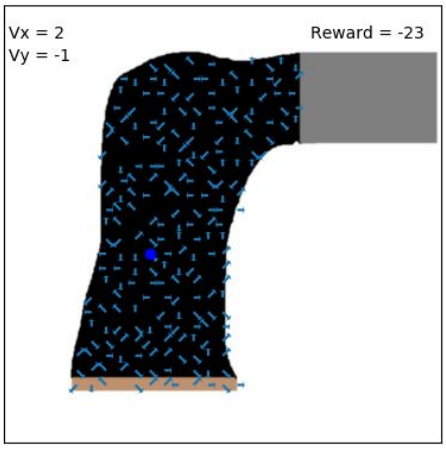
$$V(s) = \frac{\Sigma t \varepsilon \tau(s)\ \rho t{:}T(s)\ Gt}{\tau(s)}$$

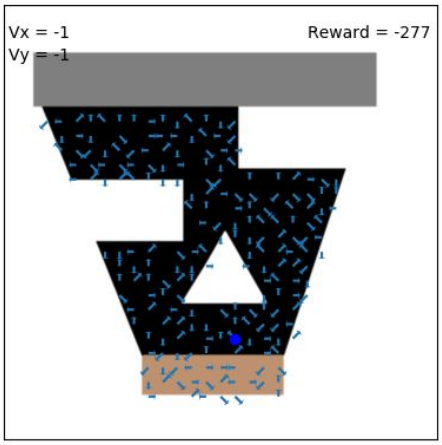
Where: $\tau(s)$ represents the timesteps where s is visited

$\rho_{t:T(s)}$ is the probability of choosing the optimal action $(1 - \varepsilon)^t$

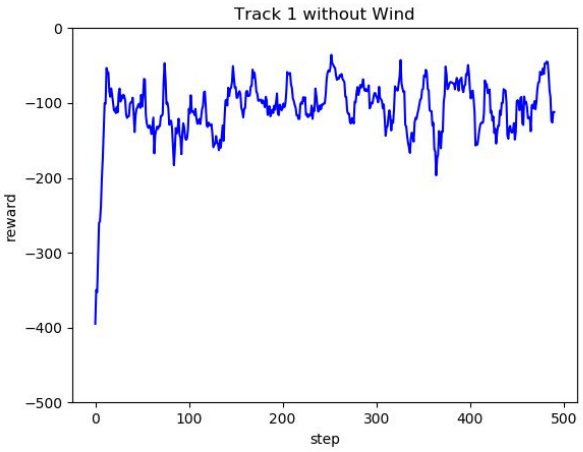
G_t is the discounted sum of rewards at t $G = \gamma G + R_{t+1}$

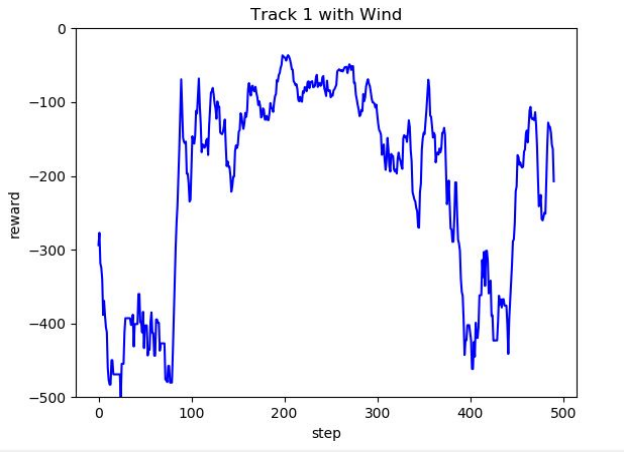
A visualization of the current policy was shown periodically to monitor the progression of the algorithm. Note that the policy represented by the arrows on the road is a function of position as well as velocity so it will update as the speed of the car changes. All trials were run with constant ε of 0.1.





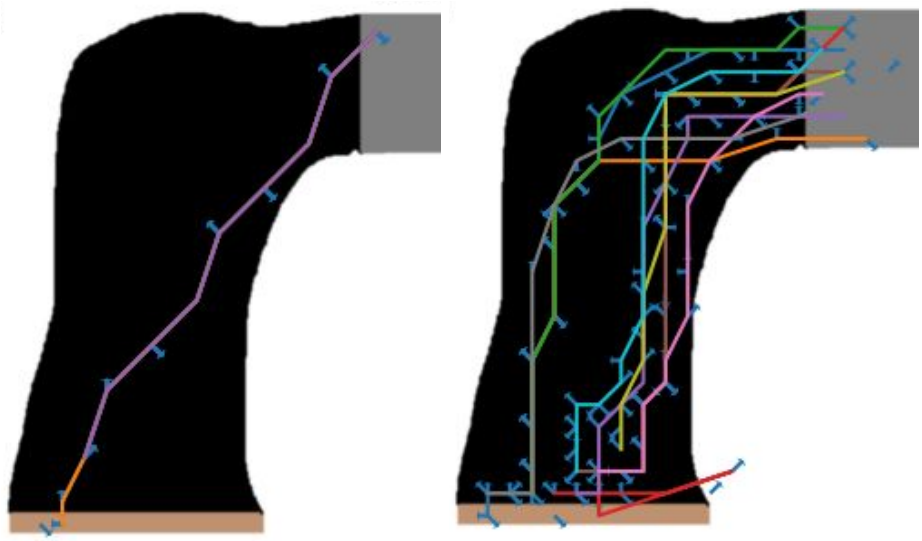
Results

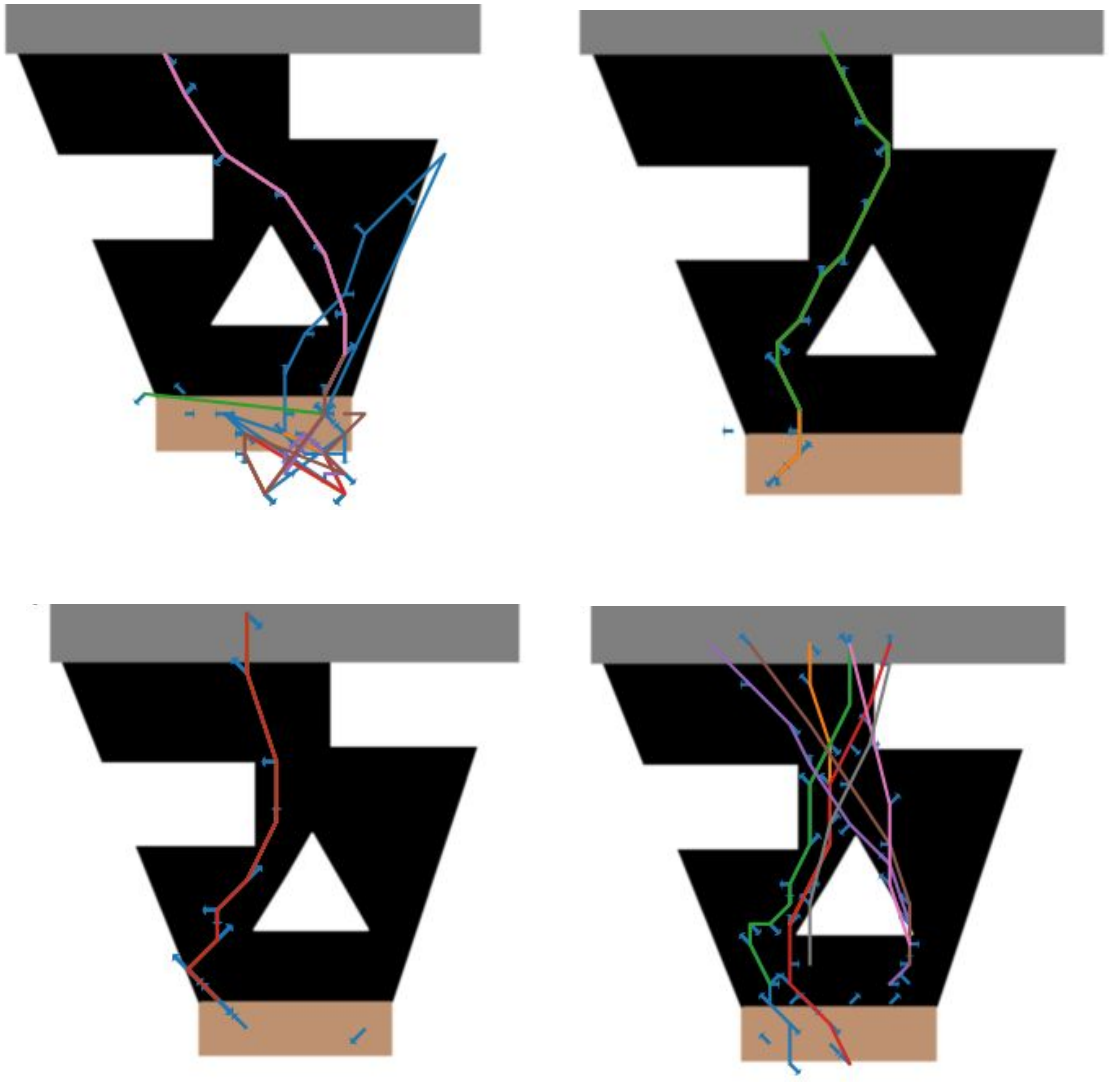




Running the simulation with Map 1 and no wind resulted in the first major jump in convergence at time = 30 while it took the simulation with wind almost 100 timesteps until any consistent performance increase. The case without wind reached a steady state error of roughly -100 while the windy case did not ever truly level off, even after 5,000+ trials. Both simulations have noticeable oscillations in their results, even after the application of a moving average filter with window size 10.

Generally, the more total timesteps a simulation took to converge, the longer each individual time step took as well. This is due to the fact that converging simulations only had to run each trial for <50 timesteps before a goal state is reached while randomly exploring movements that did not reach the finish could take upwards of hundreds of timesteps.



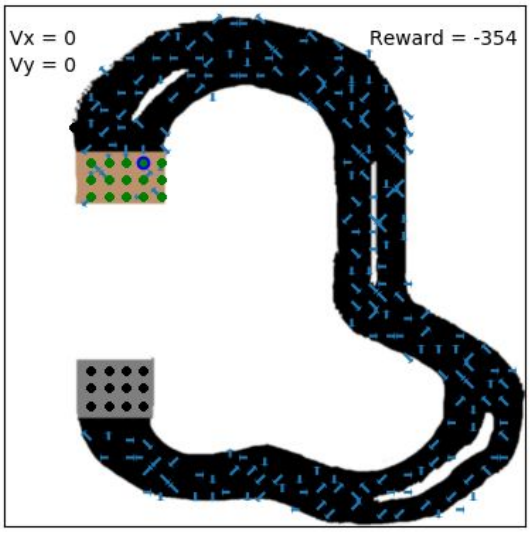


Top Left: ε = 0.01
Top Right/ Bottom Left: Result of running the simulation from the same starting point at ε = 0.1
Bottom Right: ε = 0.1

Analysis

An interesting consequence of how the rewards were distributed in this simulation can be seen in agent behavior from sub-optimal starting states. Because the penalty for falling off the edge of the map and being respawned randomly on the start is the same as any other movement action (as per the assignment instructions), there were situations where, for example, the agent figured out that it would be better off jumping into the abyss off the edge of the map and having a 50% chance of getting an optimal starting position than it would be to manually move 3 spaces to the side. This was frustrating because any deliberate attempts I made to increase punishment for falling off the map only resulted in situations where the agent would move as slowly as possible in the first problem (I added an additional penalty for getting stuck) or running in circles in the second problem.

For the second part of this assignment, the limits on the velocity of the car were lifted such that it was possible to move in both the positive and negative directions in x and y. I made a second more complicated map that had some obstacles and a split down the middle. Implementation was not as difficult as I feared and results were reasonable for the application. Similar to the right turn only case, the simulation still had difficulty finding more than one possible solution to the problem.



Using the algorithm designed for the second part of this exercise, I used a trace of Yoshi Falls; my least favorite Mario Kart map as the input image. Unfortunately, even with the most forgiving parameters for the simulation, my algorithm was unable to converge. One possible approach for this problem could be to split the map up into smaller sections and solve them individually, using non zero velocities as starting conditions and then stitching together the sub-policies into one main policy. Alternatively, it may be possible to implement some type of reward function that is based on backtracking, such that the car is penalized severely for reentering a previous section of the track.

Conclusion

For the sake of consistency, all trials were run with a constant ε value. While the value of 0.1 was determined to be useful enough in this example for the number of trials I was running, there is certainly room for improvement. At the beginning of a trial, having a large ε can be useful in encouraging the agent to search around more, however, after the target policy begins to take shape, even a 1 in 100 chance of taking random turns can be enough to cause more harm than good. One potential solution to this problem could be set $\varepsilon = \frac{1}{\sqrt{run + 1}}$ so that the early trials have sufficient randomization while later trials gradually become more and more deterministic. Many other parameters such as max trial length, number of trials, discount factor and initial value for state-action pairs can certainly be tuned in order to improve performance. I would be interested in learning what methods exist for picking smarter initial values and tuning performance mid simulation.

It should be noted that when driving a car in real life, the x and y components of velocity are not entirely separate. If a more accurate simulation is desired, it would be important to take into account the coupling between velocity and acceleration in the two directions, as well as other physical limitations such as minimum turning radius, inconsistent acceleration (it is much easier to go from 0-20mph than 20-40mph), and the fact that a real world car can not be well approximated by a point particle.

References

Sutton, Richard; Barto, Andrew (1998), *Reinforcement Learning*, MIT Press, ISBN 978-0-262-19398-6, archived from the original on 2013-12-11.