Part 2 - Project report

Årne, Runar and Lars

March 2, 2024

1 Introduction

In this project we will be using Monte Carlo and Temporal Difference to try learning different policies within a mountainous environment. We will learn and compare different algorithms, as well as visualizing their trajectories.

2 Results

We are given several trajectories from a proprietary policy π. After performing MC evaluation for $v_\pi(s)$ we have the resulting state value function visualized as a heatmap (Figure 1).

From the plot, we can see that no trajectory got more than halfway down the mountain. As every value on the right is 0, it indicates they are unexplored. This will be a problem as we can not compute an evaluation for the whole map, meaning we again can not calculate an optimal policy for the whole map.

However, for this task we can perform MC improvement, as the only thing we need for that is the state-action evaluation $q_\pi(a, s)$, which we also have computed. However, the new policy cannot possibly be optimal as we have not explored enough of the map.
However, we can not do MC control. To do MC control we need a way to generate new trajectories, which we cannot do here.

We have now received the robot used for creating the trajectory, as well as the simulator of the environment. How do we choose a starting policy for a new robot?

We already have computed the state-action evaluation $q_\pi(a, s)$ from the same environment as the one we've been given. However, as that evaluation is incomplete (only reached halfway) there would be no gain from using it to initialize our policy. That is because, as no trajectory reached the end, all routes will become more and more negative the further down the map we come. And thus, using this as out starting point will lead to our policy wanting to move left, back towards the start (as here, the values are less negative).
Therefore, it would be best to randomize the starting policy π and "restart".

After "restarting" we perform MC control and run MC for 100000 episodes with $\epsilon = 0.7$, $\alpha_{\text{decay}} = 0.99$ and $\gamma = 0.99$. On Figure 2 we can see the resulting trajectory as well as state value function $v_\pi(s)$ visualized. Did we manage to learn an optimal policy?
No, we did not. As we can see from the plot above, we did not even manage to explore an eighth of the map. It is obvious that the challenge with Monte Carlo here is that we never terminate. When performing random moves in an environment as large as this, it is very difficult to reach the end. This will lead to the robot going back and fourth a lot, and thus never exploring further areas. From the plot on the left, we see a tiny red line, which is the robot walking spinning back and forth in the starting state.

We now try to use SARSA instead to learn an optimal policy. Using the same hyperparameters, and stepsize $\alpha = 0.15$ we get the resulting policy π_{sarsa} for SARSA(0). It's trajectory is visualized in Figure 3.

As we can see, SARSA(0) learned pretty well! To compare MC and SARSA we need to use an appropriate metric.
Here, the most meaningful and appropriate metric to compare the two learning processes (MC and SARSA(0)) would

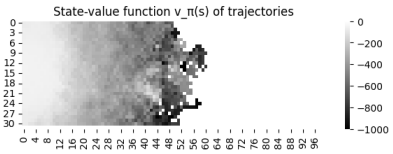

Figure 1: Heatmap of $v_\pi(s)$.

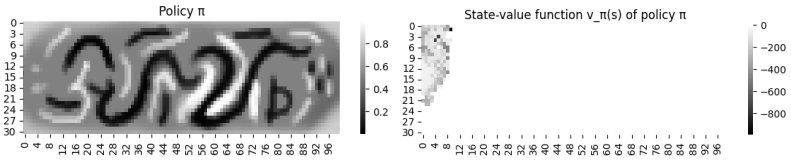

Figure 2: Trajectory and state value function of MC policy π.

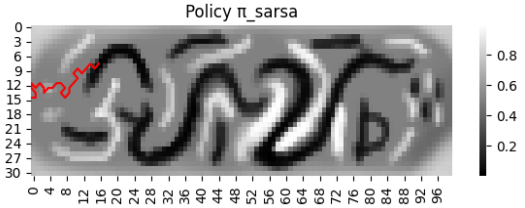

Figure 3: Trajectory of π_{sarsa}.

probably be a measure which takes into account how well it has started learning moving towards the goal. As neither policy reached the goal, we can choose to use distance from the target at the end as out measure. This would mean the policy which ended closest is appropriately chosen as the best.

As we can see, MC ended with a final distance of 99, whilst SARSA(0) ended with a final distance of 84. Therefore, we measure SARSA(0) to be the better one for this task.

We also implement SARSA(n) for values $n = 3$, $n = 5$ and $n = 10$, as well as performing Q-learning. The resulting policy trajectories are plotted in Figure 4.

From previously, we had that the best policy was SARSA(0), with a final distance of 84. Here we have that SARSA(5) has 97, whilst both SARSA(20) had 96 and SARSA(50) had 98. This would mean that, again, SARSA(0) is the best performing policy.

This makes sense, as the higher n we have for SARSA(n), the closer we get to MC. As MC apparently did not perform very well on this task, we should not expect SARSA(n) to do so either. (At least not better than SARSA(0)).

Comparing Q-learning to the other SARSA policies we see that it did very good in comparison to SARSA(n)! It ended with a final distance of 89, which is way closer to SARSA(0)'s 84. However, as SARSA(0) still had the lowest final distance, this is our current optimal policy.

Inspecting the plots, we see that both SARSA(0) and Q-learning were the only algorithms to learn a reasonable path away from the start. We can see that they learned different paths, but Q-learning seemed to get stuck in a different place, compared to SARSA(0).

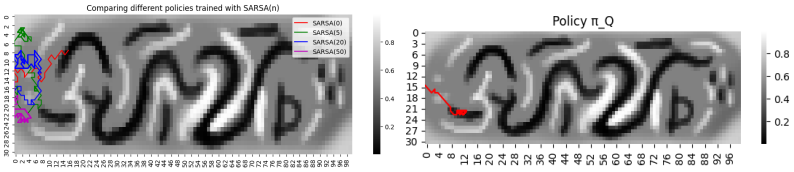

Figure 4: Trajectory of SARSA(n) and Q-learning.