COMP 138 RL: Homework 3

Mattia Danese

November 7, 2022

1 Motivation

A very important sector of reinforcement learning methods is Temporal-Difference (TD) Learning methods. In simple terms, TD methods combine model-free learning solely from organic data (i.e. Monte Carlo learning methods) and bootstrapping estimates in order to tweak previously used estimates (i.e. Dynamic Programming learning methods). As such, TD methods may have certain critical advantages over MC and DP methods. To start, a state-action pair is updated just one time-step after it was reached in TD methods; however, all state-action pairs in MC methods get updated at the end of the episode. This is not an inherent advantage over MC methods, though this aspect of TD methods can definitely become advantageous if it is desired to update state-action pairs continuously throughout the episode. Moreover, some MC methods overlook rewards of when the agents choose experimental actions which can severely slow down the learning of the agent. TD methods, on the other hand, are less vulnerable to this because the agent updates (i.e. learns) at every time-step, no matter the action taken. Lastly, TD methods do not need a complete model of the environment unlike DP methods. This is a fairly important advantage as getting a complete model of an environment, especially for "real life" scenarios that must take into account properties of physics or have continuous rather than discrete properties, may be completely impractical from a logistical and storage-based perspective. With TD methods, such scenarios can be undertaken; though, the same cannot be said for DP methods. In this report, the *Sarsa* and *Q-Learning* TD methods are used to learn how to get from one side of a board to another without going off the board or falling into a "cliff" on the lower edge of the board.

2 Introduction

2.1 Sarsa

Sarsa is an on-policy TD control method because it uses the same policy it learns to interact with the environment. In simple terms, the *Sarsa* algorithm is as follows:

- Initialize current state and pick current action from policy
- Until the terminal state is reached:
  - Take current action and observe the reward and the next state
  - Pick next action from next state using policy
  - Update current state-action pair
  - Set current state/action to next state/action

Sarsa gets its name due to its update step which uses the current state S, the current action A, the observed reward R, the next state *S'*, and the next action *A'*. *Sarsa* updates its state-action values by taking the current state-action value and adding it to the scaled (by the step-size) sum of the current reward and the difference between the discounted value of the next state-action pair and the value of the current state-action pair. How *Sarsa* picks the next action from the policy is very important and is the deciding factor if *Sarsa* will converge or not. If ϵ-soft policies are used in order to determine the next action given some state, then *Sarsa* is guaranteed to converge to both an optimal policy and action-value function (assuming all state-action pairs are visited an infinite number of times). Below is more detailed pseudocode for *Sarsa*:

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

2.2 Q-Learning

Q-Learning is an off-policy TD control method because it uses a ϵ-soft policy to pick actions but a completely greedy policy when updating state-action values. In simplest terms, the *Q-Learning* algorithm is as follows:

- Initialize current state
- Until the terminal state is reached:
  - Pick current action from current state using policy
  - Take current action and observe the reward and the next state
  - Update current state-action pair

– Set current state to next state

Evidently, the general procedure for *Sarsa* and *Q-Learning* are quite similar; though, there is one very importance difference: how the current state-action value gets updated. In contrast to *Sarsa*, the update in *Q-Learning* is the difference between the discounted *max* value of all possible next state-action pairs and the value of the current state-action pair. As such, *Q-Learning* will always base its error term on the current optimal next state-action pair while *Sarsa* will do the same most of the time but, with probability (at most) ϵ, has a chance of basing its error term on a current sub-optimal next state-action pair. Below is more detailed pseudocode for *Sarsa*:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

2.3 Agent Parameters

In this experiment, action selection for both algorithms will use ϵ-greedy policies where $\epsilon = 0.1$. The discounting factor, γ, and step-size, α, used in updating state-action values are set to 0.95 and 0.5, respectively. Usually, α is set to a fairly small value, such as 0.1 or less, to prevent the agent from making too harsh updates that would "skip" over the true state-action values in an oscillating manner. However, since this environment will be simple and stationary, having a higher α can boost learning rate since there is much less risk of making updates, big or small, that seem correct at one point but prove to be wrong sometime in the future. Additionally, the agent will perform 100 episodes for each algorithm.

2.4 Environment Parameters

This environment, known as the *Cliff Walking* environment, is a grid-world where the agent starts in the lower-left hand corner and wants to learn how to get to the goal in the lower-right hand corner. In between the start position and the goal, there is a *cliff* that runs along the lower edge of the grid-world. The agent will receive a reward of -1 for every time step; this will encourage the agent to get to the goal as fast as possible. If the agents picks an action that will take if off the grid-world, the action is not taken though a reward of

-1 is still given. If the agent goes into the *cliff*, then a reward of -100 is given (to greatly discourage it from choosing the same trajectory) and the agent is immediately repositioned back to the starting position. If the agent reaches the goal, no reward is given (i.e. $R_T = 0$). Lastly, the state space is defined as every combination of location on the grid-world and actions, (pos_x, pos_y, a) tuples, and the action space is defined as it is in the textbook.

3 Results

The figures below showcase the results of the *Sarsa* vs. *Q-Learning* Cliff Walking experiment. The results will be further discussed and analyzed.

3.1 Sarsa vs. Q-Learning

Figure 1: The figure above has been bounded between [-100, 20] on the y-axis. The initial negative spikes for *Sarsa* and *Q-Learning* go down to -2000 and -1750, respectively.

The first thing to note is that, after just about 20 episodes, both *Sarsa* and *Q-Learning* converge; in other words, both algorithms find their respective optimal policies. It is clear that *Sarsa* had much "smoother" learning than *Q-Learning* seeing as the graph of *Sarsa* has a relatively curved structure and that of *Q-Learning* has a lot more jagged spikes. That being said, *Q-Learning* found the optimal path faster since the tops of the initial spikes of its graph minimize the running average reward before the graph of *Sarsa* does. This is due to the

nature of both algorithms, more specifically their update rules. As previously stated, the error term of the update rule of *Sarsa* takes into account the state-action value derived from the ϵ-greedy policy while that of *Q-Learning* strictly considers the state-action value that maximizes the next state. Additionally, it should be stated that the true optimal path in this environment, that is the path that gets from the start to the goal with the least amount of negative reward, is the path right alongside the cliff. As such, *Sarsa* takes more time to find the optimal path while *Q-Learning* occasionally exhibits high negative reward because the former is trying to learn a policy while still exploring, in which case getting farther away from the cliff to minimize the probability of falling in is the optimal behavior, and the latter acts completely greedily, in which case staying adjacent to the cliff is the optimal behavior even though this increases the probability of falling into the cliff. Though the difference between *Sarsa* and *Q-Learning* is definitely not this black-and-white, for the sake of conceptualizing both algorithms, it may be appropriate to label *Sarsa* as a more "slow and steady" algorithm and *Q-Learning* as a more "fast-and-risk(er)" algorithm. This is due to the fact that *Sarsa* shies away from positions closer to the cliff (even though they are truly optimal) in order to reduce the chance of getting high negative reward by falling in the cliff while *Q-Learning* disregards this added risk and instead follows the truly optimal path. This also helps to potentially explain the numerous *Q-Learning* post-convergence divots in its graph. Even though, *Q-Learning* converged and found the optimal path, there is still the relatively good chance it falls into the cliff since actions are taken using the ϵ-greedy policy. *Sarsa*, on the other hand, does not have this problem since its optimal path is the path farthest from the cliff. Thus, when following its optimal path, *Sarsa* has an incredibly tiny probability of falling into the cliff due to ϵ-greedy exploration.

3.2 Optimal Policies

Below are the optimal paths found for both *Sarsa* and *Q-Learning*. It is clear that *Sarsa* found the safest path, where "safest" is defined as having the smallest chances of falling into the cliff, whereas *Q-Learning* found the true optimal path even though this path has the greatest chances of falling into the cliff. Analysis of the optimal paths found and reasoning for the differences are given the previous subsection.

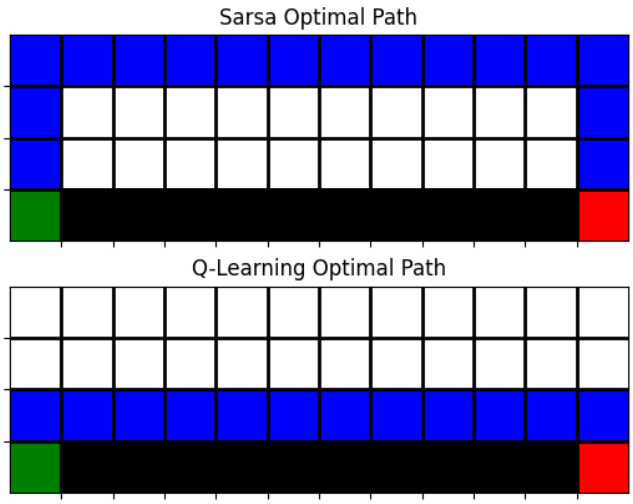

Figure 2: The figure above employs the following color scheme: Green tiles represent the Start State, Red tiles represent the Goal State, White tiles represent Valid States, Black tiles represent the Cliff, and Blue tiles represent the Optimal Path Found.

4 Conclusion

In this report, the experiment that was explored was evaluating the behaviors of *Sarsa* and *Q-Learning* in the Cliff Walking environment. By using the *Sarsa* method, the agent was able to find a path to the goal that minimized the chances of falling into the cliff, and therefore receiving a very high negative reward, at the cost of an ever so lightly higher negative reward. The *Q-Learning* method, on the other hand, found the truly optimal path to the goal, where reward is completely minimized for the average case, at the cost of occasionally falling into the cliff during exploration. Thus, *Q-Learning* proved to be best if getting to the optimal policy in the quickest manner, regardless of potentially garnering high negative reward, is the intent of the agent; otherwise, *Sarsa* is best if a safer exploration towards a more cautious path is the intent of the agent.

5 Additional Question: Non-Stationary Environment

5.1 Introduction

The previous experiment showcased the exploration and convergence differences between *Sarsa* and *Q-Learning* in the Cliff Walking environment. One key assumption of the Cliff Walking environment is that the environment is *stationary*. A *stationary* environment is one where its states, actions, and probability distributions remain unchanged. Such an environment might aid, or rather not impede, an agent with its learning, allowing it to find an optimal policy faster. This begs the question of how *Sarsa* and *Q-Learning* will behave in an *non-stationary* environment. As such, the additional experiment that will be explored is analyzing *Sarsa* and *Q-Learning* in an non-stationary Cliff Walking environment.

5.2 Agent Parameters

The agent will have the same parameters (ϵ, γ, and α) as the previous experiment. One important thing to note is that the step-size, α, is usually set to a fairly small value, especially in non-stationary environments, as this helps the agent not make relatively big (over-) corrections which may slow down learning. However, for the sake of having just one independent variable (the non-stationary environment), the step-size will remain at 0.50. Additionally, the agent will train on the stationary environment before moving to the non-stationary environment. In other words, the agent will first repeat the previous experiment exactly and then will use the policy it converged to in the non-stationary environment.

5.3 Environment Parameters

The environment will be the exact same grid-world as the previous experiment. The reward function, action-space, and state-space will also be the same as those from the previous experiment. To make the environment non-stationary, the *cliff* will move to a different location every 100 episodes out of a total of 1000 episodes. More specifically, there are 4 specific cliffs that the environment will randomly switch to: a cliff on the bottom row (like the previous experiment), a cliff on the second row from the bottom, a cliff on the second row from the top, and a cliff on the top row. It is also enforced that the *cliff* being switched to must be different than the current *cliff*.

5.4 Results

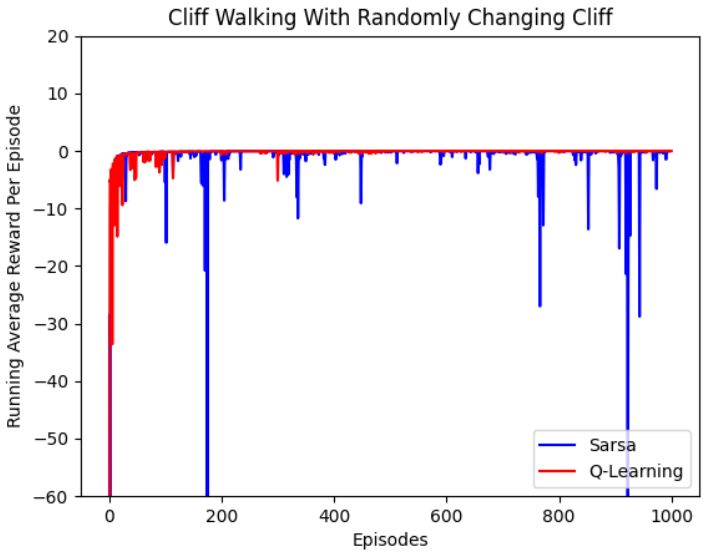

Figure 3: The figure above has been bounded between [-60, 20] on the y-axis.

It is evident that using the policy found in the previous experiment helped tremendously in a non-stationary environment. Even with the position of the *cliff* changing, both *Sarsa* and *Q-Learning* minimized their reward for the vast majority of episodes. The great negative spikes for both algorithms can be attributed to the *cliff* position changing and being in the optimal path found from each algorithm. Thus, if the agent follows its optimal path, it would be led straight into the *cliff*. When this occured, both algorithms adjusted their optimal paths fairly quickly since the spikes are relatively thin (thick spikes would signify numerous consecutive episodes where the agent fell into the *cliff*). It is also clear that *Q-Learning* handled the changing *cliff* position much better than *Sarsa* as it only has spikes of greater negative reward in the earlier episodes while *Sarsa* observed great negative reward every time the *cliff* position changed. This could be due to *Sarsa* trying to learn a policy which corresponds to just one *cliff* position; as such, when the *cliff* positions changes, the policy will most likely be very wrong. *Q-Learning*, on the other hand, only considers the best next action in its update rule. Therefore, as soon as the *cliff* position changes, the current best next action will most likely not be optimal anymore and, when it is taken, a high negative reward is observed. This immediately taints the previous next best action which causes it to no longer be picked.

5.5 Conclusion

In this experiment, the behavior of the *Sarsa* and *Q-Learning* methods in a non-stationary Cliff Walking environment were observed and analyzed. Both methods were able to get to the goal state and minimize reward the vast majority of the time. That being said, *Q-Learning* definitely performed better than *Sarsa*. This could be attributed to the nature of both algorithms, *Sarsa* learns a policy while *Q-Learning* acts greedily. Moreover, it makes sense that *Sarsa* performed worse because the policy it learns is environment specific, so when the environment changes the polcy will have to be unlearned and a new policy will have to replace it. This is not the case with *Q-Learning* as acting greedily allows for next best actions to either be disregarded (when they are no longer optimal) or discovered.