Reinforcement Learning

Nicky Acheila i6202817,
Marina Sofia Garcia Perez i6179915

November 2019

1 Introduction

In this assignment we trained an agent using reinforcement learning to solve the 'Mountain car' benchmark problem, Figure 1. We used the library 'gym' from OpenAI which provides a built-in code environment for this specific problem. In general this environment makes the implementation rather simple, once someone gets familiarized with gym.

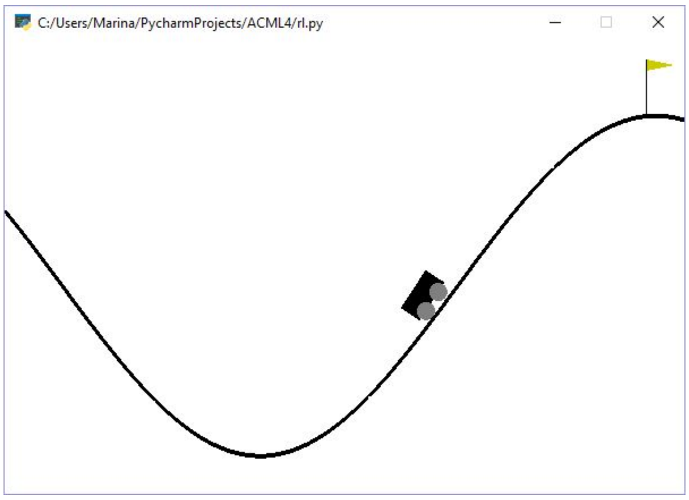


Figure 1: Mountain car graphics by the gym environment

2 Reinforcement Learning with Q-Learning

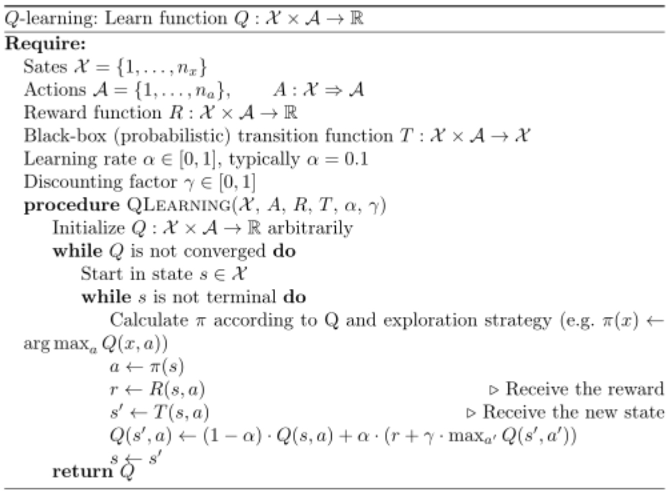
In terms of RL algorithms, we chose to use **Q-learning**, an offline training method that is powerful and straight-forward to implement, because we have an environment that can be modelled with few states and low costs, thus it provides a good search of the solution space in that setting.

For the **modeling** of the states we examine **two different scenarios**. First, we implement the classic *state=(position,velocity)* where we divide the problem space into m position-intervals and n velocity-intervals so we end up with mxn states and an $mxnx3$ Q-values table.

In the second scenario the states are only dependent on the velocity so *state=(range of current velocity)*, meaning that the state space consists of n velocity segments and the Q-values table's dimensions are $nx3$. This modeling comes from the intuition that the objective of the agent can be simplified to just acquiring a large velocity.

3 Parameters & Visualisation

For the *state=velocity* modeling, in 124 generations we obtain an agent that reaches the goal in 133 time-steps, while the initial run took 4887 iterations. Without using the visualization graphics,



```
$Q$-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$
Require:
  Sates $\mathcal{X} = \{1, \dots, n_x\}$
  Actions $\mathcal{A} = \{1, \dots, n_a\}$,        $A : \mathcal{X} \Rightarrow \mathcal{A}$
  Reward function $R : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$
  Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$
  Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
  Discounting factor $\gamma \in [0, 1]$
  procedure QLearning($\mathcal{X}$, $A$, $R$, $T$, $\alpha$, $\gamma$)
    Initialize $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ arbitrarily
    while $Q$ is not converged do
      Start in state $s \in \mathcal{X}$
      while $s$ is not terminal do
        Calculate $\pi$ according to Q and exploration strategy (e.g. $\pi(x) \leftarrow$
  $\arg\max_a Q(x, a)$)
        $a \leftarrow \pi(s)$
        $r \leftarrow R(s, a)$                          ▷ Receive the reward
        $s' \leftarrow T(s, a)$                         ▷ Receive the new state
        $Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
        $s \leftarrow s'$
    return $Q$
```

Figure 2: Q-learning pseudo-code

training is almost instantaneous. The output policy seems to depend on the initialization. The following results were obtained with learning rate = 0.1 and discount factor gamma= 0.9

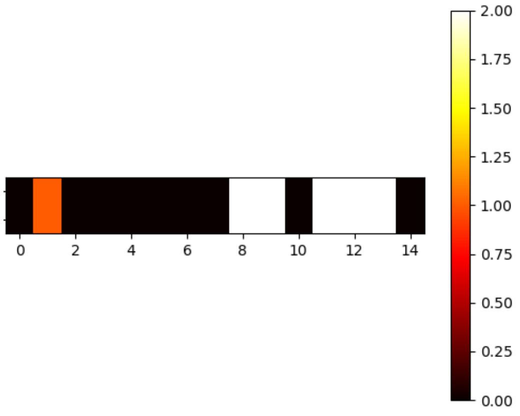


Figure 3: Policy value for state=velocities, disc_fac=0.9. 0 is full left, 1 is do nothing and 2 is full right

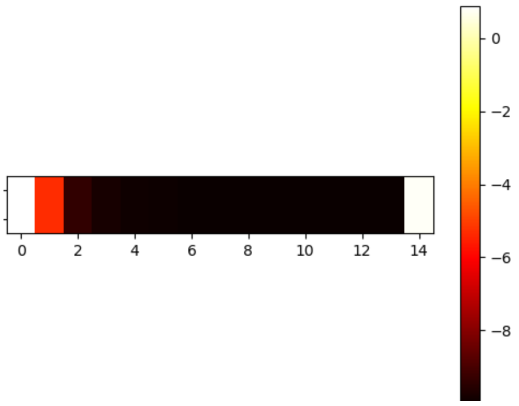


Figure 4: Q-values for each state=velocities, disc_fac=0.9

In Figure 3 we can see how the left side has a strong dominance of black, and the right side is mostly white. This means that the car has learned to accelerate to the left when it's going to the left and vice versa.

Figure 4 shows the Q-values of each state policy, i.e. the maximum Q-value of each state. We can clearly see how the higher values correspond to high velocities and how it tries to propagate (second, red, state) without much success as the reward for reaching the goal is much smaller than the penalty for the time steps.

We can try to help the algorithm by increasing the discount factor to 1, Figures 6 and 5. The final result is better in this case (only 89 time-steps) but it took more generations to achieve (352). It still run for less than a minute. It is interesting to see how the policy in the right side is very white dominated, while the left side is not actually equally black. This is probably caused by the asymmetrical environment.

Next we dive into the results of the *state=(position,velocity)* space model. In this modelling we have one extra parameter which is the segmentation of the position space. In Figure 7 the optimal policy heatmap is presented in 6 different tests with different discount factors, in a $40x40$ state space over 500 episodes and a 0.1 learning rate. The dark colors indicate low Q-values of

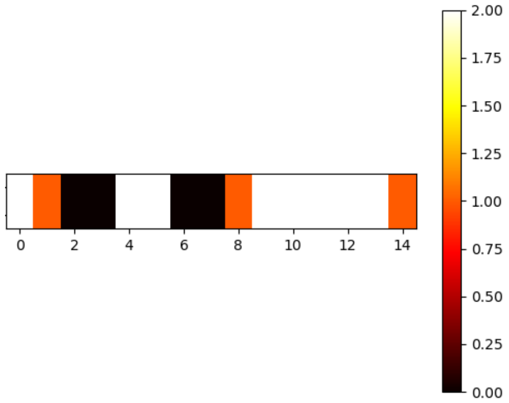


Figure 5: Policy value for state=velocities, disc_fac=1. 0 is full left, 1 is do nothing and 2 is full right

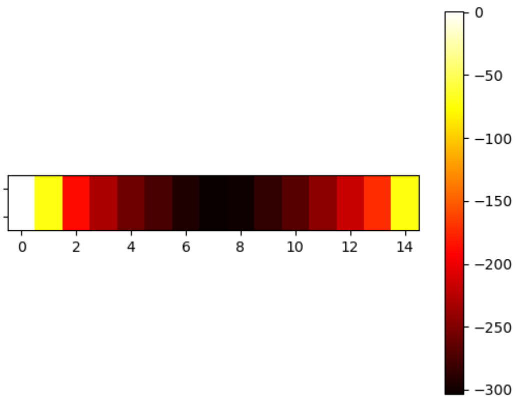


Figure 6: Q-values for each state=velocities, disc_fac=1

different scales per picture, so with the larger discount factors ($>= 0.9$) the agent does a better job of assessing the environment. Similar results were encountered when tempering with the learning rate when increasing from 0.1 resulted in a darker color dominance over the state value map. The optimal policies of a $20x10$ and a $40X40$ space with gamma=0.9 are presented in Figures 8 **??** one dominant trend that could be concluded is looking in the lower left part of the maps with positive velocity and negative position the policy indicates throttling right to take advantage of the downhill slide. The two figures show different detail due to the amount of states so the more states the more thorough modelling we get.

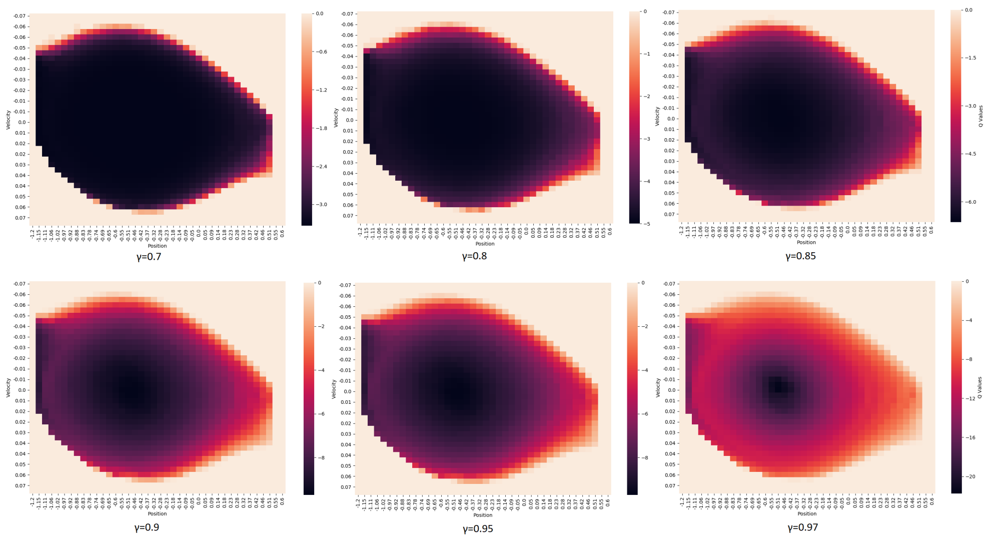


Figure 7: Heatmaps of the max Q values with different discount factors

Figure 10 presents an overview of the iterations until finish over 500 episodes. For this experiment the parameters where gamma=0.9 and learning rate=0.1, while for visualization purposes iterations over 4000 are omitted (although in initial episodes we usually get over 15000 time-steps). One interesting take-out from the results is that the algorithm is not always stable, and it also relates to the number of states and the ratio of position and velocity states. With $40x40$ states we get the most stable result although still fluctuations are rather regular. Overall, with the larger state-spaces, after sufficient training we manage to complete the episodes in less than 200 time-steps.

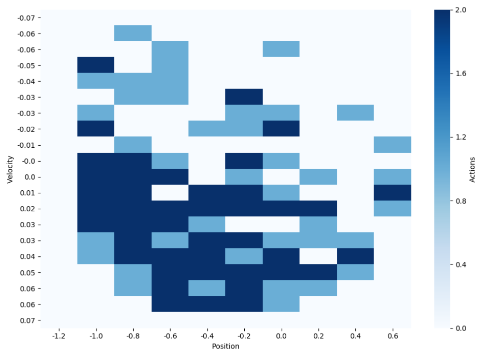


Figure 8: Optimal policy after 500 episodes 20x10 states (left)

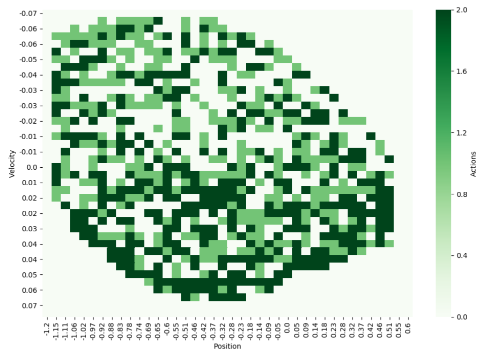


Figure 9: Optimal policy after 500 episodes 40x40 states (right)

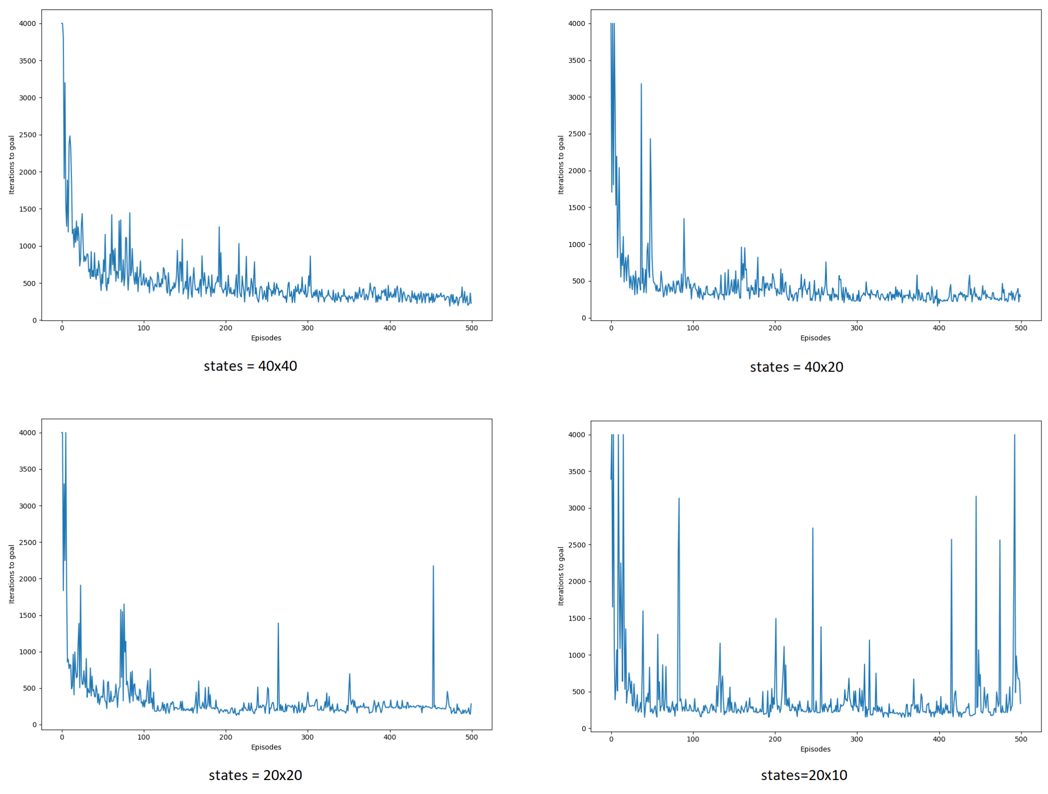


Figure 10: Iterations over training with mxn states, where m refers to # of velocity intervals and n to # of position intervals