Maastricht University

Advanced concept in machine learning
Assignment 3

Authors
Luca Brugaletta
Glauco Lorenzut

Supervisor
Kurt Driessens

November 24, 2020

1 Introduction

This assignment is based on reinforcement learning and involves the implementation of a car agent able to learn how to drive a hill.

2 Reinforcement learning

Reinforcement learning is the problem faced by an agent that must learn behaviour through trial-and-error interactions with a dynamic environment. Reinforcement learning is primarily concerned with how to obtain the optimal policy when such a model is not known in advance. The agent must interact with its environment directly to obtain information which, through an appropriate algorithm, can be processed to obtain an optimal policy[1]. There are two strategies for solving reinforcement-learning problems.

- **Model-Free:** learns implicitly and adjust the estimated value of a state based on the immediate reward and the estimated value of the next state (our project).
- **Model-Based:** has an agent that tries to understand the world and create a model to represent it.

3 Implementation

In this section are presented the choices made for our implementation. The program is realized in **Python 3.7** with the use of **OpenAI Gym** library. This library provides different environments, including the one used in this assignment **MountainCar-v0**. We implemented the **Q-learning algorithm** to solve this reinforcement-learning problem.

3.1 Q-Learning

Our Q-learning implementation uses 2-dimensional vectors to represent the **position** and the **speed** of the simulated car. Moreover, the Q-table contains another value for the action. The resulting **table Q(P, S, A)** is initialized with random values in [-2,0[. While the vector **a** has always the same dimension (3) that represents the 3 possible actions (full throttle ahead, full throttle reversed or no throttle at all), the position and speed dimension of the Q table has been tested with different shapes, this experiment is shown in section 4.3. After the initialization, the algorithm has to choose an action. This action can be the best action based on the exploration of the current state or randomly-generated, according to an **epsilon** value probability. The epsilon defined in our implementation decay over time, this decaying stops when half of the defined epochs have been completed. When the action is chosen, the Q-learning algorithm performs an observation of state and reward obtained. Then, the Q values get updated and this whole process is repeated till convergence.

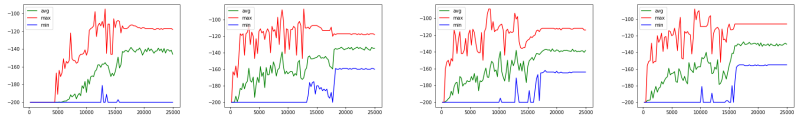
4 Experiments & results

4.1 Epochs

The first experiment took into account the number of epochs. The study was not too useful as adopting a limited number of epochs leads to too few insights. To get a clearer overview of the subsequent tests performed, we set the number of epochs at 25,000.

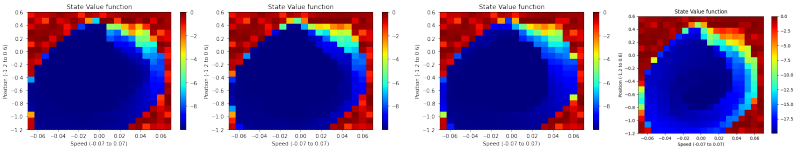
4.2 Number of steps

The second experiment focused on the number of steps the car took within the Gym environment. The default is 200, we tried running our tests using 100, 200, 500 and 1000. When testing the steps, the best solution was using the default value of 200 steps, thanks to our observations with 100 steps the car was reaching the goal for the first time after 4600 epochs and only for 15000 out of 25000 epochs. Using 200 steps resulted in the best option, the first goal was reached after 410 epochs and almost 22000 out of 2500 epochs reached the goal. Using 500 and 1000 steps worsened this result.It took 200 more epochs to reach the goal for the first time and in 21000 cases the algorithm was able to reach the goal.



(a) steps = 100 (b) steps = 200 (c) steps = 500 (d) steps = 1000

Figure 1: Number of steps



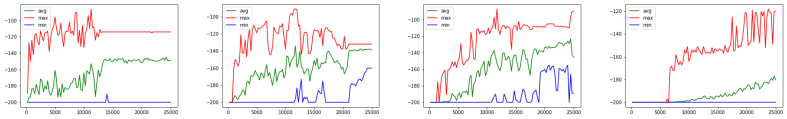
(a) steps = 100 (b) steps = 200 (c) steps = 500 (d) steps = 1000

Figure 2: Heatmap Number of steps

4.3 Q-Table size

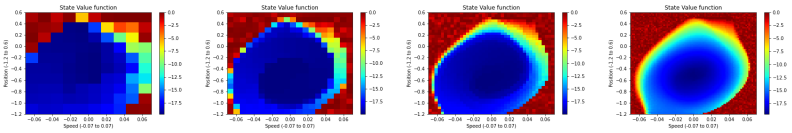
The Q-table experiment aimed to increase or decrease the size of the table to ensure greater granularity of the results. In fact, while in the Gym environment the car moves in a continuous space, in the Q learning algorithm the environment

is discretized. So we have created the Q-table which in the first two dimensions divides the environment into a set of **N x N observations**. The sizes tested were with an increasing **N: 10, 20, 40, 80**. From (Figure 3) we can see that using a size of 10 allows the algorithm to converge faster than the other sizes. But, taking a look at the heatmap (Figure 3 and Figure 4) we can notice that the one that performs better, and it is more granular, is the one with size equals to 40. Which is able to converge and to reach a better score.



(a) size = 10 (b) size = 20 (c) size = 40 (d) size = 80

Figure 3: Q-table size

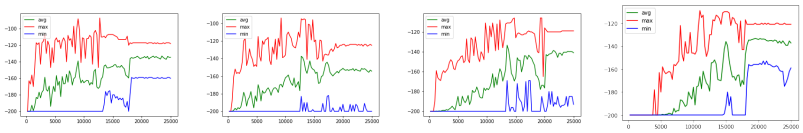


(a) size = 10 (b) size = 20 (c) size = 40 (d) size = 80

Figure 4: heatmap Q-table size

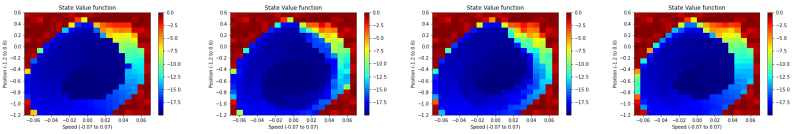
4.4 Epsilon

When we use reinforcement learning and we store a reward in one of the columns, the agent will choose that action using argmax. This can lead to an agent executing always the same move at a specific point. The Epsilon is introduced to grant more freedom to the agent. The tested epsilons are 0.1, 0.25, 0.5, 0.8 with a progressive decay that influences this value until we reach half of the epochs. With a small epsilon there is some convergence and increasing it also increases the noise. It is important to note that with the epsilon value of 0.8 we achieved the best result, but it is reasonable to consider it an inaccurate result due to the high randomness.



(a) epsilon = 0.1 (b) epsilon = 0.25 (c) epsilon = 0.5 (d) epsilon = 0.8

Figure 5: Epsilon values

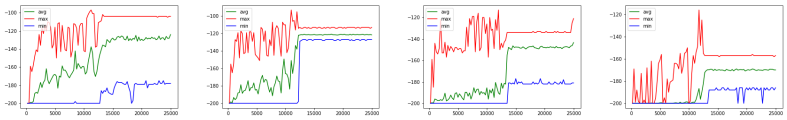


(a) epsilon = 0.1 (b) epsilon = 0.25 (c) epsilon = 0.5 (d) epsilon = 0.8

Figure 6: Heatmap Epsilon values

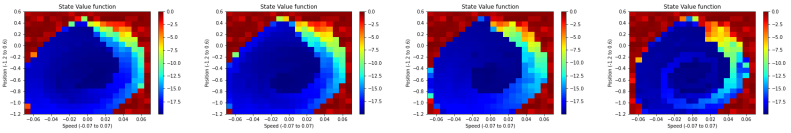
4.5 Learning rate

The learning rate determines how much the recently acquired information influenced and updated the old one. In this experiment we tested how increasing or decreasing this value could affect the agent's behaviour. The values tested are **0.1, 0.25, 0.4, 0.8**. After the evaluation, the best learning rate is 0.25 which permits the Q-Learning algorithm to have the best average. The maximum is slightly lower than the α=0.1 but converges and exceed all the other learning rates on the minimum and average rewards.



(a) $\alpha = 0.1$ (b) $\alpha = 0.25$ (c) $\alpha = 0.4$ (d) $\alpha = 0.8$

Figure 7: Learning rates

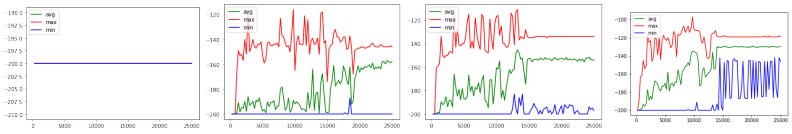


(a) $\alpha = 0.1$ (b) $\alpha = 0.25$ (c) $\alpha = 0.4$ (d) $\alpha = 0.8$

Figure 8: Heatmap Learning rates

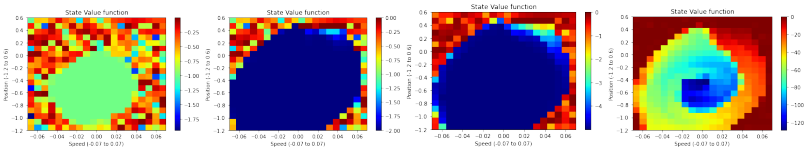
4.6 Discount Factor

This experiment analyzed the discount factor, the tested values are **0, 0.5, 0.9, 1**. With a discount factor of 0, the rewards are replaced after every single move, and only the new rewards are considered. The car never reaches the goal in this case. Alternatively, a discount factor closer to one will lead to values that show the future rewards an agent expects.



(a) $\gamma = 0$ (b) $\gamma = 0.5$ (c) $\gamma = 0.8$ (d) $\gamma = 1$

Figure 9: Discount factor = 1



(a) $\gamma = 0$ (b) $\gamma = 0.5$ (c) $\gamma = 0.8$ (d) $\gamma = 1$

Figure 10: Heatmap discount factor

5 Heatmap analysis

An interesting insight is to see how in each heat map, on top right there is a line that gets the highest possible value (dark red), around the position 0.5 and with positive speed; this is where the flag is located. The other two corner bottom right (low position, high speed) and top left(high position, high speed) have a dark colour, this makes understandable that at least the position or the speed has to be high to reach the flag.

References

[1] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.