0313_2361 *Aprendizaje Automático*

Unit 3 report: Reinforcement Learning.

Students: Carlos Castro Marín, Karla Torres Parra, Jonathan Montes Castro y Germán Álvarez López

Semester 2023-1

1. Introduction and general objective

Reinforcemet learning (RL) consists of determining how agents interact with the environment through actions to obtain the maximum reward, using the paradigm of test and error. The main goal for the agent is the learning from interaction with an environment to achieve a long-term goal related to the state of the environment. The principal elements of RL are:

- **Policy**: What to do.
- **Reward**: Which is Good for the agent.
- **Value** (new state): Stimation of the future reward
- **Model**: Environment.

The process in charge of reinforcing something (algorithm) allows updating the states in each iteration in a way that allows unlearning and learning at the same time to optimize the total future reward, this can be summarized in the figure 1*. All reinforcement learning methods are inspired by expected utility update formulas and state space exploration.

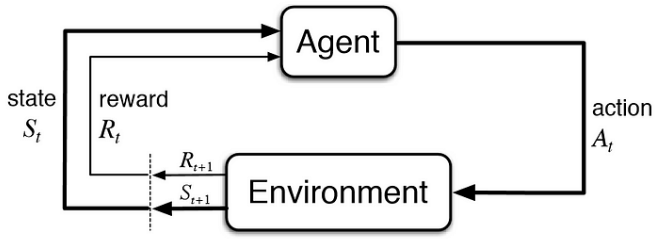
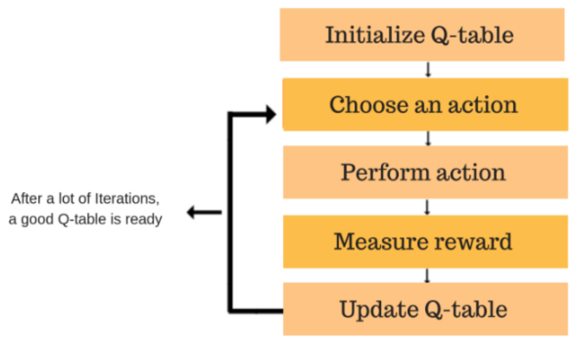


Figure 1. Scheamtic process of RL.

Q-learning is one of the most famous reinforcement learning algorithms. It is based on the Markovian theory of states and similar to SARSA (which estimates the most optimal value function for each step in the context of the Time Difference TD methods) but based on learning the policy, updating it at each step. This algorithm moves between exploration (agent must interact with the environment to learn a correct policy through trial and error) and explotation (obtains the correct action that maximizes the expected reward in a single step). Q-learning in the Markovian context makes uses of Q-Table, which is a simple lookup table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide the agent to the best action at each state. The procedure of the Q-learning algorithm can be summarized in the following diagram[1]:

*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/



The formula of this algorithm that takes two inputs: state (**s**) and action (**a**) is

$$Q_{new}(s,a) = Q(s,a) + \alpha\left[R(s,a) + \gamma\left(\max\left(Q'(s',a')\right)\right) - Q(s,a)\right],$$

with:

- Q_{new}: new state in the Q-table
- Q(s,a): actual state.
- α: learning rate. Determines the step size at each iteration by searching for mínima.
- R: reward for taking an action in that state.
- γ: Discount factor. It makes the agent look for the reward in the short (value close to zero) or long term (value close to 1).
- $\max\left(Q'(s',a')\right)$: maximum value for the new state and any action.

The main objective of this project is to demonstrate through a game from the gym library (from open AI) if this algorithm (Q-learning) works in such a way that the agent learns the policy based on the equation of states of which it is algorithm is supported.

1.1. Specific objectives

- Make a description of the configuration space of the MountainCar-v0 game and determine the normalized variables as part of the feature engineering process to be implemented in a Q-learn table.
- Evaluate, graphically through the python pygame library (in jupyter under the Anaconda software) or through the Python Virtual Display and matplotlib packages for google COLAB, if the agent (car) learns by itself the most appropriate policy, refining his states by means of reward and reaches the top of the mountain.
- Find the AAR (Average Accumulated Reward) performance metric and determine how many episodes it takes to reach the goal.

*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

2. Problem Context

For this project we use the environment Mountain Car from Gymnasium Open AI, a single agent reinforcement learning environment. This environment challenges an underpowered car to escape the valley between two mountains. The Mountain Car MDP is a deterministic MDP that consists of a car placed stochastically at the bottom of a sinusoidal valley, with the only possible actions being the accelerations that can be applied to the car in either direction. The goal of the MDP is to strategically accelerate the car to reach the goal state on top of the right hill.

As in the documentation from Gym, the core of Gymnasium is Env which is a high level python class representing a markov decision process from reinforcement learning theory It is defined a defines a "game" in which your reinforcement algorithm will compete. An environment does not need to be a game; however, it describes the following game-like features:

- **Action space**: What actions can we take on the environment at each step/episode to alter the environment.
- **observation space**: What is the current state of the portion of the environment that we can observe. Usually, we can see the entire environment.

Terminology used by this library:

- **Agent**: The machine learning program or model that controls the actions. Step - One round of issuing actions that affect the observation space.
- **Episode**: A collection of steps that terminates when the agent fails to meet the environment's objective or the episode reaches the maximum number of allowed steps.
- **Render**: Gym can render one frame for display after each episode.
- **Reward**: A positive reinforcement that can occur at the end of each episode, after the agent acts. The goal is to reach the flag placed on top of the right hill as quickly as possible, as such the agent is penalised with a reward of -1 for each timestep.

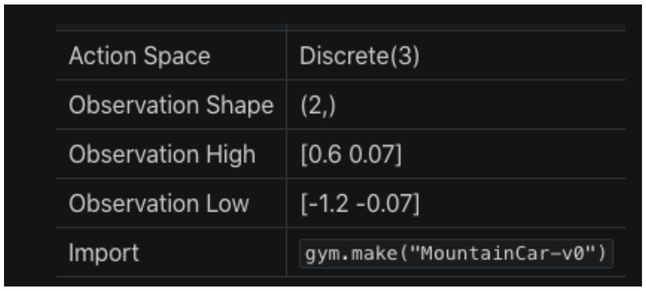
Here we can see all the Description of the Mountain Car Environment: https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py.

3. Descriptive analysis

This environment allows three distinct actions: accelerate forward, decelerate, or backward. The observation space contains two continuous (floating point) values, as evident by the box object. The observation space is simply the position and velocity of the car. Figure 1 shows a summary of the previous description. The car has 200 steps to escape for each episode. You would have to look at the code, but the mountain car receives no incremental reward. The only reward for the vehicle occurs when it escapes the valley.

*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/



Action Space	Discrete(3)
Observation Shape	(2,)
Observation High	[0.6 0.07]
Observation Low	[-1.2 -0.07]
Import	`gym.make("MountainCar-v0")`

Figure 1. Description of the MountainCar environment.

The initial position is settle randomly and the initial speed is taking as 0 by default. Figure 2 is an image taken from the gym library that represents the mountain and the goal (flag).

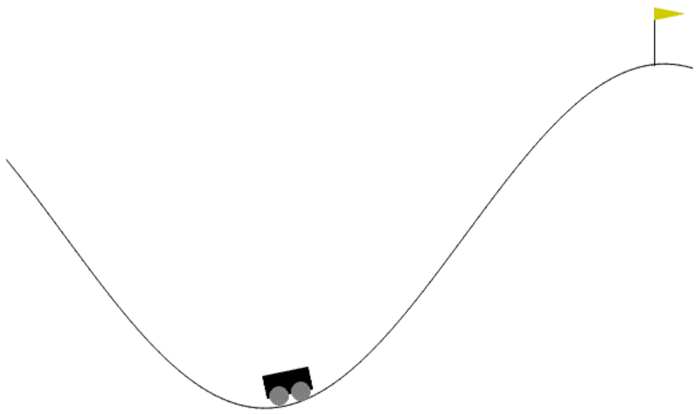


Figure 2. Mountain Car from Gym library (Open AI).

Given an action, the mountain car follows the following transition dynamics:

$$v_{t+1} = v_t + (action - 1) \cdot froce - \cos(3 * position_t) * gravity$$
$$position_{t+1} = position_t + velocity_{t+1}$$

where force=0.001 and gravitiy=0.0025. As the description from gymlibrary, he collisions at either end are inelastic with the velocity set to 0 upon collision with the wall. The position is clipped to the range [-1.2, 0.6] and velocity is clipped to the range [-0.07, 0.07].

An episode ends if either of the following happens:

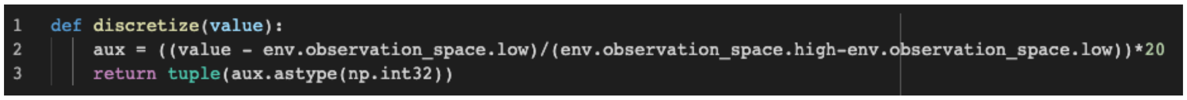
*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

- Termination: The position of the car is greater than or equal to 0.5 (the goal position on top of the right hill)
- Truncation: The length of the episode is 200

4. Data and Feature engineering

The observation space of the environment is defined in the position range $[-1.2, 0.6]$ in the "x-axis" so to speak. So, in this case we need to use all the data in this axis, which determines the domain, *i.e*, values in the x-axis in which the car is moving. We need to create the Q-table in order to apply Q-learning equation and algorithm in the context of Reinforcement learning being the car the agent in our environment, speed, force and gravity the policies of our environment and the rewards and penalties variables that we defined later. We decide to normalize the position range in order to create a table of 20 x 20 with the following normalization function:



```python
def discretize(value):
    aux = ((value - env.observation_space.low)/(env.observation_space.high-env.observation_space.low))*20
    return tuple(aux.astype(np.int32))
```

Figure 3. Discretization function, code taken from the Colab document.

A Q-table is created with random variables from -1 to 1 of a size of 20 x 20 (the code can be observed in the jupyter notebook, COLAB or jupyter with anaconda). It is important to add that the chosen learning rate was 0.1 and we use a disccount factor of 0.95 making the agent looks for the reward in a long term.

5. Environment without Reinforcemente Learning

The car in the environment fixed with a random initial position and following the policies of the environment in a markovian context with the action being random between 0 and 2 shows a truncation after some time. Figure 4 shows this behaviour.

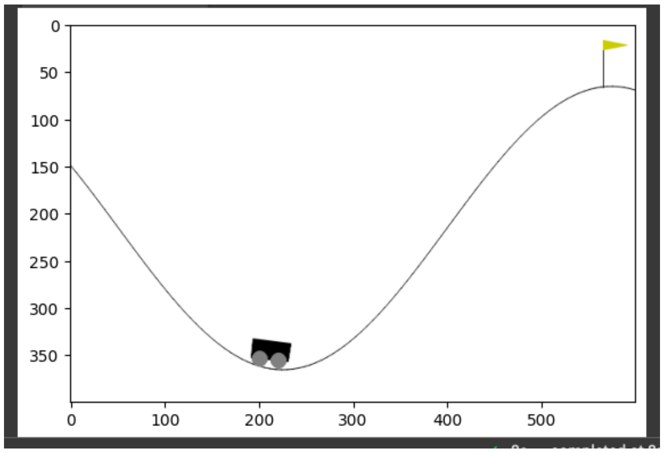


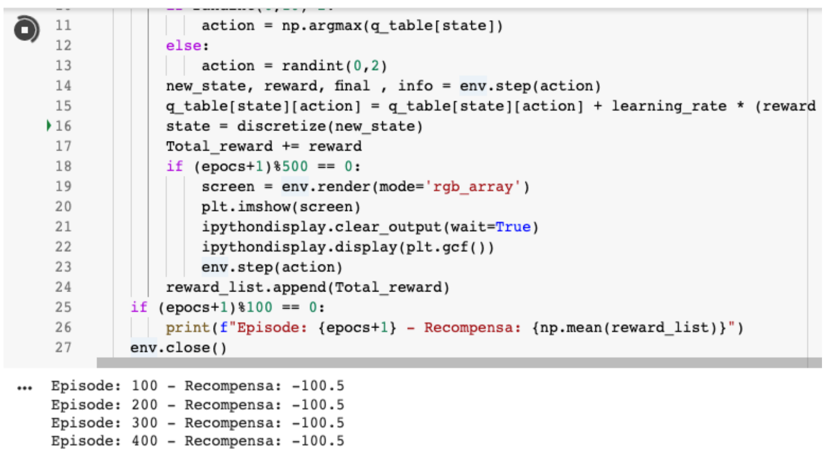
Figure 4. Agent in a random initial position and following the policies of the environment.

*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

6. Environment with Reinforcemente Learning (Q-learning algorithm)

We defined 5000 episodes (epocs in the coding) the times that the game is trained, we set our code that shows us each 500 time that the game is played. It is important to point out that our states are discretized by the function previously defined in the notebook in order to have a normalization and discretization of the configuration space in terms of the space in the "x-axis". Also our reward, being the metric in our code will be added in a list for each episode (epoc). Figure 5 shows the first episodes and the corresponding rewards for 2500 Episodes in the COLAB notebook.

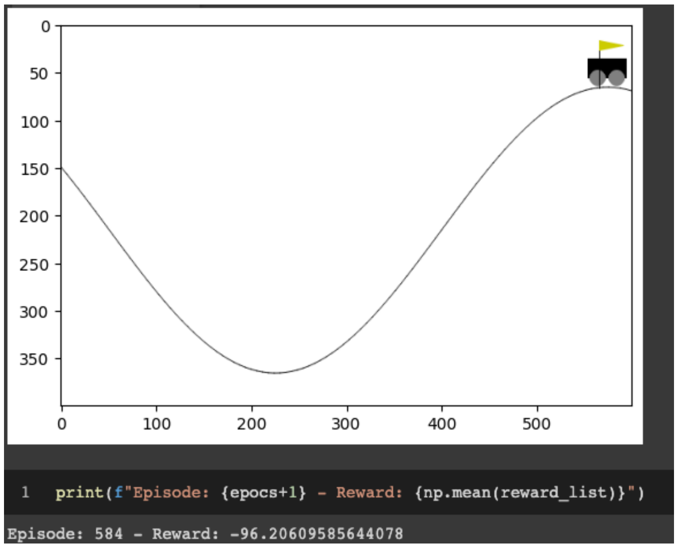


```
            action = np.argmax(q_table[state])
        else:
            action = randint(0,2)
        new_state, reward, final , info = env.step(action)
        q_table[state][action] = q_table[state][action] + learning_rate * (reward
        state = discretize(new_state)
        Total_reward += reward
        if (epocs+1)%500 == 0:
            screen = env.render(mode='rgb_array')
            plt.imshow(screen)
            ipythondisplay.clear_output(wait=True)
            ipythondisplay.display(plt.gcf())
            env.step(action)
        reward_list.append(Total_reward)
    if (epocs+1)%100 == 0:
        print(f"Episode: {epocs+1} - Recompensa: {np.mean(reward_list)}")
    env.close()
```

```
Episode: 100 - Recompensa: -100.5
Episode: 200 - Recompensa: -100.5
Episode: 300 - Recompensa: -100.5
Episode: 400 - Recompensa: -100.5
```

Figure 5. First Episodes and rewards.

Figure 6 shows how the agent learns the policy and reach the goal (flag) after 584 episodes with an average reward (ARR) of -0.9620.



```
print(f"Episode: {epocs+1} - Reward: {np.mean(reward_list)}")
```

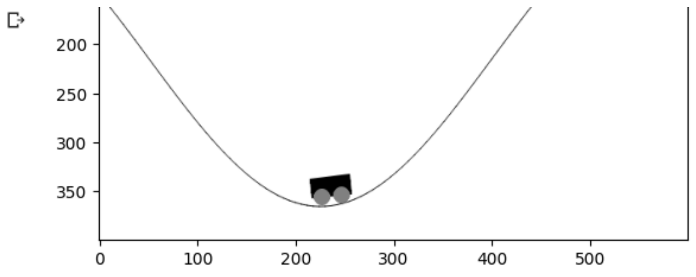
```
Episode: 584 - Reward: -96.20609585644078
```

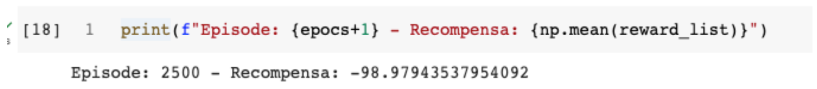
Figure 6. after 584 episodes with an average reward (ARR) of -0.9620.

*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

For 5000 episodes, it can be observed in figure 7 that in the episode 2500 the car reachs the goal (flag).





```
[18]  print(f"Episode: {epocs+1} - Recompensa: {np.mean(reward_list)}")

      Episode: 2500 - Recompensa: -98.97943537954092
```

Figure 7. after 2500 episodes with an average reward (ARR) of -0.9897.

Something important to mention. There was not a clear way to run the environment to be seen by pygame in COLAB. We had to fix this issue with libreries such as pyvirtualdisplay and matplotlib library. For this very reason, if you want to run the code in COLAB, you have to wait between 27 and 35 minutes until the agent (the car) uses Q-learning algorithm and the Q-table and preset normalization learns by the rewards to reach the goal. We suggest to use the Jupyter notebook with Anaconda software to run the codes. However in COLAB is the detailed structure of the code.

Conclusion

It is shown in this simple environment from gym library that Q-Learning algorithm works for diferente experience periods, i.e, episodes. The car, due to the Q-learning equation, learns the best policy based on the rewards in a long term. Something curious is that for les episodes the agent learns faster and reach the goal with less penalization. Finally, we can conclude that Q-learning is a good algorithm and works well in the reinforcement learning scope.

Bibliography

[1] R. Sutton, A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2005

*Image taken from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
[1] Diagram taken from https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/