# LSINF2275 - Data Mining and Decision Making - Project 2
# Markov Decision Processes

Group 10
d'Oultremont Augustin - 2239 1700 - INFO
de Voghel Brieuc - 5910 1600 - INFO
Lemaire Valentin - 1634 1700 - INFO

Link to presentation video of the project*

Sunday 16th May, 2021

## Introduction

The objective of this project is to explore different solutions of the continuous version of the Mountain Car problem[1] using OpenAI's Gym toolkit. The Mountain Car problem consists of a car on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right and reach the flag. However, the car's engine is not strong enough to climb the mountain in a single try. Therefore, the only way to succeed is to drive back and forth to build up momentum. The objective is to spend as little energy as possible to reach the goal.

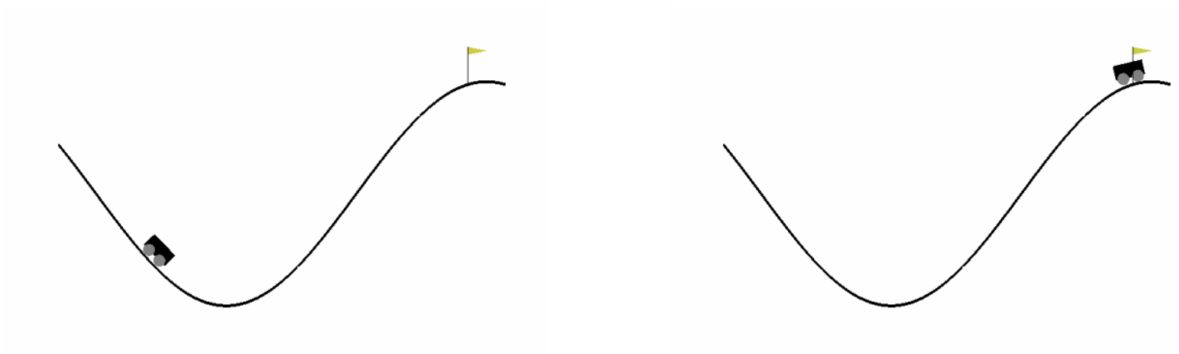


Figure 1: The Mountain Car Problem

In order to solve this problem, we first apply a basic Q-Learning approach. We then try to reproduce more advanced methods proposed in [1]. We will begin by explaining the general Q-Learning algorithm in section 2 and proceed with an explanation of the Time Adaptive $\varepsilon$ Q-Learning algorithm in section 2.1. Then, we explain the SARSA [2] algorithm in section 2.2 and we will finish with the Backward Learning [1] algorithm in section 3, which combines the other two to obtain a fast-converging and accurate algorithm.

## 1 Preliminaries

Our problem can be modelled as a Markov chain. Indeed, each state is independent of the past (memoryless), and an action can be applied at each state to reach a new one, triggering a certain reward. More on the environment in Section 4.

*Presentation video : https://uclouvain-my.sharepoint.com/:f:/g/personal/brieuc_devoghel_student_uclouvain_be/ErAgPCxOPKdGsWxO-ESza7oB6u2Z37j038q0WuVOI40zyQ?e=BDnoaQ

[1]OpenAI's Mountain Car problem : https://gym.openai.com/envs/MountainCarContinuous-v0/

In the following, notation subscript $t$ will denote the time step currently considered. $s_t$ is the state (containing both position and speed) at time step $t$, $a_t$ is the action performed at state $s_t$ and leading to state $s_{t+1}$. $r_{t+1}$ is the reward obtained by applying action $a_t$ on state $s_t$ and leading to state $s_{t+1}$. $Q(s_t, a_t)$ is the estimated reward for applying action $a_t$ to state $s_t$. $\gamma$ is the discount factor ($0 < \gamma < 1$). A low value for this parameter will penalize policies that reach the terminal state after many time steps. $\alpha$ represents the learning rate in Time Difference algorithms.

# 2 Q-Learning

The Q-Learning algorithm is the basis of the algorithms in sections 2.1 through 3. Its simplest form consists of the one-step Q-Learning and defines the following update of the $Q$ matrix :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{1}$$

## 2.1 Time Adaptive $\varepsilon$-greedy Q-Learning

One explored variation on Q-Learning is the Time Adaptive $\varepsilon$-greedy Q-Learning [3] (TAE-Q-Learning). This algorithm addresses the exploration and exploitation dilemma by adding an $\varepsilon$ factor, which essentially controls the chance of taking a random action instead of the optimal one (the action leading to the highest estimated future reward). This value decays linearly at each episode, allowing for more exploration in the beginning and more exploitation at the end.

At each timestep, a random value is generated. If that value is greater than $\varepsilon$, we choose optimal action for the current state. If it is not, a random action is chosen. The Q matrix is then updated according to eq. 1.

## 2.2 SARSA algorithm

Our second explored variation is SARSA Originally called Modied Connectionist Q-Learning, described in paper [2], the SARSA algorithm (acronym for State-Action-Reward-State-Action) consists of a slight variation on eq. 1. Note the difference in the use of the Q value when applying future action $a_{t+1}$.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2}$$

As explained in [4], the use of a future action differentiates two main classes in Reinforcement Learning: model-irrelevant algorithms such as Q-Learning or TAE-Q-Learning in which the optimal policy is immediately computed without model data and model-based methods such as the SARSA algorithm in which model-knowledge is first learned and only after, the optimal strategy is derived from this knowledge. This can be seen in eq. 2 by the involvement of $a_{t+1}$. To update a previous state, the action that will be taken in the next state must already be computed.

# 3 Backward Learning

Backward Learning [1] aims to combine the advantages of Q-Learning and SARSA to allow for SARSA's fast convergence with Q-Learning's good results by tuning the Q-values (affecting the action selection policies indirectly). It can be applied to any model-irrelevant algorithm and transform it into a model-based method.

[1] proposes to add an update rule to the Q-Learning update. It can however be extended to any update scheme and this method would then act as a wrapper around another agent using Q-learning. It lets the base agent perform its usual updates and fine-tunes the Q-values if and when the terminal state is reached. In the following developments we assume Q-Learning is the wrapped agent.

This algorithm uses 2 more parameters : $\alpha_b$ and $\gamma_b$. For each time step, the algorithm updates the states according to the initial agent's update rule (i.e. eq. 1 for TAE-Q-Learning and eq. 2 for SARSA). In addition to that, it stores, for each time step, the values of $s_t$, $a_t$, $r_{t+1}$ and $s_{t+1}$ in a vector (called $M$). If the goal state is reached at the end of the episode, the algorithm performs a backward update on the whole path by retrieving the values $s_t$, $a_t$, $r_{t+1}$ and $s_{t+1}$ from $M$ and performing the following update (for $t$ from $N$ to 1):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_b(r_{t+1} + \gamma_b \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{3}$$

The parameters $\alpha_b$ and $\gamma_b$ allow to make a bigger change in the $Q$ matrix when correclty solving the problem, thus converging faster.

We say this algorithm is a combination of Q-Learning and SARSA because as long as the terminal state is not reached, the algorithm is strictlty equivalent to Q-Learning. However, every time the agent reaches the goal, it updates all states it went through (starting from the last) and enforces a stronger learning.

This means that we update a Q-value using the maximum of the Q-values in the next state, having already updated the Q-value corresponding to the action taken in the next state. Since these updates are only done in the case we reach the goal state the Q-value corresponding to the action taken in the next state will probably have been increased and this will favour its chances of being the maximum of the actions taken in equation 3. This is analogous to SARSA in the way that it updates a Q-value by taking into account future observations and their estimated reward.

In the backward update (eq. 3) we use the expected reward of taking a greedy action rather than the actual action we have taken in this state like in SARSA. In SARSA, using the actual action makes sense as we are updating before reaching the goal state and we explicitly want to update the actual path taken to have the best estimation of rewards in the Q-table. However, in the backwards update, we have already reached the goal state. Hence, we want the Q-value for a certain state and action to hold the estimated reward for the next state. This is best estimated by computing the maximum over the next state's available actions, as this represents the greedy action which is the most probable course of action next time we reach this state.

As an extension on the algorithm proposed by [1] (wrapping Q-Learning with backward updates), we have wrapped the backward update algorithm around the SARSA agent as well. The principle is the same except that in the normal update, rather than using eq. 1 we use eq. 2.

# 4 Simulations

In order to assess the performances and compare the different algorithms, we ran several instances of the presented algorithms (agents) with multiple meta-parameters on OpenAI's Gym environment of the Mountain Car problem. Every agent was limited to 10.000 training episodes to learn to climb the mountain.

**Environment** Both the observation and action space of the problem are continuous. An observation consists of the position $p$ and velocity $v$ of the car. An action consists of the throttle $a$ of the engine, positive to accelerate forward, negative to accelerate backwards.

$$p = [-1.2, 0.6]\,;\; v = [-0.7, 0.7]\,;\; a = [-1.0, 1.0] \tag{4}$$

The observation space $(p, v)$ is quantized into $30 \times 30 = 900$ states; the action space in $10$ possible actions. The agent is rewarded $+100$ when the car reaches the flag, minus a small value

proportional to the acceleration. Steps other than the final one thus have a negative reward. Episodes where the goal is not reached within 1000 steps fail.

**Performance evaluation** The performance of an episode is the sum of the rewards for all time frames if the terminal state is reached and 0 otherwise. The performance of an agent is the average of the last 100 episodes. Thus, agents who can consistently reach the flag while consuming little fuel (by not accelerating much), will perform the best.

**Meta-parameter search** As discussed in the agents' algorithm Sections 2 through 3, several parameters influence the learning behaviour. To produce comparable results, we chose to fix the learning rate $\alpha$, discount factor $\gamma$; and for the backwards algorithms: learning rate $\alpha_b$, discount factor $\gamma_b$, to the same values for all agents. They were chosen to be feasible and reasonably good for all algorithms. The values are discussed in section 5. All agents present an $\varepsilon$ value that allows for an exploratio-exploitation compromise. It was initialized to 0.5 (every other action is random) and decreases linearly every episode until it reaches 0 after the last episode.

# 5 Results

In the following sections, we present the different results obtained with the tested parameters.

**Selection of the discount factor** An overall observation was made that the discount factor $\gamma$ performs in general best for all agents when close to 1. We chose to fix the discount factor of all agents to $0.99$. This is justified by the fact that we want our agents to strive for long-term higher rewards (i.e. consume less fuel).

**Selection of the learning rates** Experimental results show that the algorithms perform best when having a learning rate $\alpha$ of about $0.05$ and $0.07$ for the Backward Learning algorithms (i.e. $\alpha_b$).

Setting those values allow for better comparisons in between the different agents. As expected, variations have the same impact on all agents : a higher $\alpha$ tends to improve more quickly the performance but then oscillates around without converging (especially when $\varepsilon$ is still relatively high).

## 5.1 TAE-Q-Learning

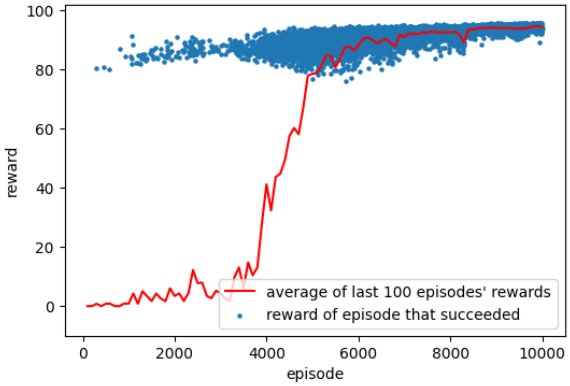



Figure 2: Results of the TAE-Q agent

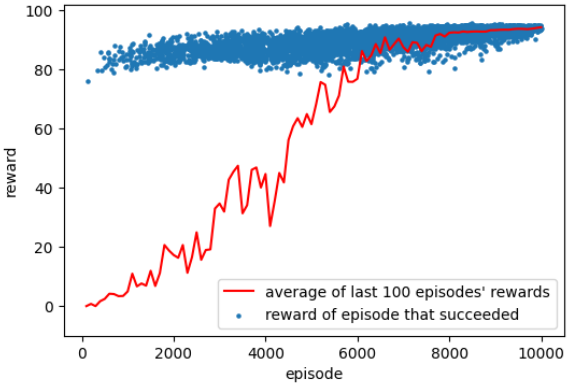



Figure 3: Results of the SARSA agent

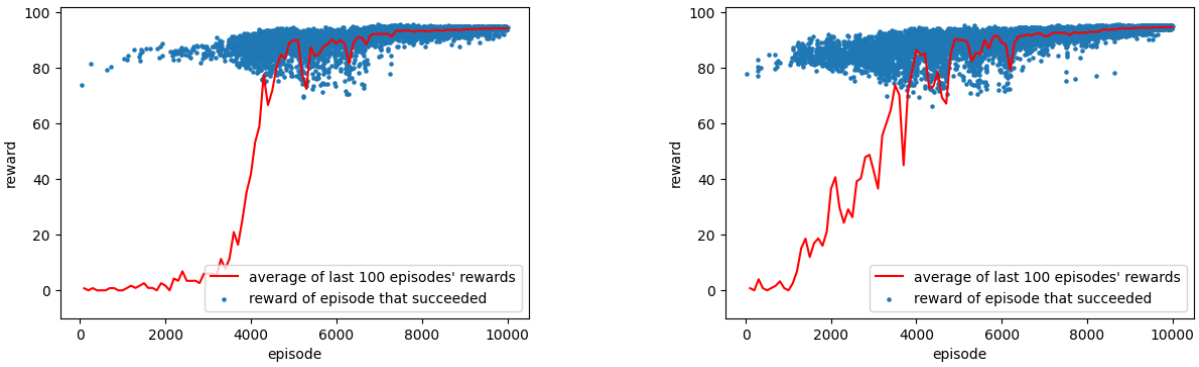
As seen in figure 2, the TAE-Q agent keeps a low performance for many iterations before climbing very fast at around 4000 episodes. After around 5000 episodes, it has reached a mean reward of around 80 and starts to slow down, but keeping a steady convergence overall.

### 5.2 SARSA

When running the SARSA algorithm on our Mountain Car problem, we obtain the convergence shown in figure 3. Immediately, we see that much more episodes result in reaching the terminal state in the early stages of learning. This leads the mean over the last 100 episodes to climb earlier than in TAE-Q-Learning.

Even though the mean climbs higher, it does not climb as steadily and it does take around 6000 episodes for the algorithm so succeed almost every time to reach the terminal state. In the last 2000 episodes we see that it has correctly converged and that the agent reaches the terminal state at every single episode with a consistent score.

### 5.3 Backward Learning




(a) on Q-Learning

(b) on SARSA

Figure 4: Backward Learning

In figure 4a we have applied the Backwards Learning wrapper on the TAE-Q-Learning algorithm. Compared to the results of the standard algorithm, we do not see that the algorithm converges much sooner in the episodes tourner à la positive, however, once the mean starts to climb it does so extremely fast. There is a stage where most episodes reach the terminal state but not all and at 6000 episodes we have reached our final convergence stage.

In figure 4b, Backwards Learning was applied to SARSA. Once again, the effect is not really a sooner convergence but rather a quicker one. We observe a much steeper ascent and the final convergence state is reached after 6000 episodes.

## 6 Discussion

On figure 5, we observe once again that when we add backwards learning to an agent, it learns slightly earlier but clearly faster. This can be explained by the fact that when we apply backward learning, every episode that reaches the terminal state has its learning strongly enforced post-experience. This will encourage future episodes to explore that path and to reinforce it (i.e. optimize that path). This will in turn lead to more episodes reaching the terminal state and applying the backwards learning rule. This speeds up the learning process quite significantly.

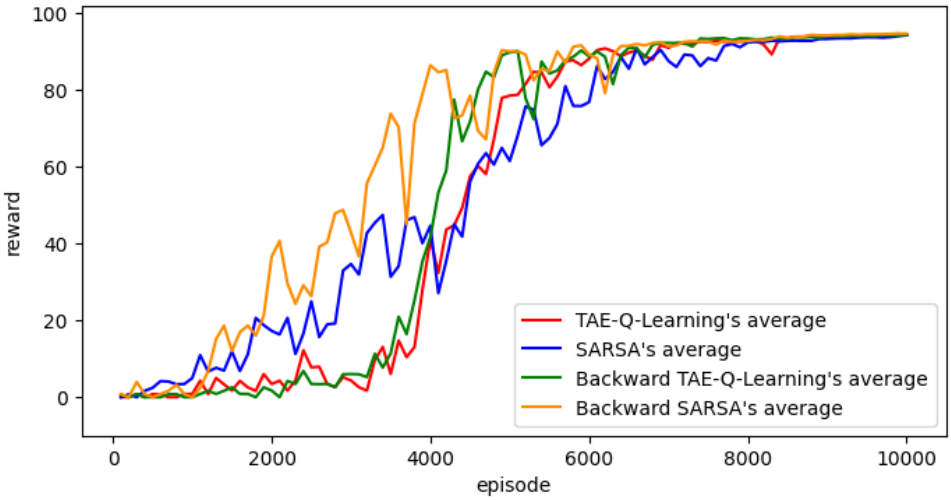



Figure 5: Comparison of the agents' learning performance (average reward over 100 episodes)

We do also observe that both SARSA based algorithms start to get better performances sooner than TAE-Q-Learning based algorithms. This can be explained by the fact that SARSA uses a future action to update a current action therefore when state leads to a high reward by applying an action, not only that state is updated but also the state before that. In TAE-Q, two episodes are needed for this update. This means that SARSA encourages exploitation more than TAE-Q.

Another observation that can be made is that TAE-Q based algorithms are much more steady than SARSA based ones. The latter show more variation in their learning curves and tend to oscillate around the optimal solution longer in the last stages of learning. This can be explained by the fact that we estimate future rewards by the Q-value of the actual action we will take in the next state. However for a future episode this might very well not be the action we will take meaning that the future reward is estimated incorrectly. In TAE-Q we estimate the future reward by taking the action that maximizes the Q-values in the next state. This corresponds to the most probable course of action in the next state. Making the learning steadier.

**Sensibility of meta-parameters** The parameter initialization has a large impact on the agent converging towards a stable output. By varying, even slightly, the different parameters (learning rate, discounting factor, and the exploration rate) we obtained very different results. Some of the experiences not converging at all to a satisfactory result even after 10 000 episodes. phrase à revoir

# 7 Future improvements

## 7.1 Adaptive Learning Rate on Backward Learning

The Backward Learning algorithm applies a greater update on the values of $Q$ when reaching a solution, meaning that when we approach a good solution, the backward update (equation 3) will get executed more and more. At the end, this means that, for each iteration (since each iteration finds a solution), the $Q$ matrix will be updated twice (performing the usual update rule and the backward update), thus making big steps at the end of the learning.

One improvement could be to have an adaptive learning rate on one (or both) updates done in the Backward Learning algorithm. The diminishing of the learning rate could start in the beginning or when a certain number of consecutive episodes found a solution.

### 7.2 Smart quantization of observation and action spaces

The current quantization consists of dividing the observation and action spaces linearly. However, some states and actions are used a lot more than others. The aim of a smart quantization would be to adjust the refinement of the grid according to the use of the different observations and actions. This would allow for more fine-tuning of the values used a lot, without changing the computation time.

### 7.3 Initial Conditions

Currently, all our algorithms are initialized with a random initial Q matrix. Ideally we would initialize them to encourage exploration in the early stages. [5] proposes to initialize the matrix Q with low (infinite) initial values in the initial state, known as optimistic initial conditions. This would correspond to all states in the region of the valley of our MountainCar problem, with a null speed.

## 8 Conclusion

In this project we set out to solve the Continuous Mountain Car problem using different Reinforcement Learning algorithms. We explored two different classes of algorithms: model-irrelevant algorithms with the basic TAE-Q algorithm and model-based algorithms with the SARSA algorithm. To delve further in the subject we implemented the Backward Learning wrapper and applied it on both of the first two algorithms.

Using the right hyper-parameters, all of these were able to converge and find the optimal solution. However by comparing the different algorithms (using the same hyper-parameters for each of them) we were able to detect some advantages and disadvantages to each of them.

TAE-Q is very slow to learn but once it starts to reinforce its known solutions it quickly converges and does not vary much after. SARSA is quicker to learn but has much more variance and it takes more episodes to reach the optimal solution. The backward learning wrapper allows to learn sooner but introduces a bit more variance in the obtained results. Therefore it is a considerable asset when applied to TAE-Q as it adresses its weaknesses but does not improve SARSA much as it enforces the qualities that algorithm already had and does not address it's issues.

## References


[1] Yin-Hao Wang, Tzuu-Hseng S. Li, and Chih-Jui Lin. Backward q-learning: The combination of sarsa algorithm and q-learning. *Engineering Applications of Artificial Intelligence*, 2013.

[2] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. September 1994.

[3] Mohit Sewak. *Temporal Difference Learning, SARSA, and Q-Learning*, pages 51–63. Springer Singapore, Singapore, 2019.

[4] Wang Qiang and Zhan Zhongli. Reinforcement learning model, algorithms and its application. pages 1143–1146, 2011.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 2018.