Artificial Intelligence Project

May 2019

Sami Lazrak

Youness El Idrissi

Abstract

OPEN AI GYM has made reinforcement learning more fun and easier to practice. This report presents state of the art Q-learning and SARSA Algorithms. This paper focuses on analyzing the differences between these two algorithms through experiments on the Frozen-lake game.

Contents

List of Figures

1 Introduction

Reinforcement learning is an area of Maching Learning.

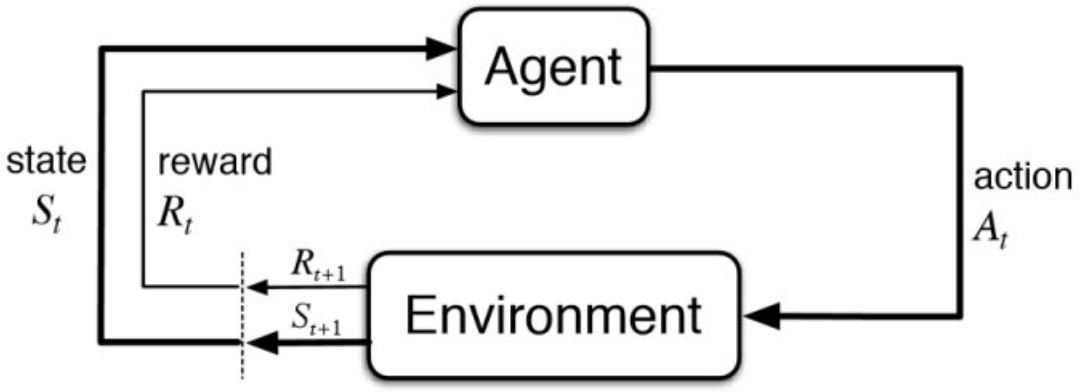

Figure 1: Reinforcement Learning Illustration

Typically, a RL setup is composed of two components, an agent and an environment. The environment refers to the object that the agent is acting on (here, the environement is the Frozen lake game) and it is fully described as a Markov Decision Process (MDP).
The agent represents the RL algorithm.

Q-Learning and SARSA are two algorithms used in RL that allows the agent to optimize his **cumulative reward**, by choosing an action according to a **Q-table** which is constantly updated.

2 Description of the algorithms

In this section, we will describe those two algorithms, and show their main differences.

> **On-policy v.s. Off-policy**
> An on-policy agent learns the value based on its current action a derived from the current policy.
> An off-policy agent learns it based on the action A obtained from another policy. In Q-learning, such policy is the greedy policy. [3]

2.1 Q-Learning

Q-Learning is an off-policy algorithm. It is based on a matrix (Q-Table) of (number of states x number of actions) size.
And uses the bellman equation, with an **action value function** to compute the Q-values $Q(s, a)$:

$$Q^{\pi}(s, a) = E_{\pi}[R_t \mid s_t = s, a_t = a]$$

The algorithms chooses an action (either he explores new solutions or exploit the ones already discovered), and learns from his actions according to the rewards received.

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}\big[\text{Target} - \text{OldEstimate}\big]$$

Figure 2: Bellman Equation Illustration

In mathematical terms, we obtain:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + Q^{\pi}_{obs}$$

with

$$Q^{\pi}_{obs}(s_t, a_t) = \alpha[R_{t+1} + \gamma max(Q^{\pi}(s_{t+1}, a)) - Q^{\pi}(s_t, a_t)]$$

$\boldsymbol{\alpha}$ is the learning rate. The **Q(s,a)** table is initialized randomly. Then the agent starts to interact with the environment. At each interaction the agent will observe the reward of its action **r(s,a)** and the state transition (new state $\boldsymbol{s_{t+1}}$). The agent computes the observed Q-value $Q_{obs}(s, a)$ and then use the above equation to update its own estimate of Q(s,a) [1]

Our Q-Learning algorithm has two main methods :

- A chooseAction() method, which is based on the epsilon-greedy algorithm : it choose between exploring new solutions (with a probability = epsilon) or exploit the ones already known (with a probability = 1 - epsilon.). If the exploitation option is chosen, the algorithm selects the action with the highest expected reward according to the current state. Otherwise, it chooses randomly an action and performs it.
- A learn() method : this method updates the Q-Table after analyzing the reward of the last action. The algorithm updates it by applying the Bellman equation with two inputs : the current state and the action taken.

2.2 SARSA

[4] SARSA is an on-policy algorithm, because we use the same policy to generate the current action at and the next action at+1; and it's difference with the Q-Learning algorithm is the way it learns and updates the Q-Table :

it updates it's Q-Table by assuming we are taking action a that maximizes our post-state Q function Q(st+1,a).

Q-learning:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + Q^{\pi}_{obs}(s_t, a_t) = \alpha[R_{t+1} + \gamma max(Q^{\pi}(s_{t+1}, a)) - Q^{\pi}(s_t, a_t)]$$

SARSA:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + Q^{\pi}_{obs}(s_t, a_t) = \alpha[R_{t+1} + \gamma max(Q^{\pi}(s_{t+1}, a_{t+1})) - Q^{\pi}(s_t, a_t)]$$

In SARSA, we use the same policy (i.e epsilon-greedy) that generated the previous action at to generate the next action, at+1 which we run through our Q-function for updates, Q(st+1,at+1). (This is why the algorithm was termed SARSA, State-Action-Reward-State-Action).

For Q-learning, we have no constraint on how the next action is selected, we pick the action a that maximizes Q(st+1,a). This means that with Q-learning we can give it data generated by any behaviour policy (expert, random, even bad policies) and it should learn the optimal Q-values given enough data samples.

They don't converge at the same rate : SARSA converges faster then Q-Learning, but they converge to the same value.

3 Practical testing

We want to precise that the code used in this part is mostly inspired by the lecture and the gists of code from [2]

The initial Q-table is initialized as an empty matrix (matrix of zeros). We should note that we use a score metric defined by:

$$\frac{\text{number of } \textbf{won} \text{ episoides}}{\text{number of } \textbf{total} \text{ episodes}}$$

The parameters used for running our algorithm are (we will specify the parameters that changed for the 8x8 version part):

1. epsilon $\leftarrow 0.9$
2. learning rate (or alpha) $\leftarrow 0.01$
3. gamma $\leftarrow 0.96$
4. decay rate (for the ϵ-greedy algorithm) $\leftarrow 0.01$, with an epsilon starting value of 0.1 and maximum value of 1

All experiments below have been conducted on 100 000 episodes. After each 5000 episodes, we average the score over 10 runs of 100 episodes using the current Q-table (i.e to take an action, we simply use **np.argmax(Q[state])**). And we estimate we solved the frozen lake game when our score is over 0.8. The pseudo code is:

Data: total episodes = 100000, arguments of the algorithm
for *episode in total episodes* **do**
  while $t \leq maxsteps$ **do**
    Run algorithm Qlearning or SARSA
  end
  if *episode is multiple of 5000* **then**
    Run algorithm 10 times over 100 runs with current Qtable;
    $scoreAverage \leftarrow Averagethe3scores$
    if $scoreAverage \geq 0.8$ **then**
      break;
      Show "Frozen lake solved";
    end
  end
end

3.1 FrozenLake 4x4

As mentionned previously, we began running on 100 000 episodes and we quickly realized that the 2 algorithms with and without decay converge after 40 000 episodes and the score remeains around 0.79 as you can observe in Figure 3. Besides the convergence happens around 25 000 episodes for both versions of SARSA and QL with decay
Q-learning solves the Frozen Lake Game (FLG) after 110 000 episodes whereas **SARSA** solves the FLG after 70 000 episodes and both **Q-learning and SARASA with decay** solve it around 50 000.

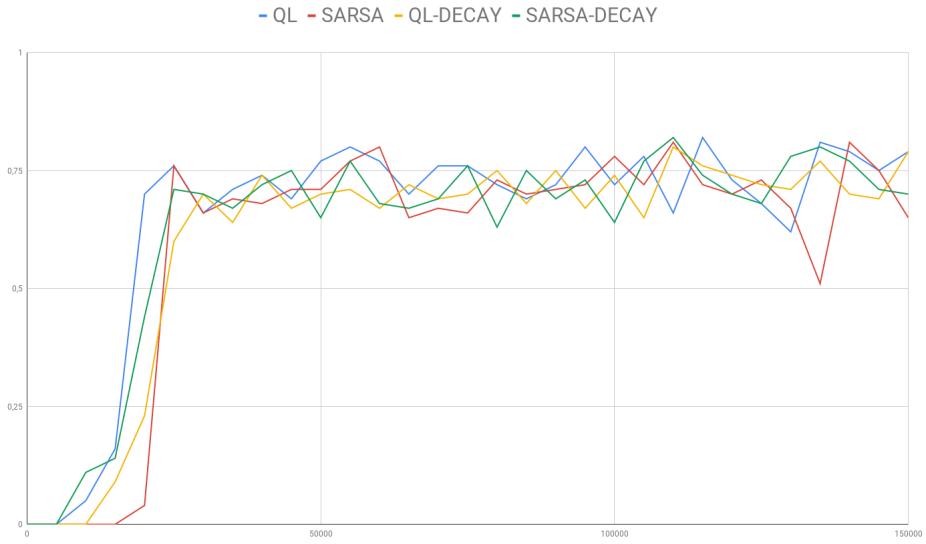

Figure 3: Performances of different algorithms depending on the computed score

Which lead us to run the same experiment with the algorithm described above on 30 000 runs with tests each 1000 run instead of 5000. As you can see in the Figure 4 below This enabled us to have a more precise illustration of how the performances of the algortihms differ.

We can clearly see in Figure 4 that the **Q-learning** algorithm with Decay performs far better with the algorithm without Decay. This was an expected result and proves that both our implementation and our approach was likely correct. Regarding the **SARSA** algorithm, we thougth that the Decayed version would perform far better than the normal version.
We did however change the learning rate to show that **SARSA with decay** is better than **SARSA** but that leanring rate was showing bad performances for the **Qlearning** algorithm.
We will discuss the learning rate parameters in more details in the end of the paper.

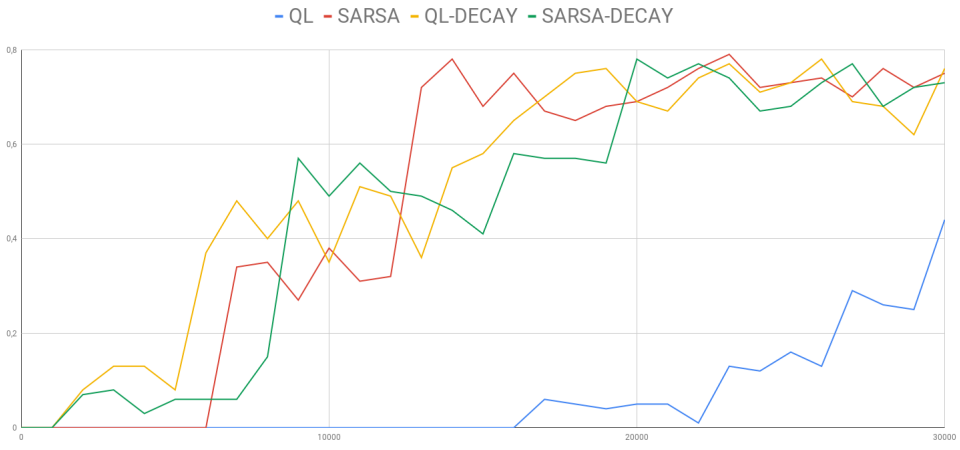

Figure 4: Performances of different algorithms depending on the computed score - smaller scale

3.2 FrozenLake 8x8

This part was quite challenging, because it took time to compute and debug. We runned on 300 000 runs to get a first feel of the performance and how far we can get in terms of score. For this version, we had to tweak the parameters to get good performance results after a short amount of learning loops (updating the Q-table). The parameters stay the same and we just need to change the:

1. epsilon $\leftarrow$ 0.5, for the non decay part

Due to the more complex version that is the 8x8, the convergence happens after a higher amount of episodes. Around 50 000 for both versions of **SARSA**, 75 000 for **Q-learning** and after 150 000 episodes for Q-learning with decay.

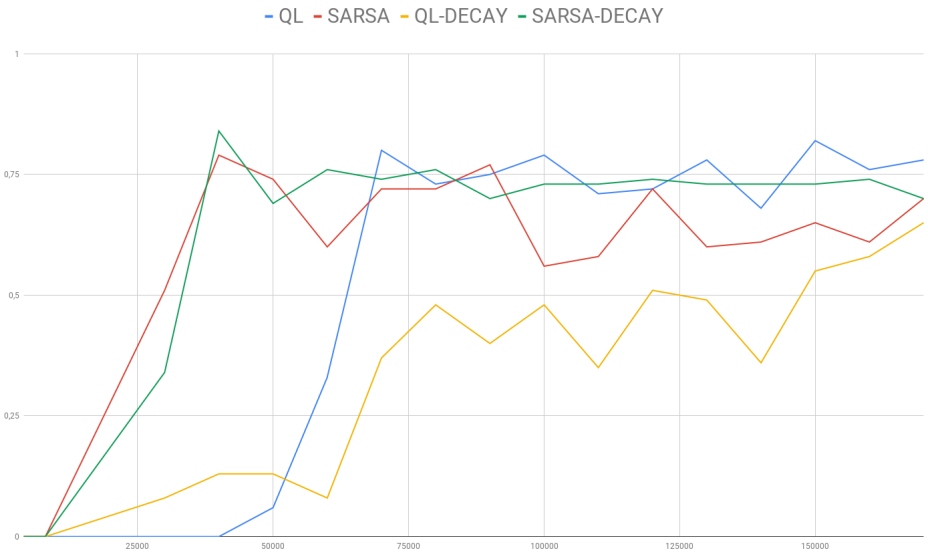

Figure 5: Performances of different algorithms depending on the computed score

4 Reflexion

Through this study case of the algorithms of **Q-learning** and **SARSA** we learned the different aspects of Reinforcement Learning.
There are a lot improvements we can later on implement to our algorithm to get even better results.

First, we can define a custom reward function that will keep a nu reward for loosing, and a positive reward for episodes where the maximum of steps is attained. This reward will be ranging from 0 to 1 depending on the distance between the player and the Goal.

Second, the learning rate was a very imprtant factor to take account of. After carefully reading [6], we understood the importance of this hyperparameter and how we can use a variant learning rate through the learning process.
Basically, the paper shows an interesting relationship between the convergence rate and the learning rate used in Q-learning where they compare a polynomial and linear learning rates in respect to the convergence rates of the Q-learning algorithm

5 Conclusion

According to our results, SARSA converges faster than Q-Learning to the optimal solution. but it doesn't mean that SARSA is a better algorithm : Q-Learning could be more useful if we were facing a problem where the policy changes a lot, because SARSA would insist on using a policy that would not be valid; so it depends on the context. The SARSA algorithm is more consistent with the FrozenLake environment because the policy (Q-Table) doesn't change significantly enough between two iterations : as a consequence, Sarsa's prediction for action2 is at most cases good, because the policy remains globally the same. If we put some parameters that could change significantly the Q-Table at each iteration, then the Q-Learning algorithm would fit better.

References

[1] Deep reinforcement learning demistified. https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa.

[2] frozen-lake repo. https://gist.github.com/adesgautam/1b298bada0e707acae32ea99fbd4c7eefile-frozen-lake$_q - learning_test - py$.

[3] Introduction to various RL Algorithms. https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287/.

[4] SARSA vs QL. https://www.quora.com/What-is-the-difference-between-Q-learning-and-SARSA-learning.

[5] Understanding RL, the Bellman equations. https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/.

[6] Eyal Even-Dar and Yishay Mansour. Learning rates for q-learning. *Journal of Machine Learning Research 5 (2003) 1-25.*