Taxi-Driver

Eloi LAFARGUE

Ryan HEADLEY

Geoffroy HUCK

EPITECH

T-AIA-902-aia_902_group69

19/06/2022

Abstract

The taxi driver game is a five by five grid environment where the taxi (agent) must pick up and deliver a passenger to one of three possible destinations, while minimizing its movements and avoiding crashes and errors. The goal of this project was to explore solving the environment with reinforcement learning, while comparing the results with a brute force method. It was found that the grid search optimized Q-learning method was able to solve the environment with an average of 10 steps, total rewards of 6, maximum cost of -10 and 0 penalties. These results are much better than the brute force method which resulted in averages of 2000 steps, -9500 total rewards, -41500 maximum cost and 1100 penalties over 100 episodes.

Keywords: Artificial Intelligence, Reinforcement Learning, Data, OpenAI

Introduction

The goal of this project was to introduce reinforcement learning through solving and optimizing the taxi game, provided through the *gym* library of OpenAI. The taxi-v3 environment is simple: pick up passengers and drop them off at their desired destination in the shortest time possible, without crashing into walls. The environment is shown in *Figure 1* below

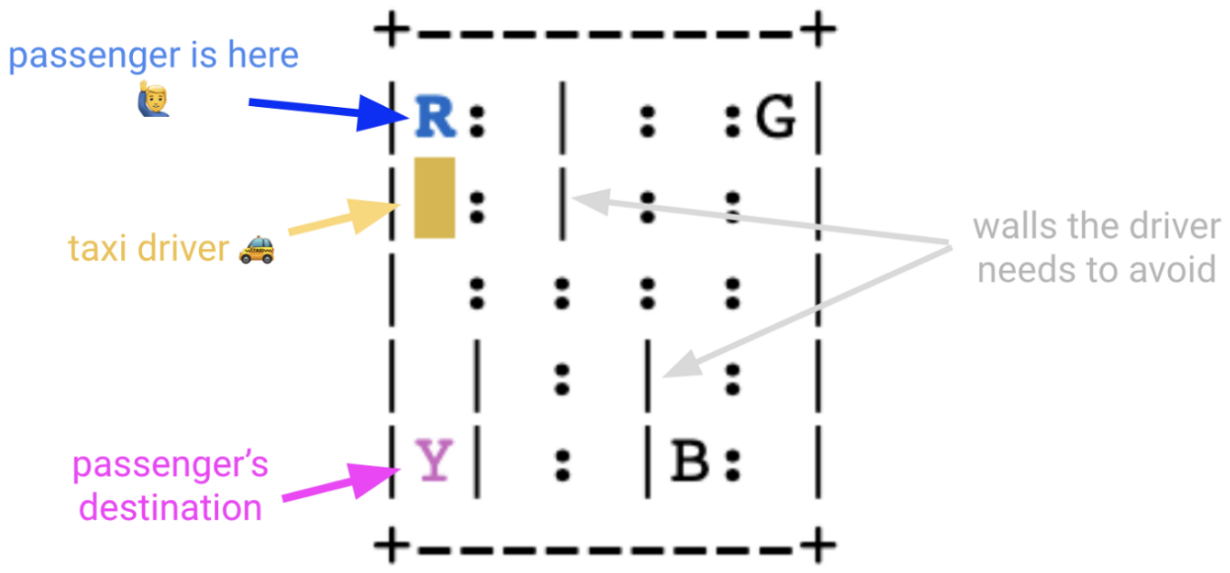


Figure 1: Taxi-v3 environment by OpenAI

Where the yellow square is the taxi, the vertical lines are the walls and the R, Y, B and G represent the four random locations that can contain a passenger. This is a 5x5 grid meaning there are a total of 25 possible taxi positions. The passenger can be at any of the four states *plus* inside the taxi, and therefore four possible destination locations. In total that gives 25 x 5 x 4 = 500 possible states.

The rewards are as follows:

- -1 for every step
- +20 for delivering the passenger to the correct destination
- -10 for doing a pickup or dropoff at the wrong location

Method

Participants

This project was equally completed by Eloi Lafargue, Geoffroy Huck and Ryan Headley. All questions were directed to Amine Bachri.

Assessments and Measures

Throughout the project, several methods were applied to validate an effective algorithm. For each method, an algorithm was produced as well as an analysis of the results. The following subsections explain the methodology applied to each algorithm, excluding the results which are provided afterwards.

Naive Brute Force

The first step of solving a problem is to start simple. Therefore the first solution written was a naive brute force method that runs an infinite loop, choosing random actions until the agent successfully completes the environment. While a solution was always found, it was quite clear that the solution time and inconsistency of the algorithm were far from ideal.

Q-Learning

The next method applied was a Q-learning algorithm. This is a reinforcement learning method that creates a table of all possible states and actions, with a q-value function that calculates the probability of each action at the current state.

The q-value function is better known as the Bellman function, which can be seen in the formula below:

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha (r_t + \gamma * max(Q(s_{t+1}, a) - Q(s_t, a_t))$$

Where Q represents the q-value, s the state, a the action, α *the learning rate, r the reward and* γ *the discount factor.*

In this formula, the new q-value is calculated by taking the q-value at the current state for the selected action, and adding the current reward plus the expected reward at the next state for all possible actions. In other words,

The bellman function, at the current time, calculates the long-term reward for an action by taking the previously calculated reward at this moment in time, and adding the expected reward from future actions.

Several hyperparameters are added to tune the learning of the agent, such as the learning rate α, and the discount factor γ.

The algorithm can be trained to iteratively improve the table through learning the correct actions. In this way, initially the agent will be exploring its new environment but later exploiting the information that it has learned from its previous runs. The exploitation vs exploration conundrum is a large problem in reinforcement learning, and Q-learning solves this problem by providing a parameter, epsilon, to play with this ratio. In addition, it is a relatively simple algorithm to learn and implement while holding the potential to yield excellent results.

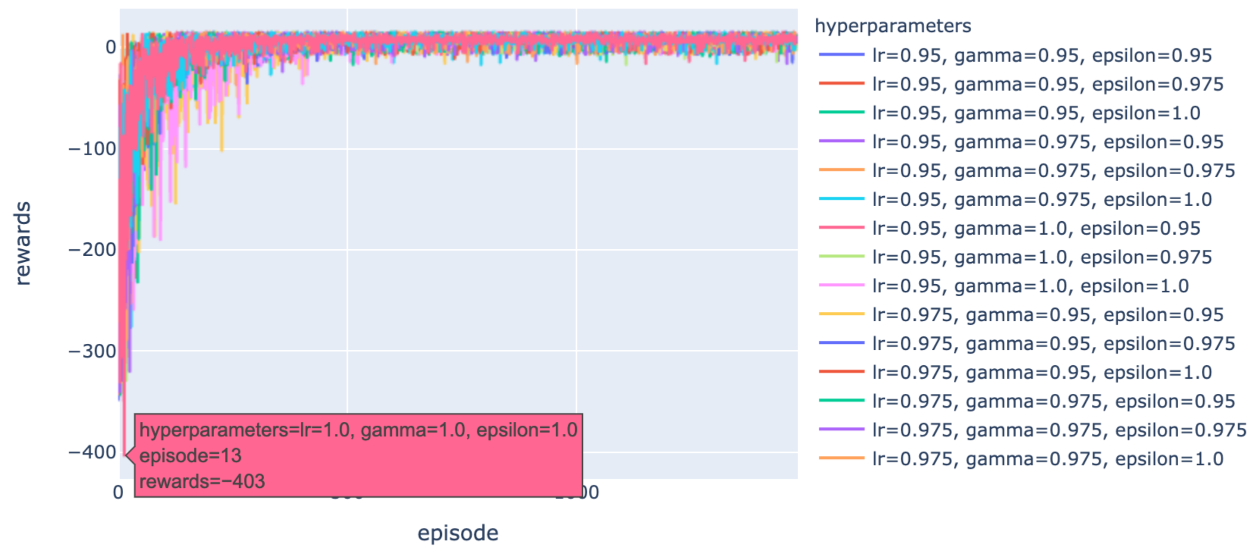
Grid-search

After each reinforcement algorithm was written, the hyperparameters were optimized using a grid search algorithm. Grid search is a method where a range of values is passed for each hyperparameter, and a loop runs through and trains the algorithm on each possible combination of the hyperparameters. The results for each iteration were stored in a DataFrame, which was later analyzed to compare and detect the optimal combination of hyperparameters.

The hyperparameters chosen were the learning rate, gamma and epsilon values. These were chosen since each hyperparameter affects the way in which the agent calculates the q-values: the learning rate determines the confidence at which the algorithms prefers what it has learned; the gamma is the discount factor, which reduces the value of the current reward learned by the agent; the epsilon is the initial value splitting the agents decision between exploration and exploitation.

Analysis

For all algorithms and optimizations, the *plotly* library was used as an interactive plotting library. This permitted the hiding and showing of superimposed plots to easily detect the best result. *Figure 2* compares a plot before and after hiding different results.



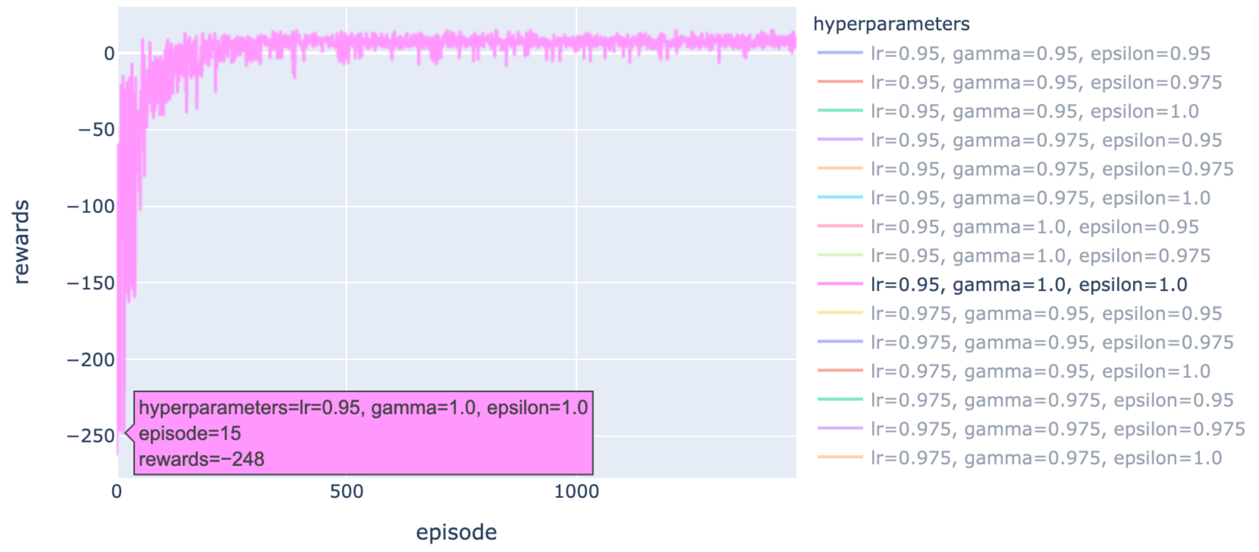


Figure 2: Comparing the highest total cost during the grid search optimization with plotly. The first image shows all hyperparameters combinations with a high total cost of -403. The second image filters through all combinations to show the lowest total cost of -248, which was undetectable in the first image.

Thanks to plotly, the algorithms and the gridsearch optimization were visually represented and easily reduced to the key indicators.

Program

The first goal of the project was to create a reinforcement learning algorithm to effectively beat the taxi-v3 environment. Afterwards, the next goal was to create a program to allow a user to test the offered solution and also modify the parameters to view new results. This was done in two modes: time mode and user mode.

Time mode. Allows a user to select the number of episodes that will be run to solve the environment. This method uses the pre-calculated q-table that was trained with the optimized parameters, and therefore offers the best solution encountered by our team.

User mode. Allows a user to modify all parameters of the algorithm, and re-calculate the q-table. This way, the offered solution can be optimized and/or challenged.

The program was written in a jupyter notebook, since it provided the fastest and most efficient way to write, test and deploy the program.

Results

The following section describes the results encountered throughout the training and inference of the reinforcement learning model and brute force method. While many indicators could be measured, the key performance indicators (KPI) were found to be:

- Maximum cost
- Average timesteps per episode
- Total penalties
- Leveling out of the reward function

Brute Force Method

The brute force method was found to be extremely inefficient at solving the environment. Each indicator was calculated over an epic of 100 episodes. An example epic of the rewards can be seen in *Figure 3*.

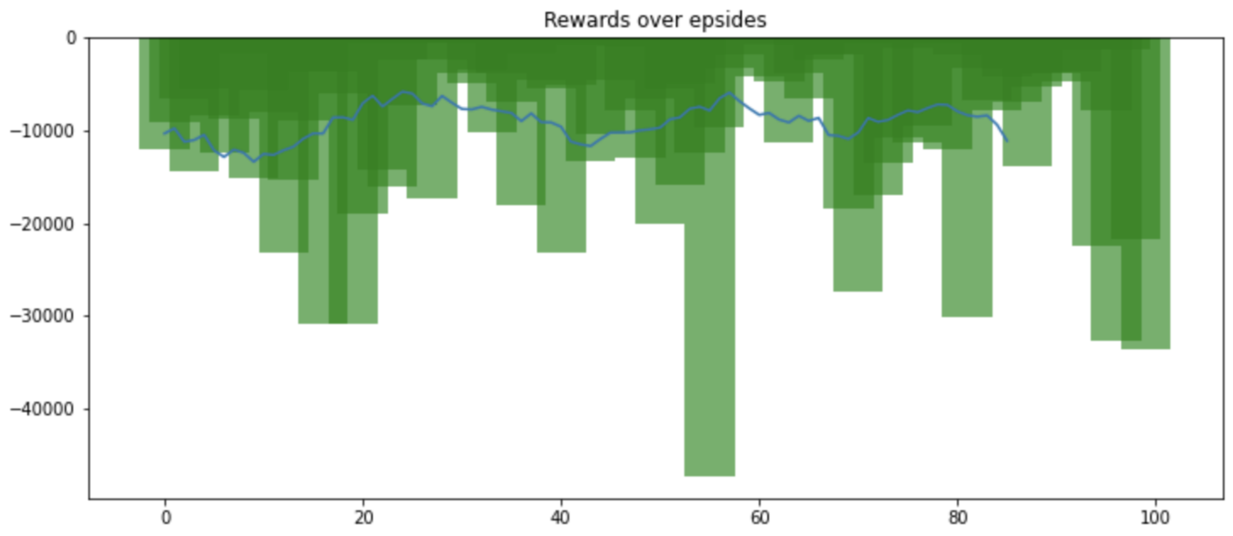


Figure 3: The total rewards per episode of the brute force method. A blue line represents the moving average over a window of 3 episodes.

After running and logging the average of these results, the average of all the runs was taken and can be seen in *Table 1* below.

Average rewards / episode	- 9 510
Average steps / episode	2 000
Average penalties / episode	1 100
Maximum cost	- 41 500

Table 1: The brute force method was run over 100 episodes, and the average for each indicator was noted. After running the method five times, the average was taken over each resulting average.

Q-Learning

Even before optimization, the Q-Learning method was found to be much more effective than the brute force method. An example of the total rewards for an epic can be seen in *Figure 4*.

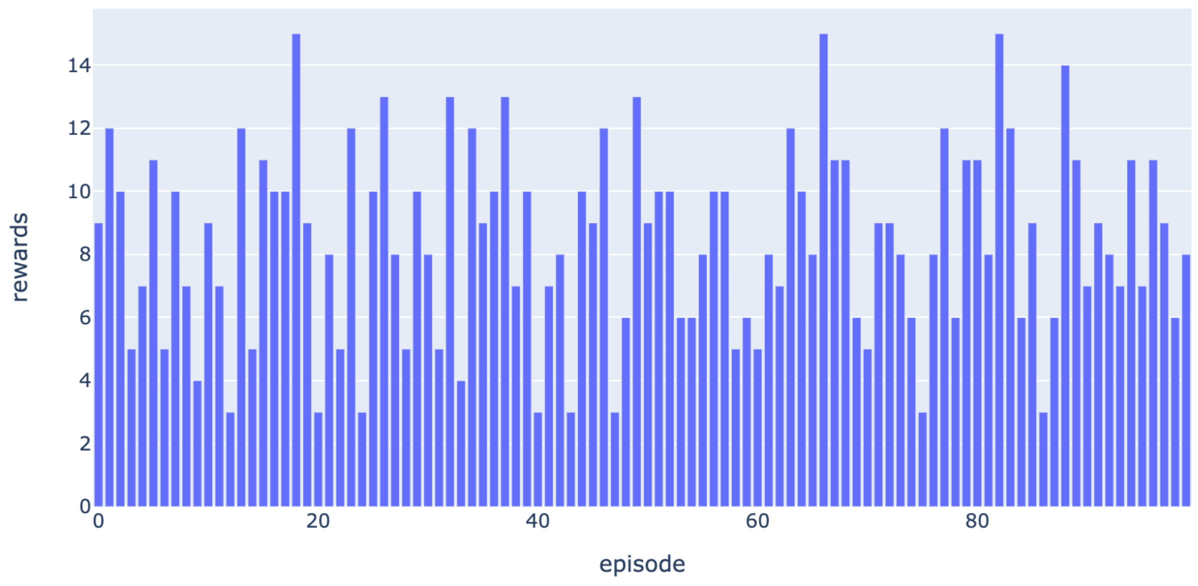


Figure 4: Total rewards over 100 episodes of the Q-Learning method.

The same methodology was used to gain an average of the KPIs over several epics, and the results are shown in *Table 2*.

Average rewards / episode	6
Average steps / episode	10
Average penalties / episode	0
Maximum cost	-10

Table 2: The Q-Learning method was run over 100 episodes, and the average for each indicator was noted. After running the method five times, the average was taken over each resulting average.

The gridsearch was run over the learning rate, gamma and epsilon hyperparameters, resulting in several hundred episodes per epoch depending on the range applied to each hyperparameter. Throughout the optimization of the Q-Learning method, all KPIs were found to level out within the first 100 episodes.

Discussion

While this project successfully determined that reinforcement learning can provide a much more effective algorithm than simply solving complex problems by the brute force method, there is still much left to be discovered. For example, reinforcement learning is full of several type of algorithms, and even though Q-Learning was proven to be a good solution, it cannot be said that it is the best. The results should be compared by applying the same methodology to other reinforcement learning methods such as the SARSA algorithm.

In addition to testing other reinforcement learning algorithms, other machine learning algorithms should be tested as well. The best solution is always the simplest solution, and it could be found that a less computationally expensive machine learning method could result in the same or better results.

References

Taxi game image - https://miro.medium.com/max/1400/1*toX5ZWcYxXtsaXE6hQJ1ag.png