IBM Deep Learning and Reinforcement Learning

Course Final Project:
Package Pick up and Drop off route optimization with Reinforcement Learning

By Javier A. Jaime-Serrano

August 2nd, 2021

Abstract

Consider the scenario where a package delivery business is looking to optimize the routes taken on every service, in order to reduce the time it takes and the associated costs per delivery.

Route optimization algorithms aim to solve the most difficult computer science problems, like the travelling salesman problem [1], by finding the most cost-effective route for a set of stops.

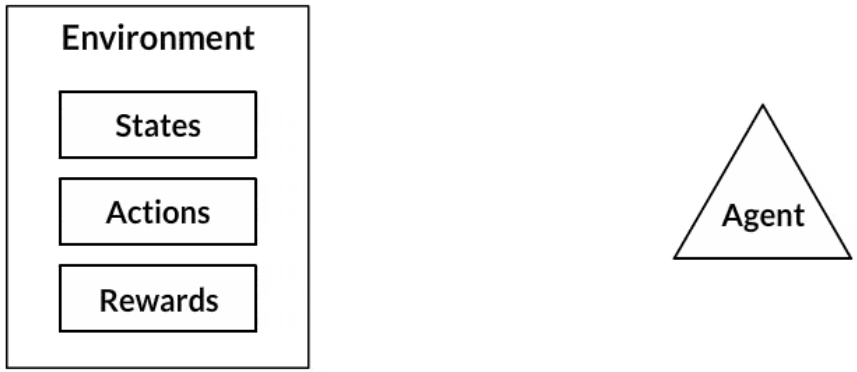
On this Project we will use a Reinforcement Learning algorithm (Q-learning) to find the best route between only two points (Pick-up and Drop-off), by adapting the OpenAI Gym Environment Taxi-V3 [2].

Reinforcement Learning Process

The process of Reinforcement Learning involves these simple steps [3]:

1. Observation of the environment
2. Deciding how to act using some strategy
3. Acting accordingly
4. Receiving a reward or penalty
5. Learning from the experiences and refining our strategy
6. Iterate until an optimal strategy is found

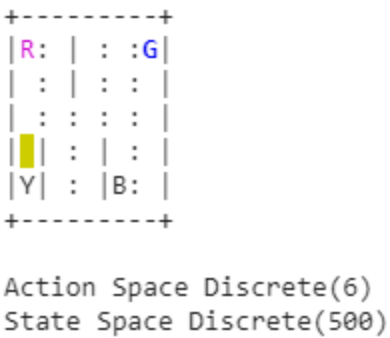
Figure 1.



Data Simulation

With OpenAI Gym (Framework) make function, we first build a new environment based on Taxi-v3 [2], and use the render function to show only one frame.

Figure 2.



State Space:
Is the set of all possible situations:
We have 4 pick up and drop off locations: R, G, Y & B.
On a 5x5 grid and 4+1 package states for a 500 possible states.
(5 × 5 × 5 × 4 = 500)

Action Space:
We have 6 possible actions:
0 = south
1 = north
2 = east
3 = west
4 = pick-up
5 = drop-off

- The blue letter is the current pickup location, and the purple letter is the current destination.
- The filled square represents the vehicle, which is yellow without a package and green with one.
- The pipe ("|") represents a wall which the vehicle cannot cross.

Implementation

Problem Statement:
There are 4 locations, and the job is to pick-up a package at one location and drop it off at another location.

Rewards:
We gain +20 points for a successful drop-off, and lose 1 point for every timestep.
There is also a 10 point penalty for illegal pick-up and drop-off actions.

Reinforcement Learning will learn the optimal action to perform in a learned mapping of the states by exploration. The optimal action for each state is the action that has the highest cumulative long-term reward.

Implementation

At creating the environment we also created a reward table, this is in the form of a Python Dictionary with the structure:
{action: [(probability, nextstate, reward, done)]}

Figure 3.

```
env.P[344] # Dictionary with structure
```

```
{0: [(1.0, 444, -1, False)],
 1: [(1.0, 244, -1, False)],
 2: [(1.0, 344, -1, False)],
 3: [(1.0, 324, -1, False)],
 4: [(1.0, 344, -10, False)],
 5: [(1.0, 344, -10, False)]}
```

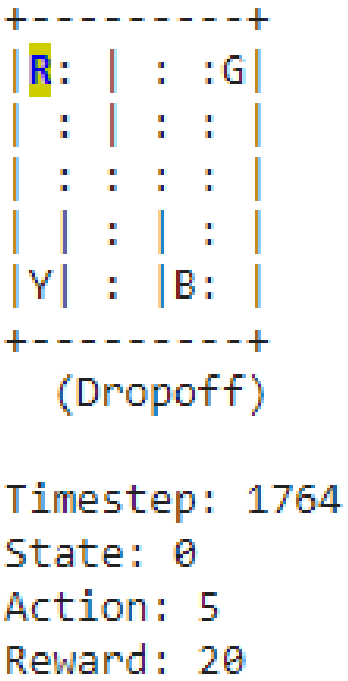
Solutions

We first tried to solve the environment with "brute force". The .sample() method automatically selects one random action from the action space. So we created infinite loop that runs until one package reach the destination (only one episode), and we receive a reward of 20 points.

Outcome:
Timesteps taken: 1764
Penalties incurred: 539

This is not Good!

Figure 4.



Solutions

We used the Q-Learning algorithm from Reinforcement Learning which give the agent some memory to learn the best action to take in a given state.

We updated the Q-values in the reward table (Q-table) with the equation:

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\left(reward + \gamma \max_{a} Q(next\ state, all\ actions)\right)$$

Where:

- α (alpha) is the learning rate ($0<\alpha\leq1$)
- γ (gamma) is the discount factor ($0\leq\gamma\leq1$)

The Q-table is a matrix where we have a row for every state (500) and a column for every action (6) see figure 5.

Solutions

Figure 5. Q-Table

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Findings

After training the agent with 500,000 episodes in total of 4min 22s, the Q-Table is updated with the q-values, where the max value is the action to take in a state. Figure 6.

```
# From the Q-Table if we are in the state 344 from above, we should go north

print(q_table[344], "max value", max(q_table[344]), "is in place 1, this is north")
```

```
[ -4.4426257   -3.75515122  -3.82461653  -4.44523844 -12.8157211
 -12.80997337] max value -3.755151224845913 is in place 1, this is north
```

We also evaluated the agent with 1000 episodes, and obtained an average of 12.9 timesteps per episode with no penalties, a considerable improvement!

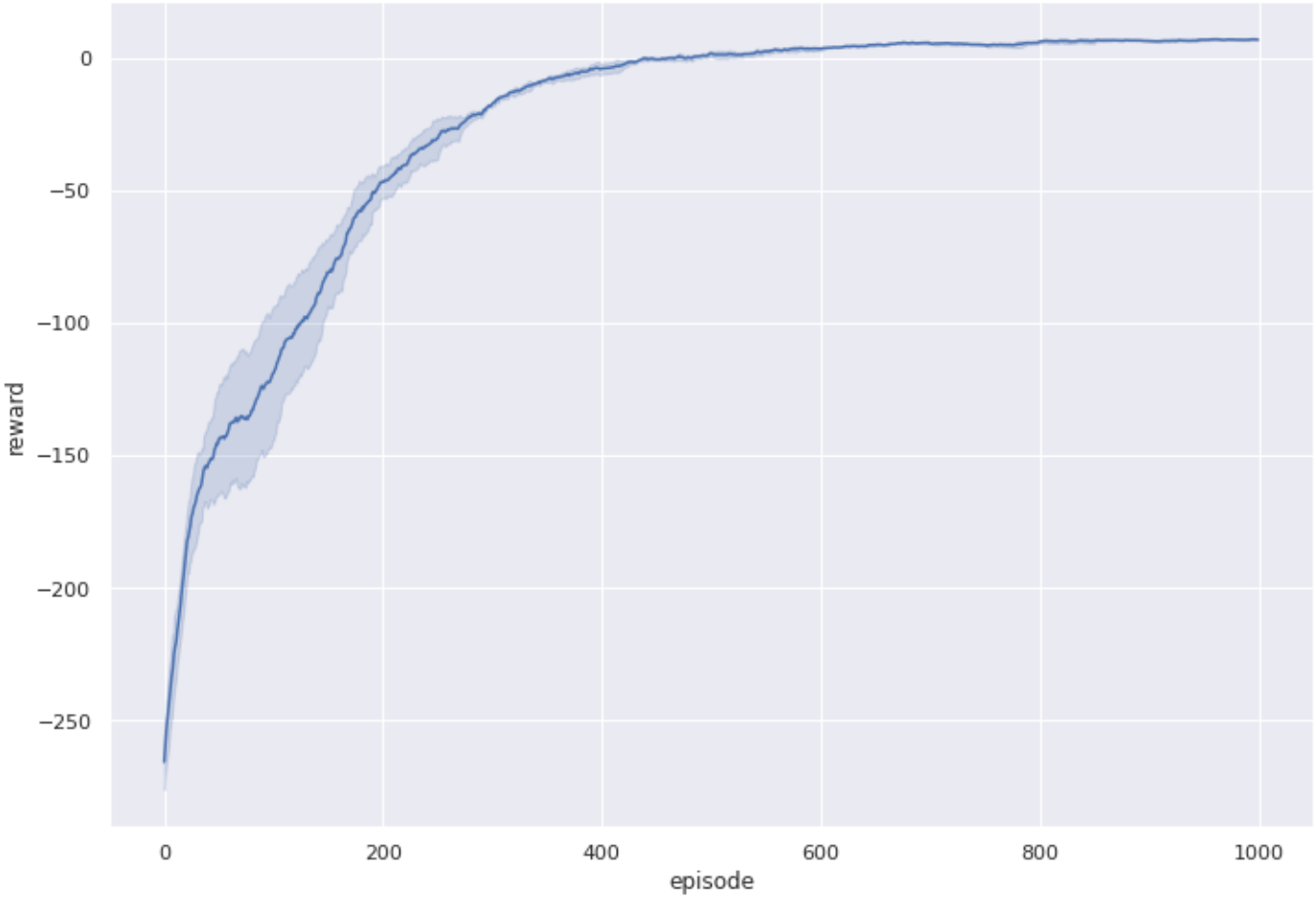
Findings

The values of alpha, gamma and epsilon at first were the same as in the tutorial[3] trial and error approach, but we found better ways to find good values.
Ideally these values should decrease over time as the agent continue to learn.

We found the best set of values with a search function (grid search) that loop over a range of parameters and test it with another function that optimize the rewards per episode, in this case using the Sarsamax algorithm that allows the decrease of the epsilon parameter over each timestep, see figure 7.

Findings

Figure 7. Reward vs. episodes



Results

We evaluate the agent with the metrics in Table 1. The reinforcement Learning algorithms initially make mistakes during exploration, but once it has seen most of the state, it start maximizing the rewards. Q-Learning is considerable better than the Random agent, and the hyperparameter tuning is also a considerable improvement, varying the epsilon with Sarsamax we found the most rewards.

Table 1.

Measure	Random (brute force) performance	Q-Learning (with given parameters) performance	Hyperparameter Tuning (Sarsamax) performance
Average rewards per move	-0.29	1.55	8.72
Average number of penalties per episode	539	0	0
Average number of timesteps per trip	1764	12.9	2.29

Conclusions

In this Project we framed the route-optimization of the package delivery business, as a Reinforcement Learning problem. Then we used the OpenAI Gym Taxi-v3 environment, where we were able to train and test our agent. We compared first our Q-Learning agent agains a Random agent with a considerable improvement. We also used functions for Hyperparameter tuning and to find the most rewards.

This is just a demo in a simulation environment. The issues with the Q-Learning algorithm start to show as the number of states increase and make it difficult to implement in a table, in this case the use of Deep Learning will be recommended instead of the Q-Table.

This is proof that Reinforcement Learning algorithms are useful beyond games.

References

[1] Travelling Salesman Problem:

https://en.wikipedia.org/wiki/Travelling_salesman_problem

[2] Open AI Gym Taxi Environment: https://gym.openai.com/envs/Taxi-v3/

[3] Reinforcement Learning Tutorial: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

[4] Jupiter Notebook: https://github.com/javier-jaime/Reinforcement_Learning_Demos