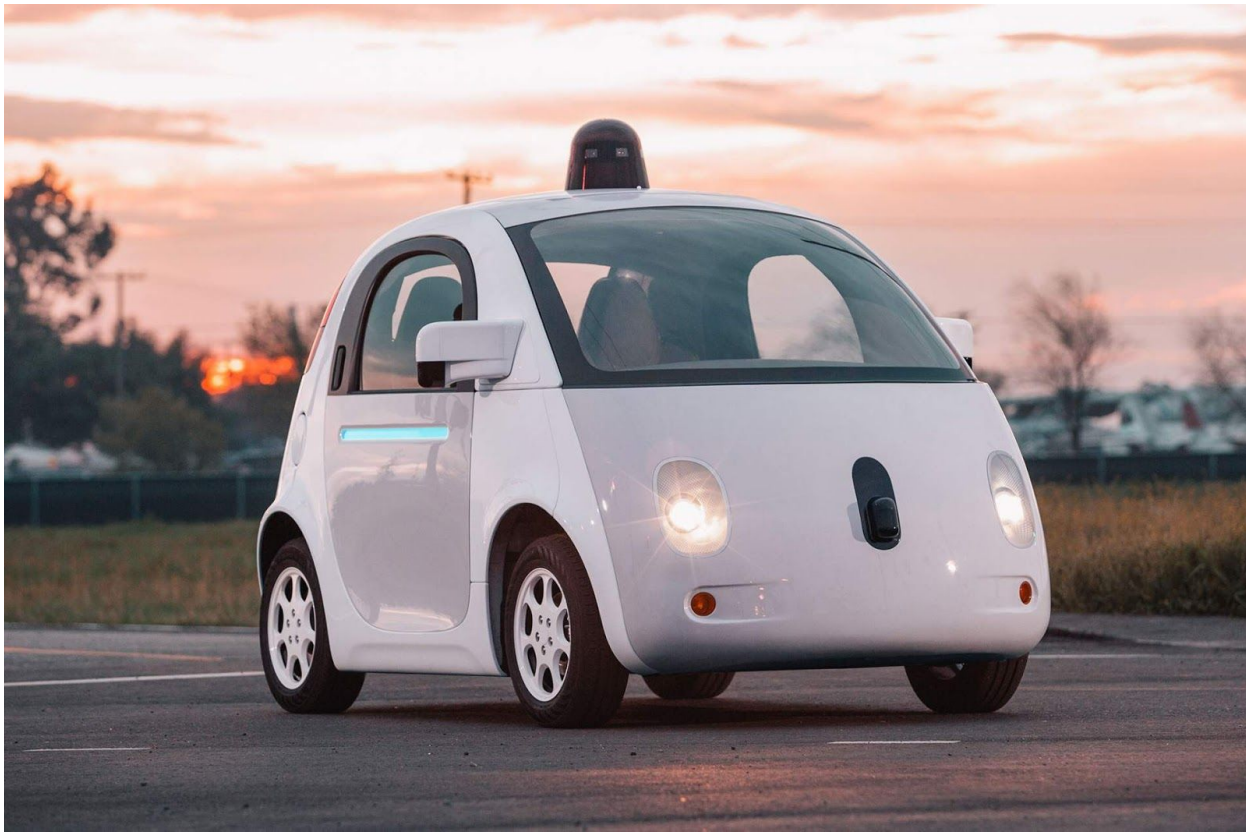


P4 - TRAIN A SMARTCAB TO DRIVE

14.06.2016

Sergio Gordillo
https://github.com/archelogos/train-smartcab

Questions

Step One: Implement a basic driving agent

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

In the first case a random-action driving agent is implemented. This means that it's not actually taking into account any input from the environment. In addition, it ignores the rewards or penalties provided by the planner so as it was said it acts like an independent random agent. However, it sometimes reaches the final goal because there are finite states and the universe created by these states is discrete. For that reason, the probability to get the goal is greater than 0 and it eventually reaches the target location.

If the Q-Learning algorithm was implemented in this step, it would be described as an example of Q-Learning with epsilon equal to 1, where the agent is trying to learning the whole time and forgets everything about its own experience.

Step Two: Identify and update state

Justify why you picked these set of states, and how they model the agent and its environment.

The states chosen are related to the inputs provided by the planner. It could be thought that a good set of states could be the absolute position of the car in coordinates, according to the grid where the car is moving around. However, the agent is acting as an independent and naive individual which is only able to consider the inputs provided by the planner.

The inputs given are related to the current traffic situation in a particular crossing and the next desired waypoint to move.

As it was said, the traffic situation of the intersection is not taken into account by the planner in terms of giving a reward or a penalty. For that reason, the inputs 'oncoming', 'left' and 'right' were not chosen as "features" of a state. In addition to that, the deadline parameter is useless to take decisions about the action taken by the agent for the same reason.

Explaining it better, since the planner doesn't punish the agent based on the value of the deadline, it is useless to consider that input as a part of the state.

According to that, the inputs “next_waypoint” and “light” were chosen as “features” of the states, describing 8 possible states.

S_0: light = “green”, next_waypoint = “None”

S_1: light = “green”, next_waypoint = “forward”

S_2: light = “green”, next_waypoint = “left”

S_3: light = “green”, next_waypoint = “right”

S_4: light = “red”, next_waypoint = “None”

S_5: light = “red”, next_waypoint = “forward”

S_6: light = “red”, next_waypoint = “left”

S_7: light = “red”, next_waypoint = “right”

Step Three: Implement Q-Learning

What changes do you notice in the agent’s behavior?

It’s clear that as soon as the agent takes actions according to a reward-penalty algorithm, the probability of reaching the destination increases. It can be seen how based on the value of epsilon the agent sometimes makes a decision to continue learning and in other cases it just takes advantage from the learning and tries to move to the next point the get a reward.

It is surprising how even without any logical programming, the agent learns and reaches the destination in most of the cases.

Step Four: Enhance the driving agent

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

It’s required to find a good set of values for the parameters alpha, gamma and epsilon. In order to reach the goal, there were made more than 50 executions based each one on 10 simulations.

Once a group of parameters that fit the specifications has been found, (penalties are not allowed beyond 90th trial and a high success score), it was made another set of 50 executions with 10 simulations in order to find the optimal values for the parameters.

This can be seen in the results.csv file where the results of the executions were saved.

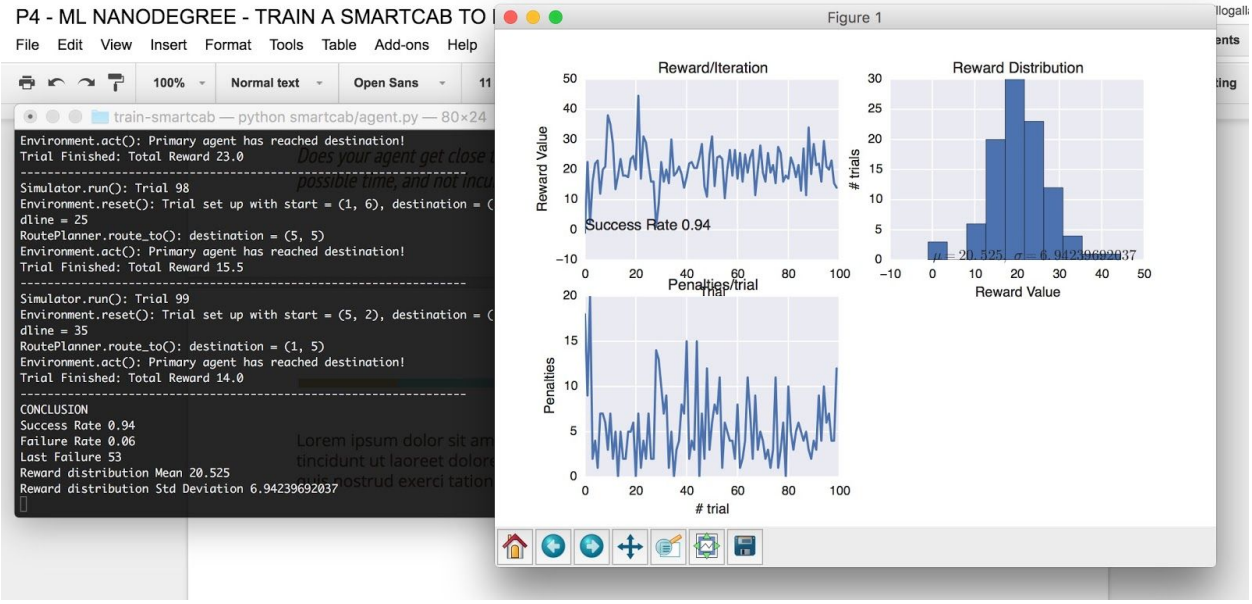
Since a decay_factor is applied to epsilon, the initial value of this parameter doesn't have a big effect on the final results.

At the end, the agent will always prioritize an action instead of another based on its own experience more than try to choose a random action.

For that reason, the exploitation-exploration tradeoff is properly managed in this problem by doing Epsilon smaller when the deadline decreases.

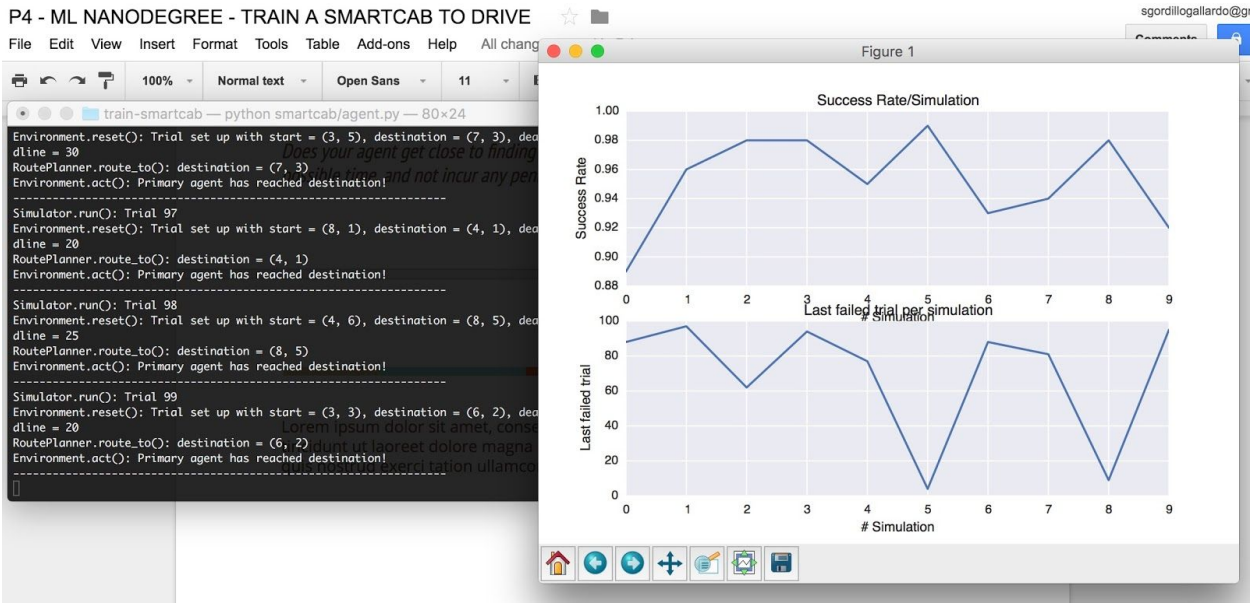
It is important to set a correct value of gamma since it's going to be static along the entire simulation.

Making simulations and tuning the value of alpha allow us to get a final score of success over 90% in series of 10 or 50 simulations.



FOR ONE SIMULATION: *success_rate ->* ***96%****; last_failure ->* ***Trial 53***

Total_Reward/Trial; Reward_Distribution/Simulation; Total_Penalties/Trial



FOR TEN SIMULATIONS: Success_Rate/Simulation (over 90% almost always)

Last_Failure (under trial number 90 almost always)

Setting the global variable N_SIMULATIONS from 1 to N it can be seen these reports.

(Default value: 1)

Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?

In general the success rate of the agent is very high (>90%) with high rewards too (mean of 20 for the reward distribution of a trial) however it's very difficult to have a stable and consistent "last failure" under 90.

It can be said that approximately the 20% of the simulations fail beyond the 90th trial, although the success rate is definitely good enough with a stable value greater than 90%.

In conclusion, the agent learns properly and really fast and almost always reaches the goal with an spectacular success rate. However, it sometimes makes a mistake with the corresponding penalty. It's clear that the tendency of the points of interest become steady with the Q-Learning method but sadly it is not perfect.