# Machine Learning Engineer Nanodegree - Udacity

Report for Project 4 - Smartcab

Gabriel Ilharco Magalhães

## Implement a Basic Driving Agent

*QUESTION: Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?*

R: The smartcab acts completely randomly choosing actions with uniform probability between None, Left, Right and Forward. The agent is sometimes able reach the destination before the hard deadline out of pure randomness. Most of the time, however, it just moves randomly until the hard deadline is reached.

## Inform the Driving Agent

*QUESTION: What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?*

*OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

R: For modeling the problem, the following information was chosen to compose the state:

- Lights state
- Oncoming traffic direction
- Left traffic direction
- Right traffic direction
- Next waypoint

Inputs of light and traffic gives the agent the ability to sense the environment and how one decision it makes in that state respecs or not the traffic rules. The next waypoint gives the agent the sense of direction, and is very important for it to learn how to consistently reach the goal. Deadline was not added as part of the state for two reasons: the first of them is that it makes learning much more expensive, since the learner would need to learn good decisions for a much larger number of states. Additionally, it shouldn't matter much for the learner the actual

deadline, since that considering a given state, what the learner should be aiming for is an action that leads it closest to its goal, respecting all traffic rules. For example, in a given state, it shouldn't matter if the deadline is 30 or 40 steps away, it should take the same decision.

There are 2 possibilities of light state, 4 for oncoming, left and right actions and *4* possible next_waypoints, for a given waypoint. This gives us a total of *2x4x4x4x4 = 512* total possible states. Since we are dealing with a gridworld problem, the average complexity should be *O(en)*, where *e* is the total number of actions and *n* is the size of the state space[1]. In our case, for *e = 4* and *n=512*, it isn't expected to be a very computationally expensive operation, and should run in the order of milliseconds.

## Implement a Q-Learning Driving Agent

*QUESTION: What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?*

R: After implementing Q-Learning, the agent seems to learn well about the actions it should take to maximize the reward. The first guess for the parameters were alpha = 0.5 and gamma = 0. The later can be explained because a greedy algorithm should (intuitively) work fine for this problem. in, this context, it is ok to learn how to maximize short-term rewards instead of long-term, because the first thing we are aiming at is not to break traffic laws (which are translated to immediate rewards), and not reaching the goal in the fastest possible way, for example (which are translated to long-term rewards). Compared to the agent that took random decisions, performance is significantly better. The new implemented agent seems to learn about the traffic rules, especially avoiding to cross red lights, and does a pretty good job reaching the goal. This smart behaviour is due to the fact that Q-learning makes the agent maximize its reward, avoiding penalties for inappropriate actions. I modified the code so I could measure the rate of successful runs. Running the simulation 10000 times, the agent was able to successfully reach its goal before the deadline in 9984 times, which represents a success rate of 99.84%. This is because the algorithm is inherently greedy as the agent receives a reward for each correct action and a penalty for an incorrect move or a traffic violation. The agent also receives a large reward for reaching the target in time. This is distinctly different finding the shortest or fastest possible route to the target - as long as the agent reaches the target location within the allocated number of moves denoted by the variable `deadline`, it received a large reward of 12.0. Therefore, it was possible to exploit the greediness of the algorithm by maximizing for short-term gains of the next best action instead of the long-term goal of reaching the target the fastest. This was inline with the project goal as well

[1] Complecity Analysis of Real-Time Reinforcment Learning - http://idm-lab.org/bib/abstracts/papers/aaai93.pdf

## Improve the Q-Learning Driving Agent

*QUESTION: Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?*

*QUESTION: 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? How would you describe an optimal policy for this problem?*

R: After implementing Q-Learning, the parameter alpha was tuned. For each value between 0 and 1, with a step of 0.1, it was simulated 10000 runs with alpha equal to that value, and the success rate was computed. Here are the results:

| $\alpha$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 |
|---|---|---|---|---|---|
| Success Rate | 0.1984 | 0.9982 | 0.9987 | 0.9996 | 0.9992 |

| $\alpha$ | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|
| Success Rate | 0.9984 | 0.9991 | 0.9988 | 0.9985 | 0.9987 | 0.9985 |

Note best alpha found was 0.3, which gave a success rate of 99.96%. Interestingly, the success rate is very high for most values of alpha, with the exception of 0, in which case the agent never learns anything and isn't able to perform much better than the random agent.

A truly optimal policy is one where the agent takes the shortest route to its goal, while respecting all traffic rules. That is, the main objective of the optimal policy is obeying all traffic rules. Whenever it is possible to do so, the optimal agent should follow the planner's next waypoint, if that doesn't make it break any traffic rule. The implementation of Q Learning makes the agent predict, at a given state, the best possible action to maximize its reward, and, since the reward is directly associated with the rules, it is expected that the agent decisions should be very similar to the optimal ones. In fact, that is the observed behaviour of the agent in the simulation. One interesting observation is that even with a lot of training, the agent doesn't seems to learn well how to handle oncoming traffic. This is because, since there are just a few external agents, encounters are not very frequent, so the agent, even after many training iterations, sometimes takes actions that lead to negative penalty. As a suggestion for diminishing this problem, perhaps a good approach would be increasing the number of external agents. In general, the performance of the agent is very satisfactory, and it is able to

successfully navigate through the environment in a very high rate (99.96% for alpha = 0.3), which indicates that the policy found by the Q Learning algorithm is very satisfactory for this problem.