Project 4 - Train a Smartcab How to Drive

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?

- After implementing the most basic driving agent, the agent's behavior becomes very sporadic since there is no method to its movements. The agent is taking a random choice between the actions `(None, 'forward', 'left', 'right')` and moving in no optimal direction. Although the agent does eventually make it to the target location since it is a finite grid, it is not efficient, rewards are not accumulating and the agent usually arrives well passed the desired deadline.

Identify and update state

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

- I chose to represent the states with all of the current inputs, in addition with the next waypoint for the agent. Since a state is something that describe the world, ie. from the lecture: a set of tokens that represent every state the one can be in, the states should be everything that best represent the world. If this was an actual self-driving car in the wild, it would be important to keep everything as the car needs to obey the rules of the road. Therefore, the agent needs to know the stop-light color (red or green), since there is penalty for running red lights, and if we don't want to get in an accident we need to take advantages of every 'sense' of the environment and know about intersection traffic with the inputs of oncoming, left, and right. I also added the next_waypoint from the Route Planner class because we need to take advantage of the best action and will be key in guiding the agent to the desired destination.

Implement Q-Learning

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

- Implementing the Q-Learning algorithm with the best available action increased the effectiveness of the driving agent, certainly after continued trials. The agent became better at finding the optimum direction from the Route Planner, in terms of not running red lights, not going in the opposite direction and reaching the destination more efficiently. The agent, however wasn't perfect as it often did not reach the destination on time and occasionally it would get unlucky and hit a high amount of red lights resulting in high negative reward accumulation. However this initial Q-Learning implementation was still much better than randomly selecting an action and with continued runs the agent began to get better and better at reaching the destination on time.

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?

- From everything I read, it said to initialize the Q matrix to zero, however when I first implemented this the agent didn't move. I believe this is because the agent was simply choosing the None action from the Environment's valid actions, therefore keeping the agent in place. After some tweaking, I settled on 3 for the Q-values, as I didn't want to encourage too much exploration. The next experimental parameters were the learning rate and the discount factor. With the learning rate, with an alpha of 0 the Q-values are never updated thus nothing is ever learned. Therefore, I wanted the opposite of this, trying the learning rate of 0.9 so learning can occur quickly. This actually ended up being a little high as it was almost learning too fast and I eventually reduced it to 0.7. The variable gamma or the discount factor is set to model the immediate rewards for the agent. If gamma is near 0 the agent will only consider immediate rewards and if gamma is closer to 1, the agent will consider future rewards with greater weight, willing to delay the reward. Since, I wanted the best of both, I began by trying 0.5, and eventually reduced it to 0.4 as I wanted a little more emphasis on the immediate rewards, since the agent wasn't performing too well in the beginning trials. After tweaking these parameters and a lot of trial and error, I settled on 0.7 and 0.4, which resulted in the agent becoming very efficient after about 20 trials, and almost perfect after 50 trials.

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?

- I think the agent is close to finding the optimal policy. I'm sure that the learning rate and the discount factor could be tuned even better. I could try plotting the efficiency rate of the agent over time or try performing something like grid search on the hyperparameters. Another idea, would be to also factor in the deadline in the current state and try to force the agent to trade off between faster and slower immediate rewards in the beginning trials and take in account for how many steps it has left to reach the destination.
- On average, the agent has about 75% success rate in the first 30 trial, about a 90% success rate between trials 30 and 60, and after about 60 runs, the agent is closer to perfect with around a 99% success rate. Therefore, by the end of the trials, the agent would resort to either having an action of None as the agent would be at a 'red light' or the agent would move optimally and reach the destination at or near the minimum possible time.