Project 4 of Udacity's Machine Learning Nanodegree

Implement a basic driving agent

Observing the agent's behavior, we see that the agent takes random actions regardless of the conditions of the environment. For example, the agent takes a 'forward' action on a red light, or takes a 'None' action on a green light both of which translate to negative reward. The agent does not learn from this to adjust its behavior.
The agent eventually reaches the destination for most cases, but sometimes it does not. I ran the simulations 10 times (n_trials=10) and the agent was able to reach the destination 6 times out of 10. The smartcab reaches the destination past the deadline each time.

Identify and update state

The following state variable have been identified:

- light: green, red
- oncoming: None, forward, right, left
- right: None, forward, right, left
- left: None, forward, right, left
- waypoint: None, forward, right, left

The total number of states adds up to **2x4x4x4x4 = 512 states**
I believe these states are appropriate for this problem because they represent the conditions of the environment the smartcab can use to learn from. The traffic laws can be interpreted by these states and the smartcab can learn how to obey them. For example, light=green and oncoming=None means it is permitted to make a left turn at the intersection.
The "deadline" state variable was not included because it could have too many different values, hence significantly increasing the number of states our agent would need. The "deadline" is calculated as a factor of the difference between the start point and the destination point. Our environment is a 8x6 grid (total of 48 points), choosing 2 points out of that could be done in 1128 different ways (48 choose 2). Including the "deadline" would mean having over half a million total states (512*1128) which is a lot.

Implement Q-Learning

The agent initially starts off the same way, seeming to take random actions. This is because the Q-table initially starts off with default values. Over time, the Q-table gets updated as the actions taken by the agent yield reward. The movements of the agent start to change and become based on the maximum Q-value of all the adjacent states. This improves the agent's ability to reach the destination by making better decisions.

Over time, the agent accumulates less and less penalties, and more positive rewards. The learning is allowing the agent to reach the destination more frequently.

Enhance the driving agent

My initial implementation of Q-Learning had the following parameters values: alpha=0.9, gamma=0.6, epsilon=0.5. I then used a trial and error approach to tune each parameter by changing its value, while keeping the other parameters values the same, and observing the agent's performance. Below are the results of the improvement process:

- Learning rate (alpha)

alpha	gamma	epsilon	success rate	penalty rate	net reward
0.9	0.6	0.5	0.57	0.39	1661.0
1	0.6	0.5	0.58	0.4	1587.0
0.8	**0.6**	**0.5**	**0.69**	**0.36**	**1872.5**
0.7	0.6	0.5	0.55	0.38	1801.5

- Discount factor (gamma)

alpha	**gamma**	epsilon	success rate	penalty rate	net reward
0.8	0.6	0.5	0.69	0.36	1872.5
0.8	0.3	0.5	0.67	0.34	1821.0
0.8	0.4	0.5	0.68	0.34	1807.0
0.8	**0.2**	**0.5**	**0.76**	**0.32**	**1838.5**

- Exploration rate (epsilon)

alpha	gamma	**epsilon**	success rate	penalty rate	net reward
0.8	0.2	0.5	0.76	0.32	1838.5
0.8	0.2	0.3	0.86	0.21	2119.0

0.8	0.2	0.1	0.99	0.08	2296.0
0.8	**0.2**	**0.05**	**0.99**	**0.06**	**2312.5**

The set of parameters for which the agent perform best are **alpha=0.8, gamma=0.2, epsilon=0.05**. With this set of parameters, the final driving agent is able to reach the destination 99% of the time (for n_trials=100) and has a penalty rate of just 6%.

The final agent performs well, though it is not 100% perfect. The agent consistently reaches the destination, but does not always obey traffic rules. Looking at the last couple of runs, the agent gets penalized very few times for traffic violations such as turning left at a red light, or taking the wrong action like going right when the next waypoint is forward.

My agent is able find a close to optimal policy. As the number of runs increases, I observed the penalty rate decrease, and consequently the net reward increase. This was also true for the reward per action, which indicates that the agent gets better and better at making good decision, hence gets closers to the optimal policy.

An optimal policy for this problem is a policy that follows the right direction at each intersection (i.e. takes the right action) and complies to traffic rules while doing so (i.e. does not incur penalties). This policy would allow the agent to reach the destination in the minimum possible time.