Train a Smartcab 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?

The driving agent in its most basic form, picks a random action out of all the possible actions. This is achieved by the following code:

```
action = random.choice(['forward', 'left', 'right', None])
```

It did not perform very well and the agent reached the destination within the threshold time only 15% of the time.

Identify and update state

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

I chose my state to be the traffic light and the waypoint. I defined it as follows:

```
self.state = (self.next_waypoint, inputs['light'])
```

Choosing the traffic light is important as the agent cannot or should not move if the traffic light is red and choosing the next waypoint is important as well as that determines the direction of the agent. Both these features factor into the calculation of the rewards.

Attributes like oncoming, left, right and deadline are not considered in calculating the reward for an action, so I have not included them in state.

Increasing the state space can lead to the curse of dimensionality problem as having more features translates to the agent having a higher number of states corresponding to the different combinations of each feature. The algorithm will therefore need more time to learn the reward of each combination. Due to the number of states that the deadline variable can take, it can blow up our state space into a size that cannot be feasibly explored by the agent in 100 trials of the game. Adding a feature to the state without considering it in

reward function will make it difficult for the smart car to relate its action to reward. Thereby lowering the agent's performance and lowering the chances of the Q learning algorithm to converge.

Implement Q-Learning

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

Upon implementing Q Learning, there was a stark change in the agent's ability to effectively reach the destination under the threshold time after a few of the initial trials. Most of the failed cases happened within the first 25 trials after which the agent was able to successfully reach the destination under the threshold time with 100% accuracy.

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?

I implemented Q learning with a Gamma value(Discount Factor) of 0.7 to ensure the algorithm was not myopic or short sighted in nature so that it would strive for long term rewards and not so much the immediate reward. I varied the value of alpha and got the following results:

Alpha(Learning Rate)	Gamma(Discount)	Percent of success
0.5	0.7	89
0.6	0.7	89
0.7	0.7	90
0.8	0.7	**93**
0.9	0.7	92
0.9	0.8	90
0.8	0.8	92

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?

Based on my analysis, it looked like increasing the value of alpha(learning rate) caused the algorithm to perform better. The learning rate determines to what extent the newly acquired information will override the old information. A learning rate value of 0 will make the agent not learn anything over each episode, while a factor of 1 would make the agent consider only the most recent information. I received the best performance with a learning rate value of 0.8 and discount factor value of 0.7 where the agent was successfully able to reach the destination under the threshold time in 93% of the cases.

The smart cab is still not able to reach goal 100% of the time and it still violates traffic laws every now and then. It also sometimes does not take the shortest possible path to reach the goal.

The smartcab does not follow the traffic laws in the following trials : Trial 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 21, 27, 28, 29, 31, 32, 34, 35 and 80.

The last 10 trials are as follows:

LearningAgent.update(): deadline = 28, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **forward**, reward = 12.0

LearningAgent.update(): deadline = 23, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **forward**, reward = 12.0

LearningAgent.update(): deadline = 17, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **forward**, reward = 12.0

LearningAgent.update(): deadline = 13, inputs = {'**light': 'red', 'oncoming': None, 'right': None, 'left': None**}, action = **right**, reward = 12.0

LearningAgent.update(): deadline = 8, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **forward**, reward = 12.0

LearningAgent.update(): deadline = 14, inputs = {'**light': 'red', 'oncoming': None, 'right': None, 'left': None**}, action = **right**, reward = 12.0

LearningAgent.update(): deadline = 18, inputs = {**'light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **right**, reward = 12.0

LearningAgent.update(): deadline = 30, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **forward**, reward = 12.0

LearningAgent.update(): deadline = 12, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **left**, reward = 12.0

LearningAgent.update(): deadline = 19, inputs = {'**light': 'green', 'oncoming': None, 'right': None, 'left': None**}, action = **forward**, reward = 12.0