Train a Smart cab to Drive - Report

Implement a Basic Driving Agent

In this part, I implemented a very basic driving agent that let the smart cab to move around. The smart cab will take a random actions from [None, forward, left, right]. I run the simulation 10 times and here is my observation:

- The smart cab always violent traffic rules and receive penalties.
- Disabling the deadline, the smart cab eventually reach the destination once out of 10 trials. In that trial, the destination was only 3 moves away from the start position. However, the smart cab took 30 moves to reach there with many penalties received.
- Car accidents happened frequently.

In this version, the smart cab only reach the destination once in 10 trials with the ignorance of deadline. Moreover, the smart cab broke the traffic rules a lot and followed inefficient route. We need to implement a more robust agent.

Inform the Driving Agent

I use a tuple to represent a state. It is a component of the following elements:

Element	Possible value	Description
`self.next_waypoint`	`[None, 'left', 'right', 'forward']`	`Inform the direction of the destination.`
`inputs['light']`	`['red', 'green']`	`The traffic light signal in the current intersection.`
`inputs['oncoming']`	`[None, 'left', 'right', 'forward']`	`The direction of the car in the oncoming intersection.`
`inputs['left']`	`[None, 'left', 'right', 'forward']`	`The direction of the car in the left intersection.`
`inputs['right']`	`[None, 'left', 'right', 'forward']`	`The direction of the car in the right intersection.`

Example of a state: *('left', 'red', None, None, 'left')*
I select the next_waypoint because it is important to lead the smart cab to the destination. Without that input, the cab will not know where it should go. The traffic light signal is also important to guarantee the cab not to break traffic rules. Finally, having information about the other cars in the next intersection vital to make sure car accident won't happen. I am not going to select deadline to the state as it is not giving any directional information to the cab.

The possible combinations of all the states:

```
4 x 2 x 4 x 4 x 4 = 512
```

The number of possible actions = 4 (None, 'left', 'right', 'forward').
In the next section, we will implement Q-Learning with a (512 x 4) table. If the smart cab met all (512 x 4) situations, we will have a well-learnt agent.

Implement a Q-Learning Driving Agent
In this section, I implemented the Q-Learning algorithm to choose the best actions. To compare this version to the basic version, I run the simulation 10 times with deadline and some guessing value of parameters: alpha=0.9, *epsilon=0.2*, *epsilon=0.5* Here is my observation:

- In 10 trials, the smart cab had 6 successes, and 4 failures.
- The smart cab violent traffic rules less compare to the basic version.
- In the last version, the car moved aimlessly while in this version, the car moved towards the destination.

Improve the Q-Learning Driving Agent
In this section, I fine tuned the Q-Learning Algorithm for a smarter agent. I set the number of trials to 100 and add some lines of code to save the penalties received in each trial. Let's see the statistic before fine tuning:

Performance **before** tuning	
alpha	0.9
epsilon	0.5
gamma	0.2
Success Rate	74%
Total Penalties received	452.50

Output Log:

```
============================================================
SUMMARY
============================================================
Total Penalities received = -452.50
        Penalities received in the first half of trials  = -213.00
        Penalities received in the second half of trials = -239.50
Success Rate: 74.00%
        Success Rate of the first half : 76.00%
        Success Rate of the second half: 72.00%
```

As we can see, the success rate is only 74%. I think one of the reason is that the epsilon is too high. Epsilon is compared to a randomly generating number, if the random number is less than epsilon, a random action is selected, else, the best action from Q-Learning is selected.

We now have epsilon=0.5 so we only have 50% chance to use the strategy from Q-learning. I decrease the value to 0.1 so we have higher chance to use the learnt strategy. I finally come up the following set of parameters that perform the best.

Performance **after** tuning	
alpha	0.9
epsilon	0.1
gamma	0.2
Success Rate	97%
Total Penalties received	91.50

Output Log:

```
============================================================
SUMMARY
============================================================
Total Penalities received = -91.50
        Penalities received in the first half of trials  = -46.00
        Penalities received in the second half of trials = -45.50
Success Rate: 97.00%
        Success Rate of the first half : 94.00%
        Success Rate of the second half: 100.00%
```

It is worth to note that the success rate of the first 50 trials is 94% while the second half is 100%. It shows that more trials made the agent learnt smarter as more feedbacks are given.

97% is a very good number already. However, I think it is still not an optimal solution. An optimal solution should guarantee zero penalties and is able to reach the destinations with the minimum possible time. This is the optimal strategy that I come up with:

Optimal Strategy

Always follow the direction of the next waypoint unless the action is breaking traffic rules.
 	
`If (breaking_rules):`
`    action = None`
`else:`
`    action = next_waypoint`
 	
`breaking_rules` is `True` if any of the following satisfies:
• `Input['light'] = 'red'`
• `(next_waypoint=='left') && (inputs['left']=='forward')`
• `(next_waypoint=='right') && (inputs['right']=='forward')`
• `(next_waypoint=='forward') && (inputs['oncoming']=='forward')`

My agent is approaching the optimal solution. The difference is that

- My agent does not always follow the next_waypoint, it still has a little probability to choose a random action that makes it far away from the destination or even fail to reach the destination in that trip.
- My agent sometimes break the rule as well, also because of the random action taken.
