Training a SmartCab

Implementing a Basic Driving Agent

Although the agent has a choice of four available actions - `None, forward, left, right` - it only makes sense to either go to the next waypoint or stay at the current location due to the reward structure. This is my solution for the first task:

```
action = random.choice([None, self.next_waypoint])
```

Informing the Driving Agent

At each intersection the smart cab receives the following inputs:

- Next Waypoint (North, South, East or West from current location)
- Traffic Light State in direction of Travel
- Vehicles from North
- Vehicles from South
- Vehicles from East
- Vehicles from West
- Deadline

The policy of the smart cab will have to maximize rewards, while reaching the goal in time. The smart cab can receive the following rewards:

- **++** — For each successful completed Trip
- **+** — For each action taken, that follows the planner
- **-** — For each action taken, different from the planned
- **- -** — For each illegal action taken

The smart cab drives in a homogenous environment. Thus, the smart cab's actions are not influenced by the current location. That's why I decided to exclude the global location from my states. This allows me to derive a reasonable policy within 100 iterations, like the rubric demands.

I originally mapped the input states to the traffic rules (i.e. risk_levels). I found that summarizing the inputs into risk levels was hurting performance and only resulted in about 85% trip completion. I decided to remove the risk level and include the traffic inputs in my states.

I originally included the remaining time in bins in my states. My thinking behind this was that the algorithm would choose to perform a move that violated traffic rules but wouldn't cause an accident. However, the environment does not allow for this behavior. That's why I ultimately decided to exlclude the deadline from my states.

This information is necessary, however, because the algorithm only has to determine whether the move to the next waypoint is legal or not based on the current traffic situation.

Final States:

- *Next Waypoint*
- *Traffic Light*
- *Traffic from Left*
- *Oncoming Traffic*
- *Traffic from Right*

Implementing a Q-Learning Driving Agent

I went back to the videos with Michael and Charles and went over the section on Q-Learning on more time. For the brain of the agent (the Q Matrix) I decided to use pandas, because DataFrames are fast and easy to use, allow for robust indexing and are easy to export for analysis. The structure of the Q Matrix is as follows:

	stay	**waypoint**
state 1	0	0
state 2	...	...

With this DataFrame set up, I implemented the q-learning update formula:

```
self.q_values.loc[self.state, action] = (1 - self.alpha) * self.get_qvalue(s
elf.state, action) + self.alpha * (reward + self.gamma * self.max_q(new_stat
e))
```

The next problem that I addressed was the exploration-exploitation dilemma. I decided to perform random restarts with the degrading probability `self.epsilon`. In order to address the problem of local optima I implemented a function that given a state will prefer actions that have not been taken over actions with a high q value.

Actions taken are more targeted and in most cases the agent reaches the destination considerably faster than the agent that's taking random actions. This is because the agent is interested in getting the final reward. Another interesting thing is that the agent often drives in circles when enough time is remaining in order to maximize rewards - I wouldn't want a taxis driver like that ;). To solve this problem small negative rewards should be given during each iteration.

Improving the Q-Learning Driving Agent

This section probably took me the longest to complete, because I challenged a lot of the assumptions that I made earlier on.

Initalization:
In the previous sections I initialized the Q-Learning matrix with zeroes. Although this gives very good results, it might take the agent a couple of iterations to reach its final destination. In order to reduce this number and with the optimal policy in mind, I decided to take another approach to initialization.

The overall goal of the agent should be to follow the `self.next_waypoint` suggestion from the planner. So whenever the `next_waypoint` in a state is equal to the corresponding action I initialize the Q matrix with 2, in all other cases to -0.5 . Because of this the agent will make invalid moves as it learns the traffic rules, but it assists the agent in reaching its destination faster and more often.

Paramters:
Based on the hint from a previous reviewer I decided to treat the problem at hand as a one-step repeated game. The q value of the next state in this case shouldn't influence the learner, which is why I set the Gamma parameter to zero. I continued to tune the two remaining parameters using the following score function:

$$score = 0.4 \cdot \mu_{trip\ completion} + 0.4 \cdot \mu_{invalid\ moves} + 0.2 \cdot \mu_{trip\ duration}$$

Alpha	Gamma	Epsilon	Score
0.9	0	0.3	1.72
0.9	**0**	**0.4**	**1.82**
0.9	0	0.5	1.74
1	0	0.4	1.66
0.8	0	0.4	1.54
0.9	0.1	0.4	1.66
0.9	0.2	0.4	1.37

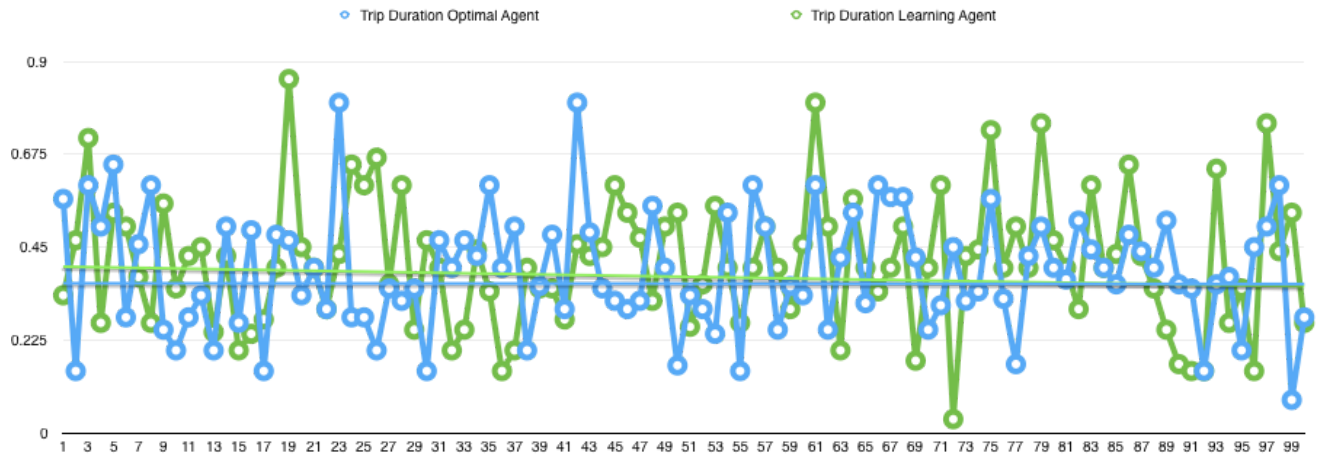
I tested various different combinations. These are the final numbers:

- Alpha: 0.9 — at the end of every trial: `self.alpha -= self.alpha * 0.05`
- Gamma: 0
- Epsilon: 0.4 — at the end of every trial: `self.epsilon -= self.epsilon * 0.69

Optimal Policy
An agent with an optimal policy has the following traits:

- The agent does not stray from the next_waypoint recommendation from the planner
- The agent does not make any invalid moves

With these two principles in mind I developed an optimal agent that follows all traffic rules and either goes to the next waypoint, if possible, or stays at the current location.



The optimal agent reaches its destination every time, using about 40% of the available time. Furthermore, it does not perform any invalid actions throughout all 100 trials. My final agent reaches its destination about 97%, using on average about 44% of the available time. However, the final learning agents performs about three invalid moves per trip. Unfortunately, this number is decreasing only very slightly over time (see chart below).