Smart Cab Project Report
Levin Jian, May 2016

1. Implement a basic driving agent

1.1. Agent accepts inputs

This part is implemented in LearningAgent::selectAction method in agent.py file.

```
def selectAction(self, state):
#         return self.next_waypoint
        return self.getRandomAction()
```

1.2. Produces a valid output

This method randomly choose an action out of list [None, 'forward', 'left', 'right']

```
def getRandomAction(self):
        listOfActions=[None, 'forward', 'left', 'right']
        return random.choice(listOfActions)
```

1.3. Runs in simulator

The driving agent can run in the simulator without errors. You can run it by launching the agent.py file. Regarding simulator display, two modifications are made to help us better understand the movement of the agents.

1) For each trial, the simulator screen will be displayed twice. The first time shows the state which the learning agent is in, and the action it’s about to take; the second time shows the reward the agent get out of its last action.
2) In the simulator screen, the texts about dummy agents are removed, and only the text about the learning agent is retained so that it is easier to identify the learning agent. Also the text now means the action the learning agent is about to take, instead of its next_waypoint. You can see it by launching the agent.py file.

2. Identify and update state

2.1. Reasonable states identified

This part is implemented in LearningAgent:: getCurrentState method in agent.py file.
The state is a tuple (self.next_waypoint, inputs['light'], inputs['oncoming'], inputs['right'], inputs['left']). This is because all these information can help the learning agent determine next action. The “deadline” information should also be useful, but it’s not included in state because, if it’s included, there will be too many possible states for the learning agent to explore during the training, and yet we are required to do only 100 trials at the utmost.

2.2. Agent updates state

The driving agent can update its state when running. You can see it by launching the agent.py file.

3. Implement Q-Learning

This part is implemented by class LearningAgent_Basic_Qtable in agent_basic_qtable.py file

3.1. Agent updates Q-values

The Q value update are done by two methods, updateQTable_1, updateQTable_2.
updateQTable_1 is executed after the learning agent move, At this point, we will not know the next state(since the traffic light would change, and the other dummy agents would move). So this method merely memorize current state, action and reward. And then after the traffic lights change, and other dummy agents complete their move, we will know the next state , and updateQTable_2 will be executed. The Q value update formula is as below:

q_new = (1- alpha)*q_old + alpha*(self.current_reward + gamma * q_max)

3.2. Picks the best action

After training, the learning agent can utilize Q table to pick the best action. This is implemented in LearningAgent_Basic_Qtable:: getQBestAction

```
def getQBestAction(self, state):
        tempList = []
        listOfActions=[None, 'forward', 'left', 'right']
        for action in listOfActions:
            tempList.append(self.qtable[(state, action)])
```

```
maxQ = max(tempList)
maxindexes =  [i for i, j in enumerate(tempList) if j == maxQ]
#if more than one action has maxQ value, choose one of them randomly
return listOfActions[np.random.choice(maxindexes)]
```

3.3. Changes in behavior explained

After applying Q-Learning algorithm, the agent's ability to choose the 'right' action significantly improves:

1) For the basic driving agent, it randomly choose action, regardless of inputs
   `Test result: Average Discounted Reward,0.38 Completion Rate,0.19 Average Penalty,-11.83 Average Deadline,1.92`
2) For the learning agent, it uses Q table to guide its action choice:
   Test result: Average Discounted Reward,27.405 Completion Rate,0.94 Average Penalty,-2.20652173913 Average Deadline,15.55

The reason for the improvement observed is very obvious, the agent is now using Q Table (being tantamount to experience obtained during its training) to guide its action, instead of acting randomly.

4. Enhance the driving agent

Grid search is used to find best hyper parameters and enhance the basic Q-learning. This part is implemented by class FineTuneQTable in agent_basic_qtable.py file

4.1. Agent learns a feasible policy within 100 trials

After applying grid search, the agent with best hyper parameter combination has below performance.

Test result: Average Discounted Reward,29.36125 Completion Rate,0.908333333333 Average Penalty,-2.45088495575 Average Deadline,13.6825

Out of the 1200 trials, the learning agent is able to reach the destination 90.8% of the time before deadline is reached, and its average discounted reward is 29.36125.

4.2. Improvements reported

Three key hyper parameters are fine tuned in order to achieve the best performance on the foundation of the basic Q learning agent.

Learning rate: The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information[1,2]. In our case, a constant training rate is used throughout the training.

Discount factor: The discount factor gamma determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward[1]. In our case, a constant discount factor is used throughout the training.

Epsilon-greedy policy: Using this policy, either we can select random action with epsilon probability and we can select an action with 1-epsilon probability that gives maximum reward in given state. it is a trading off between exploration and exploitation. With high epsilon, we end up with exploring more <state, action> pairs and exploiting less the Q table it presently has at the time, and vice versa[3]. In our case, a constant Epsilon is used throughout the training

Specifically, below parameters are used:

```
alphas = [0.1,  0.5]
gammas = [0.5,0.9]
epsilons = [None, 0.5,0.8]
And the result is as below:
```

NO	score	alpha	gamma	epsilon	finalstringrepresentation
0	23.36875	0.1	0.5	None	Test result: Average Discounted Reward,23.36875 Completion Rate,0.985 Average Penalty,-0.88330170778 Average Deadline,15.9175
1	29.36125	0.1	0.5	0.5	Test result: Average Discounted Reward,29.36125 Completion Rate,0.908333333333 Average Penalty,-2.45088495575 Average Deadline,13.6825
2	28.2725	0.1	0.5	0.8	Test result: Average Discounted Reward,28.2725 Completion Rate,0.934166666667 Average Penalty,-2.14386584289 Average Deadline,15.1083333333
3	29.17708	0.1	0.9	None	Test result: Average Discounted Reward,29.1770833333 Completion Rate,0.909166666667 Average Penalty,-2.4185840708 Average Deadline,13.7875
4	28.76417	0.1	0.9	0.5	Test result: Average Discounted Reward,28.7641666667 Completion Rate,0.926666666667 Average Penalty,-2.37190812721 Average Deadline,14.5433333333

5	23.37	0.1	0.9	0.8	Test result: Average Discounted Reward,23.37 Completion Rate,0.99 Average Penalty,-0.890398550725 Average Deadline,16.3183333333
6	25.26917	0.5	0.5	None	Test result: Average Discounted Reward,25.2691666667 Completion Rate,0.934166666667 Average Penalty,-1.5993705036 Average Deadline,15.3775
7	18.16542	0.5	0.5	0.5	Test result: Average Discounted Reward,18.1654166667 Completion Rate,0.99 Average Penalty,-6.15627782725 Average Deadline,16.8883333333
8	28.81708	0.5	0.5	0.8	Test result: Average Discounted Reward,28.8170833333 Completion Rate,0.944166666667 Average Penalty,-2.22626441881 Average Deadline,14.6366666667
9	23.62333	0.5	0.9	None	Test result: Average Discounted Reward,23.6233333333 Completion Rate,0.988333333333 Average Penalty,-1.01114649682 Average Deadline,16.1983333333
10	21.93167	0.5	0.9	0.5	Test result: Average Discounted Reward,21.9316666667 Completion Rate,0.704166666667 Average Penalty,-1.63327370304 Average Deadline,10.1841666667
11	15.85333	0.5	0.9	0.8	Test result: Average Discounted Reward,15.8533333333 Completion Rate,0.995833333333 Average Penalty,-7.54418197725 Average Deadline,15.985

Best hyper parameters(the one with highest discounted reward)

1	29.36125	0.1	0.5	0.5	Test result: Average Discounted Reward,29.36125 Completion Rate,0.908333333333 Average Penalty,-2.45088495575 Average Deadline,13.6825

Basic Q learning

0	23.36875	0.1	0.5	None	Test result: Average Discounted Reward,23.36875 Completion Rate,0.985 Average Penalty,-0.88330170778 Average Deadline,15.9175

4.3. Final agent performance discussed

An optimal policy should always move towards the direction of next_waypoint unless a move is not allowed due to traffic light or oncoming dummy agents, in that case, it should perform "None" action.

The final driving agent has below parameters and test result:

1	29.36125	0.1	0.5	0.5	Test result: Average Discounted Reward,29.36125 Completion Rate,0.908333333333 Average Penalty,-2.45088495575 Average Deadline,13.6825

Visualize the final agent in the simulator display, we can see that the final agent roughly act by the optimal policy. It generally go straightly to the next_waypoint, while try not violating traffic rules.

The actual policy used by final agent is outputted to file qtable_policy.csv, which can be automatically generated by running agent_enhanced_qtable.py file. The content of final agent's policy is attached in the appendix section.

One interesting point to note is that although the final agent(NO 1) has higher Average Discounted Reward than the basic q table agent(NO 0), it has lower completion rate. This is because Q learning is all about maximizing discounted reward. To push the final agent towards having higher completion rate, we will need to adjust relevant reward mechanism like penalizing not being able to reach destination.

5. Reference

1. https://en.wikipedia.org/wiki/Q-learning
2. http://stackoverflow.com/questions/33011825/learning-rate-of-a-q-learning-agent
3. https://junedmunshi.wordpress.com/2012/03/30/how-to-implement-epsilon-greedy-strategy-policy/

6. Appendix---Final agent's policy

Below is the actual policy used by the final agent. It is obtained by

1) Load the final Q table
2) Group by state
3) Extract <state, action> pairs which has the highest Q values

In the action column, blank indicates 'None' action. Also as introduced earlier, state = (self.next_waypoint, inputs['light'], inputs['oncoming'], inputs['right'], inputs['left'])

	state	action	qvalue
0	(None, 'green', None, None, None)		0
1	(None, 'green', None, None, None)	right	0
2	(None, 'green', None, None, None)	left	0
3	(None, 'green', None, None, None)	forward	0
4	(None, 'green', None, None, 'left')	right	0
5	(None, 'green', None, None, 'left')	forward	0
6	(None, 'green', None, None, 'left')		0
7	(None, 'green', None, None, 'left')	left	0
8	(None, 'green', None, 'forward', None)		0
9	(None, 'green', None, 'forward', None)	left	0
10	(None, 'green', None, 'forward', None)	forward	0
11	(None, 'green', None, 'forward', None)	right	0
12	(None, 'red', None, None, None)	forward	0
13	(None, 'red', None, None, None)		0
14	(None, 'red', None, None, None)	left	0
15	(None, 'red', None, None, None)	right	0
16	(None, 'red', 'left', None, None)		0
17	(None, 'red', 'left', None, None)	left	0
18	(None, 'red', 'left', None, None)	right	0
19	(None, 'red', 'left', None, None)	forward	0
20	('forward', 'green', None, None, None)	forward	4.16646
21	('forward', 'green', None, None, 'forward')	forward	0.832517
22	('forward', 'green', None, None, 'left')	right	0.305338
23	('forward', 'green', None, None, 'right')	forward	0
24	('forward', 'green', None, None, 'right')	left	0
25	('forward', 'green', None, None, 'right')		0
26	('forward', 'green', None, 'forward', None)	forward	2.741485
27	('forward', 'green', None, 'left', None)		0.090556
28	('forward', 'green', None, 'right', None)	forward	0.462786
29	('forward', 'green', 'forward', None, None)	left	0.057437
30	('forward', 'green', 'left', None, None)	forward	0.892137
31	('forward', 'red', None, None, None)		1.362523

32	('forward', 'red', None, None, 'forward')		0
33	('forward', 'red', None, None, 'forward')	left	0
34	('forward', 'red', None, None, 'forward')	right	0
35	('forward', 'red', None, None, 'left')		0.006565
36	('forward', 'red', None, None, 'right')	forward	0.137513
37	('forward', 'red', None, 'forward', None)		0.139287
38	('forward', 'red', None, 'right', None)	left	0
39	('forward', 'red', None, 'right', None)		0
40	('forward', 'red', 'forward', None, None)	right	0.287907
41	('forward', 'red', 'left', None, None)		0.072546
42	('forward', 'red', 'right', None, None)		0
43	('forward', 'red', 'right', None, None)	left	0
44	('forward', 'red', 'right', None, None)	right	0
45	('left', 'green', None, None, None)	left	5.273028
46	('left', 'green', None, None, 'forward')	right	0.246795
47	('left', 'green', None, None, 'left')	left	4.675783
48	('left', 'green', None, None, 'right')	left	0.255313
49	('left', 'green', None, 'forward', None)	left	0.264347
50	('left', 'green', None, 'left', None)	left	2.712862
51	('left', 'green', None, 'right', None)	left	0.742188
52	('left', 'green', 'left', None, None)	forward	0.013728
53	('left', 'green', 'right', None, None)	forward	0.257689
54	('left', 'red', None, None, None)	right	1.678347
55	('left', 'red', None, None, 'left')		0.016809
56	('left', 'red', None, None, 'right')	left	0
57	('left', 'red', None, None, 'right')	right	0
58	('left', 'red', None, None, 'right')		0
59	('left', 'red', None, 'left', None)		0
60	('left', 'red', None, 'left', None)	right	0
61	('left', 'red', None, 'left', None)	forward	0
62	('left', 'red', None, 'right', None)	left	0.090345
63	('left', 'red', 'forward', None, None)		0.409673
64	('left', 'red', 'left', None, None)	right	0.174415
65	('left', 'red', 'right', None, None)	right	0.025942
66	('right', 'green', None, None, None)	right	4.000492
67	('right', 'green', None, None, 'forward')	forward	0.455125
68	('right', 'green', None, None, 'left')	left	0.14131
69	('right', 'green', None, None, 'right')	left	0.042716
70	('right', 'green', None, 'forward', None)	right	1.791598
71	('right', 'green', None, 'left', None)	right	0.396417
72	('right', 'green', None, 'right', None)	right	0.387742
73	('right', 'green', 'left', None, None)	forward	0.416146
74	('right', 'green', 'right', None, None)	right	0.44975

75	('right', 'red', None, None, None)	right	4.809836
76	('right', 'red', None, None, 'forward')		0.054145
77	('right', 'red', None, None, 'left')		0.208035
78	('right', 'red', None, 'forward', None)	right	0.355903
79	('right', 'red', None, 'left', None)		0
80	('right', 'red', None, 'left', None)	forward	0
81	('right', 'red', None, 'left', None)	right	0
82	('right', 'red', None, 'left', None)	left	0
83	('right', 'red', None, 'right', None)	left	0.094135
84	('right', 'red', 'forward', None, None)	right	0.793345
85	('right', 'red', 'left', None, None)	right	1.828893