# Report

April 15, 2016

*1. In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?*

Answers: By applying random actions to the cab, it will generate random actions on the map, but it will still obey the traffic rules since the act function in enviroment.py will not perform that random action if the agent breaks the rule. The angent will finally ready the target location by a small chance since the map in finite when set enforce_deadline to false. The rewards are negative.

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

Answers: The states feature I choose include traffic light, incoming traffic, left traffic, right traffic and the action the agent intend to perform. Basiclly it's the inputs structure and self.next_waypoint. I choose these features because when people driving, most of time those features describe what we need to consider.

*3. Implement Q-Learning. What changes do you notice in the agent's behavior?*

Answers: It begins to reach the destination on time and get positive final reward. But with a high chance the agent will stuck in a rut to just make right turns in a loop all the time and miss the deadline. Maybe it's because I set the learning rate(0.5) and gamma(0.7) too high so that the agent by a change make a lot of right turn at the begining and quickly learns it can make more right turns to collect a reward of 2. Even in the situation it should make a left turn, it sacrifise to get a negative reward -0.5 (since action != agent.get_next_waypoint()) in order to make three more right turns because the q table shows more action values.

One time when I run the 1000 trials the agents get really stuck in loop with 977 times failures to get destination, which means 97.7% times the agent fails.

*4. Enhance the driving agent*

*4. 1 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?*

Answers: To prevent the agent stucking, one way is to do a grid search of alpha and gamma. The other way is to us a exploration function. I choose the second one and implement it in function *next_action_index_with_exploration* and also decrease the alpha(0.2) and gamma(0.5). The basic idea is cache the time of state->action the agent have tried. If it doesn't meet a certain threshold(*self.NE*) then we use a certain positive q_value for that state->action (*self.reward_plus*). After we meet enought numbers of that state->action status, we begin to use the values we've learned in q_table.

This time I never see the agent stucks again. It still may loop a little but will get out again. Percent of failure drops to around 0.83%

*4. 2 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?*

Since I use a exploration function, I enlarge the trials number to be 10000 in order to get more possible state. Here's the result:

Total number of states(size of q_table): 97

The agent can learn some good policy, for example:

'green###*#forward': [0.0, 6.467462471732415, 0, 0]

- Green light, no traffic, agent plans to go forward, then just go(6.46)

'red###*#forward': [2.0150303065266995, -0.75024, -0.5500800000000001, -0.30724185569920004]

- Red light, no traffic, agent plans to go forward, then better to stay(2.015)

'green###*#left': [0.0, -0.37512, 5.592063572513188, 0.2883501921024001]

- Green light, no traffic, agent plans to go left, then just go(5.59)

'green#forward###left': [1.5008308907230599, 1.0484784867742991, 0.7969750027123902, 0]

- Green light, on coming traffic going forward, agent plans to go left, better to stay(1.5) to avoid collapse with on coming traffic.

'red#right#forward#*#right': [0.5036166307367694, 0, 0, 0]

- which means the light is red, on coming traffic is trying to go right, left traffic is going to go forward, the prime agent is going to make a right turn. In this situation, the agent learns the best next action is to stay(0.5036) because there's traffic from left are going forward.

'red##*right*##left': [0.887672123724325, -0.02997396157598211, 0.21298760278363635, 0]

- For this situation, the light is red and the prime agent plan the make a left turn. It learns better to stay(0.8876) and wait.

'red#forward#*#right#forward': [0.09584415424058786, 0, 0, 0]

- Right light, better stay.

It can learn some bad policies:
'red#forward###left': [0.48620042443791445, -0.39305946182028867, 0.13736588618086382, 1.9325663854034965]

- If the prime agent trying to go left on a red light and only have oncoming traffic going forward, it will make a right turn instead(1.9)

'red###*#left': [0.0, -0.692180163624, -0.3911499794281595, 2.042660093585707]

- Red light, no traffic, agent plans to left. q table succesfully learns to punish the forward(-0.69) and left(-0.39) actions. But rewards to turn right.

So in general, when the agent meets a red light and trying to make a left turn, it still rewards going right instead of staying and waiting, since in the future it may make more right turns to come back and get more rewards. I think to totally solve this problem is to involve the deadline feature in state and gives a negative reward when missing the deadline. So that the agent will try to stay on the best path in order to meet the deadline.