Train a Smart Cab

Implement a Basic Driving Agent

At the beginning, I set policy to return a random action selected from {None, forward, left, right}. Most of trials failed because the agent move around with no particular destination. However, sometimes smartcab has reached destination though it took time far more than deadline. It is just a luck to be in the goal. Actions were not corresponding to actions of planner.

Select and update states of agent

The agent have to obey traffic rules and follow the direction given by route planner.
So, I chose four features below to represent each state.

1. Next waypoint:
The agent itself doesn't know its destination and which way to go next. So, it needs to be taught the route from route planner to get to the goal.

2. Traffic light:
The agent needs to learn traffic rules. All valid actions of agent, move forward, turn right, turn left and stay there, need to know whether traffic light is green or not to obey traffic rules. For example, going forward action takes priority over all other agent's action if traffic light is green, it means the agent can decide whether it may go forward or not only by the state of traffic light.

3. Boolean for turn left
In the traffic rules for this project, the agent can turn left when traffic light is green and there is no oncoming traffic making a right turn or coming straight through the intersection. So, I defined the variable that shows whether the agent can turn left or not. By representing it by only one boolean variable, the size of agent state space can be decreased.

4. Boolean for turn right
In the traffic rules for this project, the agent can turn right when traffic light is green. Also, when traffic light is red, a right turn is permitted if no oncoming traffic is approaching from agent's left through the intersection. So, as same as boolean for turn left, I defined the variable for turning right.

With the variables above, the agent can choose to stay there when traffic light is red, boolean for turning either side is false. So, all possible actions by the agent can be represented by those four variables.

Also, defining variables in this way, I could decrease space of Q-Learning value table. If I chose all variables to represent states of the agent, the space of Q-Learning value is 512 (4 possible waypoint for 4 direction and the state of traffic light). However, with the variables I chose, next waypoint takes 4 possible value, traffic light takes 2 possible value, boolean for turning each direction takes 2 possible value. So, it has only 32 states.

Implement a Q-Learning Driving Agent

Q-Learning algorithm was implemented to help agent to choose best action depends on its current state and behavior was significantly improved. I used equation below to update Q value for each agent state and action.

$$Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha[r+\gamma Q(s',a')]$$

α represents learning rate and γ represents discount factor. Although there might be optimal value, I set these parameters to $\alpha = 0.6$ and $\gamma = 0.9$.

The agent takes reward when it took appropriate action with its current state. Also, the agent takes negative reward when it didn't take right action or broke traffic rules. With Q-Learning algorithm, the agent takes an action which has the most highest Q(s,a) value for the given state. Q values are updated by each trial and improved to take appropriate action. So, it gets better and better to take right action, when the agent which takes random action was not improved by any of their action.

Improve the Q-Learning Driving Agent

With Q-Learning algorithm, the policy always choose an action that has the best value on Q-Learning table. So, some of actions might never be chosen if it isn't chosen at early trial.
To avoid this, I used ε greedy method. In ε greedy method, the policy choose random action with probability ε, while it choose the best action from Q-Learning table with probability $1-\varepsilon$.
By doing so, the agent sometimes takes random action and exploit new way.
It is desirable to take more random action in the early stage and take less random action in the late stage. So, ε should be high at first and decrease gradually. I set ε to decrease by the trial times increase.

$$\varepsilon = 1 / (trialtimes + 1)$$

I tried some parameter combinations to find the best performance of agent. The parameters I tried was below.

$$\alpha \{0.4, 0.6, 0.8\}$$
$$\gamma \{0.2, 0.5, 0.9\}$$

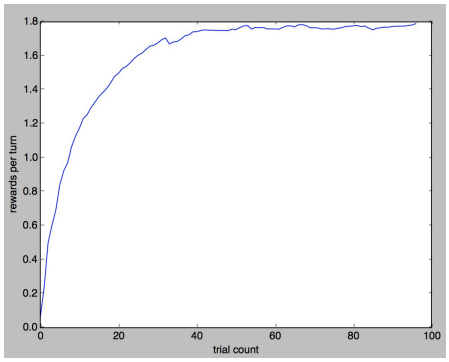
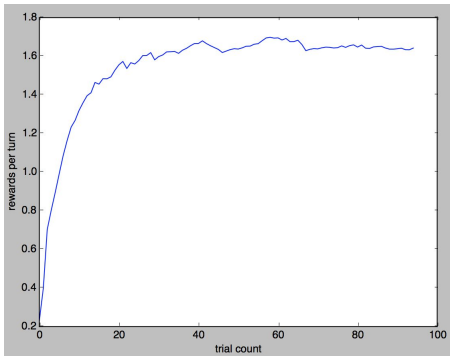
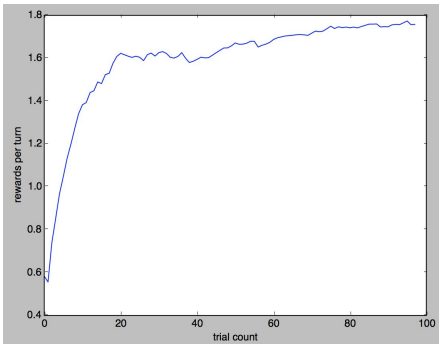
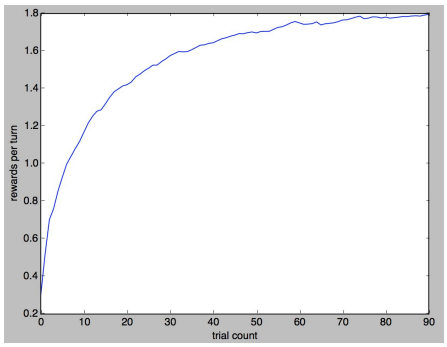
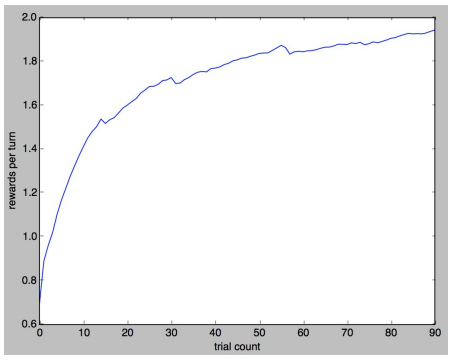
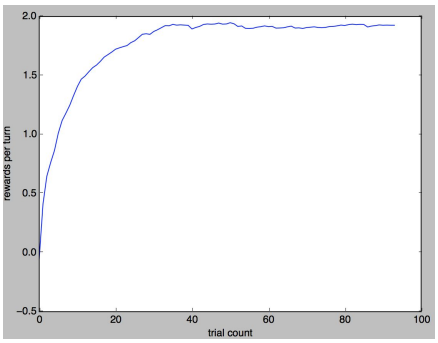
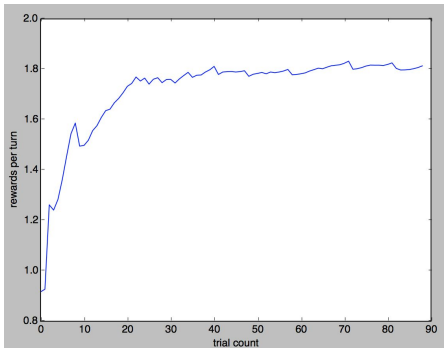
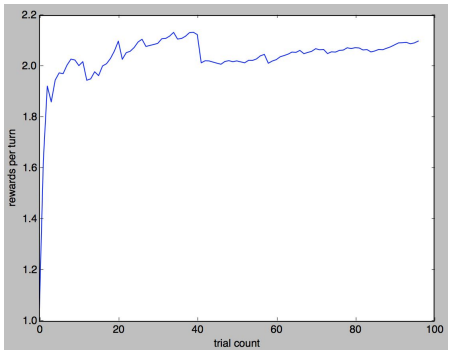
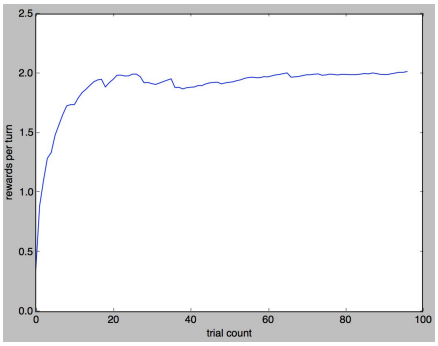
Rate of reaching destination (100 times trial)

	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$
$\gamma = 0.2$	97	95	98
$\gamma = 0.5$	91	91	94
$\gamma = 0.9$	89	97	97

Average value of reward per move (100 times trial)

	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$
$\gamma = 0.2$	1.7897	1.6425	1.7576
$\gamma = 0.5$	1.7961	1.9438	1.9262
$\gamma = 0.9$	1.8137	2.1000	2.0176

Graphs of average value of reward per move (100 times trial)

	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$
$\gamma = 0.2$			
$\gamma = 0.5$			
$\gamma = 0.9$			

For α, I found that higher value results in high rate of reaching destination. High α value means the agent update its Q-Learning value a lot from Q-Learning value from next state and action. Especially, it is important to learn Q-Learning value fast at the beginning of trials, so the higher α resulted into high accuracy.

For γ, I can see that higher γ value tend to get high average reward. Although the agent with lower γ value reaches destination as similar rate as higher γ value, it seems to drive inefficiently to get to destination.

In conclusion, I chose $\alpha = 0.6$ and $\gamma = 0.9$ to the best parameters for Q-Learning method.
It shows high rate of getting destination 97%, and average value of reward 2.1. Also, shown in the graph, the agent with these parameters seems to converge at the early stage.