# **Train a Smartcab to Drive**

Xiangwei Wang

### **1. Identify and update state**

### Firstly I change the code to be the follow:

**self.next\_waypoint** = self.planner.next\_waypoint()

*inputs* = self.env.sense(self)

deadline = self.env.get\_deadline(self)

action = None

ran=random.randint(1,4)

if ran == 1:

actino = None

elif ran==2:

action = 'forward'

elif ran==3:

action = 'left'

elif ran==4:

action = 'right'

else:

action = none

reward = self.env.act(self, action)

where **self.next\_waypoint** is the next waypoint location, *inputs* in intersection state and deadline is current deadline value. And they are all the inputs to our agent. For the output or the action lf the agent, it is just random from none, ‘forward’. ‘left’ or ‘right’.

When running in similar, we found that agent just drive blind and random, not approaching to the target, and can not arrive the target within a short time(I observe for about 5min, and it does not make it), the reward is varied about form -1 to 2.

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

I only choose **light condition** and **next\_waypoint** as our robot states.

There are six independent state to choose model our agent:

1. next\_waypoints, it is direction form global router which can direct agent to his target with the reasonable and shortest path, so we must choose it as a state if we want to be able to reach the target.
2. Light condition, it is the traffic light condition which can be right and green, agent must obey the rule or it can not move. So it is another necessary state to model.
3. Left, right, oncoming, the three values is the intend of the car froming three different direction, for example, the oncoming is what the car in oncoming direction want to do, ‘go forward’, ’turn left’, ‘turn right’ or ‘stay there’. For general purpose, they should states for model robot, however for this problem, they are necessary. Because, in the reward function, I find that the reward do not take it in account. Even though we add them as states, the agent cannot learn something right by Q method.
4. Deadline, it is the left time for robot to get to target. For this question it is not as necessary as usual. Because agent will not learning something relative to Deadline by Q method as well. Because each available move will get a reward and no extra penalty for each move.

As a result, I only choose **light condition** and **next\_waypoint** as our robot states.

AvailableInformation=[('Next Way Point:', self.next\_waypoint),

('light',inputs['light']),

('On Comming:',inputs['oncoming']),

('Left:',inputs['left']),

('Right:',inputs['right']),

('Deadline',deadline)

self.state=AvailableInformation[0:2];

And because I do not change the action of agent now, so the agent still move randomly.

**3. What changes do you notice in the agent’s behavior?**

**3.1 Implement Q-learning**

I implement a Q-learning by the following three step.

1. Firstly, intial a Q value table by ordereddict and the states include light condition and next-waypoint. When initialize I set all value to zero.
2. Update the Q-value after every move by the following equation

Where alpha is the learning rate and lamda is the time discount and r is the reward of the action according to state before the action, and s’ is the state after the action.

1. Get the best action according the Q-value and current state. Just choose the action which can get biggest Q value. If there two or more action with the same Q value, I choose the action from those actions by random.

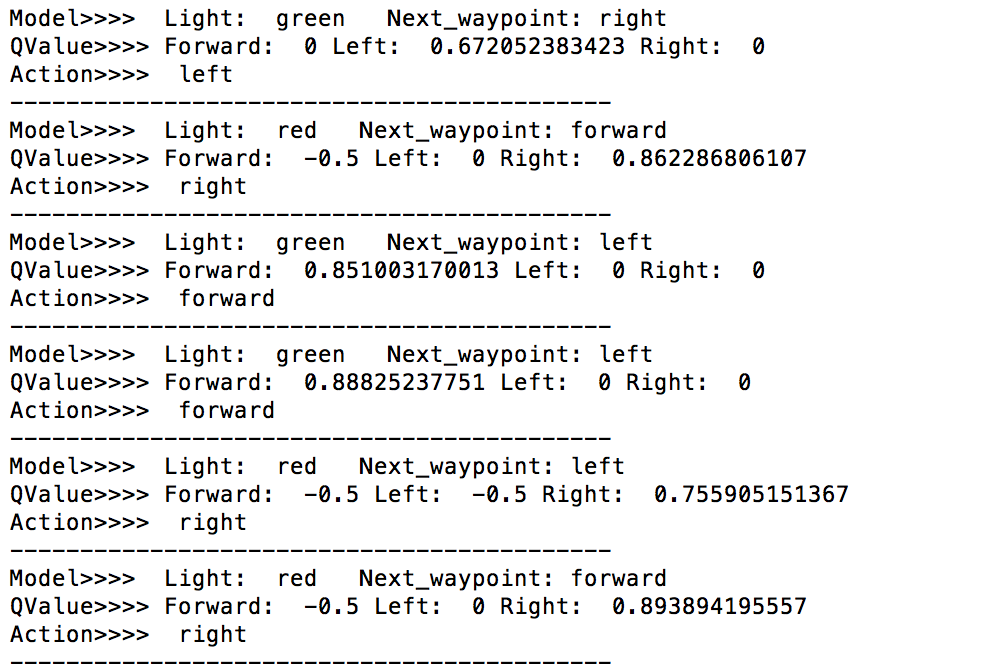
Note: for the first step, there is no Q value, so it is chosen by random.

**3.2 Behavior Changes**

After using Q learning, the agent no longer act by random, instead, he act by his current knowledge.

For example:

The following picture shows the agent’s action. Model is the current state of the agent; QValue is the q value when agent choose different action; Action is the action agent choose. We can know that the action only depend on the QValue and current state. Agent learn how to move by Q learning.



However because agent the best action available from the current state based on Q-values, so he learn less and may stuck in local optimal. And it really does, because he always get reward if he turn right, so sometimes, he always turns right and move in a local circle.