Smart Cab – How to Drive

Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

Next waypoint location, relative to its current location and heading, Intersection state (traffic light and presence of cars), and, Current deadline value (time steps remaining), And produces some random move/action (None, 'forward', 'left', 'right'). Don't try to implement the correct strategy! That's exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce_deadline set False (see run function in agent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

Observe what you see with the agent's behaviour as it takes random actions. Does the smart cab eventually make it to the destination? Are there any other interesting observations to note?

Yes, the agent finally reached the destination. As it is the greedy approach where is is choosing the actions randomly, thus it took a lot of time and does not care whether there is an oncoming vehicle or whether that vehicle is at its right or left and what is the state of red light. Which leads to take a lot of trials and time. It takes so long that there is a significant difference from the deadline.

Identify and Update State

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

What states have you identified that are appropriate for modelling the **smartcab** and environment? Why do you believe each of these states to be appropriate for this problem?

```
if(light=red):
        next_waypoint = [right,left(-),forward(-)]

        if(next_waypoint=forward)
                action=[none(+) or right (-)]
else if (light=green)
        next_waypoint = [right(+), left, forward(+)]

        if(oncoming=forward)
                action=[none(+) or left (-)]
```

And also the car seems to turn right when oncoming car is turning left and the light is red because the next_waypoint is right everytime.

States as per US Laws

1. *Light: red or green*
2. *Oncoming: forward, left, right or none*
3. *Left: Forward, left, right or none*
4. *Action: Take the next waypoint or none*
5. *Next waypoint: forward, left or right*

*It does not make sense to use deadline to fold into each state as the agent seems to learn the correct set of rules under **100** trials and reaching the destination safely, maximizing the rewards it got by following the next_waypoint thus deadline does not make any sense.*

Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behaviour.

What changes do you notice in the agent's behaviour when compared to the basic driving agent when random actions were always taken? Why is this behaviour occurring?

The agent reaches the destination much faster. As while exploiting the earlier trials the agent was also exploring while collecting the positive and negative rewards. Thus, it helped to collect more rewards in the later trials and reach the destination quickly i.e. in initial trials there are some negative rewards but later on there were no negative rewards. This behaviour is observed because agent gradually learning to follow the best next_waypoint. It is not random now and is learning to follow traffic rules as well though initial trials are being penalized.

Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

Q-Learning alpha and gamma parameters are being trained from values [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9] and reward_per_action and penalty_per_action is being calculated and analysed in excel to sort the data(qLearningTunning.csv). Thus alpha=0.9 and gamma=0.3 is being considered to the best value in this range of values. Though there is not much difference between gamma=0.3 and 0.5 but at the same time penalty_per_action is better, which is also analysed after repetitions.

At alpha=0.9 and gamma=0.3 the agent reaches the destination 97/100 times thus the performance of the driving agent is really good. (alpha_0.9_gamma_0.3.csv)

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? How would you describe an optimal policy for this problem?

Yes, obviously the agent reached to the optimal policy. In 100 iterations, it consistently reaches the destination, and minimally incurs penalties (especially in the beginning). After iterations, it seems to consistently take correct actions. One thing I would add is that the "learning" could probably be accelerated by creating more dummy agents (i.e. it would speed up convergence for states that were happening less often by making those events happen more frequently). By increasing the number of dummy agents, the car would probably interact more with them, and would accumulate penalties to avoid earlier than later. This would then translate to less "training time", and would be a nice enhancement.