P4: Smart Cab Project Report

By Himanshu Dongre

Task 1

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 to 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.

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

Ans:

A simple driving agent (available in agent.py) is designed. At first I ran this agent for 10 trials and with enforce deadline set as true. In such cases the agent almost never reaches the destination on time. After changing the enforce deadline to false I found that the driving agent was indeed reaching the destination, but was taking too long and the path was not at all optimal. Many times it did not move even when the roads were clear. Also the agent was taking action at random as was hoping to reach the destination without any solid logic. This was happening because the agent was not learning anything from its past journeys. It simply took an action randomly and was not trying to optimize reward by choosing correct actions.

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.

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

Ans:

A basic naïve agent should at least be aware of the traffic at current intersection and traffic light status of that intersection. It would be like driving completely blind if the car doesn't know of the incoming traffic and traffic light status. To update the state, I therefore used a combination of traffic light inputs and oncoming traffic inputs at each intersection along with the next_waypoint. next_waypoint is used so that the overall goal of the agent still remains to reach the destination. If next_waypoint is not used, then the car will not be able to understand where it has to ultimately reach. Also I did not use the input for right oncoming traffic as according

to the US traffic rules, right side traffic does not affect the agent's choice. As the agent was supposed to run with enforce_deadline as false. I did not use deadline input too as it gives no relevant information that might help agent drive to destination. Including deadline would also increase the overall state space of the problem slowing down the computation. As the approach of the agent was very random, it was clear that it was going to take very long time to reach destination. I just wanted to check if the agent does reach the destination if given lot of time.

Task 2

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 behavior.

What changes do you notice in the agent's behavior?

Ans:

A Q-learning agent (available in Qagent.py) is designed. After implementing Q-learning algorithm, I ran the agent for 100 trials with enforce deadline set to True. I saw an immediate increase in the intelligence of the agent. It failed to reach the destination within specified deadline a couple of times at start. However, as the agent was now using the results from its past journeys, it quickly started learning to take correct turns which maximized rewards. Also it was now reaching destination in fairly less number of steps. The agent safely reached destination almost 67% of the times.

Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

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?

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?

Ans:

To further enhance the driving agent, I ran the driving agent with different combinations of learning rate, discount parameters and epsilon value. Below are few parameter values with the result that they produced:

Learning_rate	discount	epsilon	No. of successful trips out of 100 simulations
0.5	0.5	0.5	67
0.6	0.4	0.5	61
0.9	0.35	0.5	73
0.9	0.35	0.4	77
0.9	0.35	0.3	82
0.9	0.35	0.2	85
0.9	0.35	0.1	90
0.8	0.35	0.1	89

When I ran the agent with the above settings for 100 trials, it reached destination with higher total rewards 90% of the times for learning rate =0.9, discount=0.35 and epsilon 0.1. However most of the unsuccessful attempts were in the beginning indicating that the agent is still behaving randomly at start as it doesn't have any states populated in Q table. As initially the Q values are zero, it will start preferring action "None" so as to accumulate positive reward. However, this exploitation problem is solved up to some extent by changing epsilon to 0.1 which then imparts some randomness to the initial decision making process helping the agent to escape local minima. Also, with more number of trials, accuracy of the agent seems to increase. Ex.- with 200 trials successful trips rate bumps to 94%.

After initial few attempts, agent starts learning from its past mistakes and starts reaching destination by taking correct turns and in less number of steps. Initially as the cab starts reaching destination, the total rewards are high. However, the rewards in the end are mostly low as the agent has learned quite a lot by then and reaching destination in shorter period of time than before. Therefore, I can say that the agent is now behaving optimally and completing the trip in minimum time with no penalties.

Reference:

- www.dmvdriverseducation.org
- Left Turn Yield on Green, how to properly turn Left-YouTube video
- http://mnemstudio.org/path-finding-q-learning-tutorial.htm
- https://webdocs.cs.ualberta.ca/~sutton/book/ebook/node65.html