Udacity Machine Learning Nanodegree

P4: Train a Smartcab to Drive

1. Implement a basic driving agent

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

I only implemented a simple agent. The action = random.choice [None, 'forward', 'left', 'right'])' makes the agent go in random directions. It makes it to the target location just by chance – this can take a long time or a short amount of time. It solely relies on chance and there is no learning involved yet.

2. Identify and update state

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

I picked the recommended waypoint and environment inputs left, right, oncoming, and light. They model the agent as they are all things that the sensors record and that contribute to the action decision. I didn't use deadline or time step and used them for learning adjustments and the exploration rate. This is also because they aren't needed for the action decision and would just increase the amount of states.

3. Implement Q-Learning

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

The agent initially starts off the same way. It seems like the actions are random. This is because the Q-table starts off with a random initialization. There isn't a learning rate, exploration rate, or discount value. The Q-table gets updated if the actions of the agent yield a reward. The movement of the agent is based on the maximum Q-value of the adjacent states. However, the agent after about thirty runs, takes specific actions in particular states. This improves its ability to reach the target (sometimes). The learning is allowing the agent to reach the target more frequently. Other times it gets stuck in loops and not moving at all. This is detrimental because it takes time for the agent to get

out of doing these useless actions. Overall, I notice that the agent is learning, but it is slow – this and the fact that it gets stuck in some cases (ex. Gets stuck not moving).

4. Enhance the driving agent

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?

There were multiple changes that I made to my implementation that improve the agents ability to reach the final target:

Removed states & keys in Q table: I removed the 'right' input from the state as it is already dealt with by the right-of-way of the traffic lights. The result of this action removes a number of states from the Q table.

Random Q table: I initially had a table that was initialized with zeros. This causes the agent to repeat its actions and then get stuck in cycles on the environment. By randomly initializing the values of the Q table, this happens a lot less frequently. The agent originally has an average of 1.6 average rewards per action with the zero table in 30 runs, but with a random table - it is about 1.8. I multiplied the random values with four (since the agent has four actions) to get a better spread. By trying to spread it further, I noticed no additional benefit.

Evaluate future Q values: I made it so that the Q table considers the next possible states and what their Q values are. This allows the high Q values from future states to impact the agents decision on choosing what action to make. This improves the overall learning of the agent.

Added exploration rate: I added an exploration rate that helps cover more states in my Q table. This ensures that the agent is not under time pressure and increases the average reward per action after thirty runs by 1.0 with an exploration rate of 5% (found to be optimal by trial and error) compared to when there was no exploration rate.

Added discount rate: I added a discount rate that trades off the importance of sooner rewards for later rewards (future Q values). I tested it and reached an optimal value of 40% through trial and error. This value gave the least amount of penalties and increased the speed of learning for the agent. The quantitative result is an increase of 0.4 for the average reward per move after 30 runs compared to when there was no discount rate.

Added learning rate: I added a learning rate that allowed the agent to improve its results with fewer runs. The quantitative result is that with a learning rate of 80% (by trial and error), the average reward per action increases by 0.5 from 30 runs compared to when there was no learning rate.

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?

My agent was able to find a close to optimal policy. As the number of runs increases, I witnessed the average reward improve significantly – this was also true for rewards per action. The increase in the number of runs leads to an increase in the reward per action with some fluctuation. The overall indication of this is that the agent gets stuck less in cycles and just not moving.

If I were to continue to try and improve my policy, I would have to do so by testing various Q tables and possibly other learning rates, discount values, and exploration rates.