P4: Train a Smartcab to Drive

Section 1: Implement a Basic Driving Agent

The initial simulation was run with enforce_deadline=False (there was no time limit on each trial each trial was going to run until the destination was reached or user terminates the simulation), random actions selected from action list (none, forward, left, right), the simulation was ran n_trials=4 (it was given 4 tries to reach the destination). As to be expected the actions of the agent observed are random and there is no learning going on of the agent as no matter what the inputs are the agent selects the next action randomly. As observed in the simulation run, the agent reached the destination every time (as that is the only termination criteria for the simulation with the above settings not counting user termination of the simulation). In each case the agent took a while to reach the goal with 53 time cycles as minimum and 217 time cycles as maximum.

Trial 0: 53 time cycles
Trial 1: 54 time cycles
Tral 2: 217 time cycles
Tral 3: 91 time cycles

Reference file: run01.txt

Section 2: Identify and Update State

In order to identify agent states to use in our simulation we basically need to identify minimum number of useful states to reach our goal (dropoff location). In practice this means we need minimum number of states to encode the rules that the agent must follow. The reason we say minimum number of states is that we don't need superfluous or redundant states, as this will only complicate our logic and will take longer to run.

- 1st rule: is smartcab must follow traffic light rule so we need to know **light: red/green** state
- 2nd rule: on green light, you can turn left only if there is no oncoming traffic at the intersection coming straight so we need to know **oncoming: none/forward** state.
- 3rd rule: on red light, you can turn right if there is no oncoming traffic turning left or traffic from the left going straight so we need to know **oncoming: none/left** state and **left: none/forward** state.
- although this is not a rule we need **next_waypoint** location the agent will use this to reach the goal (dropoff location).

Finally combining all the states we need: **light: red/green**, **oncoming: none/forward/left**, **left: none/forward**, **next_waypoint**.

Note we do not need **right: none/forward/right/left**, **oncoming: right**, **left: right/left** states according to the traffic rules we are working with, we are working with traffic lights, if for example we had stop signs we'd need some of the states we are discarding, as well as other states that we do not currently possess, like for example order of approaching of intersection.

For ease of computation we will keep all states from **oncoming** and **left** and only remove **right**. So in the end we end up with **light**, **oncoming**, **left**, **next_waypoint** states.

The state changes for each turn was observed, as the state is taking its values from inputs and next_waypoint and those tend to change most of the time (each turn), the state changes too.

Section 3: Implement Q-Learning

Q-Learning algorithm was implemented in the agent.py file under class **QLearner**. Some changes have been made to **update** function in **LearningAgent** class to accommodate code from section 1 & 2 (basic = True - will run the code from section 1 & 2, where basic = False - will run the Q-Learning algorithm, Q-Learning will be the default).

The simulation was run with basic=False, enforce_deadline=True, n_trials=20. The QLearner was run with default alpha (0.5), gamma (0.7), epsilon (0.05) values. The default values have been selected by intuition and are expected to be changed to improve the learning.

Out of 20 trials observed only one did not reach the destination, this looks to be a big improvement over the random run in section 1, considering that this new run was using a deadline. One can see how the learning is progressing, for example in trial 3 one can see how sometimes the car moves forward on the red light, where in trial 6 one does not notice the same behavior, as the agent has been penalised for moving forward on the red light and the agent adjusted its strategy (it was noticed later, that on trial 18 the agent did run (moved forward) a red light, but it looks like it did so because the goal (destination) was on the next move, we might want to adjust the penalty for running a red light to be greater than the reward for reaching the destination to avoid this). Additionally the agent does move on green light where with random selection action=None was always as good a choice as action=forward.

Reference file: run03.txt

In order to evaluate enhancements added to the Q-Learning some additional code was added to the original QLearner test runs in order to compare the QLearner with enhanced QLearner.

Runs with (Alpha = 0.5, Gamma = 0.7, Epsilon = 0.05) - intuition values 1st selected
Mean for the last 25 trials (5 runs of 100 trials): (14.44, 20.6, 19.72, 14.32, 13.4)

Section 4: Enhance The Driving Agent

An attempt was made to iteratively find the optimal alpha, gamma, epsilon for the Q-Learning algorithm, for this purpose **find_optimal** function was created in agent.py file. This function iterates through all possible combinations of the given alpha, gamma, epsilon values. The runs have been made with n_trails=100, in order to evaluate the results mean was taken of the last 50 trials for each run. Max value was found of all the means collected.

Given all the combinations it was necessary to run multiple iterative runs to try out different combinations, each time adding/removing some values of alpha, gamma, epsilon depending on the results, otherwise the processing time would have been to great.

1st iterative run:

alpha_a = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)
gamma_a = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)
epsilon_a = (0.02, 0.05, 0.07, 0.1, 0.13, 0.15, 0.18, 0.2)

Total combinations: 392
Optimal Combination of Alpha, Gamma and Epsilon: (0.5, 0.1, 0.05)
Optimal Average Steps: 10.96

2nd iterative run:

alpha_a = (0.3, 0.4, 0.5, 0.6, 0.7)
gamma_a = (0.05, 0.07, 0.1, 0.2, 0.3)
epsilon_a = (0.02, 0.05, 0.07, 0.1, 0.13)

Total combinations: 125
Optimal Combination of Alpha, Gamma and Epsilon: (0.4, 0.3, 0.07)
Optimal Average Steps: 10.76

3rd iterative run:

alpha_a = (0.3, 0.4, 0.5, 0.6)
gamma_a = (0.05, 0.07, 0.1, 0.2, 0.3, 0.4, 0.5)
epsilon_a = (0.05, 0.07, 0.1)

Total combinations: 84
Optimal Combination of Alpha, Gamma and Epsilon: (0.5, 0.07, 0.05)
Optimal Average Steps: 11.5

4th iterative run:

alpha_a = (0.4, 0.5)
gamma_a = (0.07, 0.1, 0.2, 0.3)
epsilon_a = (0.05, 0.07)

Total combinations: 16
Optimal Combination of Alpha, Gamma and Epsilon: (0.4, 0.2, 0.07)
Optimal Average Steps: 10.94

Reference file: run04.txt (1st iterative run) note this file only includes the last two runs (combinations of alpha, gamma, epsilon) of 392

Note: attempting to run all the combinations it was necessary to turn off the graphics of the simulation as it was taking too long to render (in the simulatior.py file) also last line in the **update** function of the **LearningAgent** had to be commented out for the same reason.

After running the above combinations we selected the relative optimal (they are only optimal from the subset given to alpha, gamma, epsilon in reality the total set of values is much larger and has a range of 0.0 to 1.0) value for alpha, gamma, epsilon.

Alpha = 0.5
Gamma = 0.2
Epsilon = 0.07

Finally we can run the **QLearner** with the found “optimal” values of Alpha = 0.5, Gamma = 0.2, Epsilon = 0.07 and other values as follows: n_trails=100 and enforce_deadline=True. The result was observed to be 5 out of 100 failed to reach the destination all others succeed, with last failure on trial 38 of 100. One also can see that the agent hardly ever receives a negative reward, infact it only happens a few times in the later half of the 100 runs, and none of the trials are net negative in terms of reward. This result is consistent enough to argue that the agent will arrive at the destination most of the time and do so with positive total reward. For the last 25 trials the mean steps to reach destination was 12.5 vs (10.96, 10.76, 11.5, 10.94) from the iteration minimum means, I would argue that this number is within that range.

Reference file: run05.txt

Satisfied now with the results from the new optimal values we can now directly compare the runs from the original QLearner, below are the numbers for initial results (from section 3) and new results from the optimal values.

Runs with (Alpha = 0.5, Gamma = 0.7, Epsilon = 0.05) - intuition values 1st selected
Mean for the last 25 trials (5 runs of 100 trials): (14.44, 20.6, 19.72, 14.32, 13.4), mean = 16.5

Runs with (Alpha = 0.5, Gamma = 0.2, Epsilon = 0.07) - optimal values found
Mean for the last 25 trials (5 runs of 100 trials): (12.5, 13.96, 12.8, 13.72, 10.48), mean = 12.7

As one can see there is a substantial decrease in the number of steps to destination from the original QLearner values to the optimized QLearner values. The average mean for original was 16.5 vs average mean for the optimal is 12.7 a substantial decrease in the number of steps to destination.

One can also see that there was some luck involved with the original values (intuitive values) selection they are actually not that far off from the optimal values. As a test let's just pick another set of values that might now work as well if we wanted to try to see how good or bad the optimal values are, still we are not going to select very bad values.

Runs with (Alpha = 0.8, Gamma = 0.6, Epsilon = 0.2) - not very good values (test)
Mean for the last 25 trials (5 runs of 100 trials): (22.36, 19.52, 21.04, 19.76, 21.68), mean = 20.88

As one can see the average mean is 20.88 vs. 12.7 (optimal) that is another relatively large jump away from the optimal, even though intuitively it might not look that way, or does it? For example making Epsilon 0.2 makes it that a random direction is selected 20% of the time, surely that will not go so well for the agent trying to logic their way to a given destination.

In the end I would argue that QLearner with (Alpha = 0.5, Gamma = 0.2, Epsilon = 0.07) values, after 100 trials is relatively close to the optimal route to the destination.

References:

https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/

http://mnemstudio.org/path-finding-q-learning-tutorial.htm