Implement a Basic Driving Agent

To begin, your only task is to get the smartcab to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

The next waypoint location relative to its current location and heading. The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions. The current time left from the allotted deadline. To complete this task, simply have your driving agent choose a random action from the set of possible actions (None, 'forward', 'left', 'right') at each intersection, disregarding the input information above. Set the simulation deadline enforcement, enforce_deadline to False and observe how it performs.

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

Eventually the agent reached the destination, but it took him 102 moves, way above the 30 moves deadline for that specific problem. Optimally, he could have reached with the destionation with around 10 moves only.

The agent moves with no regard to other variables received as input:

- traffic lights, it may take the decision of moving forward even though the traffic light prohibits it
- other vehicles
- next waypoint
- how many moves left to complete

Acting randomly is it as if the actuator is unaware of the sensor inputs or any representation of state.

Sidenote: What I noticed about the grid specifically (and it seems not important for the exercise) is that there are no margins. On the left side, if the agent moves left, he will end up on the right side, as if it is a continuation of the road. The same thing if the agent moves up on the upper magin, the agent will move to the lower part of the grid.

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and

update the agent's current state using the self.state variable. Continue with the simulation deadline enforcement enforce_deadline being set to False, and observe how your driving agent now reports the change in state as the simulation progresses.

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

To derive the state, I need to look at the variables identified by the sensor. These are:

- next_waypoint: {'forward', 'right', 'left'}
- inputs
  - traffic_light: {'red', 'green'}
  - direction of oncoming car, if any: {None, 'forward', 'right', 'left'}
  - direction of car on the right, if any: {None, 'forward', 'right', 'left'}
  - direction of car on the left, if any: {None, 'forward', 'right', 'left'}

My state will be a combination of these variables.

I could apply an elaborate state transformer function to further reduce the space dimension, but that would imply some domain knowledge upfront. For example, I could state that I only care if a car is coming in my direction ('forward'), and that all others states are irrelevant for the problem solution. That would greatly reduce the number of different possible states, but as mentioned, implies an assumption.

Time could also be a relevant variable, but in this project time or deadline should not affect the calculation of the best action, so I will not incorporate this dimension into state.

So how many possible states?

I will not make any assumptions in this problem and consider all variables. So to arrive at the number of possible states, I need only to multiply the set sizes: 3 x 2 x 4 x 4 x 4 = 384 possible spaces.

Reviewer: for this section can you also provide some brief justification for why each of the next_waypoint, traffic light, direction of oncoming car, direction of car on the right, and direction of car on the left are actually needed to represent. Therefore why does the agent need to keep track of these variables?

Let's now consider we have access to some domain knowledge upfront, and will apply it in order to speed up Q-learning convergence.

The agent needs to keep track of the variables to know if the action will obey the traffic rule or not. These are U.S.rules of traffic:

1. On a green light, a left turn is permitted if there is no oncoming traffic making a right turn or coming straight through the intersection.

2. On a red light, a right turn is permitted if no oncoming traffic is approaching from your left through the intersection.

Let's see how useful each variable is for the agent to know if the action obeys traffic rules:

- Traffic Light: Required for rule 1 and rule 2.
- Oncoming car direction: Required for rule 1. It only matters whether the oncoming car is moving forward or making a right turn, or not, so it could be converted to binary.
- Car approaching from the left: Required for rule 2. It only matters whether the oncoming car from the left is moving forward or not, so it could be converted to binary.
- Car approaching from the right: Not required

Applying domain knowledge, I know that there is no need for the variable 'right', and that I can transform 'ongoing' and 'left' into binary variables. That can be used to reduce the number of spaces.

OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

With domain knowledge applied in the state transformation function, we could reduce the number of states, which would make Q-Learning converge quicker. The current number of 384 states is not large, but it can quickly become intractable as we add more variables to our sensors.

If we consider domain knowledge of the U.S traffic rules, summarized in the two rules described in the problem, I know there is no need to know the direction of cars coming from the right. We can also reduce the variable 'oncoming' and 'left' to binary variables: whether or not the oncoming car is moving forward or making a right turn, and whether or not a car approaching from the left is moving forward.

That will reduce the number of possible states to: 3 x 2 x 2 x 2 = 24 possible spaces.

As seen, with a little domain knowledge we can greatly reduce the number of possible spaces which will make Q-learning converge faster.

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by

the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement enforce_deadline to True. Run the simulation and observe how the smartcab moves about the environment in each trial.

The formulas for updating Q-values can be found in this video.

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

The agent is goal oriented, since next_waypoint is one of the variables used in the state. It moves towards the goal, only changing the default behavior when other factors are in place, such as red traffic lights or a car coming in its direction.

The behavior is due to the knowledge encoded in the q-values. In layman's terms, what the car is learning is it will get a positive reward if it follows the next waypoint direction, respect the traffic signals, and don't crash into nearby cars; and a negative reward if it breaks one of these rules.

It is a simplist example, and there are more subtle behaviors that could be learned with lots of iterations, considering q-value encodes both current and future rewards.

Reviewer: Can you go into a bit more detail in terms of some of the actual behavior seen from the agent. Therefore is the agent following the rules of the road? Going in circles at times? Getting stuck in local minima? What do the initial trials look like? Random exploration still? Does the agent learn to follow the best next_waypoint? etc... As there must be some issues(at least in the beginning trials, with random exploration with epsilon).

The basic driving agents does not follow the rules of the road, as the action taken are random and does not take reward into consideration. The Q-learning agent gets a -1 negative reward whenever it fails to obey the traffic rules, so it learns to obey the traffic rules through a feedback system, even if these rule are unknown to the agent.

As for direction, when the agent moves randomly, it gets stuck in small sections of the map, moving in circles. It eventually finds the waypoint if it is close to the starting point, but if it is far it will take a long time to reach it. With enough trials, the Q-learning agent learns to move torwards the next waypoint (which will give a reward of 12 when it reaches) except if that direction disobey a traffic a rule.

When the exploration rate is high, such as in the first trials, the behavior of the Q-learning agent is similar to the random action agent. The exploration is important for the agent to learn the Q-values for each pair; it can't know beforehand what is the reward for the pair unless it experiences it, or it is defined as domain knowledge in the problem formulation. As the exploration rate decays with the number of of trials, its behavior (action

selection policy) converges to the optimal behavior.

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your smartcab:

Set the number of trials, n*trials, in the simulation to 100. Run the simulation with the deadline enforcement enforce*deadline set to True (you will need to reduce the update delay update_delay and set the display to False). Observe the driving agent's learning and smartcab's success rate, particularly during the later trials. Adjust one or several of the above parameters and iterate this process. This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

QUESTION: 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?

The three parameters I've tuned are:

- learning rate (alpha)
- discount factor (gamma)
- exploration rate (epsilon)

Learning rate is the rate at which the agent updates its q-value. An agent with 0 learning rate will never update its q-value, while an agent with 1 learning will always replace the existing q-value with the new calculation.

Discount factor is the the rate at which future rewards are discounted. It determines the importance of future rewards for the model. If discount factor is 0, future rewards do not impact the model. If it is 1, it will be as relevant as the current reward. If there is no terminal state, with a discount factor of 1, the future rewards will generally be infinite.

Finally, exploration rate determines the willingness of the agent to explore its environment, despite its knowledge of the best action encoded in its q-value. If exploration rate is 1, the agent will always explore, choosing a random action. If the exploration rate is 0, the agent will never explore, always deciding which action to take based on the maximum q-value calculation. This tradeoff is generally known as the exploration/exploitation dilemma.

Other parameter I could have optimized is the decaying rate for the exploration factor, but I decided to set it as a function of number of trials, so it decreases with every new trial.

I've optimized the model using a grid search approach, similar to the approach used to optimize parameters for a supervised/unsupervised learning model. I've tested values from learning rate and discount factor from 0 to .9 exclusive, with interval of 0.05, and exploration rate from 0 to 1 inclusive with interval of 0.1. That equals 2816 variations (16x16x11).

The metrics I used to evaluate are number of failures, average relative time of completion and average number of penalties. The relative time of completion is calculated by *time / (time + deadline)*, so it measures how much of the available time the car used to reach its destination.

The reason I've decided not to use total reward as a relevant metric is because it is highly dependent on the distance between the starting point and end point, and that distance is random in each trial.

For each parameters combination, out of 100 trials I only used the result of the last 10 trials to calculate the metrics, considering the first 90 trials were used for learning.

The best result has an alpha of 0.8, gamma 0.7 and epsilon 0.1. It takes an average of 27.63% of the available time for the car to reach its destination, with 0 failures among the 10 trials, and 0 penalties in all of the evaluated trials.

Reviewer: Please include the top 20 combinations of parameters that yielded the best results

Here are the top 20 combinations of parameters that yielded the best results:

Alpha	Gamma	Epsilon	Avg time to complete	Number of failures	Avg number of penalties
0.80	0.70	0.10	27.63%	0	0.00
0.85	0.20	0.60	28.14%	0	0.00
0.20	0.60	0.00	28.78%	0	0.00
0.40	0.80	0.10	28.87%	0	0.00
0.35	0.10	0.20	28.99%	0	0.00
0.40	0.15	0.70	29.15%	0	0.00
0.10	0.65	0.20	29.35%	0	0.00
0.65	0.15	0.90	29.70%	0	0.00
0.15	0.75	0.20	29.88%	0	0.00
0.05	0.05	1.00	29.88%	0	0.00
0.45	0.20	0.00	29.90%	0	0.00
0.55	0.30	0.80	30.01%	0	0.00
0.05	0.30	0.40	30.18%	0	0.00
0.45	0.40	0.20	30.23%	0	0.00
0.70	0.60	0.00	30.24%	0	0.00
0.25	0.35	0.20	30.27%	0	0.00
0.30	0.50	0.40	30.47%	0	0.00
0.85	0.85	0.90	30.48%	0	0.00
0.80	0.75	0.80	30.57%	0	0.00
0.80	0.25	1.00	30.81%	0	0.00

QUESTION: 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, it gets close to finding an optimal policy. The optimal policy (for this example) is where the agent reaches the destination with the minimum time required, respecting the traffic rules and not crashing the car.

In the first example we mentioned a test where there were 30 moves available, but a quick glance would tell you it only needed 10 moves to reach the destination. The final agent is using on average 27.63% of the available time, so it would complete the first track in approximately 8 moves (8/30 = 26,7%).

References:

- https://en.wikipedia.org/wiki/Q-learning
- Udacity RL classes
- Artificial Intelligence course, available on edx (Berkeley)
- An introduction to Multiagents Systems (Michael Wooldridge)
- Artificial Intelligence: A Modern Approach (Stuart Russel, Peter Norvig)