Project 4 Train a Smartcab to Drive

- Author Neo Xing, 2016/09

Implement a Basic Driving Agent

Instructions

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?

ANSWER

1. The smartcab makes it to the destination with a small chance, as it drives randomly.
2. Given enough trial times, it will make it, otherwise it usually aborts after running out of time. Based on `random_agent_log.txt`, 28 positive results out of 50 trials, namely a success rate of 56%.
3. The agent simply ignores restrictions to follow traffic lights and avoid collision with other cars.

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?

ANSWER

- I have identified `next_waypoint` and `inputs` for the states. The `deadline` is not appropriate as it is not crucial for the functionality, and to complicated for optimization due to its huge quantity of discrete states.

- The `next_waypoint` represents road map, while `inputs` contains information of environment such as traffic light and other cars. The smartcab will need them to make decision to plan the trip while following traffic rules, managed by the reward and punishment rules of the algorithms.

OPTIONAL QUESTION

> 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?

ANSWER

- According to the following table of states, the total number is `3*2*4*4*4=384`. It's unreasonable as it'll take too much time for Q-Learning algorithm to visit all possible states.

feature	states
next_waypoint	left, right, forward
light	red, green
left	None, left, right, forward
right	None, left, right, forward
oncoming	None, left, right, forward

- After apply traffic rules, the states table can be reduced greatly to a decision tree with only 9 possible states. With `n_trials=100` it makes Q-Learning algorithm able to learn and make informed decisions.

light	next_waypoint	left	right	oncoming
red	right	left car not turning right	-	-
red	right	no car on left	-	-
red	right	left car going forward	-	-
red	right	left car not going forward	-	-
green	right	-	-	-
green	forward	-	-	-
green	left	no car oncoming	no car oncoming	-
green	left	left car oncoming	or right car oncoming	oncoming car turn left

green	left	left car oncoming	or right car oncoming	oncoming car not turn left

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?

ANSWER

- The smart agent has a much higher success ratio to reach the destination, and much lower chance to break traffic rules.
- This is because the smart agent learns from its experience, which is evaluated and improved by feedback of reward for positive actions and punishment otherwise. The next time the smart agent meets the same state, it can choose the action with the highest reward.

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?

ANSWER

- I firstly run a search with `n_trails=20` to find the rough impact and ranges of three parameters. It turns out most epsilons will go to values around 0, which means the optimal result is not much related with `epsilon`.

Searching Path

success ratio	percentile time	alpha	gamma	epsilon	final_epsilon
0.35	0.879	0.0	0.0	0.0	0.0
0.4	0.807	0.0	0.1	0.7	0.7
0.4	0.762	0.0	0.4	0.3	0.3
1.0	0.415	0.1	0.0	0.0	0.0
1.0	0.411	0.1	0.3	0.0	0.0
1.0	0.366	0.1	1.0	0.0	0.0
1.0	0.34	0.2	0.4	0.0	0.0

Best Result

success ratio	percentile time	alpha	gamma	epsilon	final_epsilon
1.0	0.34	0.2	0.4	0.0	0.0

- The decay rate set to constant `1.0`, with `epsilon` set to `0.01`. It's the most simple and efficient choice, without affecting the effectiveness of algorithm. Then fine search is conducted on different combinations of `alpha` and `gamma` with `n_trials=100` for best parameters. The following table shows its learning path and final result.

Searching Path

success ratio	percentile time	alpha	gamma	epsilon	final_epsilon
0.12	0.934	0.0	0.0	0.01	0.01
0.24	0.888	0.0	0.1	0.01	0.01
0.25	0.896	0.0	0.4	0.01	0.01
0.27	0.866	0.0	1.0	0.01	0.01
1.0	0.441	0.1	0.0	0.01	0.01
1.0	0.39	0.1	0.3	0.01	0.01
1.0	0.384	0.3	0.0	0.01	0.01
1.0	0.376	0.7	0.1	0.01	0.01

Best Result

success ratio	percentile time	alpha	gamma	epsilon	final_epsilon

1.0	0.376	0.7	0.1	0.01	0.01

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?

ANSWER

- My agent is approaching the optimal policy with n_trials increasing. However, it does have penalties due to occasionally violating traffic rules such as breaking lights and conflicting with other cars.
- However, it breaks traffic rules at the beginning but as it learns from experience, it avoids making the same mistakes. However, it still makes mistakes when it is in a circumstance that it never sees before. The number of times it breaks traffic rules decreases and total rewards increase. Also, the optimal policy will never have loops in the routes.