Train a Smartcab How to Drive

Project 4: Reinforcement Learning

Install

This project requires Python 2.7 with the pygame library installed:

https://www.pygame.org/wiki/GettingStarted

How to run the program

Make sure you are in the top-level smartcab directory.

Then run:

`python smartcab/agent.py`

or:

`python -m smartcab.agent`

Tasks

Download smartcab.zip, unzip and open the template Python file `agent.py` (do not modify any other file). Perform the following tasks to build your agent, referring to instructions mentioned in `README.md` as well as inline comments in `agent.py` .

Also create a project report (e.g. Word or Google doc), and start addressing the questions indicated in italics below. When you have finished the project, save/download the report as a PDF and turn it in with your code.

1.Implement a Basic Driving Agent

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.

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?

The smartcab eventually make it to the destination, however the smartcab does not always hit the waypoint within the time frame. The resort of random agent is present in `agent.txt` . The success rate of the random agent are respectively 19% and 24%. The result obtained was what I was expecting it is because the agent takes random action. However, the smartcab ignore the traffic lights and the two cars collided with each other. What is important that the agent does not learn own actions. It means that

the agent does not improve itself.

2. Inform the Driving Agent

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.

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?

I identified two state variables before implementing a Q-Learning driving agent.

- *Each intersection*

I choose these intersection states because there are some rules in US we

have to obey. Each intersection include the traffic lights (red / green), the oncoming traffic (left , right or None) from 2 other cars (oncoming , left).

***US right-of-way rules:** On a green light, you can turn left only if there is no oncoming traffic at the intersection coming straight. On a red light, you can turn right if there is no oncoming traffic turning left or traffic from the left going straight.*

- *next_waypoint*

I choose this state because we can not directly choose destination and next waypoint informs which way we would prefer to travel.

I don't include deadline. There are a lot of possible deadline values, thus it will be large in memory. Additionally, the deadline could possibly influence the agent in making illegal moves when the deadline is near.

3. Implement a Q-Learning Driving Agent

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 when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

Q-Learning is implemented in QLearningAgent.py . I implemented the Q-learning algorithm as follows.

The Q-learning Algorithm Steps:

$$Q_{t+1}(s_t, a_t) \leftarrow \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right)$$

- ***Initialize the Q-values table, Q(s, a).***
- ***Observe the current state, s.***
- ***Choose an action, a, for that state based on one of the action selection policies explained here on the previous page (-soft, -greedy or softmax).***
- ***Take the action, and observe the reward, r, as well as the new state, s'.***
- ***Update the Q-value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.***
- ***Set the state to the new state, and repeat the process until a terminal state is reached.***

Reference: http://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html

I noticed that the agent's behavior is too different from what it used to be. The agent is learning the US right-of-way rules and is trying to manage to get to the waypoint before the deadline. Also, the smartcab getting to the destination more quickly. This is because the agent examines history of own behavior and what sort of rewards have occurred when it's taken actions in given states before.

4. Improve the Q-Learning 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 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?

I turned parameters alpha, gamma, epsilon and epsilon degradation. Just by looking at the results from the agent in Qlearn.txt .

Here are some performances:

alpha	gamma	epsilon	epsilon degradation	penalties	rewards	moves	success rate
0.5	0.5	1	0	695	9.4	31	19%
0.5	0.5	1	0.1	35	22.0	15	92%
0.5	0.5	0.1	0.01	28	18.0	8	98%
0.2	0.8	0.1	0.01	19	22.5	9	89%
0.8	0.2	0.1	0.01	23	18.0	6	99%
0.9	0.2	0.1	0.01	30	26.0	26	98%
0.7	0.2	0.1	0.01	30	20.0	16	98%
0.7	0.4	0.1	0.01	30	23.5	16	98%
0.8	0.4	0.1	0.01	22	22.0	12	100%

The best set of parameters are below:

- ***Learning Rate: 0.8***
- ***Discount Value: 0.4***
- ***Epsilon Value: 0.1***
- ***Epsilon Degradation Rate: 0.01***

The Smartcab reached the destination around 95~100%. This performance is much better than `agent.py` *.*

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?

I believe the agent get fairly close to finding an optimal policy, but it is not perfect. The smartcab is not always arrive on time, and it still takes illegal actions sometimes(Refer to the answer to the previous question). Actually, my best agent parformed 22 illegal actions.

Here is an example:

LearningAgent.update(): deadline = 31, inputs = {'light': 'red', 'oncoming': 'forward', 'right': None, 'left': None}, action = left, reward = -1.0

LearningAgent.update(): deadline = 30, inputs = {'light': 'red', 'oncoming': 'forward', 'right': None, 'left': None}, action = right, reward = -0.5

Additionally, I noticed the smartcab reaches the destination quickly with less number of moves. The overall rewards are shorter towards the end due to the fact that the agent makes it to the destination early. However, In the real world, it causes car crash when the cab takes illegal actions. Therefore, I can say that an optimal policy taking the smallest route without performing many illegal actions.