smartcab-Copy1

September 7, 2016

1 P4: Smartcab

1.1 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?*

ANSWER:

The instructions for this question explicitly state that these random actions disregarding *the state of the traffic light*. Since the already-coded `Environment.act()` flags `move_okay` depending on what action the agent wants to take and the state of the traffic light, I would have to change this method or write a new one (which does not make sense) to follow these instructions fully.

However, we can more easily disregard the other inputs assessed outside of `Environment.act()` and assign random actions to our agent. By doing this we can see that our agent does make it to the destination in most cases. In the other cases it runs up against the `hard_time_limit` set to `-100`.

1.2 Inform the Driving Agent

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:

The determining factors for our state description are what aspects of a state are tied to rewards for an action from that state.

Since rewards rely on whether the learning agent has followed (or not followed) traffic and safety laws and whether it has moved toward the destination as per the planner's `next_waypoint` returned value, we can combine inputs into a succinct description:

```
self.State = namedtuple('State', ['next_waypoint', 'okay_moves'])
self.OKAY_MOVES = ('all', 'all but left', 'right on red', 'none')
```

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?

ANSWER: With only 12 possible states given its description (3 * 4), Q-Learning can quickly determine the optimal policy based on our model's rewards

1.3 Implement a Q-Learning Driving Agent

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:

As the learning agent progresses through the trials, it begins to take actions toward the destination more frequently. These actions also begin to follow traffic rules more often.

This behavior changes are occur because: 1. The agent is making more moves toward the destination (positive rewards). - The agent is driving more safely (negative rewards)

1.4 Improve the Q-Learning Driving Agent

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:

Summary of parameter tuning:

```
self.epsilon = 0.5  # tested .1, .3, .5 and .8
self.decay_rate = self.epsilon / 400  # adjusted with eps

self.learning_rate = 0.2  # alpha, tested .1 and .3
self.discount = 0.2  # gamma
```

Details of tuning `self.epsilon`: - `self.epsilon = 0.1` - 64.7% of trials where destination is reached - `self.epsilon = 0.3` - 92.8% of trials where destination is reached - `self.epsilon = 0.5` - 96.5% of trials where destination is reached - `self.epsilon = 0.8` - 95% of trials where destination is reached

With these parameters assigned above, our **smartcab** will reach its destination about 95 times out of our 100 trials with the deadline enforced. (Average determined from sample of 10 batches of each set of 100 trials.)

Negative rewards in the latter 3/4 of the 100 trials were rare, in the range of 0 to 3 instances.

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:

I would say my agent is getting close to the optimal policy, but since there is no default penalty for every move (or non-move), the agent is not incentivized to minimize travel time. Perhaps, as well, the penalty for traffic violations should lower than a separate reward for one that causes a collision. These penalties could also be increased to ensure that no violation occur in later trials.