smartcab-Copy2

September 13, 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

Instructions:

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: The determining factors for our state description are what aspects of a state are tied to the 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 the following description:

```
State = namedtuple('State', ['next_waypoint', 'light',
                             'oncoming', 'left'])
```

Notes on excluded variables:

- The `inputs['right']` variable is not included because the learning agent's options for legal moves will not be affected by the presence and direction of the next action of another agent on the right-hand side.
- The `deadline` variable used in environment.py is not included either, but unlike `inputs['right']`, it has an impact on rewards albeit an indirect one. That is, when the deadline is reached the learning agent *does not* obtain the 10 points for reaching the destination. However, because the `next_waypoint` variable *is* included in the state description, the learning agent is incentivized to move toward the destination through the `+2` reward for an action taken that matches this variable. In addition, if the `deadline` variable were included, it would increase the total number of possible states exponentially, and thus, drastically increase the training time the learning agent would require to make decently informed decisions.

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: Every possible combination of values for each state variable (listed below) is: $2 * 4^3 = 128$

```
light: 'green', 'red'
next_waypoint: 'forward', 'left', 'right', None
oncoming: 'forward', 'left', 'right', None
left: 'forward', 'left', 'right', None
```

This is reasonble, although not ideal, since even after running 10000 trials the length of the `q_table` indicates that not all states have been assessed. Granted, this is also a result of only having only 3 other agents present in the environment.

1.3 Implement a Q-Learning Driving Agent

Instructions: 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 q-learning algorithm does change the behavior of the primary agent from random to a progressively more "intelligent" or "informed" decisions. As the primary agent learns from 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 occur because of the reward structure's impact on how the agent perceives the 'best' action given its current state. The negative rewards discourage unsafe actions, while the positive rewards encourage actions that take the agent towards its destination. This incentive structure is integral to the q-value, which is defined as "the value for arriving is s, leaving a, [and] proceeding optimally thereafter" (from the Reinforcement Learning lesson). This q-value is the measure by which we compare different actions that could be taken in a given state to determine the "best" action for our learning agent.

To "learn" which actions to take, the agent updates the q-value through a process called *value iteration*. As the agent visits a state repeatedly, the q-value update formula uses a discount value gamma, where $0 > gamma < 1$, to create a geometric series that gradually converges on a given action's q-value for that particular state. As these q-value converge on their actual value, the learning agent begins to take actions that earn more rewards based on the incentive structure.

1.4 Improve the Q-Learning Driving Agent

Instructions:

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: *Somewhat optimal parameters based on a cursory investigation:*

```
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
```

Sample tuning of `self.epsilon`, all other params held constant:

- `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: An optimal policy would always take actions to move toward the destination unless this action would violate traffic laws. In this case, the optimal policy would then determine the next best move (with regard to the direction of the destination) that does *not* violate traffic laws. I.e. first priority is safety, second is reaching the destination.

I would say my agent is getting close to the optimal policy. As mentioned in the previous question, with appropriate parameters, the learning agent almost always reaches the destination by the deadline after the first few trials. In addition, when measuring the frequency of negative rewards (i.e. traffic violations), I found very few occured after the agent learned over 25 trials.

However, this 0 to 3 range for violations for the latter 75 trials is much too high according to our definition of an optimal policy which prioritizes safety over reaching the destination.

Also, while the learning agent consistently reached the destination by the deadline, often it made non-optimal moves with respect to the destination. This caused it to travel longer than necessary, even while driving safely. 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.