UDACITY

PROJECT

Train a Smartcab to Drive

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW 6

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

3 SPECIFICATIONS REQUIRE CHANGES

You have a good agent here and very impressed with your understanding of these reinforcement learning techniques. Just have a few minor issues, but should not be too difficult to update(and should help increase your understanding even more). We look forward to seeing your next submission. And keep up the hard work!

Implement a Basic Driving Agent

Student summarizes observations about the basic driving agent and its behavior. Optionally, if a visualization is included, analysis is conducted on the results provided.

Question 1: OPTIONAL REQUIRED

> "It is getting rewards when it stay idle on a red light and it gets punished when it gets idle and the light turns green."

Nice work with a red light(positive reward) and a negative reward when there is a green light without other traffic. However, there is actually one more instance that should also be discussed. What are the rewards when there is a green light **with** other traffic?

Question 2

> "In the environment.py file, the function step() is called to update the state of every agent and the environment for each step taken. By teh otrh hand the function act() is also called to validate the action taken to know if that action was good or wrong to take."

It is the `act()` function when an agent performs an action

Question 3

> "I would say that as the number of trials increases, the outcome change slightly but it is not significantly and it is just a variation result of the random executions."

Good! We do see some random noise here, but this is really not a trend, as this smartcab really isn't that smart yet. This lack of success does make sense as the agent is only exploring and not learning, collecting many negative rewards. As this behavior is similiar to a random walk.

Inform the Driving Agent

Student justifies a set of features that best model each state of the driving agent in the environment. Unnecessary features not included in the state (if applicable) are similarly justified.

Please also check out the code review, section. But this is a valid state. As it is always an important thing to determine a good state before diving into the code, as this will pave the way to an easier implementation. And nice discussion for the need of all these features.

And good ideas your deadline comments. As if we were to include the deadline into our current state, our state space would blow up, we would suffer from the curse of dimensionality and it would take a long time for the q-matrix to converge. Also that including the deadline could possibly influence the agent in making illegal moves when the deadline is near.

The reason this is marked as *Requires Changes* is that lastly for this section make sure you also provide some justification for why you did / why we can omit right traffic `inputs['right']`.

Question 5: Nice calculation. Just note that including the full state and not hard-coding states only results in a total of 96 total states, which really isn't too many to learn with a feasible number of training trials and a good epsilon decay rate. Maybe another idea to confirm this would be to run a Monte Carlo simulation(considering that all these state are uniformly randomly seen). Would 750 be enough? How many training trials could represent 750 steps? What about every state, action pair? Try changing the step

```
from sets import Set
from random import choice

def chance_of_visiting_all_states(iterations, k, n=24):
    r = range(n)
    total = 0
    for i in range(iterations):
        s = Set()
        for j in range(k):
            s.add(choice(r))
            if len(s) == n:
                total +=1
                break
    return float(total)/iterations

steps = 750
print "Chance of visiting all states in {st} steps: {ch}".format(st = steps, ch = chance_of_visiting_all_states(2000, steps, 96))
```

The driving agent successfully updates its state based on the state definition and input provided.

The driving agent successfully updates its state in the pygame window!

Implement a Q-Learning Driving Agent

The driving agent chooses the best available action from the set of Q-values for a given state. Additionally, the driving agent updates a mapping of Q-values for a given state correctly when considering the learning rate and the reward or penalty received.

Some minor code issues, check out the code review.

Student summarizes observations about the initial/default Q-Learning driving agent and its behavior, and compares them to the observations made about the basic agent. If a visualization is included, analysis is conducted on the results provided.

Nice observations! Thus especially with an epsilon decaying implementation, typically the behavior for the first few trials isn't any different from the randomly acting agent's behavior. Thus once the agent starts to learn and build up a sufficient q-learning table and the agent tabulates rewards and updates Qvalues for each state action pair, the agent will start to hone in on the correct waypoint while more highly weighting the smartcab's best actions.

Improve the Q-Learning Driving Agent

Student summarizes observations about the optimized Q-Learning driving agent and its behavior, and further compares them to the observations made about the initial/default Q-Learning driving agent. If a visualization is included, analysis is conducted on the results provided.

Great parameter tuning here. As it is crazy how the number of bad actions / accidents / violations seem to be directly correlated to your epsilon value in these graphs. Another interesting epsilon decaying rate to check out would be the Gompertz function. And use a negative of it.

And very cool idea to use a version of alpha decay and reduce the learning rate over time. Thus as time elapses, the agent becomes more confident with what it's learned in addition to being less persuaded by the information.

The driving agent is able to safely and reliably guide the *Smartcab* to the destination before the deadline.

Congrats on your ratings! One thing to look into is to determine how reliable your agent actually is. Try changing the number of testing trials to something like 50 or even 100. How does your agent perform then? Is it ready for the real world? Do you need more training?

Student describes what an optimal policy for the driving agent would be for the given environment. The policy of the improved Q-Learning driving agent is compared to the stated optimal policy, and discussion is made as to whether the final driving agent commits to unusual or unexpected behavior based on the defined states.

> "Example 1 ('right', False, True, True): For this example the optimal policy would be to turn right even though it has red light"

We actually DON'T have a red light here, as the `False` refers to that the light is `green`.

```
is_red_light = True if inputs['light'] == 'red' else False
```

Therefore the `right` action is still the optimal action, but this is why you see a negative reward for the `None` action in this state. One of the reasons why I recommend not self-creating states.

> "As far I could see, there is no state where the policy is different than what would be expected from an optimal policy."

When I look at your `sim_improved-learning.txt` file. I found the state of

```
('right', True, True, True)
 -- forward : -9.89
 -- right : 1.31
 -- None : 1.62
 -- left : -14.93
```

What is the action the agent took here? Is this really optimal? Isn't this why you created your `can_go_right`

Student correctly identifies the two characteristics about the project that invalidate the use of future rewards in the Q-Learning implementation.

Nice ideas here

- This is due to egocentric nature of the agent, as well as the random aspects of state that cannot be predicted ahead of time. Were the simulation changed to an allocentric view, where the agent could sense where it was in relation to the destination, etc, then gamma term would be a more important parameter for optimal performance and policy generation.
- Also that the destination itself moves after every trial. If we attempted to propagate reward away from the goal, we would eventually propagate reward away from every intersection.

RESUBMIT

DOWNLOAD PROJECT

6 CODE REVIEW COMMENTS ›

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

Student FAQ