UDACITY

PROJECT

Train a Smartcab to Drive

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

You have a good agent here and very impressed with your understanding of these reinforcement learning techniques. Just have a couple of sections to look into, but should not be too difficult to update(and should help increase your understanding even more). Check out the last section based on your submitted comment and we look forward to seeing your next submission. And keep up the hard work!

Getting Started

Student provides a thorough discussion of the driving agent as it interacts with the environment.

> "When the light is red the driving agent gets a postive reward since it is in stop postion. And when the light is green the agent is getting negative reward because it is not moving when it should."

Nice job examining the rewards for the agent! One way to force the agent to move is through simulating annealing.

Student correctly addresses the questions posed about the code as addressed in Question 2 of the notebook.

Good job checking out the environment! As there are many tuning knobs we can check out here and play around with.

Implement a Basic Driving Agent

Driving agent produces a valid action when an action is required. Rewards and penalties are received in the simulation by the driving agent in accordance with the action taken.

Agent produces a valid action!

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

> "No , since the agent is driving randomly , it make no sense to consider the rate of reliability."

With this finite grid, even a random agent can still stumble upon the destination from time to time, but still quite bad. So these numbers **make sense**.

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. Students argument in notebook (Q4) must match state in agent.py code.

You have modeled the environment very well.

```
state = (waypoint, inputs['light'], inputs['left'], inputs['right'], inputs['oncoming'])
```

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.

Good justification for the omission of the deadline. Also note that 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. But correct that including the deadline could possibly influence the agent in making illegal moves when the deadline is near.

The total number of possible states is correctly reported. The student discusses whether the driving agent could learn a feasible policy within a reasonable number of trials for the given state space.

> "Currently there are 20 training trials , i think we would either need to increase the number of trials or decrease some state space for the agent to learn a policy easily."

Definitely need more than just 20 training trials, but still too too many states. As a total of 384 states 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, 384))
```

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: (1) Chooses best available action from the set of Q-values for a given state. (2) Implements a 'tie-breaker' between best actions correctly (3)Updates a mapping of Q-values for a given state correctly while considering the learning rate and the reward or penalty received. (4) Implements exploration with epsilon probability (5) implements are required 'learning' flags correctly

Code looks great! Nice work randomly selecting an action when we have multiple actions with the same Q-Value and nice work with your Bellman equation here, as this refers to the current reward only

```
self.Q[state][action] = (1-self.alpha) * self.Q[state][action] + self.alpha*(reward)
```

For future reference you can also check out this version of the Q-Learning algorithm

```
self.Q[state][action] += self.alpha * (reward - self.Q[state][action])
```

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.

> "Yes , with increase in the no of training trials the number of bad actions is decreasing as is plotted with black dotted line in the above plot. And the average reward is increasing."

Nice observations here. The agent is getting a bit better here, as it is actually starting to learn as the trials pass by. As this is definitely expected with a decaying epsilon. Therefore we can reduce the chances of random exploration over time, as we can get the best of both exploration vs exploitation.

Maybe with a slower decay rate and with more training trails, this default Q-Learning driving agent could actually improve and explore and learn more states. But for now we can use this agent as a good benchmark.

Improve the Q-Learning Driving Agent

The driving agent performs Q-Learning with alternative parameters or schemes beyond the initial/default implementation.

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. This might be an interesting one to check out in terms of a sigmoid function

```
self.trial_count = self.trial_count + 1
self.epsilon = 1 - (1/(1+math.exp(-k*self.a*(self.trial_count-t0))))
```

- Where k determines how fast the agent performs the transition between random learning and choosing the max q-value. k also determines how fast the sigmoid function converges to 0.
- The t0 value can also be chosen empirically; by using the sigmoid function, we can make sure that the agent would have sufficient time to explore the environment completely randomly, in order also to fill the Q-value matrix with the correct values.

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.

Good intuition! As more training trials == more exploring == more learning == more confident agent. As more training trials with increased exploration is exactly what the agent needs, since the main thing in the training phase is to explore, learn and fill up the Q-Values!

> "Since i have used
> ..... function : at , where a = alpha, and t = trialno.
> the value needed for a was large . Because with increase in the no of trials the decaying factor was decreasing with good amount. So to maximize the no of training samples and to carry the decrease of decaying factor for a long time , large value of a was required. And since 0<a<1 , therefore i choose 0.95 at first then finally 0.9 for better performance."

Just note that the learning rate (alpha) should not be used in your epsilon decaying function. `self.a` (should be a new variable) is completely different than the learning rate (`alpha`). Remember that the learning rate is used in the actual Q-Learning and determines the how much the Q-table updates from one iteration to the next.

```
self.Q[state][action] = (1-self.alpha) * self.Q[state][action] + self.alpha*(reward)
```

What does a high learning rate of 0.9 represent?
(https://en.wikipedia.org/wiki/Q-learning#Learning_rate)
(http://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html)
(http://www.jmlr.org/papers/volume5/evendar03a/evendar03a.pdf)

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 100 or even 200. How does your agent perform then? Is it ready for the real world?

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. Student presents entries from the learned Q-table that demonstrate the optimal policy and sub- optimal policies. If either are missing, discussion is made as to why.

> "If the waypoint is in any direction and the light is red , then it should always select None i.e wait."

Remember that the agent can turn right on a red light if there is no traffic coming from the left going forward (i.e. if `inputs['left'] != 'forward'`). From the `environment.py` file.

```
# Agent wants to drive right:
elif action == 'right':
    if light != 'green' and inputs['left'] == 'forward': # Cross traffic
        violation = 3 # Accident
    else: # Valid move!
        heading = (-heading[1], heading[0])
```

> "If the waypoint is in any other direction than forward and the signal is green, then it should choose None to avoid accidents on the other lane."

Not really correct. The only instance when the light is green that the agent needs to account for is that the agent must to yield to oncoming traffic going `forward` or `right` when the agent wants to go left at a green light. Can check this out here.

> "4) In this example from the text file generated the smartcab is not following the optimat policy.
> ('left', 'green', 'forward', None, 'forward')-- forward : -0.20-- None : -4.74 -- right : 0.38-- left : -19.27
> Here signal is green and the way point is left, so cab should move left, but the cab is awarded negatively for left."

As mentioned above, the agent must to yield to oncoming traffic going `forward` or `right` when the agent wants to go left at a green light. You can actually still make the argument that this can be considered a 'sub-optimal' policy, but the `left` action is not the correct action at this state.

Optional Question 9

Therefore some ideas to think about

- Think about what type of environment the agent is a part of. Does the agent have a full view of the entire grid? Only a certain aspect of it?
- How about the destination itself? Does it change throughout the trials runs? Why would this matter?

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

Student FAQ