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

5 SPECIFICATIONS REQUIRE CHANGES

You have a good agent here and and impressed with your understanding of these reinforcement learning techniques(one of the best analysis I have read). Just have a few section to perfect, but should not be too difficult(and should help increase your understanding even more). We look forward to seeing your final submission. And keep up the hard work!

Based on your submitted comment: "*I don't understand what is meant by including visualizations. The visualizations are already present in the Notebook and seem sufficient. Do I need to provide additional visualizations?*"

- I think it is referring to if you add any additional visuals, but I have actually never seen anyone do this. Your submission is fine!

[OPTIONAL] Getting Started

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

Nice work examining the reward system. With your comment of " *Probably the code that allows it to move is currently missing or incomplete.* " As we have `action = None` with this agent so it doesn't move. One way to force the agent to move is with simulated annealing.

Student correctly addresses the questions posed about the *Train a Smartcab to Drive* code.

Love the thoroughness of your answers, greatly exceeds expectations here! 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.

Your agent does well in producing a valid output of None, 'forward', 'left', 'right'

A visualization is included that correctly captures the results of the 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.

"*I definitely hope not to find myself aboard such a smartcab!! For example in Las Vegas I found there are more than 27 million taxi drives per year. With 6% of major accidents this means 1.62 millions major accidents every year!.*"

Would agree here, definitely don't want to be in this cab. 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. Correct that this behavior is similiar to a random walk.

Note: Based on the question of -- How many of those bad decisions cause accidents? Should calculate the sum of major accidents and minor accidents(could enhance your analysis even more!)

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.

Impressive discussion for the omission of the deadline here. 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 note that including the deadline could possibly influence the agent in making illegal moves when the deadline is near.

However for this section make sure you also discuss why the agent does need the rest of the features here. Therefore what does the `waypoint` tell the agent? Why is this needed? Why does the agent need to keep track of left traffic? etc...

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.

Nice calculation here and glad that you have brought up " *It would be tempting to use domain knowledge to reduce the state space but in this case it would be cheating. The purpose of the exercise is to have the agent learn by himself the correct actions without domain specific knowledge.....*" You are correct that the goal of reinforcement learning is to learn these rules without them being hard-coded into the algorithm. We include these states directly and see that Q-matrix will converge as the agent will be able to learn these rules on its own when the algorithm has been implemented correctly.

Note: We can actually omit one of the inputs here based on US traffic laws, if you do this make sure you update the corresponding sections.

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

Your agent changes state while running, nice work!!

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.

Nice work here with your Q-Learning algorithm

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

Few minor issues here. In your `get_maxQ()` function and code of

```
maxQ = max(self.Q[state].iteritems(), key=itemgetter(1))[0]
```

You should be returning the maximum Q-value for the state, not necessarily the action associated for it. As your function right now return something like `right`, as we should be returning a number such as `0.837`.

Also in your `choose_action()` function and code of

```
action = self.get_maxQ(state)
```

your agent should be choosing a random action from a choice of actions that have the highest Q-value. For example, since all actions are initialized with a reward of zero, it's possible that all four actions are considered "optimal". Not having the agent choose a random action from this would imply that it always chooses, perhaps, the first available option. This is incorrect behavior:

```
STATE X:
-- 'forward' : 0.00
-- 'left'    : 0.00
-- 'right'   : -1.023
-- 'None'    : 0.00
```

The agent should choose one of 'forward', 'left', or 'None' with equal probability, since they are all considered optimal with the current learned policy. Using a list would be a good idea.

Lastly In your `learn()` and `createQ()` functions make sure your Q-Learning algorithm and Q-Table are only updated and initialized when Learning = True(it seems as you have deleted the comments, but they were)

```
# When learning, implement the value iteration update rule
```

```
# When learning, check if the 'state' is not in the Q-table
```

A visualization is included that correctly captures the results of the initial/default Q-Learning driving agent.

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.

"*If we have a look at the Q values in the txt file, we see that many cells in the Q matrix are zero and there is often a single nonzero value per state. This suggests that the sample size (number of trials) is too low to get a significant improvement. Since the number of trials is based on the decay function for epsilon, we need to use a function that decays in a slower manner, allowing for many more trials to be collected.*"

Awesome job checking out the log files. As only 20 training trials with 384 states is infeasible to learn a good policy. I bet with a slower decay rate and even more training trials the agent could actually encounter even more of these state, resulting in even more efficient max action selection and higher confidence. As this default Q-Learning driving agent could actually improve.

The reason this is marked as *Requires Changes* is that make sure you also address the question

ARE THERE ANY OBSERVATIONS THAT ARE SIMILAR BETWEEN THE BASIC DRIVING AGENT AND THE DEFAULT Q-LEARNING AGENT?

Improve the Q-Learning Driving Agent

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

Very cool dynamic decay epsilon value, as it is crazy how the number of bad action/accidents/violation seem to be directly correlated to your epsilon value in these graphs. It seems as you have a good decay rate. Could actually look into using a slower epsilon decay and more training trails, as I bet you could get A+ for both ratings here.

Also nice idea to try and tune the learning rate, another idea to look into would be to try out 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.

A visualization is included that correctly captures the results of the improved 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.

Can you please provide a more complete comparison of your optimized Q-Learning driving agent and the initial/default Q-Learning driving agent. How do they also compare in terms of safety and reliability ratings? How do they also compare in terms of number of bad actions, violations, etc... There should be discussion about each panel and how the agent improved compared to the previous iterations of the driving agent.

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?

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.

Nice job with your shown states here. I would recommend providing a bit more analysis for these. i.e. what do these q-values represent? What action did the agent take? Why? etc... But based on your analysis throughout your project, I can clearly see that you understand this.

The reason this is marked as *Requires Changes* is that for this section, please also discuss what an optimal policy for the driving agent would be for the given environment in general terms. Could look into discussing the US traffic laws.

Note(Nothing Needed): One thing to note here, with your state of

```
('forward', 'green', 'right', 'forward', 'left') -- forward : 0.83 -- right : 0.00 -- None : 0.00 -- left : 0.00 Correc
t. Goes forward (luckily...).
```

Could also notice that there are multiple dummy agents in this state. Therefore maybe look into increasing the number of dummy agents to learn more of the states in the beginning trials.

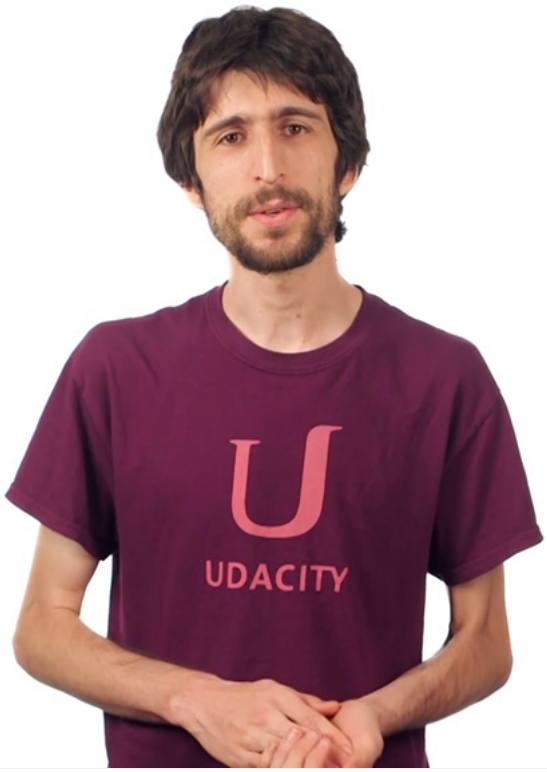
Student correctly identifies the two characteristics about the project that invalidate the use of future rewards in the Q-Learning implementation.

Awesome ideas and your comment of "*The goal of the SC is to reach the destination in time, but the SC never knows how far it is from the destination and there is no reward in reaching it. Furthermore, the destination changes at each trial, so we can't build a memory.*" Is spot on!

- First Characteristic - 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.
- Second Characteristic - As the destination itself moves after every trial. We could possibly (given enough trials), propagate reward away from every intersection, and hence would effectively not be making use of knowing where the goal is at all.

RESUBMIT

DOWNLOAD PROJECT



Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Rate this review