UDACITY

PROJECT

Train a Smartcab to Drive

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Great work!

[OPTIONAL] Getting Started

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

You can see how the software handles reward when the agent is idle by looking at the code here.

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

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.

Great job using `self.valid_actions`. Not using this could have the agent trying to make invalid moves.

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.

The key here is that it is not learning. Nice work.

Suggestion: It is more "pythonic" to use `random.choice(self.valid_actions)` (line 107).

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.

You are correct in noting that `deadline` is not necessary, and in fact, causes a drastic increase in the dimension of the state space.

Nice job identifying that US traffic laws make `right` unnecessary.

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.

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

Think about this:

The only danger from traffic to the left is cars going straight. Cars to the left turning left would be a danger if you are going straight, but in this case one of you will have a red light. Cars to the left turning right are never a danger.. With this knowledge, this is a sufficient input:

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

Now your state space dimension is 48.

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.

Pretty convoluted way to handle action selection. Shouldn't you just make a list of possible action that have the maximum Q (line 113)?

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.

Good analysis. If you have a look at your log file `sim_default-learning.txt` you can see that there are many states that your agent has not yet learned. This would show these states have not been visited by the agent. Your agent *is* learning, but it hasn't done enough exploration.

Improve the Q-Learning Driving Agent

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

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.

Nice work. It looks like your agent has done enough exploration to become reliable.

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

Nice job! The agent has to be extremely reliable. In a real-life situation, this is a zero-fault tolerance application. Even one mistake can result in deaths!

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 discussing optimal and sub-optimal policies. You can read more about learning rates for Q-learning here:

- http://www.jmlr.org/papers/volume5/evendar03a/evendar03a.pdf
- http://karpathy.github.io/2016/05/31/rl/
- https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0
- http://mnemstudio.org/path-finding-q-learning-tutorial.htm
- https://www-s.acm.illinois.edu/sigart/docs/QLearning.pdf
- http://www.umiacs.umd.edu/~hal/courses/ai/out/cs421-day10-qlearning.pdf

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

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ