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

Congratulations for meeting all specifications for this project! I added some comments and suggestions below, I hope they'll be useful. Keep up the good 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.

Comment

> he rate of reliability, although greater than I expected, makes sense as we have 4 moves so with a random decision we have a probability of 1/4 to make the right choise, so we could be making the right decisition to get to the goal in some cases as we see in the left-lower diagram.

Note, however, that reaching the destination implies getting several decisions right, since the agent must go through a path from its current position to the destination, one step at a time. On the other hand, the deadline is quite generous for this problem (see here), so it can get some decisions wrong and still reach the destination.

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.

Suggestion

As an exercise, I think it would be interesting to study two "extreme cases" for this problem:

- What if you were allowed to use only the **raw inputs**, without manipulating them in any form? In other words, your state would be a list of the original inputs provided to the agent (you could still decide to not add some inputs to the state, though). Would the agent be able to learn a feasible policy then? After how much time, and with which parameters?
- What if **no learning took place**? Which set of instructions should you pass the agent for it to never receive a penalty?

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

Suggestion

Another way to initialize `self.Q[state]` is using a dict comprehension:

```
if (self.learning) and (state not in self.Q):
    self.Q[state] = {action: 0.0 for action in self.valid_actions}
```

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.

Awesome

Excellent job implementing both the Q-update function (without future rewards) and the policy (selecting at random one the best actions given the Q-values for the state). Congrats!

Suggestion

Here's an alternative approach to picking the best action, based on list comprehensions. Given `maxQ` is the maximum Q-value for `state`, we can write:

```
best_actions = [action for action in self.valid_actions if self.Q[state][action] == maxQ]
action = random.choice(best_actions)
```

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.

Comment

Notice how, given the simplicity of the state you chose (i.e., given how small the state space is), 20 training trials is enough for the agent to learn a very good policy! Properly defining your state is fundamental in a reinforcement learning task. :)

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.

Comment

Given that we have well defined training and testing phases, the best strategy for the exploration rate decay may be to keep `self.epsilon == 1` for as long as you wish your agent to explore, and then drop `self.epsilon` to below the tolerance level for testing to begin. This way you guarantee that the agent is exploring as much as possible of the state space until the time comes for it to be tested.

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

Awesome

Congratulations for getting the maximum grade for both safety and reliability!

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.

Comment

> As far I could tell this could be resolved by changing the rewards. This case the agent prefers to stay as the reward for not passing the red light is greater than the reward for turning right in a red light when you can.

I don't think this is the case: the reward for following the waypoint is *usually* larger than the reward for staying idle when this action is allowed by the environment. But there is some amount of randomness in the rewards received by the agent, so it is possible that, in some occasions, the agent would indeed receive a larger reward for staying idle. Visiting the state more times should help getting to Q-values that better represent the true value of each action in the state.

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

Rate this review

Student FAQ