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

Congratulations for a great submission. You seem to have grasped both the concepts discussed in this project and how to use them. There are small fixes needed in your policy and Q-update function, but I have no doubt your next submission will meet all specifications. 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.

Comments

- The initial agent will stay put all the time, and it will receive (positive or negative) rewards purely as a function of how the environment changes.
- It's interesting to realize that this means the agent *could already be learning*, since it could acquire knowledge on the rewards it receives when it does nothing on different states of the environment.
    - However, such learning would be limited by the fact that the agent is always performing the same action (`None`, or staying put). It needs to *explore* the action space in order to know which is the best action for each situation.

- The plots for the basic (random) agent show no upward or downward trend: most values stay stable, apart from some fluctuation expected given the random nature of both the environment and the agent.
- This means that the agent will behave the same, no matter which trial or step it finds itself in.
- This makes sense since there is no learning going on: the agent *cannot* do anything apart from randomly picking an action, so it won't change its behavior. And the environment also behaves in the same way as the trials go by.
    - This second condition is important, and not necessarily true: the environment *could* change over time, perhaps even in response to behavior from the agent.
        - Think of an adversarial environment, such as a game: an opposite player can change the way it plays as it spots weaknesses and strenghts in the agent.
        - In this case, it's important for the agent to **keep learning** over all trials, to avoid getting stuck in a predictable pattern that its adversary can then exploit.

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.

Comment

- One way the `deadline` *could* be useful is by getting the agent to violate traffic rules **without causing any accidents** to get to its destination.
- For instance, if the `deadline` is approaching the agent could run a red light if no other cars were in the intersection.
- This would depend on the reward structure: does the reward the agent gets for reaching the destination compensates the penalty for commiting a violation?
    - Given the fact that we are not considering future rewards in our implementation of Q-learning, what would the answer be here?

- To avoid an increase in the state space size, `deadline` could be transformed into a boolean such as `deadline_approaching`, which would be `True` if the deadline were close enough that violating some traffic rules would start looking interesting.

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 dict comprehension:

```
if self.learning and state not in self.Q:
    self.Q[state] = {action: 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.

Required

- The `get_maxQ()` method should return the maximum Q-value for a given state, not an action associated to this value.
- Then, in the `choose_action()` method, one of the actions associated to the maximum Q-value for the state should be selected at random when the learning agent is in exploitation mode.

- In the Q-update equation, the updated Q-value should be a **weighted sum** between the new Q-value and the old Q-value.
  - The weights should sum up to 1 to avoid the Q-value from growing indefinitely.
- Given that in this setting `gamma == 0` (that is, the discount rate is zero), we don't need to worry about the discounted Q-value for the next state. In other words, the new Q-value is simply `reward`.
- The new Q-value should be multiplied by the learning rate `alpha`.
- Therefore, considering the weights should sum up to 1, which factor should multiply the *old* Q-value in the Q-update equation?

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

There are clear signs in the plots that your agent is learning:

- A downward trend in the frequency of bad actions
- An upward trend in both reward per action and rate of reliability

As an experiment, you can try setting up more training trials for the agent to see how it fares.

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.

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

Awesome

Congratulations for reaching 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

> The car should wait for this car to pass by not moving, and then checking whether it can go left on the next step. Instead, the learned policy is for the car to move forward.

One point to keep in mind is that, according to the environment, staying idle in an intersection when the light is green constitutes a minor violation (see here). This may not be a good representation of US traffic rules, but it means the best action for this state according to the Q-table makes sense. :)

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

Required

(This is an optional question; I'm only marking it as requiring changes because you will need to resubmit based on another rubric item.)

> Future rewards wouldn't work in this scenario because the smartcab does not have enough information to know whether a future state is closer or farther to the destination, it only tries to move towards the waypoint.

Excellent observations. Two more thing to keep in mind regarding this question:

- Do the agent and the destination begin each trial in the same place, or do they move around? Can the agent learn the best route by rote?
- Is there an "extra reward" for reaching the destination? In other words, is there anything to propagate back, other than the reward for obeying traffic rules and following the planner?

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

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

Student FAQ