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

Impressive adjustments here in your submission, you have great intuition in these reinforcement learning techniques. Wish you the best of luck in your future!

[OPTIONAL] Getting Started

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

Nice work examining the reward system. Could also mention how big or how negative the rewards would be.

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.

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

Another thing to think about is what type of environment do we have here?

- deterministic
- stochastic

Would this matter in relation to the q-learning algorithm and/or parameters?

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 have modeled the environment very well.

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

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 very nice discussion for the need of all these features. We could actually omit right traffic if desired, as this is not needed based on US traffic laws!

You are also correct that using the deadline in its full form is unreasonable. Maybe the only way we could actually use the deadline would be to encode it as a binary variable, say +10 time steps and -10 time steps. But this would still double the number of states. Something to think about when working with continuous values.

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 calculations here. Another idea to think about is how the amount and actions of the other dummy agents would affect the agents exploration. Would all these states be seen in a randomly uniformly distributed manner? How would the number of other agents on the road affect this? Think about if we only had a couple other agents? How often would we see instances with left and oncoming traffic?

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 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.

Great code adjustments here and great work with your Q-Learning equation

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

As this refers to the current reward only. For future reference you could also check out this version of the Q-Learning algorithm

```
 self.Q[state][action] = 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.

Nice ideas. Thus typically the behavior for the first few trials isn't any different from the randomly acting agent’s behavior. Thus once the agent starts to learn and build up a sufficient q-learning table and the agent tabulates rewards and updates Qvalues for each state action pair, the agent will start to hone in on the correct waypoint while more highly weighting the smartcab's best actions.

Improve the Q-Learning Driving Agent

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

Great decay rate, glad that you have used long exploration(high epsilon for an extended amount). Here might be another interesting epsilon decay rate to play around with

```
self.trial_count = self.trial_count + 1
self.epsilon = 1 - (1/(1+math.exp(-k*self.alpha*(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 was also 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.

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

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 addition here and throughly analyzing a sub-optimal state. With your comment of "*We see there are still several 0 values in the Q-matrix which means the potential for error exists and can be exposed increasing the number of test rounds.*" As I think you mean "increasing the number of **training** rounds" here, as the training phase is where the agent is learning. Which you are correct with, as we still see multiple 0.0 q-values. Thus probably more training trials would help the agent to encounter this state a bit more. Good intuition!

Another idea to think about in terms of an optimal policy

- When the agent is blocked by a red light or traffic, the best course of action is always to wait until they can follow the waypoint. However, it is often the case that they could be taking alternate routes that decrease the distance to the goal (proceeding along the step-wise diagonal path to the goal). A vector waypoint system could greatly aid navigation by rewarding paths that decrease the 2D distance to the goal instead of just rewarding following a single waypoint. In general, it may also be the case that patterns in traffic signals can be learned, but not in this environment.

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

Think you have captured these quite well. Can also note that gamma propagates reward from each future state that the agent could travel to. Since the smartcab has only an egocentric view of the world, we find that the propagation of reward comes in the form of simply propagating the Q-value of the best action that would be available at the given intersection/state. Hence, even a not-so-optimal state would receive additional reward from the backpropagation simply because each state has at least one optimal, positively-valued state-action pair. In effect, gamma can make the driving agent learn sub-optimal policies because it is incorrectly attributing positive future rewards to “better” states (which is untrue since we have no idea where the goal is).

DOWNLOAD PROJECT