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

1 SPECIFICATION REQUIRES CHANGES

Very nice job here with your analysis and code implementation. Just have one last section to provide a bit more detail for and you will be good to go. Keep up the great work!!

Implement a basic driving agent

✓

Student is able to implement the desired interface to the agent that accepts specified inputs.

✓

The driving agent produces a valid output (one of None, 'forward', 'left', 'right') in response to the inputs.

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

✓

The driving agent runs in the simulator without errors. Rewards and penalties do not matter - it's okay for the agent to make mistakes.

"*There aren't many observations to notice, except for the fact that the smartcab drives in circles.*"

You could also mention that the agent is not following the rules of the road and pilling up penalties.

Identify and update state

✓

Student has identified states that model the driving agent and environment, along with a sound justification.

Good justification for the states that you have decided to represent, as we definitely need to include light, oncoming, left, and next waypoint. Also nice addition with the comment for the omission of the deadline with "*I also decided not to include the deadline as a state, because it can take so many values, that it will make a space state gigantic.*" 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.

Note: Nice catch with "*We are using here the US driving laws, which means that traffic to the right don't affect our behavior on a light-regulated intersection.*" As we don't need this.

✓

The driving agent updates its state when running, based on current input. The exact state does not matter, and need not be correlated with inputs, but it should change during a run.

Your agent changes state while running, nice work

Implement Q-Learning

✓

The driving agent updates a table/mapping of Q-values correctly, implementing the Q-Learning algorithm.

You have correctly implemented the Bellman equation

```
self.Q[self.state][action] = (1 - learningRate) * currentQ + learningRate
 * reward + self.gamma * newAction[1]
```

✓

Given the current set of Q-values for a state, it picks the best available action.

Great work in your `chooseAction` function. And nice idea to implement an e-greedy approach with epsilon. As a more advanced version of this would be epsilon decay. As we can reduce the chances of random exploration over time, as we can get the best of both exploration vs exploitation. As this typically works very well. As we typically see `1 / t` for the decay factor.

✓

Student has reported the changes in behavior observed, and provided a reasonable explanation for them.

You are correct here. I would also recommend describing some of the actually behavior of the agent as well. For instance the behavior for the first few trials does not appear significantly different from the randomly - acting agent's behavior. As it tabulates rewards and updates Qvalues for each state action pair, correct actions become more highly weighted and the smartcab chooses these actions more often.

Note: We can now use the statistics of "*achieves 80%+ success rate*" as a good benchmark for the Enhance the driving agent section

Enhance the driving agent

✓

The driving agent is able to consistently reach the destination within allotted time, with net reward remaining positive.

When I run your code, in the last 10 trails, your agent is able to reach the destination in all 10 of them. Nice implementation!!

Specific improvements made by the student beyond the basic Q-Learning implementation have been reported, including at least one parameter that was tuned along with the values tested. The corresponding results for each value are also reported.

Love the chart of all of the different parameter values tested and your comment of "*100 experiments for parameters Alpha 0.8, Gamma 0.1, Epsilon 0.005 it consistently performs with 99% success rate: 12.3 moves on average, 22.9 rewards average, 0.4 penalties average*" is great justification for your final parameters. However in your code I see that you have implemented alpha decay

```
learningRate = 1.0 / (t + 1) ** self.alpha
```

(which is a great idea). Therefore please also provide some discussion of this and why is it used.

✓

A description is provided of what an ideal or optimal policy would be. The performance of the final driving agent is discussed and compared to how close it is to learning the stated optimal policy.

Nice discussion of what an optimal policy would be and your agents performance. As checking out the q-learning table is great justification for "*Penalties received are probably because of random actions, as the car didn't have knowledge for what could be the best action*". Thus if you were to remove random actions in the later trials this may be even more optimal(as epsilon decay would be very cool).

Note: With the comment of "*First of all, looking at the value we clearly see that we haven't explored all action/value pairs, so 100 trials are not enough for convergence.*" As with more training time this maybe be much improved. You could also look into increasing the number of cars on the road.

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

☆☆☆☆☆

Student FAQ