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 adjustment in this report. As alpha decay is a great thing to have in your toolbox. Wish you the best of luck in your future!!

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.

✓

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

Nice work here with your chart, as 100 trials is definitely a sufficient number of trials.

Identify and update state

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

Solid justification for your represented states. Would also recommend discussing the total number of possible states we would have with this implementation.

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.

The driving agent updates its state when running in the pygame window

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. For future reference you could also check out using the basic Q-Learning

$$Q_{t+1}(s_t, a_t) \leftarrow \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right)$$

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

Very clean way of implementing your argmax

```
bestAction = max(self.Q[self.state].iteritems(),
key=operator.itemgetter(1))[0]
```

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

Enhance the driving agent

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

The driving agent is able to consistently reach the destination

✓

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.

Again love the chart with all of the different parameter values tested, as I am glad that you have also checked out the penalties as well. Nice adjustments with your discussion of your alpha decay. You can check out more learning rate ideas here

- alpha ideas

Another tunable parameter would be the initial q-values. As this refer to “optimism in the face of uncertainty”. The idea is that high initial Q-values (implying an “optimistic” agent in the sense that it initially believes all possible actions will yield excellent rewards) lead to an exploratory-leaning agent, because it will delay exploiting familiar paths, since those will end up with lower Q-values than its initial estimate.

Based on your submitted comment and analysis comment of: "*According to the lecture, it was said that it is always a great idea to decay Alpha so we can avoid local maxima/minima, which means that our car should be more confident with what it’s learning, while should be less convinced by the information as it prevents the car from changing what it knows already. With the decrease of learning rate our car becomes more convinced that it’s found an optimal solution.*"

As this is a great think to know, as you are exactly correct. Just also consider the environment we are working with. As

- In fully deterministic environment(as we have here), a learning rate of 1 is actually considered optimal.
- When the problem is stochastic, the algorithm still converges under some technical conditions on the learning rate, that require it to decrease to zero. In practice, often a constant learning rate is used, such as 0.1 for all.

✓

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.

Would recommend cutting out random exploration at a certain point. As we could use the first 90 trials as the training data(use exploration) and the last 10 as the testing(no exploration) data, and evaluate the testing set for your optimal policy.

Another idea to think about how to implement:

- For instance, we could imagine that the final destination is one block forward and one block right of us, and we are at a red light. If the waypoint is “forward”, the agent will wait, but it might easily be faster to turn right (and incur the small penalty) and then hope to get a left.

DOWNLOAD PROJECT

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Student FAQ