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

5 SPECIFICATIONS REQUIRE CHANGES

This is probably the single best report I have had the pleasure to read and absolutely love the videos, plots and ideas. Should be proud to show this one off! You have a good agent here and very impressed with your understanding of these reinforcement learning techniques. Just have some minor issues, but should not be too difficult to update(and should help increase your understanding even more). We look forward to seeing your next submission. And keep up the hard work!

[OPTIONAL] Getting Started

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

Love how you have throughly analyzed the environment here, definitely needed and you are the first person I have seen put a video in the notebook, very cool. However in terms of the question of *How does the light changing color affect the rewards?* There are three instances that should be mentioned with this stationary agent.

- What are the rewards when there is a green light with other traffic?
- What are the rewards when there is a green light without other traffic?
- What are the rewards when there is a red light?

Student correctly addresses the questions posed about the *Train a Smartcab to Drive* code.

"*num_dummies : Discrete number of dummy agents*
Changes the number of dummy agents in smartcab environment to 10"

As the default number of dummy agents is 100, but you can definitely change this to 10 if desired.

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.

Your agent produces a valid action while driving.

A visualization is included that correctly captures the results of the basic driving agent.

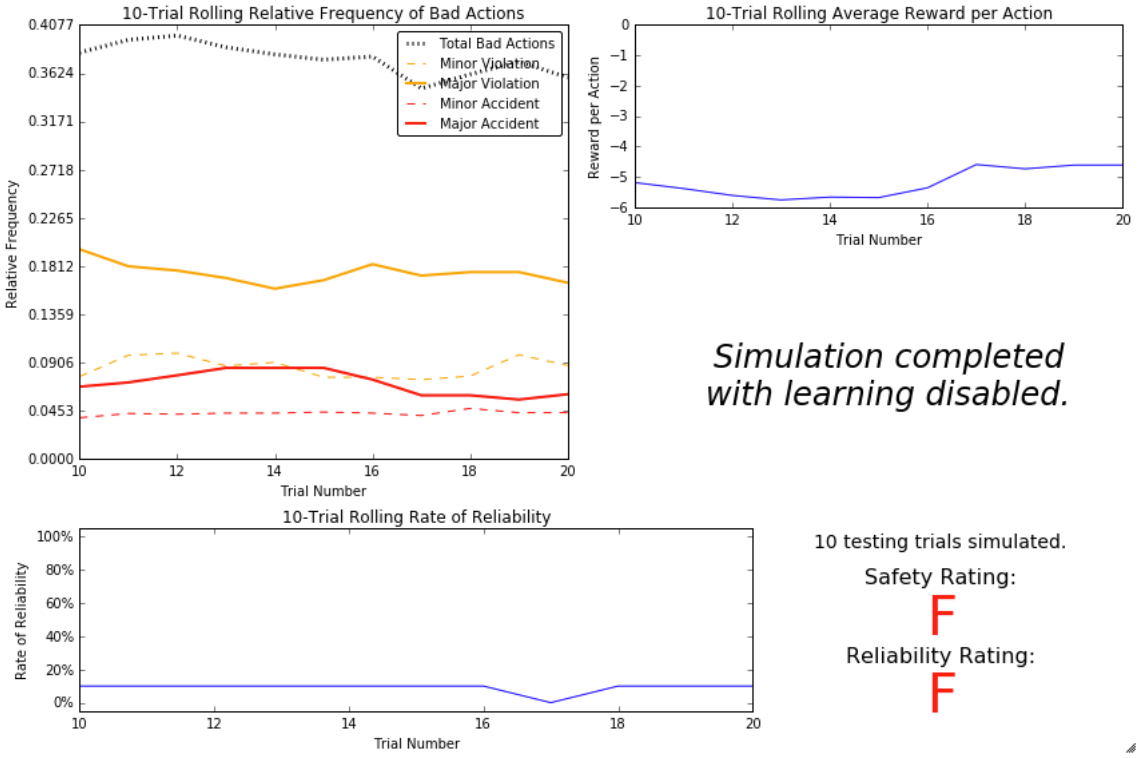
It seems as your visual is not coming through here.

```
# Load the 'sim_no-learning' log file from the initial simulation results
vs.plot_trials('sim_no-learning.csv')
```

As you need to run your basic driving agent with learning = False with the flags of

- 'enforce_deadline' - Set this to True to force the driving agent to capture whether it reaches the destination in time.
- 'update_delay' - Set this to a small value (such as 0.01) to reduce the time between steps in each trial.
- 'log_metrics' - Set this to True to log the simluation results as a .csv file in /logs/.
- 'n_test' - Set this to '10' to perform 10 testing trials.

To populate this log file. Should look similar to this



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

Love the analysis here and love the visual support, we don't see too much of this, impressive! However you may need to update this analysis when you actually have a visual shown in this report, so this will be re-evaluated.

Note: With your comment of "*Over the initial 30 trials,*" As we should actually only see 20 training trails in this basic driving agent.

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 in your **report**. 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 nice discussion for the need of all these features and good idea to omit right traffic, as this is not needed based on US traffic laws.

And awesome justification for the omission of the deadline. 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.

The reason this is marked as *Requires Changes* is that in your code I see that you have used

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

Without the `inputs['oncoming']` feature, which is clearly needed. Therefore make sure your code matches your analysis.

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 calculation! As a total of 96 total states really isn't too many to learn with a feasible number of training trials and a good epsilon decay rate. Maybe another idea to confirm this would be to run a Monte Carlo simulation(considering that all these state are uniformly randomly seen). Would 750 be enough? How many training trials could represent 750 steps? What about every state, action pair? Try changing the step

```
from sets import Set
from random import choice

def chance_of_visiting_all_states(iterations, k, n=24):
    r = range(n)
    total = 0
    for i in range(iterations):
        s = Set()
        for j in range(k):
            s.add(choice(r))
            if len(s) == n:
                total +=1
                break
    return float(total)/iterations

steps = 750
print "Chance of visiting all states in {st} steps: {ch}".format(st = steps, ch = chance_of_visiting_all_states(2000, steps,
 96))
```

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.

Very clean code and nice with your list comprehension while randomly selecting an action when we have multiple with the same q-value. I wish I could add more insight, but this is essentially exactly how I would code this up!

And great work with your Bellman equation, as this refers to the current reward only

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

For future reference, could also check out this version of the Q-Learning algorithm

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

A visualization is included that correctly captures the results of the initial/default Q-Learning driving agent.

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.

Love the analysis! As the agent is getting a bit better here, as it is actually starting to learn as the trials pass by. As this is definitely expected with a decaying epsilon. Therefore we can reduce the chances of random exploration over time, as we can get the best of both exploration vs exploitation. Good!

However the reason this is marked as *Requires Changes* is that with your comment "*The Q-Learning agent received a A+ safety rating and a F reliability rating*". As I actually don't see an A+ safety rating and a F reliability rating in your visual here, but I see an F and a D. But simple adjustment.

Improve the Q-Learning Driving Agent

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

Love the gridSearch code and visual, as we don't see this too often. As more training trials with increased exploration is exactly what the agent needs, since the main thing in the training phase is to explore, learn and fill up the Q-Values! As it is crazy how the number of bad actions / accidents / violations seem to be directly correlated to your epsilon value in these graphs. This would be an interesting one to check out(reversed sigmoid)

```
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.

Another idea to play around with in your gridSearch code would be alpha decay and reduce the learning rate over time. Thus as time elapses, the agent becomes more confident with what it's learned in addition to being less persuaded by the information.

A visualization is included that correctly captures the results of the improved 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.

Such an impressive comparison to the other agents here and really glad that you have included some additional metric here in your report. As it seems as more training trials with more exploration allowed the agent more time for learning resulting in a more confident agent!

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

Congrats on your ratings! One thing to look into is to determine how reliable your agent actually is. Try changing the number of testing trials to something like 50 or even 100. How does your agent perform then? Is it ready for the real world?

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.

Love the analysis and code snippets for a few *optimal* states here. Therefore lastly for this section, please also try to discover at least one state where the policy is different than what would be expected from an optimal policy(same idea as above with an incorrect result)? This would be necessary to discover where the agent/algorithm is still having a bit of trouble and maybe we can find a way to improve the efficiency of it. If you cannot find one, please mention this(but look hard as we will as well)

Note: I am a bit confused with the state spaces, you have decided to use throughout your report. As in Question 4 you use waypoint, light, left, and oncoming. Then in this section you only use waypoint, light, oncoming. And you `agent.py` file has waypoint, light, left. As these should all be consistent. Or please mention this.

RESUBMIT

DOWNLOAD PROJECT