UDACITY

PROJECT

# Train a Smartcab to Drive

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW 2

NOTES

SHARE YOUR ACCOMPLISHMENT!

## Meets Specifications

Excellent work on this project! I'm impressed with your understanding of the concepts used to train the smartcab. Congratulations on meeting specifications!

### Getting Started

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

You do a nice job of succinctly summarizing the framework in which rewards are awarded, and accurately explained the different parameters and functions within agent.py, environment.py, simulator.py, and planner.py.

**Student correctly addresses the questions posed about the code as addressed in Question 2 of the notebook.**

### 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.**

Excellent--your driving agent produces a valid action when an action is required.

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

Your random driving agent performed as expected -- the reported frequencies of bad actions (~40%) and low (~20%) rate of reliability is typical of a random action performance. Well done! Your analysis of the behaviors of your basic driving agent is also thoughtful and detailed.

> The trend does small peaks and troughs, but overall doesn't change significantly. The changes are not big enough to result in a grade change. For instance, the reliability is constant except for an intermittent peak of about 10% for a couple of trials in between.

This is absolutely the case--since the agent is not set to learn, there is no trend and consistencies we see in the data are coincidental. At this point, increasing the number of training trials will not improve this metric.

If you are curious about how rewards are allocated, you can see the breakdown of it in environment.py:

```
# Agent attempted invalid move
        else:
```

```
            if violation == 1: # Minor violation
                reward += -5
            elif violation == 2: # Major violation
                reward += -10
            elif violation == 3: # Minor accident
                reward += -20
            elif violation == 4: # Major accident
                reward += -40
```

## 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. Students argument in notebook (Q4) must match state in agent.py code.**

Nice analysis and justification of the set of features you've chosen to include in your default-QLearner! You are correct that `deadline` is not as important of a feature to include since including this will drastically increase in the dimension of the state space. Also, including `deadline` could influence the agent to make illegal moves to meet the deadline.

**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.**

Your state size calculation is correct!

**The driving agent successfully updates its state based on the state definition and input provided.**

Running your code, your driving agent is changing states and taking actions in response to each step of inputs. The driving agent is functional and is able to move around the grid.

## Implement a Q-Learning Driving Agent

**The driving agent: (1) Chooses best available action from the set of Q-values for a given state. (2) Implements a 'tie-breaker' between best actions correctly (3)Updates a mapping of Q-values for a given state correctly while considering the learning rate and the reward or penalty received. (4) Implements exploration with epsilon probability (5) implements are required 'learning' flags correctly**

Your code is able to iterate over the set of Q-values in a state and select the maximum value in order to implement the best action. Good work!

Also, setting your states into tuples is a good way to set them as immutable values so that the parameters won't be accidentally changed.

**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.**

Your default Q-Learner looks like it's learning a policy! We begin to see trends forming here of decreasing frequency of bad actions and increasing rewards, which indicate that the agent is learning. The graphs also show a correctly implemented linear epsilon decay function.

> Overall, the smartcab is still performing poorly. However, there has been clear improvements in performance (with all bad actions dropping) and reliability and evidence that the model has been learning. This suggests that optimization and increasing trials could yield a stronger, better rated model.

Your observation is exactly on point. We see some improvement shown when compared with the results of the random agent. However, at this point the agent has not yet explored enough states to break the 0 pt barrier in average reward per action. I would recommend generating summary statistics for each run as you begin optimizing your agent's performance -- a running count of the trial count, success rate, and net reward/penalties could be useful in zoning in on where your q-learner is getting stuck in local optima or how your errors are distributed.

## Improve the Q-Learning Driving Agent

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

**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.**

Nice job in getting this cab up to A+/A ratings!

> An epsilon tolerance of 0.001 was used, and an alpha of 0.002. The very low epsilon tolerance and alpha were used to allow the model sufficient trials to train with, and caused the 2304 of trials. A large number of trials is important to increase the safety rating to a maximum.

Excellent choice! I appreciate that you've explained your thought process here in detail. When adjusting your alpha rate, an alternative to choosing one constant rate is to implement a decay function, which will allow you to begin with a high learning rate and decrease it over the course of training trials. Beginning with a higher learning rate will attribute more weight to new results, and steadily lowering it to a relatively small number will allow your cab to gradually switch to exploitation of the Q-table and policies.

Your analysis is excellent--great job!

Below are some links for further reading on the exploration-exploitation trade-off, if you're curious:
http://papers.nips.cc/paper/1944-convergence-of-optimistic-and-incremental-q-learning.pdf
https://articles.wearepop.com/secret-formula-for-self-learning-computers
http://ftp.bstu.by/ai/To-dom/My_research/Papers-2.1-done/RL/0/FinalReport.pdf
http://www.jmlr.org/papers/volume5/evendar03a/evendar03a.pdf

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

Congratulations on achieving A+/A ratings!

**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. Student presents entries from the learned Q-table that demonstrate the optimal policy and sub- optimal policies. If either are missing, discussion is made as to why.**

Nice job on including a very clear, specific optimal policy. You've also included a good discussion of the examples you've extracted from the log.

> A study of some more of the examples suggests that the Q-Learner has more or less (as we do see some suboptimal choices) learned to take the right actions given the four traffic patterns in a cross section, and somewhat chooses to move, rather than wait till it's safe to follow the waypoint. It also suggests a priority of safety over reliability, which coincides with the scoring method and the overall performance of A for reliability, and A+ for safety.

Your analysis is very detailed and perceptive. It is also worth noting that, as the time draws nearer to the deadline, the agent is more heavily penalized for taking bad actions. In this situation, it might very well be that moving in any direction that does not result in a violation is the preferable action.

Additionally, if you examine some of the entries with sub-optimal policies, we can see that the agent learns a suboptimal policy because it first explored an action which is sub-optimal, but did yield positive rewards. Consequently, the agent might repeatedly exploit this action until exploration occurs again. Although it may later randomly explore the optimal policy, the suboptimal policy will have acquired a higher value in the q-table.

In any case, identifying instances where the agent is implementing a sub-optimal policy can allow us to discover the places in which the agent/algorithm is still having a bit of trouble and find a way to improve its efficiency.

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

RETURN TO PATH

Student FAQ