UDACITY

Back to Machine Learning Engineer Nanodegree

Train a Smartcab to Drive

REVIEW

CODE REVIEW 1

HISTORY

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.

Good observations of the simulation environment in PyGame.

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

You do a nice job of succinctly summarizing the different parameters and functions within agent.py, environment.py, simulator.py, and planner.py.

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.

> As the number of trials increases, the outcome of results didn't change at all because the agent is not learning.
>
> According to the testing result from the 10 testing trials, both reliability and safety get F. so the random agent is not reliable neither safe.

This is absolutely the case. Since the agent is not set to learn, it is performing poorly and any consistencies we see in the data are coincidental. At this point, increasing the number of training trials will not improve these metrics.

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 agent! While `deadline` is important for the agent to improve its efficiency, it must be dropped from the model as including this feature 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.

Additionally, very nice observation that the right state can be dropped. Omitting this feature will help reduce the feature space and make your agent faster!

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! You can also complete a Monte Carlo simulation to calculate the general number of steps needed in order to visit the different possible states. You can also read about it in more detail here.

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

The driving agent changes state while running the program -- nicely implemented.

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!

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 agent 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 and the default number of 10 testing trials.

> As the number of training trials increased, the number of all kind of bad actions decreased and the average reward increased.
>
> Safety and reliability have been improved but not sufficient.

Your observation is correct. 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 build a stable Q-table. 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.

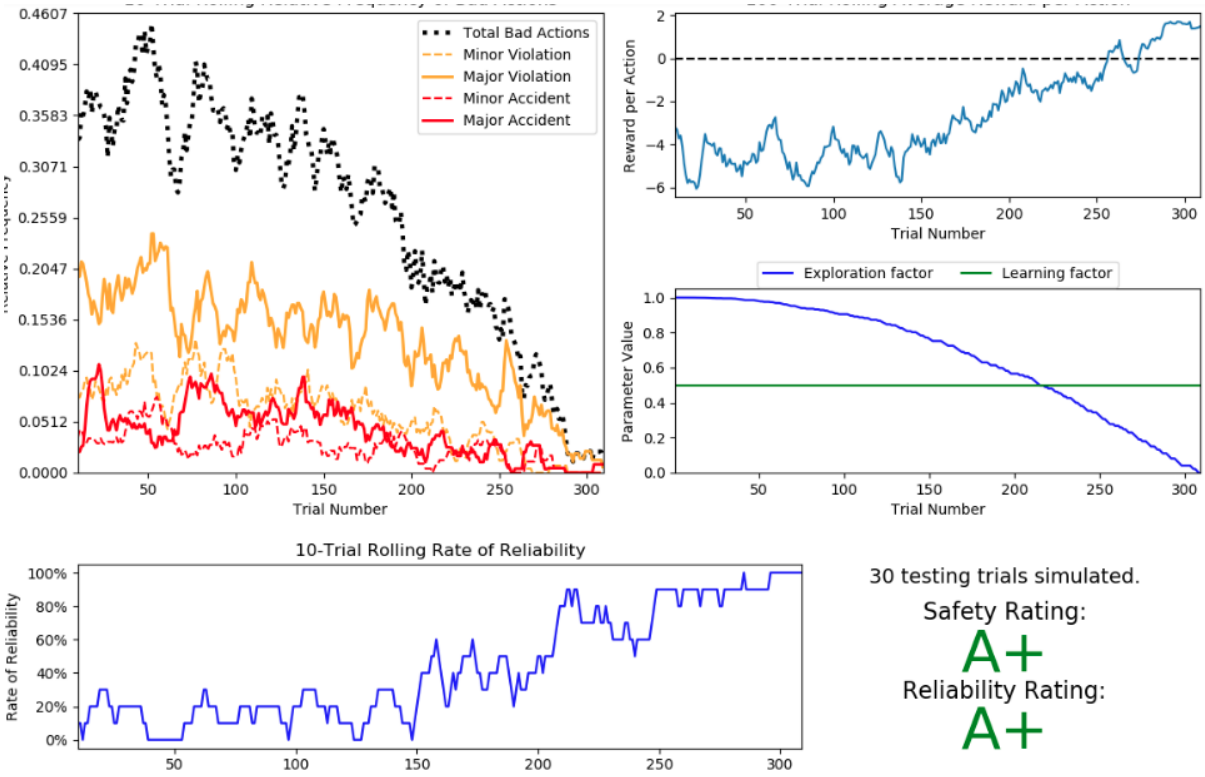
Improve the Q-Learning Driving Agent

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.

Your results are impressive here! Achieving A+ ratings is no small feat. To further improve the confidence of your model, you may want to consider increasing the number of testing trials to 100 or more trials. When your agent consistently achieves A ratings with a high number of testing trials, this means your model is robust and your agent's performance will likely continue to yield similar results in future tests.

> Solution 2: gamma = 0, alpha=0.5, only 100-200 trials:



Good to see you were able to run a solid number of training trials for your agent. There are actually two ways to increase the number of training trials: by adjusting the epsilon decay function and by lowering the epsilon-tolerance. These two methods focus on different parts of the agent's training. When epsilon is decayed slowly, the agent is able to repeatedly explore most of the state-action pairs for all the states because epsilon is kept at a higher value for a long duration of time. The more slowly epsilon decays, the more the agent is able to explore and refine its policies in the Q-table. This is quite different from lowering the tolerance level to increase the number of trials.

When the tolerance level is lowered, the speed of the epsilon decay doesn't necessarily change. That is, epsilon can still decrease to a low value over the same number of trials. However, since this value has not hit the lowered tolerance level, the agent still remains in the training phase. The agent will not explore additional states because of the low epsilon value. It will repeatedly exploit the policies in the Q-table until epsilon finally hits the tolerance level. Often, the tricky part of tuning the model is deciding on the balance between exploration and exploitation to achieve optimal results.

I appreciate that you've experimented with an alpha decay function and tested its impact on the agent's performance. I would also suggest quickly checking the log entries for the number of optimal/suboptimal policies for your final model, even though the ratings are both A+. Tweaks that may not yield in a change in

the ratings may remedy some of the states with different types of suboptimal policies (please see comments for Question 8 for further explanations).

Finally, another aspect that you can consider for further parameter tuning is to pair an alpha decay function with a cosine function for your epsilon decay. This might allow your agent to further fine-tune its performance.

Below are some links for further reading (in particular, the last paper explores the relationship between the alpha used and convergence rates):

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.

Excellent work on achieving 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 discussion and good job exploring your q-table. We see that a good agent generally learns the optimal policy, although there may be exceptions. Often these exceptions are due to the fact that the agent simply has not explored enough. Even with thousands of trials due to the random nature of state distribution, the agent may not have explored every state successfully.

In your suboptimal policy, we may see that the optimal action was never explored. This is why it's important to run enough trials in order to let the agent explore all the different actions. Longer exploration would remedy situations where a suboptimal policy of this type is learned.

Other times, the agent learns a suboptimal policy because it first explores an action which is suboptimal but does yield positive rewards, and then repeatedly exploits that action. Later it may randomly explore the optimal policy, but at that point, the suboptimal policy will have a higher value in the Q-table.

For example, this might be "going forward at a green light" instead of following the waypoint at a green light. We will get some reward for simply moving on green, regardless of the waypoint, but it's not optimal. However, it will be regularly exploited until exploration occurs again. During the exploitation period, it will build up a significant lead on the optimal policy. With enough training trials, the agent will eventually correct its policy and learn the optimal policy.

However, we also see that our agent has learned the policies well enough, to meet specifications (A ratings) for this project. This shows that while our agent may not learn the entire optimal policy, it might learn enough to be adequate for a given job.

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

You've made some good points and are right about the one for the environment. Since the environment is stochastic (the start and end points are randomly placed for every trial), it is difficult for the agent to perform rote learning since the goal changes with every trial.

For the one about the agent, you were very close. The reason why the agent cannot use future rewards is because it cannot see beyond the current intersection, so it is unable to evaluate different possible routes and calculate rewards.

DOWNLOAD PROJECT

1 CODE REVIEW COMMENTS ❯

RETURN TO PATH

Rate this review