Tasks

Implement a Basic Driving Agent

Question: Observe what you see with the agent's behaviour as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

Answer:

- The cab drives randomly
- Its actions may or may not be in sync with the planner
- It receives positive/negative rewards, its state is empty
- The cab may eventually make it to the destination unless hard time limit is reached first

Inform the Driving Agent

QUESTION: What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?

Answer:

There are numerous states that are required for learning because each state includes a combination of the following variables from the environment:

- Waypoint is required for the smartcab to get to its destination. Without the waypoints the cab will wonder randomly instead
- Light is required to learn traffic rules to prevent accidents. In real-life situation, if a car gets into an accident, the goal may become irrelevant as there is no car to drive anymore. In smartcab simulation, the agent receives negative rewards which would enable the agent to choose other, more favourable, actions in the future.
- In some situations the information from traffic light is not sufficient to avoid accidents. The awareness of other nearby cars and their travel direction becomes important. For example, the cab should not make actions with the following light/car/action states:
    - Oncoming car, moving any direction, green light, left action
-
    - Left car and/or right car, red light

OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

Answer:

In total, there are 384 possible states (3 * 2 * 4^3), because we have

- 3 waypoint states (forward, left, right)
- 2 light states (green, red)
- 4 possible directions of each nearby car (none, forward, left, right) with 3 possible positions of nearby cars (left, right, oncoming)

Since there is a chance for each of those 384 states to occur and we want the smartcab to learn the traffic rules so as to avoid accidents but also to reach its destination, the number of states is reasonable.

Implement a Q-Learning Driving Agent

QUESTION: What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

Answer:

With the changes implemented, the smartcab can now reach the destination in more predictable number of steps, and with fewer traffic violations. The reason is that the cab accumulates the knowledge of traffic rules via rewards that the environment assigns to it. The more the cab conforms to traffic rules and follows travel directions, the more it is rewarded. On the other hand, if the cab violates the rules or gets into accidents, it is penalized.
The weight of each action outcome/reward is recorded into a model. The next time the cab decides to act, it selects an action given the state that it is in based on a action-selection policy. Most of the time, the best action is selected, which indicates that if it is taken, the outcome will be most favourable, or more rewarded.

Improve the Q-Learning Driving Agent

QUESTION: Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

Answer:

In order to monitor performance of the agent, I added a few metrics to be output while the agent is running. An additional script (stats.py), parses the results and creates graphs. The metrics include:

- Steps. The number of steps per trial that was used to reach the destination. This variable is reset to 0 every trial.
- Total steps. The number represents total steps for 100 trials
- Epsilon. The epsilon value used for the duration of the experiment (100 trials). When decaying epsilon is used, this displays the effective value correlated with the total steps
- Alpha. This shows the effective alpha value correlated with the total steps. The variable is used to get the feel as to how the agent reacts to different alpha values in terms of steps per trial and rewards
- Rewards. The obtain the trend in the number of rewards across all trials

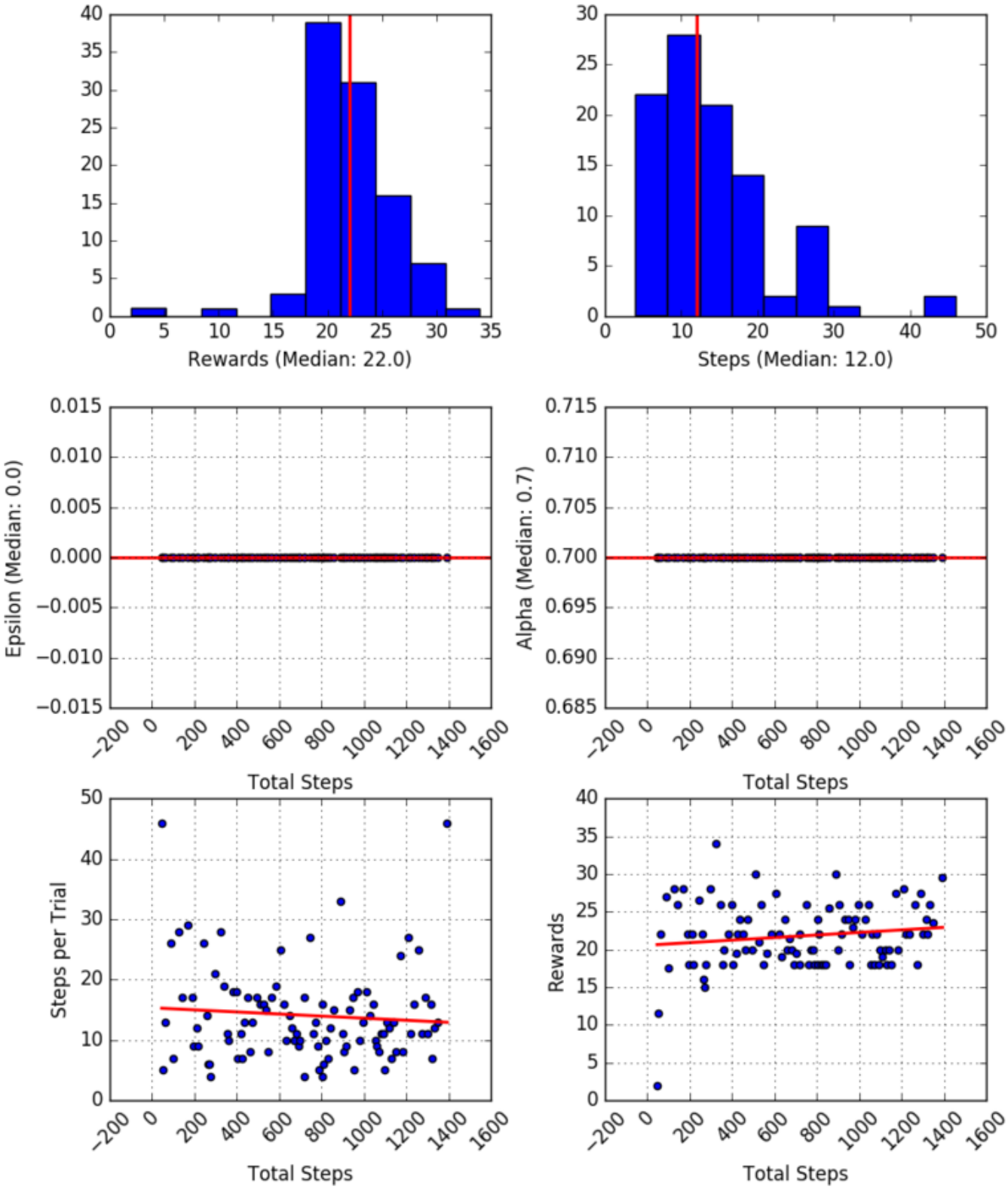
The stats script also displays the number of trials completed successfully (agent reached the destination within window), and the number of timed out trials (failed to reach destination within time window). To use:

- Pipe the output from the agent to a text file: *python agent.py > output.txt*

- Display stats: *cat output.txt* | *python stats.py*

Example output:

Reached: 98, timed out: 2



The biggest impact on agent's performance has a non-zero alpha value. In the above graphs a value of 0.7 was used. The results show that during 100 trials:

- The steps per trial has a downward tendency, which suggests that it takes the agent fewer steps to reach its destination
- The number of rewards has a slight upward tendency. This would mean that the agent makes better decisions as time progresses
- On average it takes 12 steps to reach the destination, and earn 22 rewards in the process

In this experiment, epsilon value is 0. The algorithm in choosing the action (see agent.py) uses optimistic long-term reward of 0.1 for unseen state/actions.

Changing epsilon to decaying, or a small constant value (e.g., 0.10) does not have much effect on agent's performance. Changing the gamma value also doesn't have much effect. Variation in alpha values (from 0.1 to 0.9) produce similar results with non-significant differences.

With the current configuration, the agent consistently reaches destination within allotted time window, which is over 95% in 100 trials.

QUESTION: Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties? How would you describe an optimal policy for this problem?

Answer:

The graphs above suggest that the agent is able to reach its destination in fewer steps with positive reward value over the experiment duration. Since the trend is not as pronounced towards the end of the test, the policy, given the current parameters and number of trials, reached overall good state but not optimal. An optimal policy is the one that enables the agent to reach its destination in fewer steps with most positive rewards.

To reach the optimal state, either one or more modifications should be made:

- Increase the number of trials to a large number so that the agent explores all 384 states at least a few times each so as to learn most-rewarding action. The drawback here is that without any other input into the agent, the learning process may last indefinitely
- Create a more controlled environment to enable the agent to visit each state in a deterministic number of trials
- Modify action selection algorithm in the agent to explore all states for each action in pre-determinate manner. Since this environment is stationary, the algorithm should be reverted to exploiting behaviour after the learning stage is complete