Train a SmartCab to Drive

Paul Heraty

Implement a Basic Driving Agent

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

To begin with, I just set the action to be random as follows:

```
action = random.choice(Environment.valid_actions)
```

Now the red car moves randomly around the grid, not obeying any lights or worryinig about on-coming cars. For example, you can see from the log snippet below that lights are ignored, where it decides to move forward even though the lights are red.

```
LearningAgent.update(): deadline = 30, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = -1.0
```

It also is not specifically moving in the direction of the destination. I ran the program multiple times, and once it did actually make it to the finish:

```
Environment.act(): Primary agent has reached destination!
LearningAgent.update(): deadline = 12, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 12.0
```

But this is a pure coincidence, as there is no intelligence in the LearningAgent yet that would cause it to move torward the destination. In all cases, the car just drives around randomly, and only gets to the destination by pure chance.

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?*

I believe that the appropriate states to model for the smartcab are the env inputs and also the next_waypoint coming from the planner.

The reason I chose these is that between them, the car can determine a) where it should go next and b) whether it is safe to go in the desired direction.

In fact, when I examine the rules more, it shows that I never actuaully need to use the inputs 'right' value to determine any of the valid actions for the smartcab. So I can drop that.

I did not choose the deadline state, as it does not add much value to the state space. In fact, it could have the adverse affect of causing the agent to break traffic rules in order to meet the time deadline, and this would not be an acceptable action. So for that reason, I am not including the deadline state.

Therefore, the final states that I am selecting are

1. next_waypoint

2. inputs 'light', 'oncoming' and 'left'

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?*

Using the env inputs and the planner next_waypoint, the total number of states for this environment is 96. I get this by multipling the following:

3 : number of possible next_waypoint values ('right', 'forward', 'left')

2 : number of possible input 'light' values ('red'. 'green')

4 : number of possible input 'oncoming' values (None, 'forward', 'left', 'right')

4 : number of possible input 'left' values (None, 'forward', 'left', 'right')

This seems like a reasonable nmber of states, as we should be able to visit each of these states by running a finite (reasonably small) number of training simulations.

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?*

The agent is now learning from mistakes. At the beginning, all values in the Qvalues are initialized to zero, so the agent essentially has no idea what the rules of the road are, or how moving towards the destination is better than moving awway from it. For the first few runs, we can see the agent making poor decisions, breaking traffic rules and also moving away from the destination. This also leads to it taking a long time to get to the destination.

However, as the Qvalues are updated after every move, the agent starts to learn what actions are appropriate to take. By scoring 'poor' decisions negatively, and scoring 'good' decision positively, the agent can now choose better decisions when in states that it previously was in by looking at what it learned from it's previous decisions. This has the effect of the agent starting to behave more appropriately, and getting to the destination faster.

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?

Strangely, I'm not seeing much difference by playing with the alpha and gamma parameters. For example, I'm always seeing between 98-100% successful runs (i.e. destination reached within limit) out of 100, regardless of the alpha and gamma parameters.

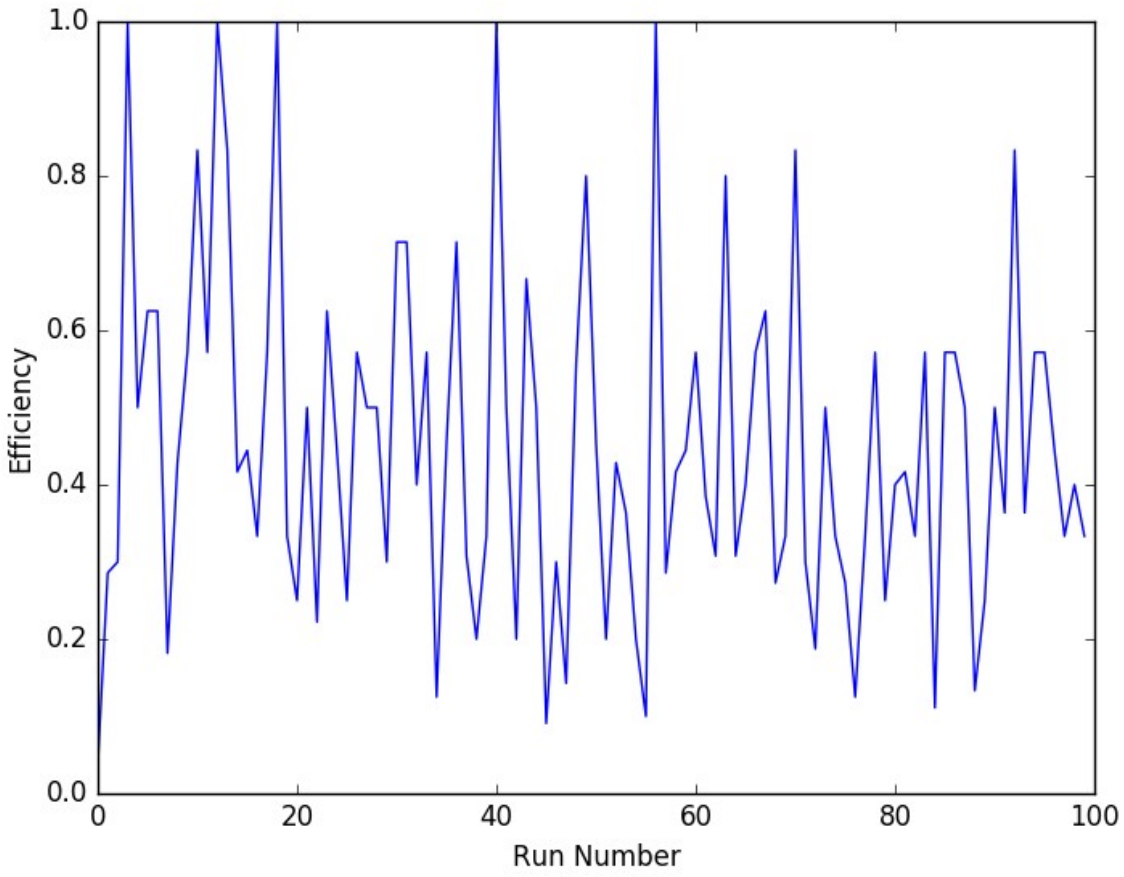
Here's some data from various runs:

Alpha	Gamma	Percentage Complete
1.0	0.1	99%
0.5	0.1	100%
0.5	0.5	100%
0.1	0.1	100%

I added in an efficiency metric to try to look at this closer. I defined efficiency as

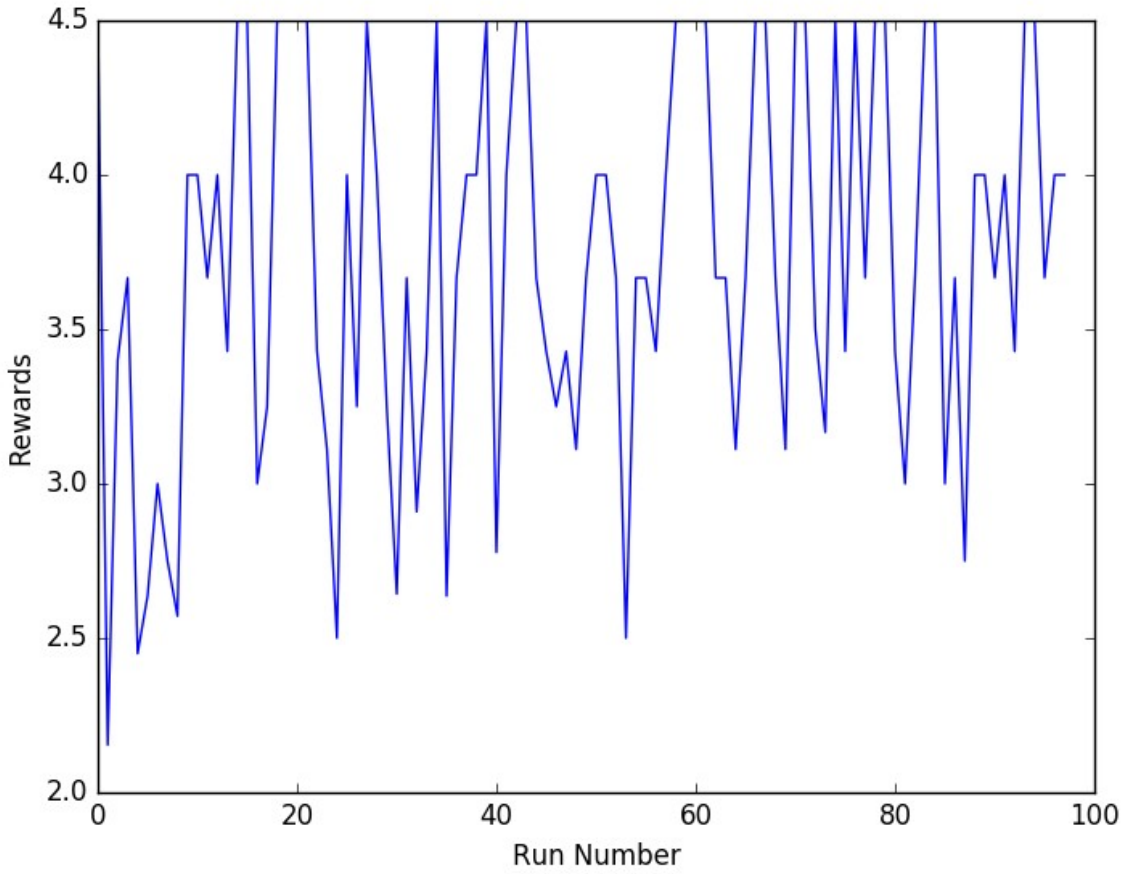
Efficiency = # of moves made / # of shortest possible moves

When I plot this over the 100 runs, I see similar graphs for all alpha/gamma combinations:

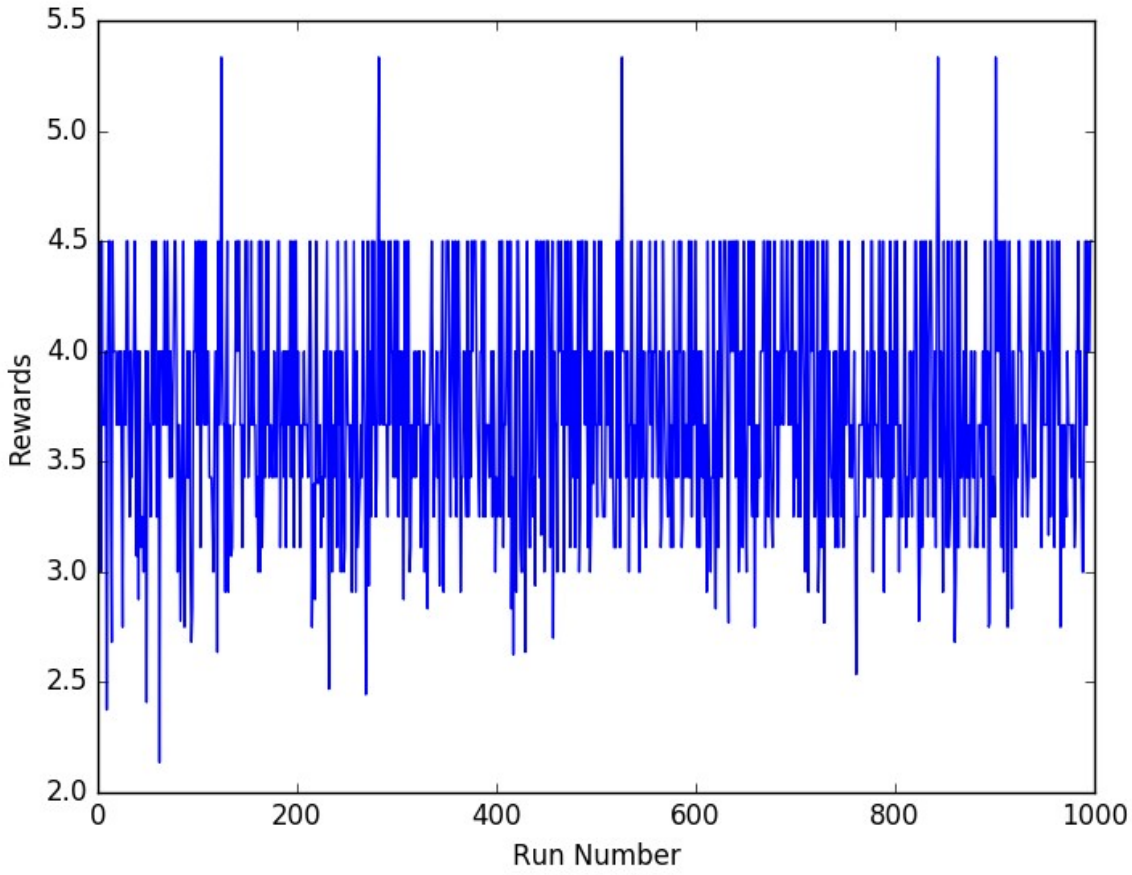


This seems to suggest that the agent is not really getting any better over time. However, on reflection, there is nothing in the implementation that would guarantee the agent to get more efficient over time. All the agent cares about is making decisions that give it the most reward, and our rewards are based on not breaking the rules of the road rather than on being efficient.

I also added a counter to track the total number of penalties accrued per run by summing up all the rewards for any given run. When I look at these, I also don't really see any trend towards avoiding penalties. I then normalized these by dividing by the number of moves made per run. Here's the table of normalized rewards after 100 runs:

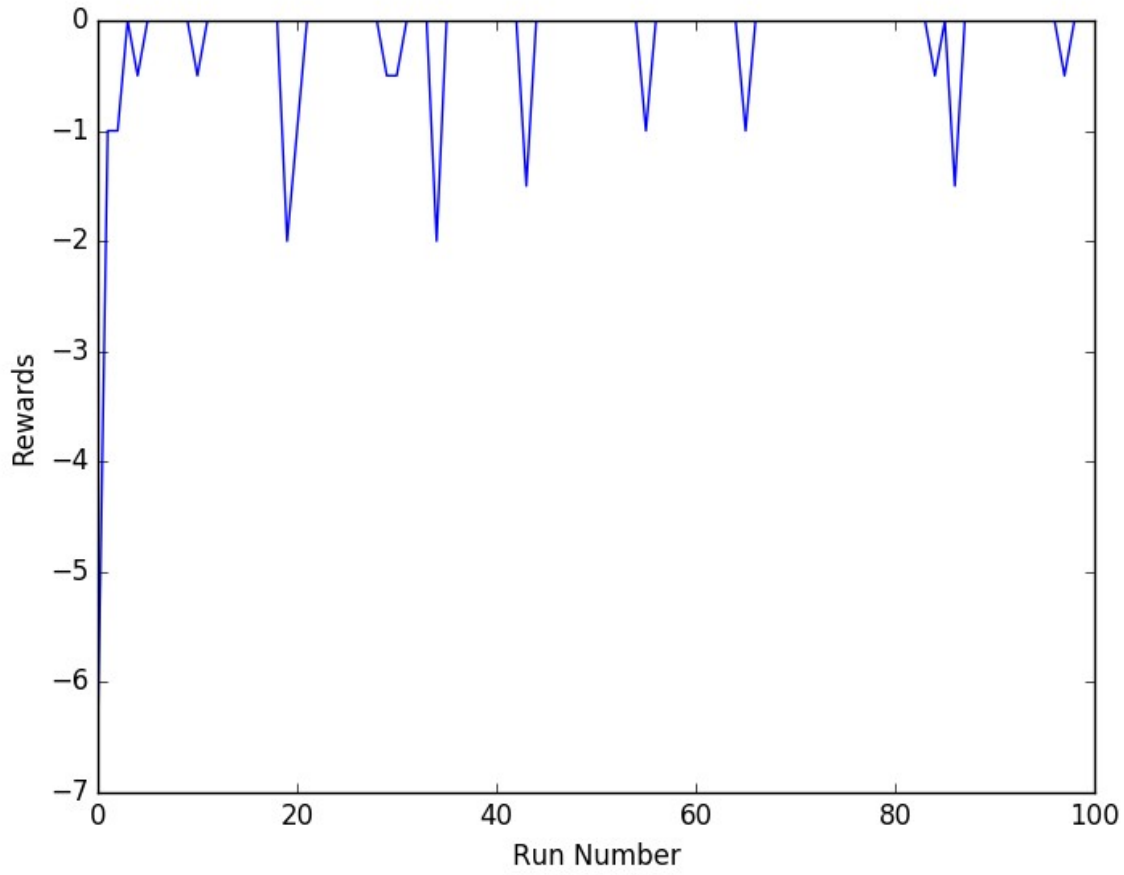


I wasn't sure if you could actually infer an upward trend here, so I re-ran over 1000 runs.



I think this shows that there is no upward trend. So how do I see the learning progression?

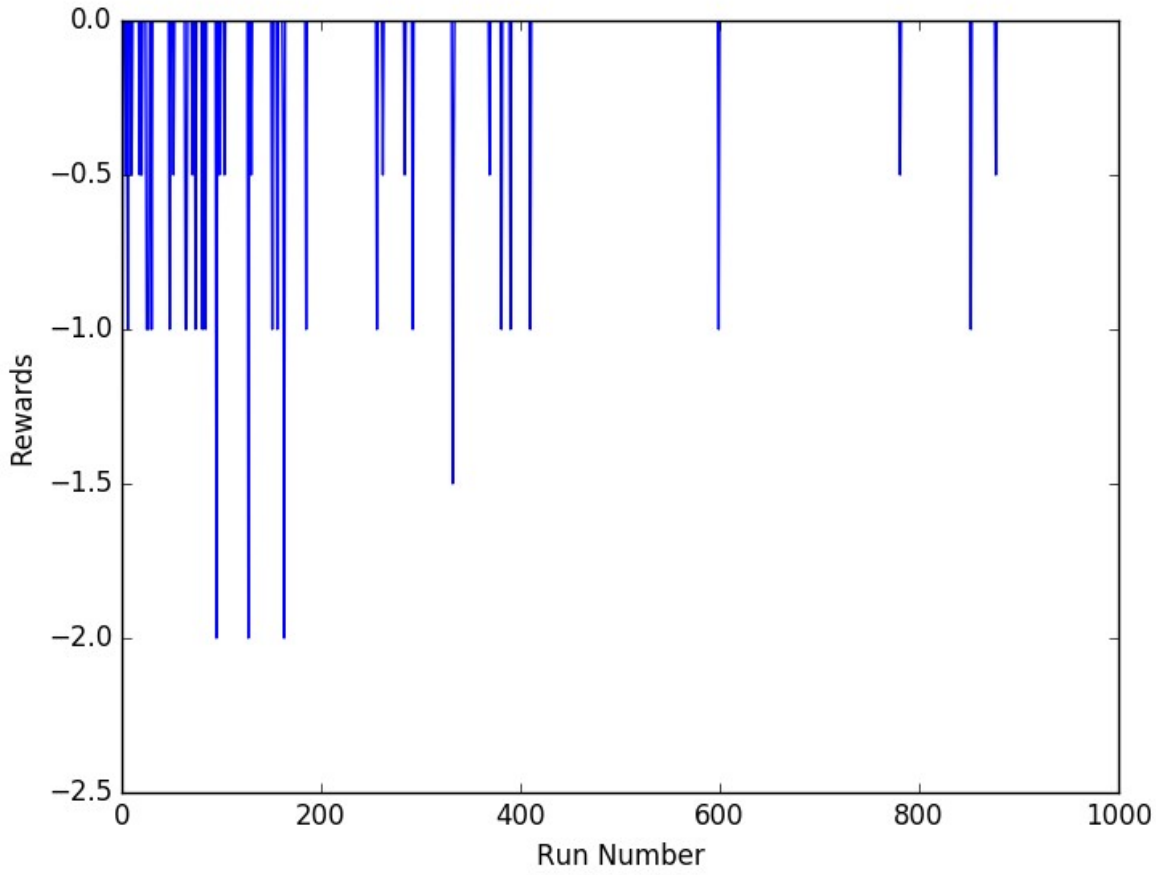
Finally, I think I figured it out. I decided to keep track of all penalties accumulated over each run. The graph of these is as follows:



So here we can see that the penalties start quite high, at -7, and quickly reduce to show smaller and smaller accumulated penalties while still getting to our destination.

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?*

An optimal policy would be one where the agent always gets to the destination on time with zero penalty points. As you can see from the graph above, I think I am pretty close to finding an optimal policy. In fact, when I run over a larger number of iterations, 1000, you can see that after about 400 runs I start to see very few penalties accrued while still achieving 99.7% success rate of destinations reached.



The penalties that occur after about 400 runs must be due to extreme corner cases, and given that the learning algorithm will ensure that they will not happen again, we can say that the agent is approaching the optimial policy.