Smartcab Submission Report

After Random Action implementation

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?

So far, the cab (as expected) moves about at random. Sometimes it happens upon a waypoint, and I imagine with sufficient running time and no obstacles it will always *eventually* hit the waypoint given infinite time, but there's also no assurance that it will do so within any finite timeline no matter how large you make it.

After State Storage

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 the things that should go in local state for the agent should be bits of data that are useful for deciding what to do next. In addition to the desired direction of travel from the planner, almost every input qualifies with the exception of what any car to the right in the current intersection is planning to do.

I can't think of a scenario in which we care about that. If we want to go straight, then if the light is green we can go, and if red we can't. If we want to go right, then on green we just go, on red we care what the car from the left wants to do. If we want to go left, then on red we stop, and on green we care what the oncoming car wants to do. In no case do we care about the intentions of the car to the right (unless that car is not a reliable rule follower, something we might actually want to consider in a more realistic simulation).

We also don't really care about deadline, since regardless of how long is left on the clock we still want to take the optimal action at each step (we would not want to make a car that would break safety laws when in a hurry, this wouldn't be a lot better than a human driver).

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?

This seems like a combinations problem, though I don't know by heart the formulas for arriving at a solution without thinking it through. If we just care about the intended direction of travel, then there are 3 states (since as long as we aren't on the waypoint right now we will want to travel in *some* direction). We also care about the color of the light, which means we can be in 2 light states (red or green) for each direction we might want to travel, 6 possible states. Now we add in the state of oncoming traffic, for which there are 4 states (often there is no car there resulting in "None"); this is also independent which means 4 options for each of the existing 6 states, or 24 possible states. Then finally we can account for the state of the car to the left, which can be in any of 4 states, once again independent of any other data point, so multiply by 4 again to result in 96 possible combinations. This is a lot more than many of the simple examples, but it's still tractable even on an underpowered piece of hardware to keep track of state/action values for 96 possible states. The number of possible actions is 4 ("None" being an option if the safety circumstances of the current environment prevent taking the desired action immediately), so that means we have 96 states and 4 possible actions, or 384 state/action pairs.

After Q-Learning stage 1

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?

At first the behavior is not very different at all, for at least the first episode or two. Quickly the agent begins to manage to get to the waypoint before the deadline, though at least for this iteration it's behavior is a bit strange (more on that in a minute). Over time the agent also begins to drive more safely and legally, though not always. This is occuring because it examines it's history of behavior and what sort of rewards have occured when it's taken actions in given states before. When deciding whether or not to go "forward" when the light is red, it notices that it's been penalized for this before and that taking the "None" action in this state is better, for example.

This is not totally as I'd expect, though. For one thing, the agent appears to prefer to keep driving rather than hold still; so if it can't drive forward like it would prefer because of a red light, but it *can* turn right, it seems to do so even though it will get hit with a -0.5 reward. I suspect this is because I've set the discount rate for future rewards to be 0.8, and it's valuing the future 2.0 rewards it will get for driving back towards the waypoint too highly. I think I should reduce the discount rate signficiantly.

Additionally, it still takes illegal actions sometimes because I'm using an exploration value of 10% and so even when it knows the right thing to do, 10% of the time it takes a random action, often an unsafe or illegal one. I think I need to have some exploration path that doesn't just pick a random value; maybe we start with a higher epsilon value and reduce it to 0 over time by reducing it each time the exploration path is hit by a tiny amount.

During Q-learning parameter iteration:

Learning Rate: 0.5 Discount Value: 0.8 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.0 Trials: 100 Deadlines Missed: 23 Successful Arrivals: 77 Traffic Infractions: 71

attempts to make epsilon value degrade over time

Learning Rate: 0.5 Discount Value: 0.8 Initial Epsilon Value: 0.15 Epsilon Degradation Rate: 0.01 Trials: 100 Deadlines Missed: 62 Successful Arrivals: 38 Traffic Infractions: 5

Learning Rate: 0.5 Discount Value: 0.8 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 26 Successful Arrivals: 74 Traffic Infractions: 25

decrease discount value to make strange looping behaviors reduce

Learning Rate: 0.5 Discount Value: 0.3 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 12 Successful Arrivals: 88 Traffic Infractions: 27

increase learning rate to value immediate rewards more highly

Learning Rate: 0.65 Discount Value: 0.3 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 14 Successful Arrivals: 86 Traffic Infractions: 22

crank down the discount value further, that made a big impact on success rates

Learning Rate: 0.65 Discount Value: 0.2 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 35 Successful

Arrivals: 65 Traffic Infractions: 22

actually that was worse, bring discount value up a bit and learning rate down

Learning Rate: 0.6 Discount Value: 0.25 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 11 Successful Arrivals: 89 Traffic Infractions: 26

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?

The numbers are available above. The last set performs best, of the sampled parameter combinations, and behaves more rationally than the initial agent when it's route is blocked. Once it's gone through around 12 iterations, it begins to behave reliably, and by the time it's gone through the first 30 episodes it's traffic infraction rate drops significantly. I suspect over time it performs quite well, so I'm going to run another set of 5,000 trials because I suspect it has converged already and will perform more or less successfully from this point forward.

Learning Rate: 0.6 Discount Value: 0.25 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 5000 Deadlines Missed: 331 Successful Arrivals: 4619 Traffic Infractions: 24

This approach is working ok, but there are some edge cases. When we run into another car at an intersection in the oncoming lane, we tend to lock up. It seems we've learned that it's simply safest to let other cars go, and if both agents decide that then we get stuck. We could up the experimentation rate, but that would result in more infractions and we don't want that. I think the next step might be to have the epsilon value scale out the current q values by random amounts rather than just go away entirely so that if we're stuck for long enough the car will pick another "valid" direction eventually. Another possibility might be to attach a slight negative reward to holding still; less than the penalty of committing an infraction, of course, but high enough to encourage us out of just being parked and waiting. Increasing the discount rate would also help with this. That's probably the easiest to mess with so we'll try that one first.

Learning Rate: 0.6 Discount Value: 0.4 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 14 Successful Arrivals: 86 Traffic Infractions: 23

This doesn't seem to be helping enough when I scale it up over a large number of trials, we still suffer failure > 10% of the time because of locking up in certain cases. I'm going to revert the discount value and try instead accumulating a small negative reward for staying still.

Learning Rate: 0.6 REWARD FOR STAYING STILL: -0.1 Discount Value: 0.25 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 100 Deadlines Missed: 4 Successful Arrivals: 96 Traffic Infractions: 28

That seems way better. Let's see how it does scaled up over 500 trials.

Learning Rate: 0.6 REWARD FOR STAYING STILL: -0.1 Discount Value: 0.25 Initial Epsilon Value: 0.1 Epsilon Degradation Rate: 0.001 Trials: 500 Deadlines Missed: 38 Successful Arrivals: 462 Traffic Infractions: 26

We're still missing some deadlines unnecessarily, but fewer, and the infractions are stable. This is probably good enough.

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?

I believe this is fairly close, but that an optimum policy would probably make some cleverer decisions when blocked. I think that by re-implementing the epsilon degredation to instead have an action randomly selected from the actions known to not be illegal that could be slightly better. Still, this is fairly close specifically because we care about both not breaking the law/hurting people and getting there on time, but far more about the first than the second thing. If we can get there on time 90% of the time, and never break the law or crash, that's better than a higher time-efficiency rating at the expense of causing traffic accidents occasionally. The most optimal policy (if it exists) would arrive on time 99.9% of the time (when deadline is reasonable) is never crash or break the law.