Project 4: Training a Smartcab

Implement a Basic Driving Agent

We can make the agent pick a random valid action by adding the following: `action = random.choice(environment.valid_actions)`

Observing the agent, it's clear that it's wandering with no clear direction (which is to be expected because it ***really is*** random). If we disable deadlines entirely, it will always reach its destination eventually, just through random chance. Using the hard limit of 100 moves, it succeeded in 18 out of 20 trials. Using the regular constraint, it succeeded in 4 out of 20 trials.

Inform the Driving Agent

First, note the information that the car recieves stored in `inputs`: location, heading, light, oncoming, left, and right. There is also `deadline` (how many more actions can be taken), and the next waypoint given by `planner`.

The states that I want to store are the following:

- `light`: whether the light at the intersection is 'red' or 'green'
- `oncoming` whether there are cars going forward from any direction
- `left`: whether there are cars turning left from any direction
- `waypoint`: the direction in which `planner` suggests

The reason why I chose to include light, oncoming, left, and waypoint is straightforward: these are all inputs required to obey the rules of traffic. For example, one cannot drive straight through a red light, one cannot turn left at a red if there is oncoming traffic, one cannot turn right if there is traffic coming from the left, etc.

There were a couple measures that I chose to ignore, such as `deadline`, `right`, and `location`, which all deserve explanation.

`deadline` is excluded because I feel any incremental improvement in the agent's state involving a measure of 'urgency' would be offset by the increase in steps for the Q-learning algorithm to converge (since this will exponentially increase the number of possible states for the agent). The rationale for `location` is similar, with the added fact that `waypoint` already gives the agent a sense of direction (i.e. it basically provides no extra information, and would make the agent practically unlearnable within a reasonable number of steps).

Right is excluded because if the other agents obey the law, `right` provides no relevant information with which the agent can make better decisions. To see why, just note that regardless of which direction one goes, cars turning right will not affect you.

Implement a Q-Learning Driving Agent

My implementation of Q-learning is as follows:

- initialize all original Q-values as 5
- set the learning rate to 0.3
- added a 30% chance for picking a random move
- set the discount rate to 0

I added a 30% chance for picking a random move to encourage the agent to explore more. This also helps it avoid getting stuck in local optimas, by forcing it to expose itself to new situations.

The reason why I set the discount rate to zero is because I didn't think the extra complexity is meaningful for the agent's training. This is mostly due to the artificial constraints of the simulation. For one, the car only recieves information for the traffic light ahead of it, and whether cars are coming from any intersection within immediate view. In other words, our agent does not have a view of the ***bigger picture***. The car does not know where it is going, or where it is inside the city; it only knows the direction it must know, and most importantly, this is really all it needs to know within this simulation.

The behaviour changes are immediate after a few trials: the agent starts to obey traffic rules, and starts to head towards where the planner tells it to. There are some strange behaviours though: for example, it may wander aimlessly in street-legal circles along its way to the destination. This is probably because street-legal moves give it points, even if it's not necessarily heading towards its destination at the time. The reason why these behaviours emerge is that as more information is gathered, the agent starts to pick moves that will maximize its reward.

Improve the Q-Learning Driving Agent

Before tuning, the results were already *fairly* good, with a success rate hovering around 85%. There were several tweaks that I made to help improve the performance.

For one, I gradually decreased the learning rate and random choices a little bit for each trial. This is the fundamental trade-off between exploring and exploiting: I figured since we were only being measured for successes within the first 100 trials, the agent had to take less risks to maximize total reward and success rate, even if it would learn a bit less. Also, it seems to me as if most of the learning will take place at the beginning, so it's not too large of a tradeoff anyways.

Something else I did that also improved the success rate was to tweak the "random" move to be less random. What I did was make the chance of picking any random move proportionate to the expected reward that it would give. This way, its exploration is more productive: this effectively means the agent will be less inclined to take more calculated risks.

learning_rate	randomness	error rate

learning_rate	randomness	error rate
0.30	0.30	4.5%
0.35	0.35	5.167%
0.40	0.40	2.1667%
0.45	0.45	2.5%
0.50	0.50	3.83%

I also changed the learning rate and chance to take a random move to 40%. Over 6 trials of 100 runs each, I got an average success rate of approximately 98%. I should note, however, that the initial learning_rate and randomness do not seem to make too large of a difference. In all honesty, I'm inclined to attribute the approximately 2% error rate to random chance. Using any parameter between 0.3-0.5, we seem to get a success rate of between 95%-98% consistently in the first 100 runs.

I also ran tests with and without decay enabled, and decay seemed to produce consistently better results over 4 trials of 100 runs each:

learning_rate	randomness	decay enabled	error rate
0.1	0.1	yes	9.5%
0.1	0.1	no	10.5%
0.2	0.2	yes	4.75%
0.2	0.2	no	5.25%
0.3	0.3	yes	4.75%
0.3	0.3	no	6%
0.4	0.4	yes	2%
0.4	0.4	no	3.25%

The Optimal Agent

It seems to me as if the optimal agent would be one in which the agent simply follows the planner the entire time, while following the traffic rules. Since the agent has no global awareness, turning either left or right has a 50% chance of going in a direction that will move the agent in the correct or wrong way. However, this decision would not have neutral value, since going in the right direction will reduce the commute by 1 unit, whereas going in the wrong direction could increase the commute by as much as 4 units - the expected value *should* be negative.

Near the end of my training cycles, the agent seems to follow my (hypothetical) optimal strategy fairly closely. This is with the exception of the rare event that it wanders in a circle or two before once again heading in the direction of the planner. I should note, however, that it almost does always reach the destination within the alotted time.