Train a Smartcab to Drive

§1 Implement a basic driving agent

Initially we implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),

And produces some random move/action `(None, 'forward', 'left', 'right')`.

We run this agent within the simulation environment with `enforce_deadline` set to `False` (see `run` function in `agent.py`), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by the agent and reward/penalty earned are shown in the simulator.

§1.1 Agent's behaviour

The agent traverses the grid in a random fashion. The agent does reach the destination if we leave the program running long enough (in one case this was 274 actions) but not before the deadline.

§2 Identify and update state

We identify a set of states that are appropriate for modelling the driving agent. The main source of state variables are current inputs, but not all of them are worth representing.

At each time step, we process the inputs and update the current state. We run it again and observe how the reported state changes through the run.

§2.1 Picking states

The rewards and penalties need to be sufficiently represented in choosing a set of states in order for the agent to learn a feasible policy. The sensory inputs which are important for determining the reward are as follows:

<u>The next waypoint</u>:

This is the direction we would ideally travel in to reach the destination as quickly as possible and can be any one of three values ('left', 'right' or 'forward'). We get a reward of 2 every time the action we take matches next waypoint, provided the move was legal.

Illegal moves incur a penalty of -1. For legal moves which are not the next waypoint we get a reward of 0.5. Doing nothing (action = None) is always legal and yields a reward of 1. Notice doing nothing is preferable to travelling in a direction other than that given by next waypoint.

<u>The traffic light colour</u>:

This can be either 'red' or 'green' and determines if we can legally traverse the junction or not.

<u>Other cars at the junction and which direction they are travelling in</u>:

The legality of our move is also determined by how we behave in relation to the other vehicles at the junction. There is potential for other cars approaching the junction from three possible directions ('left', 'right' or 'oncoming') and each car might be going in one of three different directions or there may be no vehicle ('left', 'right' or 'forwards', None).

<u>The deadline</u>:

This can be any value between 0 and 70 since it is five times the distance from the destination and the destination can be a maximum of (8+6=) 14 intersections away. If we reach the destination within the deadline we receive a reward of 10.

Including all of these, results in a state space size (4 x 2 x 3 x 4 x 71=) 6816. We know that for our Q-learning algorithm to converge, we have to visit each (state, action) pair infinitely often so the smaller the state space the better. We can reduce the size of the state space in the following way.

When arriving at a junction we need not know what traffic coming from the right is doing since it has no impact on whether we can traverse the junction legally or not, regardless of which direction we wish to go in. In-fact with regards to the other vehicles at the intersection, we only need to know the following two things:

1. If there is traffic coming from the left which intends to go forwards or not – this determines whether we can legally turn right or not
2. If there is oncoming traffic going forwards or turning right – this determines whether we can turn legally left or not

We can also significantly reduce the contribution to the state space by the deadline reducing the values of the deadline that we consider. The reward is only dependent on whether the deadline has been passed or not.

So our state becomes the following five dimensional variable:

1. Next waypoint (None, 'forward', 'left' or 'right')
2. Light ('red' or 'green')
3. There is oncoming traffic either going forwards or turning left (Boolean)
4. There is traffic coming from the left going forwards (Boolean)
5. Deadline has not passed (Boolean)

Using these variables in place of our previous variables relating to other traffic and the deadline results in a state space size (4 x 2 x 2 x 2 x 2 =) 64 and we have reduced its size by more than a factor of 100 without losing any information of consequence.

§3 Implement Q-Learning

We implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, we pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero) which the agent takes into account when updating Q-values. We run it again, and observe the behaviour.

§3.1 Changes in the agent's behaviour

The update rule we use to propagate Q-values across the state, action space is

$$Q_{t+1}(s,a) = (1-\alpha_{t+1})Q_t(s,a) + \alpha_{t+1}\left(r + \gamma \max_{a'} Q_t(s',a')\right).$$

Here r is the reward for taking action, a, at time t, in state, s, to arrive in state, s' at time $t + 1$. From state, s', we proceed optimally by choosing the action which has the largest Q-value, $\max_{a'} Q_t(s', a')$. γ is the discount factor and α_t is the learning rate.

In our basic implementation of Q-learning we chose to initialise our Q-function values to zero, we chose the discount factor, $\gamma = 0.5$ and learning rate $\alpha_t = 1/t$.

The resulting behaviour of the agent depends very much on the values with which we initialise our Q-function together with the initial action (the action we assume to have the maximal Q-value before searching for a larger one). We start by describing the extreme case for our problem in which we choose None as our initial action.

<u>initial action = None</u>:

This is a poor choice since it results in us getting stuck repeatedly choosing None over all other actions. This is because in our environment every time the agent takes the action None it is guaranteed a positive reward of 1. The first time the agent visits a state it takes the initial action None and receives a reward of 1. Eventually this positive reward makes the values in the Q-table (initially zero) positive for all (state, action = None) pairs and leaves the Q-values zero for all the other actions. Every time we revisit a state, action = None has the largest Q-value so we again take action None. One could argue here that the problem here is that the agent is greedy to its detriment. That is to say it chooses to exploit what it has learnt over exploring more to find better possible solutions and thus gets stuck in a local maxima of the Q-function.

<u>initial action = random choice</u>:

A better choice is to make the initial action random. Doing this unfortunately does not in this case resolve the problem of local maxima of the Q-function which arises because the agent is greedy. Now, when the agent gets a positive reward from an action in a given state the first time we visit it, that action will be the one chosen when we revisit the state in the future because that action will have the maximal Q-value. Choosing that action again further propagates that positive Q-value to neighbouring states resulting in that action being taken over others in an increasing number of states.

As before action = None gives a guaranteed reward of 1 so creates a local maximum of our Q-function, resulting in the agent often choosing to do nothing over taking other actions. Another local maximum occurs when the agent takes a legal action which is not the next waypoint and gets a reward of 0.5. In particular, because there is very little traffic and one can legally turn right even if the traffic light is red, we often find our agent going around in circles turning right repeatedly for several iterations at a time.

§4 Enhance the driving agent

We apply the reinforcement learning techniques we have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of the agent. The goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

§4.1 Improved Q-learning

The main problem we encountered with our basic Q-learning agent was that it was not exploring enough of the state, action space. Once it had found an action which gave a positive reward (and hence positive Q-value) it would continually choose that action over others rather than exploring to see if there was a better policy for that particular state.

One way to encourage more exploration by the agent is to increase the value with which we initialise our Q-table so that most actions, the first time they are taken, decrease the Q-value rather than increasing it – only the most rewarding actions might result in an increase. This means that in future iterations, the agent chooses a new action over the ones chosen previously rather than vice versa as we were seeing when we chose the initial Q-values to be zero. This approach is effectively increasing the level of reward for which the agent will settle. More concretely, the agent will not settle for a reward of 1 for taking no action or 0.5 for taking a legal action other than next waypoint when the initial Q-value is 2. Choosing a value of 2 rather than zero takes the performance of our agent from managing to not reach the destination in time for any of the 100 trials to reaching the destination in time for 98 of the 100 trials.

In Tables 1 and 2 below we look at the number of trials out of 100 which arrive at the destination in time for 5 Runs and take the average to compare different parameter choices. In Table one we compare results for varying choices of the discount factor γ, in Table 2 we look at varying the initial Q-values, $Q_{t=0}(s,a)$.

γ	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Average**
0.2	79	90	87	44	81	**76.2**
0.4	90	89	82	84	91	**87.2**
0.5	97	98	99	63	96	**90.6**
0.6	7	25	99	85	0	**43.2**
0.7	32	0	0	34	1	**13.4**

Table 1: The number of trails out of 100 for which the agent reached the destination in time for varying values of gamma on 5 separate runs

$Q_{t=0}(s,a)$	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Average**
2	97	98	99	63	96	**90.6**
2.1	100	91	96	100	100	**97.4**
2.2	98	97	85	97	90	**93.4**
2.3	86	96	83	84	91	**88**
2.4	89	92	87	91	85	**88.8**

Table 2: The number of trails out of 100 for which the agent reached the destination in time for varying initial Q-values on 5 separate runs

In our final version of the agent, based on Tables 1 and 2 we choose, $\gamma = 0.5$ and $Q_{t=0}(s,a)$ = 2.1

§4.2 Optimal policy

We know from the reward structure that the optimal strategy (assuming that we do not know our location or that of the destination) is to take the action which matches the next waypoint if the action is legal (collect reward 2), otherwise take action None (collect reward 1). We note that the deadline is set to 5 times the distance to the destination. So provided for every trial, on average, we are able to take the next waypoint, rather than do nothing, at least one in five moves we will reach the destination within the deadline and collect an additional reward of 10. Note that this is always possible because the traffic light is set to change randomly every 3, 4 or 5 moves. Using this policy, there will be no penalties incurred by the agent.

Note that this strategy is not necessarily the fastest route. There may be one or more faster routes to the destination. Take for example the case where neither co-ordinate of our location matches that of the destination. In this case there are two possible directions we could take which would bring us closer to the destination, only one of which is the next waypoint. It may be that by taking the other option, rather than doing nothing because action = next waypoint is illegal, will get us to the destination faster.

However, this strategy will collect fewer rewards along the way, since doing a legal move which does not match the next waypoint results in a reward of 0.5 instead of 1 for doing nothing.

Another example of where a faster route exists is where the location and destination are close to opposite edges of the grid. Because the grid wraps around on itself (agents can leave one side of the grid and appear on the other in one move) and the next waypoint does not utilise this, there is in these cases a faster route than that indicated by next waypoint. The information the agent has is not sufficient for it to find these faster routes consistently.

The policy found by the learning agent is one that allows it to reach the destination in time with a high success rate, however it does not find the optimal policy. In the worst case I could find, the agent reaches the destination in time for 91 of the 100 trials, the failures occur on trial numbers 11, 26, 30, 51, 53, 71, 79, 96 and 100. In the best cases where the agent reached the destination on every one of the 100 trials, it would still occasionally incur penalties. The learning agent consistently (for every trial in 100) finishes with net positive reward.