Project 4: Train a Smartcab to Drive

Stefan Dulman

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?*

The result of the simulation is that our agent performs a random walk until it finally hits the destination. Yes, the cab will hit the destination as with this kind of behavior, there is a non-zero probability for every point in the simulated space to be reached.

The behavior resembles more a drunk person walking by ignoring all rules than an intelligent smartcab. The traces reflect also the fact that the agent ignores all information, for example crossing on red:

```
LearningAgent.update():
  deadline = -4,
  inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': Non
e},
  action = forward,
  reward = -1.0
```

or stopping on green:

```
LearningAgent.update():
  deadline = -5,
  inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': No
ne},
  action = None,
  reward = 0.0
```

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?*

The states we identified are defined by two sources of information: time information (derived from `deadline`) and traffic state information (derived from `light`, `oncoming`, `right`, `left` and `next_waypoint`).

We chose to ignore the `deadline` variable as it increased the number of states too much for the amount of simulations at hand.

Not all values for the identified variables are relevant, so the number of states is smaller than 2*4*4*4*4 = 512. For example, for the `next_waypoint` variable, only three values are usefull - as if it ever hits 'None' the simulator halts the algorithm. We can reduce the states defined by `light`, `oncoming`, `right`, `left` observing the fact that various combinations lead to two main cases from the perspective of `next_waypoint`: is it ok to cross the intersection or not. The states are presented in the following table:

State variable	next_waypoint	light	other
forward_free	forward	green	
forward_busy	forward	red	
left_free	left	green	oncoming = left or None
left_busy	left	green	oncoming = forward or right
left_busy	left	red	
right_free	right	green	
right_free	right	red	left = not forward
right_busy	right	red	left = forward

In short, we defined the traffic-related states by observing the the DummyAgent used in the simulation never crosses on red, so the color of the traffic lights is respected by the other objects in the simulator. The state `forward_free` corresponds to the `next_waypoint` being `forward` and the intersection being clear (green traffic light). Similarly, `forward_busy` shows the need of going forward but intersection blocked (red traffic light). The other variables are irrelevant here, as the driver going forward has priority.

The state `left_free` showcases the need of turning left and the intersection being cleared (green traffic light and oncoming traffic `None` or `left`). The state `left_busy` happens when either the traffic light is red or when it is green but we have oncoming traffic crossing our path.

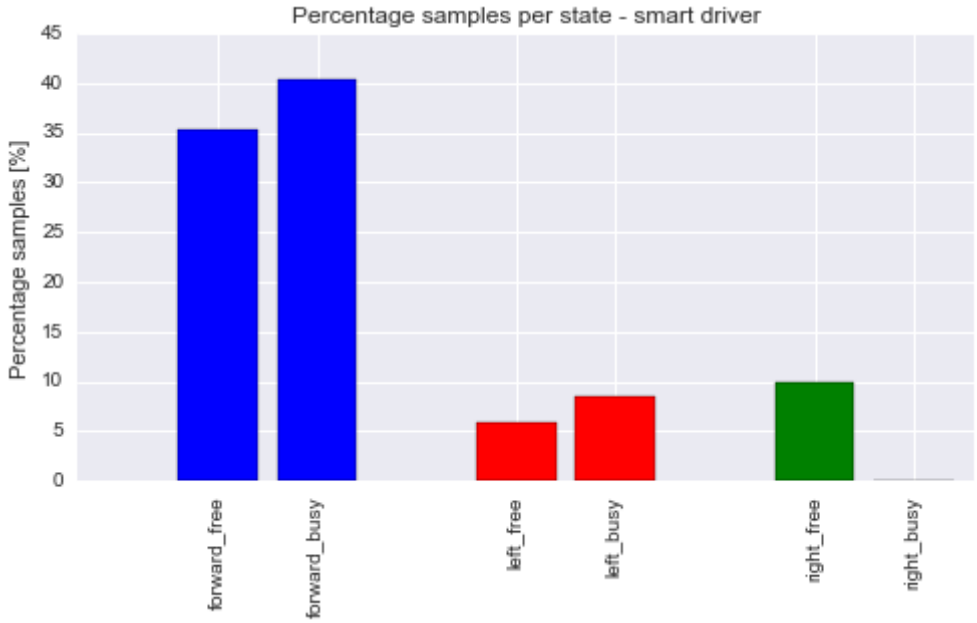
The state `right_free` combines the need of going right with the possibility of doing so. This appens when the traffic light is green or when it is red but no traffic from the left interferes. `right_busy` happens when the agent can't really go right (it is red light and there's traffic from the left interfering).

It is interesting to notice that the 'right' variable was not needed for defining the previous states, so we removed it as well.

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?*

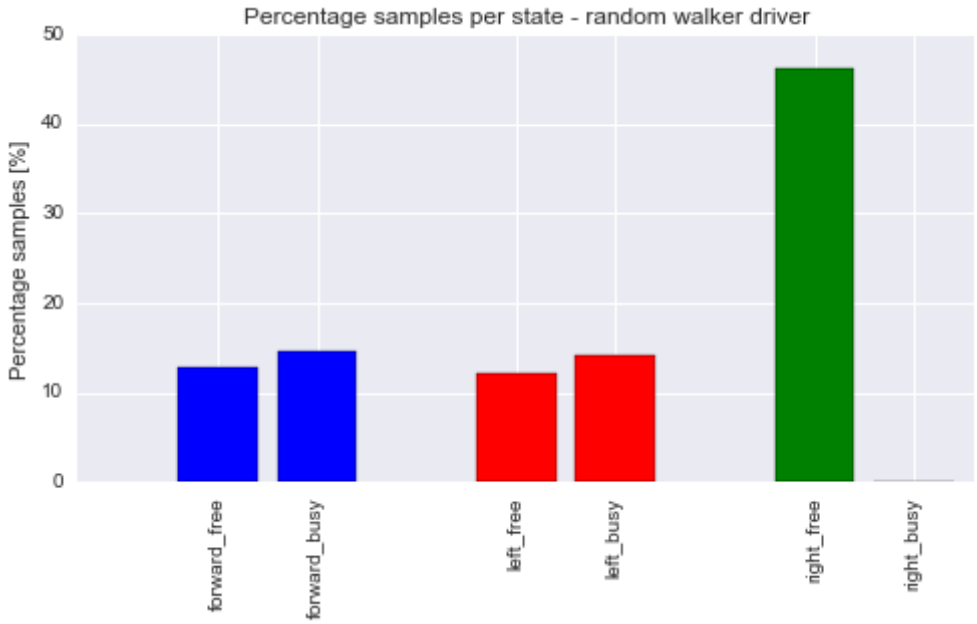
The overall number of states when we consider all variables is pretty large: 512 for `light`, `oncoming`, `right`, `left` and `next_waypoint`. Fortunatelly, we were able to reduce the problem to six states only, which is a small sized problem.

A major problem is the amount of samples the algorithm has for learning. For example, let's consider the case of a perfect driver from the perspective of traffic, with no constraint on time. The cab respects the traffic rules and moves closer to the destination if allowed. In this case, the percentage of states the cab sees is presented in the figure below.



We ran the simulation 10000 times and aggregated the results from all simulations. The behavior is as expected: most of the states require the cab to drive forward (after aligning towards the target, it should just go straight). Because the cab simply drives on green and needs to stop on red the `forward_busy` is larger in expected value then `forward_free`. On the other hand, due to sparsity of cabs, `right_busy` state is barely present at all.

Let's look now at the random walk cab. Initially, the cab will still need to learn what to do. Due to the little amount of samples for some states it will learn those wrong, so the final number of states seen by the cab will be somewhere in between the smart and random walk cabs.



As an overview, the number of states for forward, right and left movement is different now from the previous figure, `forward` not being dominant anymore. This happens because the cab ignores the needed direction and the distribution among the three categories should be more or less identical. We notice a dominance of the right movement states - this is due to the right long u-turn suggested by the planner when the cab faces the opposite direction (left u-turns are not implemented). Also with respect to the right movement states, the states in which a right turn is not possible is very low (due to the low density of agents), the cab being able to turn on right on both green and (almost always) on red.

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?*

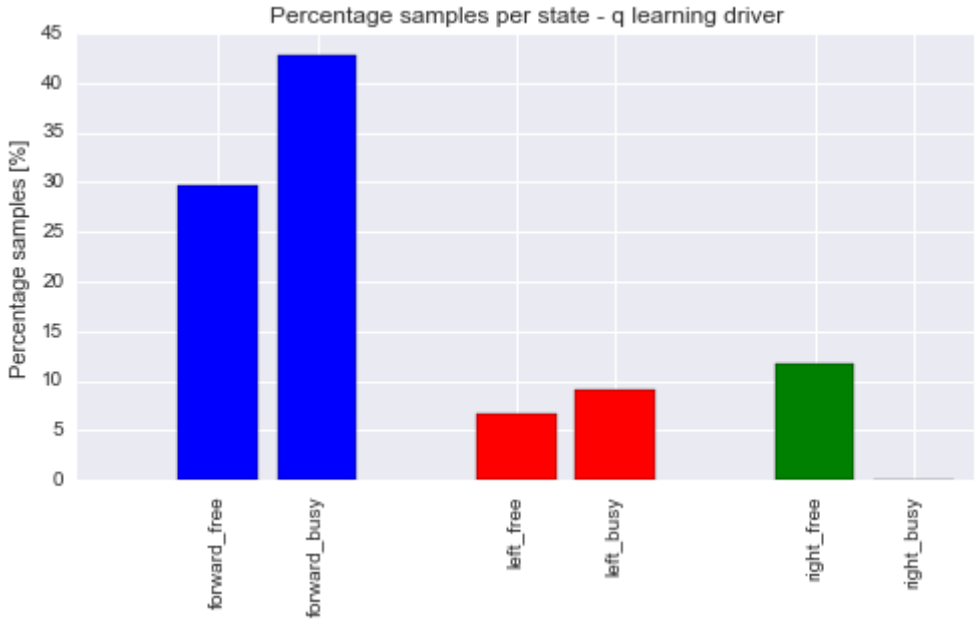
After implementing the Q-learning mechanism, we noticed that the agent improved with time vey fast and was able to complete the assignment with high rewards (in roughly 98% of times!). This is in contrast with the random agent which was hitting the destination based on the properties of random walk.

The Q matrix for one of the runs was:

	None	Forw.	Left	Right
forward_free	0.39	**2.12**	-0.04	-0.10
forward_busy	**0.27**	-0.73	-0.90	-0.30
left_free	0.00	-0.10	**2.27**	-0.30
left_busy	**0.31**	-0.57	-0.67	-0.16
right_free	0.41	-0.39	-0.44	**2.22**
right_busy	0.00	-0.34	0.00	0.00

The agent learned the proper rules - if it needs to go towards a certain direction and the intersection is free it proceeds. Otherwise it stops. In the previous table we highlighted the maximum Q values on each line to confirm this. Exception is the `right_busy` state where in 100 runs 1-2 samples are seen, so the Q marix values are meaningless.

Regarding the number of samples seen for each state, the graph below shows that the agent basically converged to the smart driver above (which was the desired behavior):



As for the total reward gained by the agent, in the firt five runs, the values were: `[-1.0, 17.5, 22.0, 20.0, 24.0]` while in the last five runs: `[18.0, 17.0, 28.0, 23.5, 20.0]` confirming the very fast convergence of the algorithm.

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?*

For the section above we used the parameters: `alpha` = 0.6, `gamma` = 0.2 and `epsilon` = 0.05. In the file `agent.py` we used the function `scanparameters()` to perform a grid search for a better set of parameters. `Better` is actually dependent on the optimization metric chosen. We tried several combinations:

- for the metric set to the shortest amount of (relative) time taken for the last ten trips, the grid search algorithm outputed the parameter set: `alpha` = 0.7, `gamma` = 0.355, `epsilon` = 0.05
- for the metric set to the maximum reward gained in the last ten trips, the parameter set was: `alpha` = 0.6, `gamma` = 0.3, `epsilon` = 0.07

We noticed that the parameters are very close to each other (within the noise boundary actually). For this parameter set, the agent behaves better than the original q learning driver above. For comparison, these is the total reward gained in the last experiment in the last five runs: `[27.0, 22.0, 32.5, 29.0, 34.0]`. These values are larger than the default q learning agent above.

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?*

Let's start with the second part of the question: an optimal policy for the problem. Given the data available to the agent, I think the optimal policy is driving toawrds the destination by obeying the traffic rules. So, if the agent needs to move forward and it meets a busy intersection, it should stop and wait for it to clear. When free, it should continue. The same holds for all directions of movement. *Smarter* ways of moving around (avoiding red traffic lights, congestions, etc.) are not really possible as the agent cannot learn the state of the traffic lights above. From the perspective of Q matrix, an optimum policy would be represented as having all the elements in the matrix 0 (for example) and 1 on the following positions: (`forward_free`, `forward`), (`forward_busy`, `None`), (`right_free`, `right`), (`right_busy`, `None`), (`left_free`, `left`), (`left_busy`, `None`).

Moving on to the first part of the question, yes, the agent comes very close to the optimal policy. When we ignore the time component, the optimal policy is given by the `smart driver` agent described above. Looking at the table for the Q values above we conclude that the agent actually maps onto the `smart_driver` with two differences:

- for the right_busy state the action will be more or less random as the agent did not see enough states. The positive part is that these states do not really occur in the simulations.
- the agent has a probabilistic behavior. So, for (100*`epsilon`)% of the times it will guess what to do. Of course, after training we could remove this randomness.