Smartcab project report


```
author: miroslav.karpis@gmail.com
project: Udacity Reinforcement Learning project (Machine Learning Engineer Nanodegree)
```


Simulation details

Valid actions

[None, 'forward', 'left', 'right']

Rules

If an agent is at a green light, it should be allowed to:

- Go forward
- Turn right
- Turn left, yielding to any oncoming traffic that is going straight or turning right

At a red light:

- Turn right, yielding to any traffic from the left that is going straight

Rewards and Goal

- The smartcab receives a reward for each successfully completed trip.
- Smaller reward for each action it executes successfully that obeys traffic rules.
- Smartcab receives a small penalty for any incorrect action.
- Larger penalty for any action that violates traffic rules or causes an accident with another vehicle.

Task 1: Implement a Basic Driving Agent

Question 1: Does the smartcab eventually make it to the destination? During my observations (approximately 10 cycles of new target placement), I have observed only 1 case, when the car reached the target (10% success rate).

Question 2: Are there any other interesting observations to note? So far I didn't observe very much of interesting behavior.

Task 2: Inform the Driving Agent

Question 1: What states have you identified that are appropriate for modeling the smartcab and environment?

Currently we have available combinations between following states/inputs:

- **light**: ['red', 'green']
- **oncoming**: Information whether a car is approaching from oncoming(forward) direction, and the direction where the vehicle is heading. [None, 'right', 'left', 'forward']
- **right**: Information whether a car is approaching from right direction, and the direction where the vehicle is heading. [None, 'right', 'left', 'forward']
- **left**: Information whether a car is approaching from left direction, and the direction where the vehicle is heading. [None, 'right', 'left', 'forward']
- **waypoint**: The next waypoint location relative to its current location and heading. ['right', 'left', 'forward']
- **deadline**: Decrementing iteration counter given to smartcab reach the goal/target.

Generally, the best and in our scenario also safest would be to consider all the given input states (light, oncoming, right, left, waypoint). Down side from choosing all the states is, that it might take more time (learning curve will be not very steep), which means that it might take more time to find global optimum. It will be worth to perform more tests and observe the relationship between number of states and learning curve.

> Note: After trying to include all the states into the Q_table I could see that after relatively long time some of the state values have still not been used (had initial value), and the results did not show some bigger improvements.

Selected states:

- **waypoint**
- **light**

Question 2: Why do you believe each of these states to be appropriate for this problem? Below are reasons why I selected following states. The reason why I have chosen them is that they give most information and possibility value.

Selected states:

- **waypoint**: gives the most Important information about our goal (recommended future step)
- **light**: our smartcab will be facing this state every iteration, so invalid action would cause to negative reward every time step.

Not included states:

- **right**: The state information is important/used only in the combination with red-light. Otherwise this information should be not important. Because of the low use and probability of occurrence I have decided to not include this side into our model. Including the model would cause longer exploration phase without very much added value.
- **left**: Reason why didn't include left state, is the same as described above. The chance that the state during the simulation is very low.
- **oncoming**: Similar as above the chance of occurring the state together with green-light and left state is very low.
- **deadline**: Deadline is randomly chosen value from the environment. In some cases the value can be up to 45, what means that to train a Q_table with extra 45 states would require a lot of extra training time.

OPTIONAL 1. *How many states in total exist for the smartcab in this environment?* If we consider that our states have following parameters/combinations we have total **129 states**. 129 is calculated from following values (2 x lights) x (4 x oncoming) x (4 x right) x (4 x left) 2. Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? I would say, that yes. It might take longer time to find global optimum, but in real-life situation it would be necessary to include all the states. 3. Why or why not?

Task 3: 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?

Main difference that I observed is that overall agent predictions and arrives to goal are different after several iterations. At the beginning the initial several iterations are relatively similar, because of the Q_table is at initially empty and actions are randomly selected (the same as when we chose random actions). After a while we can see that the behavior of Q-Learning Driving Agent is starting to act according to waypoint (the smartcab is starting to nicely move towards the goal). So after several iterations, we can clearly see the difference between the random and Q-Learning agent, that in the random action scenario the car arrives to the goal only very seldom and in Q-Learning case it arrives nearly every time (after the Q-table values have been taught).

I have also implemented a slow decay/decrease logic to our exploration rate (epsilon). Current implementation increases the exploration distance by 1 for every 10 successful destination. This implements to the agent the transition between exploration (learning the environment) to exploitation (using the knowledge).

Task 4: 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?

Below is a final Q_table after 50 simulations of 100 iterations with states **waypoint + light**

State	Action	Value
{'light': 'green', 'next_waypoint': 'left'}	'None'	0.13284552719759363
	'forward'	-0.4138769560481279
	'left'	3.6721705432802976
	'right'	-0.21665451109787753
{'light': 'red', 'next_waypoint': None}	'None'	2

	'forward'	2
	'left'	2
	'right'	2
{'light': 'red', 'next_waypoint': 'left'}	'None'	1.496838296197925e-10
	'forward'	-0.16999999999999998
	'left'	-0.575849375
	'right'	-0.18955632421875002
{'light': 'red', 'next_waypoint': None}	'None'	2
	'forward'	2
	'left'	2
	'right'	2
{'light': 'red', 'next_waypoint': 'right'}	'None'	0.2545723197572889
	'forward'	0.6
	'left'	-0.16999999999999998
	'right'	2.2232554786862897
{'light': 'green', 'next_waypoint': 'forward'}	'None'	0.14941584689645235
	'forward'	7.425291203052964
	'left'	-0.2674857421875
	'right'	-0.4499045108327552
{'light': 'green', 'next_waypoint': 'right'}	'None'	0.18975000000000003
	'forward'	-0.20218877438843852
	'left'	-0.3998796987982248
	'right'	2.5999017995259983
{light': 'red', 'next_waypoint': 'forward'}	'None'	6.24207904822194e-68
	'forward'	-0.9430592890624999
	'left'	-0.9465663887097309
	'right'	-0.5145314759644966

Below are some results from different combinations of state:

waypoint + light

- overall_iterations: 693
- overall_simulations: 50
- total_sucess: 48
- **sucess_rate: 96.0 %**

waypoint + light + left + right + oncoming

- overall_iterations: 681
- overall_simulations: 50
- total_sucess: 45
- **sucess_rate: 90.0 %**

waypoint + light + oncoming

- overall_iterations: 649
- overall_simulations: 50

- total_sucess: 47
- **sucess_rate: 94.0 %**

waypoint + light + left + right

- overall_iterations: 737
- overall_simulations: 50
- total_sucess: 46
- **sucess_rate: 92.0 %**

From the above results we can see that the best results of 96%, we have got with the waypoint and light state combination. We can also see from the results where we used all the states, that the success rate was the worst. Most probably the reason is that the the number of iterations was not enough for training the most complex model. On another side we can see that shortest time (number of overall simulations) was achieved with waypoint + light + oncoming combination.

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?

The chosen states waypoint and light gives the best prediction rate but it reaches the destination in an average possible time. Optimal policy would be to have 100% accuracy and use as input all the states. In theory this should be achieved after a lot of iterations.

A very useful control of our trained model is our Q-table, where we can see what actions will our model choose with specific sates combination. By observing the final Q-table we can see that all the actions are properly taught if we consider only the included states (light and next_waypoint).

On other side we can observe during the simulation that the model does not handle well situations where left, right or oncoming states are involved. These are the states that we didn't include into our model.