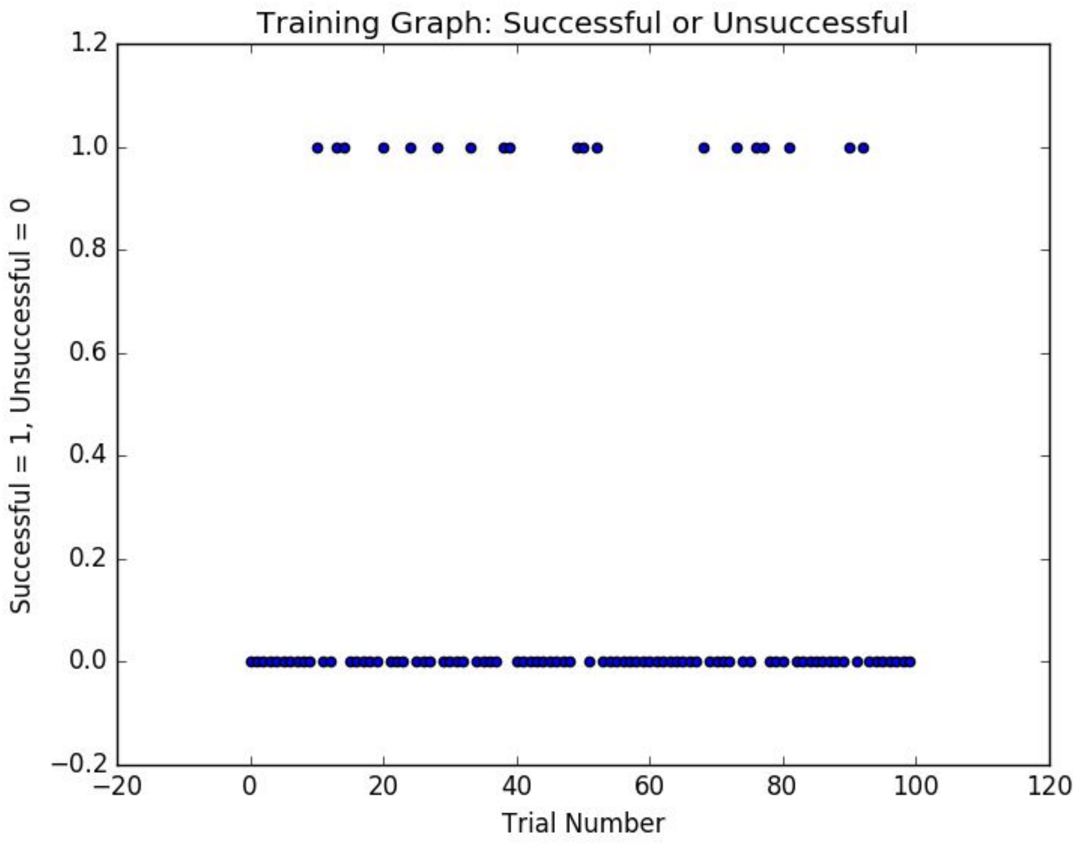
Implement a Basic Driving Agent:

A single change of code is required for action to choose an action randomly through the list of that is valid_actions = [None, 'forward', 'left', 'right'].

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?

Graph of successes and failures:



Observations from the above graph:

If the graph is observed,the car only reached the end with a success rate of 19% taking into consideration the following conditions:

The car was successful within deadline and rewards above 10 and unsuccessful will be as seen in the code in the following circumstances

1. The car reached the destination but was either too late or it clashed with other cars or it made illegal moves
2. When the car is successful it reached the destination 19 times to be precise

Inform the Driving Agent

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?

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?

Answers:
Appropriate States

I make the following assumptions while approaching this problem such as:
1.creating tuples for states since they are hashable
2.State:It means the environment that the smart cab encounters at every intersection which may depict the following:
Status of the Traffic lights i.e. whether the traffic light is green or red.
Status of the Traffic (no of cars blocking the path) at that intersection.
Deadline:The arrival time at the required destination.
The Next Way Point:The direction or path to reach the destination
Let us classify these as **Relevant** and **Irrelevant inputs**

Relevant inputs:
Status of the Traffic lights i.e. whether the traffic light is green or red since we need to obey the rules of the road to reach the destination safely
The Next Way Point:The direction or path to reach the destination,This is relevant too because it concerns where the car should go.
Status of the Traffic (no of cars blocking the path) at that intersection.

Irrelevant inputs:

Deadline:The arrival time at the required destination,which can be ignored since there are no rewards for the deadline but rather for the passenger in the cab to reach the destination safely.

Number of States:

As such, we would have states that factor in next_waypoint (Left, Right and Forward), light (red or green), incoming traffic (None, 'forward', 'left', 'right')This would result in 3 (next_waypoint) x 2 (light) x 4 (incoming traffic) = 24 states,24 states per position seem like a reasonable number given that the goal of Q-Learning is to learn and make informed decisions about each state because we have to understand that we face an exploration-exploitation dilemma here.This is a fundamental trade-off in reinforcement learning.It seems like a good balance of exploration and exploiting 6 states.

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?

Answer:

Parameters Initiated:

Alpha (learning rate), is arbitrarily set at 0.3.Gamma (discount rate), is arbitrarily set at 0.3. Epsilon (randomness probability), is arbitrarily set at 10 such that it is 10%.This is achieved by randomizing the values of p from 0 to 100.and if p < epsilon, the smart cab would take a random action.Q initial values set at 4.We will suffer more initial penalties trying out every action but this will matter more only when scaling up this reinforcement learning problem to include more dummy agents.This technique is referred to as optimistic initialization.

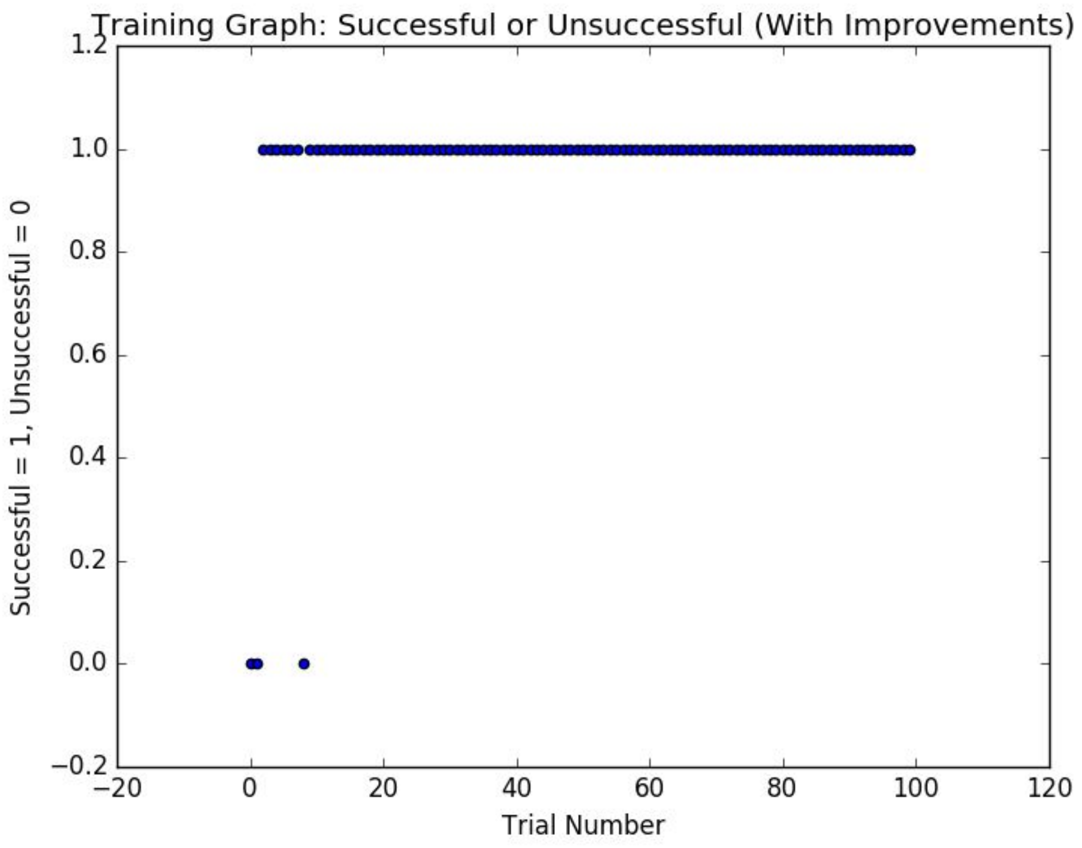
Results of trials:

```
# Format of results
(light, oncoming, next_waypoint)[None, 'forward', 'left', 'right']

('green', 'right', None) ['4.00', '4.00', '4.00', '4.00']
('red', 'right', 'left') ['4.00', '4.00', '4.00', '4.00']
('red', 'forward', None) ['4.00', '4.00', '4.00', '4.00']
('green', 'right', 'left') ['4.00', '4.00', '4.00', '4.00']
('red', None, 'left') ['0.01', '-0.37', '-0.50', '0.57']
('green', None, None) ['4.00', '4.00', '4.00', '4.00']
('green', 'left', 'left') ['3.16', '2.94', '3.40', '4.00']
('red', 'left', 'left') ['3.16', '2.86', '2.86', '4.00']
('red', 'forward', 'right') ['3.16', '2.86', '2.86', '3.64']
('green', 'forward', 'right') ['3.16', '4.00', '4.00', '4.00']
('green', 'forward', None) ['4.00', '4.00', '4.00', '4.00']
('red', 'right', 'forward') ['4.00', '4.00', '4.00', '4.00']
('green', 'forward', 'left') ['3.16', '4.00', '4.00', '4.00']
('green', 'left', None) ['4.00', '4.00', '4.00', '4.00']
('green', 'left', 'forward') ['3.16', '3.64', '2.90', '2.65']
('green', 'left', 'right') ['3.16', '2.97', '4.00', '4.00']
('red', None, None) ['4.00', '4.00', '4.00', '4.00']
('red', 'forward', 'left') ['3.16', '2.86', '2.86', '2.61']
('green', None, 'forward') ['1.66', '7.46', '0.47', '0.40']
('green', 'right', 'right') ['3.16', '4.00', '4.00', '4.00']
('red', 'right', 'right') ['3.16', '4.00', '4.00', '4.00']
('red', 'left', None) ['4.00', '4.00', '4.00', '4.00']
('red', 'left', 'right') ['3.16', '2.06', '2.86', '4.00']
('red', None, 'right') ['1.07', '0.15', '0.63', '2.80']
('red', None, 'forward') ['0.00', '-0.95', '-0.98', '0.40']
('red', 'left', 'forward') ['3.16', '2.86', '2.86', '4.00']
('green', None, 'left') ['1.34', '0.90', '3.41', '1.30']
('green', 'right', 'forward') ['3.16', '3.71', '2.86', '4.00']
('red', 'right', None) ['4.00', '4.00', '4.00', '4.00']
('green', None, 'right') ['1.07', '0.76', '1.65', '3.32']
('green', 'forward', 'forward') ['3.16', '3.88', '2.86', '2.93']
('red', 'forward', 'forward') ['2.50', '2.86', '2.86', '2.18']
```

Inference:

The smart cab reaches the destination more frequently and We've achieved a success rate of 97% with a random assignment of parameters



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?

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?

Answers:

Parameter Tuning

1. With trial and error, it seems the following parameters allow the agent to perform best.
2. Alpha (learning rate): 0.8
3. Gamma (discount rate): 0.2
4. Initial Q = 4
5. Success rate: 98%

Results of Final Trial :

```
# Format of results
(light, next_waypoint)[None, 'forward', 'left', 'right']

('green', 'right', None) ['4.00', '4.00', '4.00', '4.00']
('red', 'right', 'left') ['4.00', '4.00', '4.00', '4.00']
('red', 'forward', None) ['4.00', '4.00', '4.00', '4.00']
('green', 'right', 'left') ['1.44', '4.00', '4.00', '4.00']
('red', None, 'left') ['0.00', '-0.91', '-0.99', '-0.07']
('green', None, None) ['4.00', '4.00', '4.00', '4.00']
('green', 'left', 'left') ['1.44', '4.00', '4.00', '4.00']
('red', 'left', 'left') ['0.93', '0.64', '0.64', '0.14']
('red', 'forward', 'right') ['4.00', '4.00', '4.00', '4.00']
('green', 'forward', 'right') ['4.00', '4.00', '4.00', '4.00']
('green', 'forward', None) ['4.00', '4.00', '4.00', '4.00']
('red', 'right', 'forward') ['4.00', '4.00', '4.00', '4.00']
('green', 'forward', 'left') ['4.00', '4.00', '4.00', '4.00']
('green', 'left', None) ['4.00', '4.00', '4.00', '4.00']
('green', 'left', 'forward') ['1.44', '2.42', '0.80', '0.40']
('green', 'left', 'right') ['1.44', '0.78', '0.83', '4.00']
('red', None, None) ['4.00', '4.00', '4.00', '4.00']
('red', 'forward', 'left') ['0.93', '0.64', '0.64', '4.00']
('green', None, 'forward') ['0.67', '12.39', '0.20', '-0.43']
('green', 'right', 'right') ['4.00', '4.00', '4.00', '4.00']
('red', 'right', 'right') ['4.00', '4.00', '4.00', '4.00']
('red', 'left', None) ['4.00', '4.00', '4.00', '4.00']
('red', 'left', 'right') ['4.00', '4.00', '4.00', '4.00']
('red', None, 'right') ['0.73', '-0.56', '-0.56', '2.09']
('red', None, 'forward') ['0.00', '-1.00', '-1.00', '-0.04']
('red', 'left', 'forward') ['4.00', '4.00', '4.00', '4.00']
('green', None, 'left') ['0.77', '-0.19', '3.72', '0.24']
('green', 'right', 'forward') ['1.44', '2.40', '0.64', '4.00']
('red', 'right', None) ['4.00', '4.00', '4.00', '4.00']
('green', None, 'right') ['1.44', '0.12', '1.04', '2.10']
('green', 'forward', 'forward') ['1.44', '11.04', '4.00', '4.00']
('red', 'forward', 'forward') ['0.19', '-0.57', '-0.57', '0.41']
```

Results of Prior Trials :

```
Simulator.run(): Trial 95
Environment.reset(): Trial set up with start = (6, 4), destination = (2, 1), deadline = 35
RoutePlanner.route_to(): destination = (2, 1)
LearningAgent.update(): deadline = 35, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = left, reward = 2.0
LearningAgent.update(): deadline = 34, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 33, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 32, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 31, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 30, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 29, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 28, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 27, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = forward, reward = -1.0
LearningAgent.update(): deadline = 26, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 25, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 24, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = left, reward = -1.0
Environment.act(): Primary agent has reached destination!


Simulator.run(): Trial 97
Environment.reset(): Trial set up with start = (4, 5), destination = (1, 1), deadline = 35
RoutePlanner.route_to(): destination = (1, 1)
LearningAgent.update(): deadline = 35, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': 'left'}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 34, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 33, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 32, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 31, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 30, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 29, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 28, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 27, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 26, inputs = {'light': 'red', 'oncoming': 'forward', 'right':
None, 'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 25, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 24, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 23, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
Environment.act(): Primary agent has reached destination!

Simulator.run(): Trial 98
Environment.reset(): Trial set up with start = (4, 6), destination = (8, 1), deadline = 45
RoutePlanner.route_to(): destination = (8, 1)
LearningAgent.update(): deadline = 45, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 44, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 43, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 42, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = forward, reward = -1.0
LearningAgent.update(): deadline = 41, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 40, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 39, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 38, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = left, reward = 2.0
LearningAgent.update(): deadline = 37, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 36, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 35, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 34, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 33, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
```

```
LearningAgent.update(): deadline = 32, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 31, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 30, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 29, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 28, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 27, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 26, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 25, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
Environment.act(): Primary agent has reached destination!


Simulator.run(): Trial 99
Environment.reset(): Trial set up with start = (3, 2), destination = (4, 5), deadline = 20
RoutePlanner.route_to(): destination = (4, 5)
LearningAgent.update(): deadline = 20, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 19, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = right, reward = 2.0
LearningAgent.update(): deadline = 18, inputs = {'light': 'green', 'oncoming': None, 'right':
None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update(): deadline = 17, inputs = {'light': 'red', 'oncoming': None, 'right': None,
'left': None}, action = None, reward = 0.0
Environment.act(): Primary agent has reached destination!
```

Optimal Policy

The agent does reach to the final absorbing states in the minimum possible time while incurring minimum penalties.

The optimal policy would be:
Minimum possible time.
Obey all traffic rules.
No clashes with other cars.

In this project, there are two issues resulting from the rewards' system. It benefits the agent to do the following :
Disobey traffic when the time is running out to reach the destination.
This can be seen from the trials' summary above where when there is a red light, the car might still make a move instead of remaining stationary.which would cause accidents and this is undesirable in the real world.
Going in circles when there is a lot of time left to have more rewards before reaching the absorbing state.
This can be observed how the cab takes a particular long time initially when there is a lot of time. Thereby explaining the high number of moves in the trials' summary above.This would inturn cause the smart cab to increase the cab fare in the real world and also delay the trip for the passenger, both of which are undesirable in the real world.