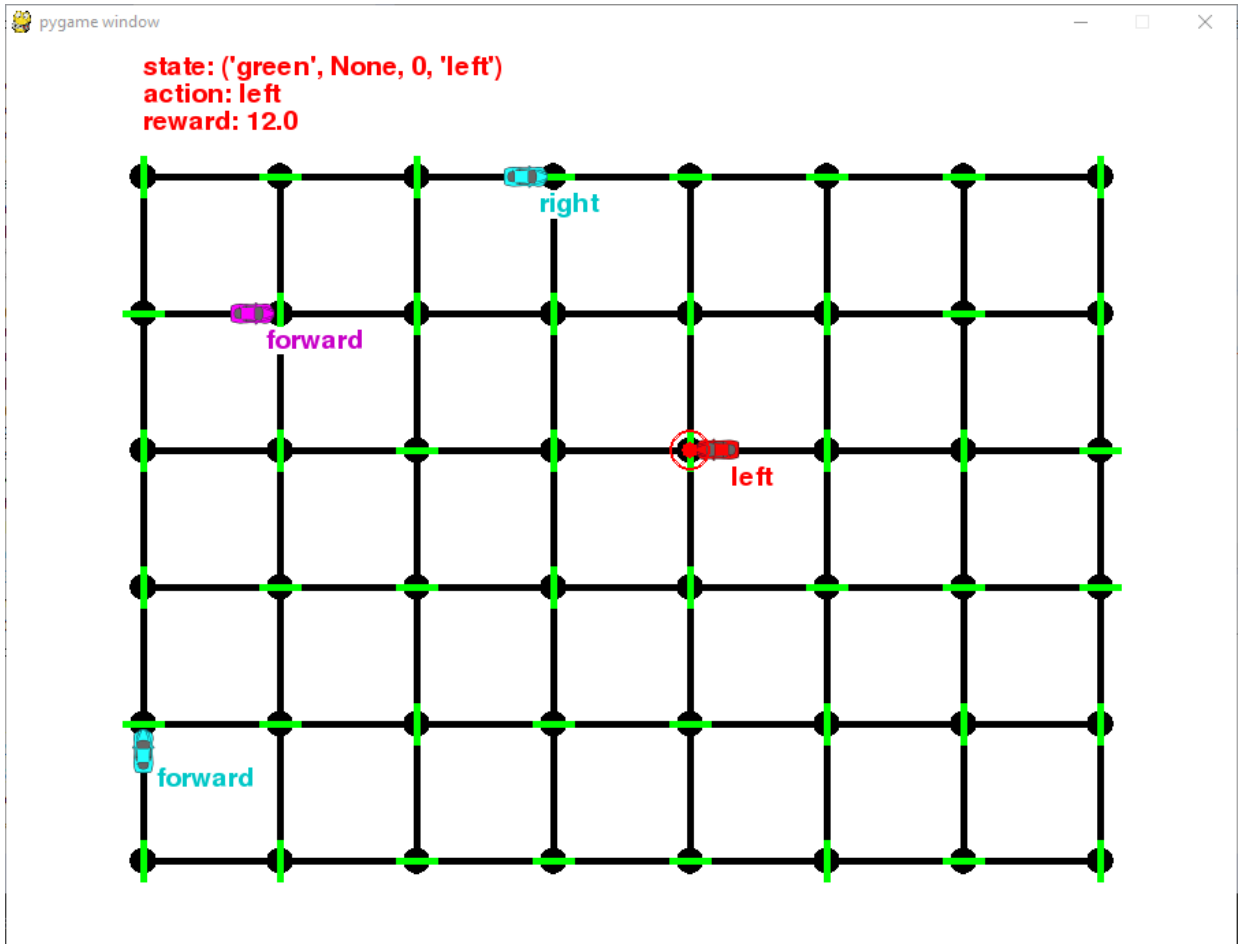
MLND P4: Train a Smartcab to Drive Project Report

23 Sept 2016



Implement a Basic Driving Agent

*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?*

ANSWER: The agent had eventually arrived at its destination, but it the time taken to reach the destination is completely random. The number of iterations it takes to reach its destination varies after every reset.

The agent does not account for reaching its destination before the allotted deadline, and does not take the fastest nor the safest route. It completely disregards its route planner suggestions (`next_waypoint`), does not pay heed to any traffic rules (traffic light rules, right-of-way) and is insensitive to the rewards for any action it took.

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?*

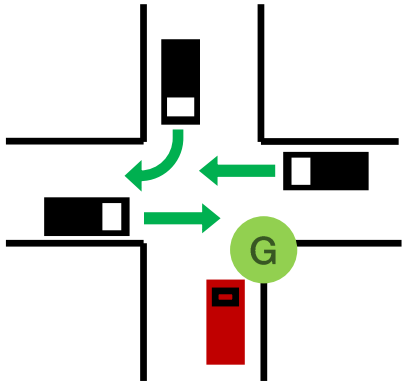
ANSWER: The identified inputs for `self.state` are:

```
light = ['red', 'green']
oncoming = [None, 'forward', 'left', 'right']
left = [None, 'forward', 'left', 'right']
next_waypoint = ['forward', 'left', 'right']
```

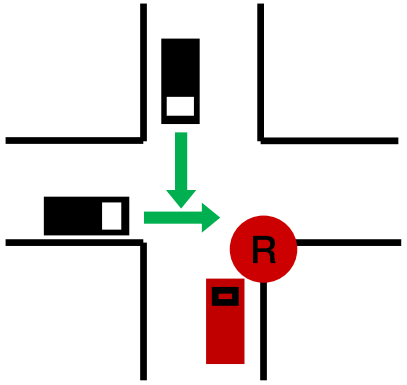
In which the states will be a combination of any element in the 4 input variables. `light` will represent the fundamental modelling of traffic lights in an intersection. This is combined with `oncoming` and `left` to model the U.S. Right-of-Way rules, where in general, higher priority of oncoming and straight-through traffic is given. The illustration below details the reasoning. As for `next_waypoint`, since the agent only has an egocentric view of the environment, including this in the state will help to guide the agent toward its destination.

Inform the Driving Agent

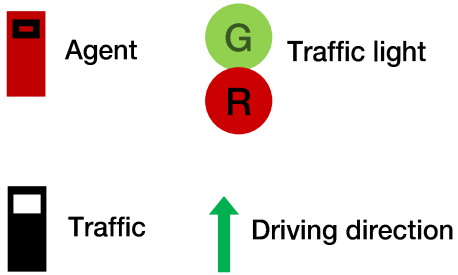
1) Agent is NOT permissible to make a left turn



2) Agent is NOT permissible to make a right turn.
Yield to oncoming traffic when turning left.



Legend:



From the illustration of the two scenarios above, the agent will need to sense the presence and directions of `oncoming`, `left`, and `right` traffic including the traffic light (`light`) in order to learn rule-abiding actions. However, to allow for faster agent learning, the state space can be simplified by removing the `right` input, as it is only applicable in one of two scenarios we would want to model. `oncoming` and `left` play a greater role for the learning usefulness for the agent.

Inform the Driving Agent

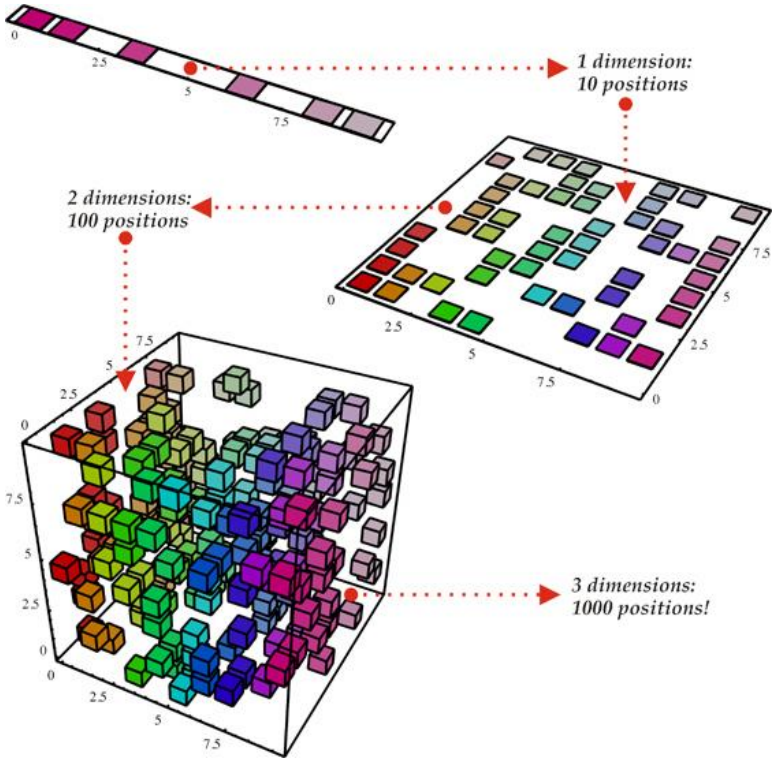
*OPTIONAL: How many states in total exist for the **smartcab** in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

ANSWER: Based on the `self.state` inputs defined earlier, we have 2 inputs for `light`, 4 for `oncoming`, 4 for `left` and 3 for `next_waypoint`, giving us the total number of states, 2 x 4 x 4 x 3 = 96.

`right` could have been included into `self.state`, however, this would increase the state space to 384. With higher dimensions, the number of iterations needed will also increase greatly in order for the agent to visit each state for effective Q-Learning. Given that the deadline constraints of 100 trials and about 20-40 steps per trial, this gives us about 2000-4000 possible steps that the agent will make when learning, and hence it will optimistically visit each state about $2000/384 \approx 5$ to $4000/384 \approx 10$ times. Contrast that with a state space of 96, this gives us $2000/96 \approx 20$ to $4000/96 \approx 40$ times. Increasing the chances of the agent visiting a particular state will improve its Q-Learning as it learns better from more past actions taken.

Inform the Driving Agent

ANSWER Con't: Hence, the state space is a tradeoff between the most accurate representation for Q-Learning (by modelling the environment as best as we can with higher dimensions) versus speed and effective Q-Learning (getting the agent to reach its destination within the stipulated deadline) by only modelling the rules which the agent will most likely encounter during its trials.



Implement a Q-Learning Driving Agent

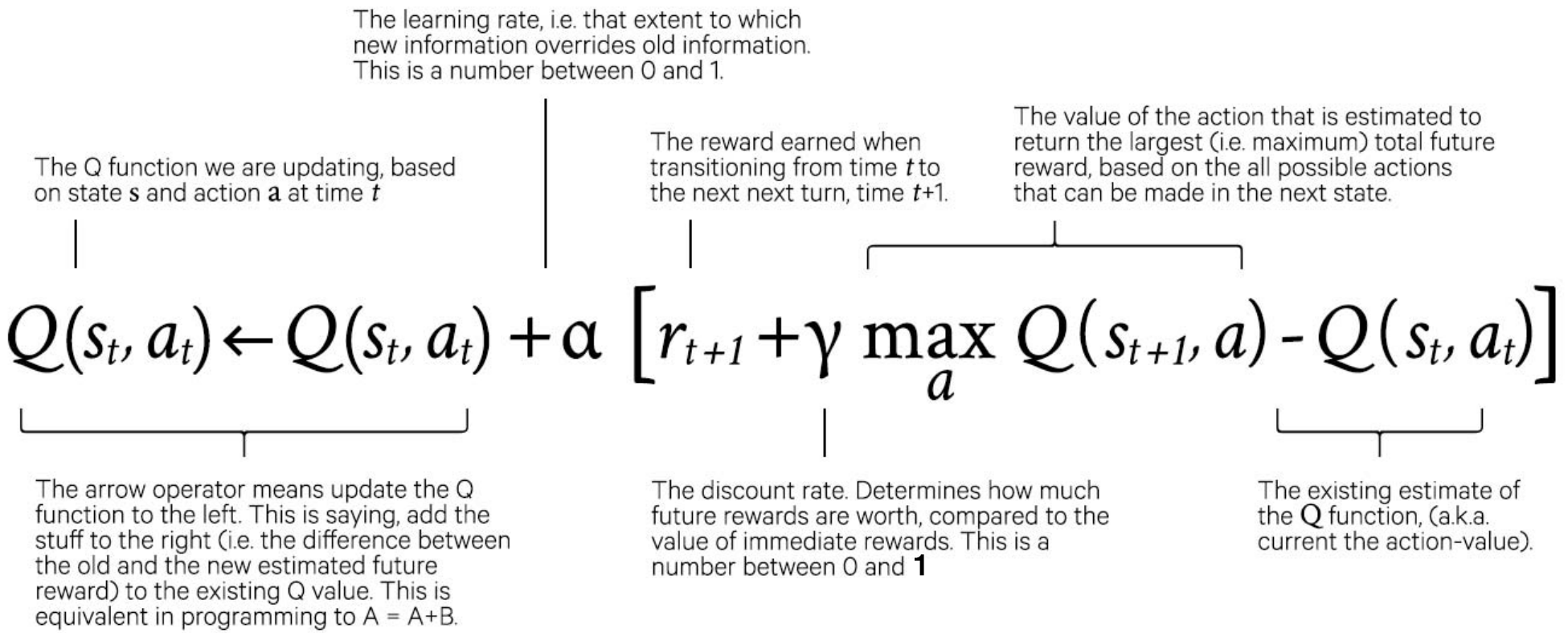
QUESTION: What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

ANSWER: The agent progressively gets better at reaching its destination within the deadline, and progressively reacts appropriately to traffic rules. It seems to be ‘learning’ from its past mistakes from exploring the environment with various states and actions.

Previously, the agent does not take the consequences of its actions into account when taking random actions. With Q-Learning, the agent senses its environment for inputs, takes an action, and receives a reward or penalty for that particular action. It is rewarded for obeying traffic rules and reaching its destination and is penalized otherwise. This information is logged into a Q-table any time an agent takes and action from a particular state. If the agent encounters the same state repeatedly enough, the particular action’s value (Q-value) will be updated (increase or decrease) based on how much accumulated rewards/penalties. The agent will pick the best action for that state which maximizes the rewards (max Q-value) it receives. Thus, over multiple iterations, the agent learns to choose high reward actions over high penalty ones, that lead toward its end goal (its destination).

Implement a Q-Learning Driving Agent

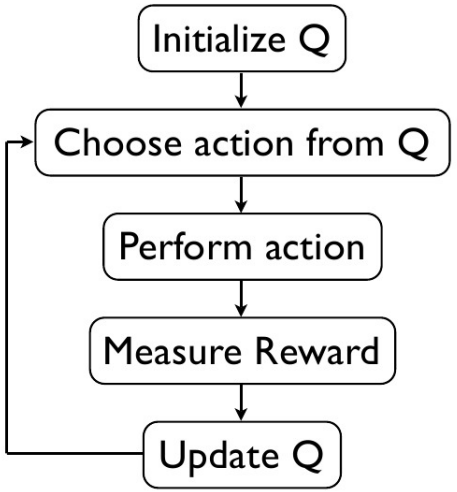
The Q-Learning function



Implement a Q-Learning Driving Agent

The Q-Learning algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until s is terminal



Implement a Q-Learning Driving Agent

The Epsilon-Greedy Strategy

ε-Greedy Action Selection

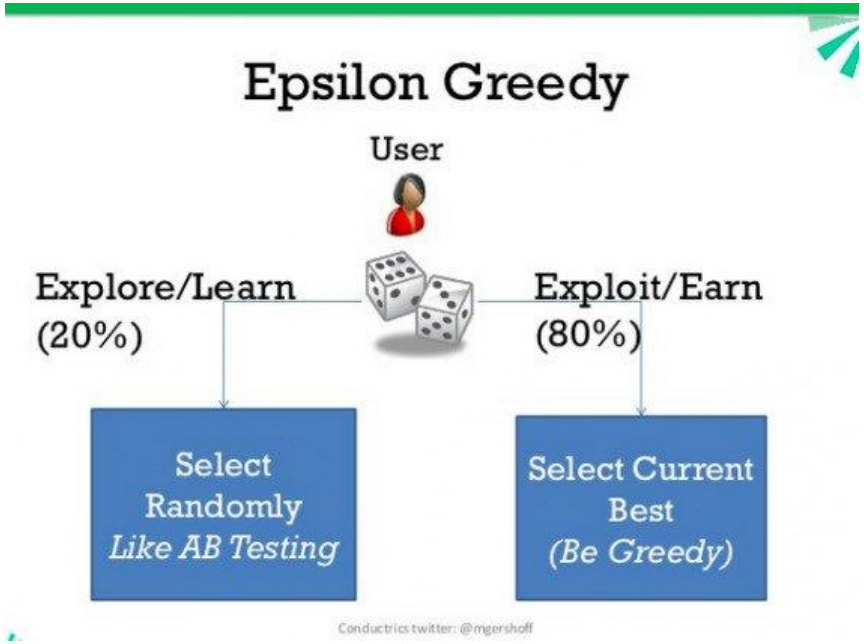
- Greedy action selection:

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- ε-Greedy:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\varepsilon \\ \text{random action with probability} & \varepsilon \end{cases}$$

. . . the simplest way to try to balance exploration and exploitation



The Epsilon-Greedy Strategy is useful for balancing Exploring/Learning and Exploiting/Earning. In the beginning of the trial, the agent has not learnt anything from its environment, hence, an epsilon value favoring Exploring/Learning is desirable, as the agent takes random actions from time to time to gauge the rewards it will receive. Thus, the agent would have explored more state options in this manner, building a robust experience for Q-Learning. As the trial continues and the agent has learnt enough states to get itself around effectively to its destination, a lower epsilon value is desirable, where the agent will favor Exploiting/Earning as much rewards as possible (hence improving overall success rate). Epsilon has to be decayed throughout the trials for best results.

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?

ANSWER: The metric defined to measure agent success is as below:

$$Results = \frac{Number\ of\ times\ agent\ reaches\ destination\ within\ deadline}{Total\ number\ of\ trials}$$

Where the range is from 0 – 100. This is averaged over 5 runs with the same parameter set to give the Avg Results.

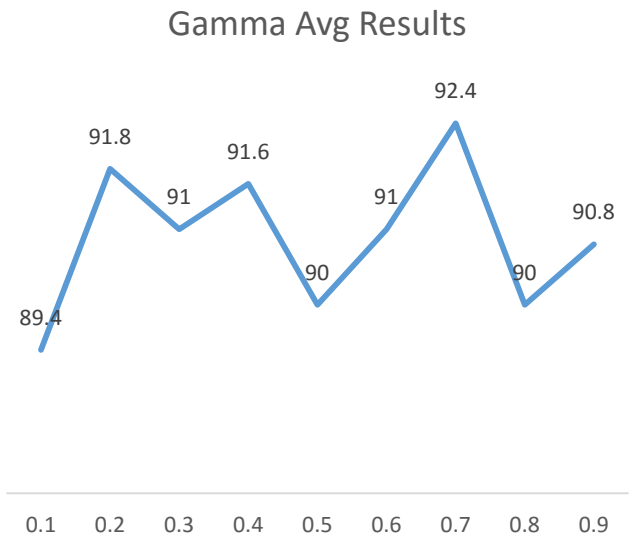
Improve the Q-Learning Driving Agent

ANSWER Con't: A sample of the report is as below (full results in Excel file smartcab_results.xlsx):

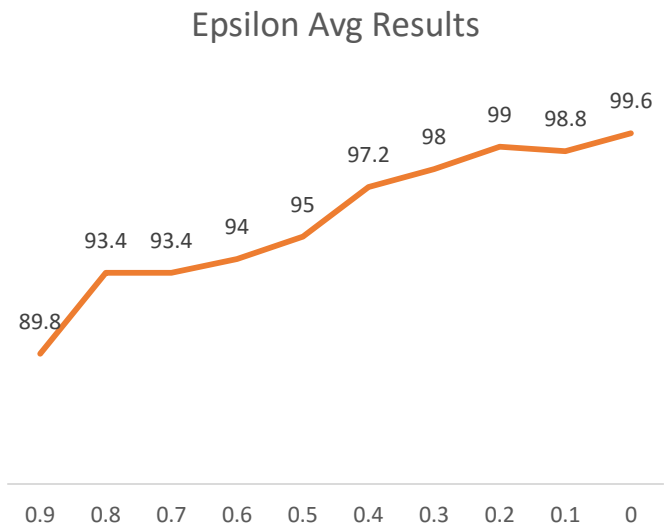
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	run_num	test	epsilon	alpha	gamma	Initial Q	alpha decay	epsilon decay	reset mechanism	decay mechanism	inputs	results 1	results 2	results 3	results 4	results 5	Avg results
2	1	a	0.9	0.9	0.1	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	91	93	90	86	87	89.4
3	2	a	0.9	0.9	0.2	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	93	94	91	90	91	91.8
4	3	a	0.9	0.9	0.3	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	89	89	90	92	95	91
5	4	a	0.9	0.9	0.4	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	95	89	91	89	94	91.6
6	5	a	0.9	0.9	0.5	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	86	93	93	91	87	90
7	6	a	0.9	0.9	0.6	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	92	92	92	90	89	91
8	7	a	0.9	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	90	93	93	95	91	92.4
9	8	a	0.9	0.9	0.8	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	90	90	91	89	90	90
10	9	a	0.9	0.9	0.9	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	91	92	90	91	90	90.8
11	10	a	0.9	0.9	1	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	92	85	92	89	91	89.8
12	11	b	0.9	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, right, next_waypoint)	92	87	93	95	91	91.6
13	12	b	0.9	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, right[0,1], next_waypoint)	86	89	90	92	90	89.4
14	13	b	0.9	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left , next_waypoint)	91	89	95	89	91	91
15	14	c	0.8	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	97	95	94	92	89	93.4
16	15	c	0.7	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	94	91	93	92	97	93.4
17	16	c	0.6	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	96	96	91	92	95	94
18	17	c	0.5	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	93	93	95	95	99	95
19	18	c	0.4	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	97	96	98	98	97	97.2
20	19	c	0.3	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	97	99	98	99	97	98
21	20	c	0.2	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99	100	98	99	99	99
22	21	c	0.1	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99	98	100	98	99	98.8
23	22	c	0	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	100	100	100	99	99	99.6
24	23	d	0	1	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	100	100	100	99	99	99.6
25	22	c	0	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	100	100	100	99	99	99.6
26	24	d	0	0.8	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99	98	99	100	99	99
27	25	d	0	0.7	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99	100	99	99	99	99.2
28	26	d	0	0.6	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99	100	99	100	99	99.4
29	27	d	0	0.5	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)						99.2
30	28	d	0	0.4	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)						99
31	29	d	0	0.3	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)						99.4
32	30	d	0	0.2	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)						99.4
33	31	d	0	0.1	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)						99.6
34	32	d	0	0	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)						21.4

Improve the Q-Learning Driving Agent

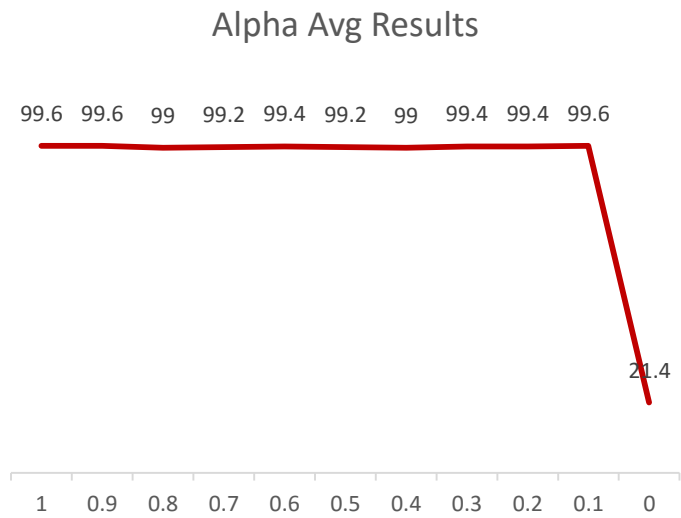
ANSWER Con't: Tuning for the parameters gamma, alpha and epsilon has been done as illustrated below. Note that the values tested for alpha and epsilon are their *initial* values, as they decay after every trial is completed (0.9 for alpha, 0.95 for epsilon). Gamma values are not decayed at all.



Best gamma is found to be 0.7 (92.4 Avg Result). This means the agent performs best when there is a high importance placed on future rewards versus immediate rewards.



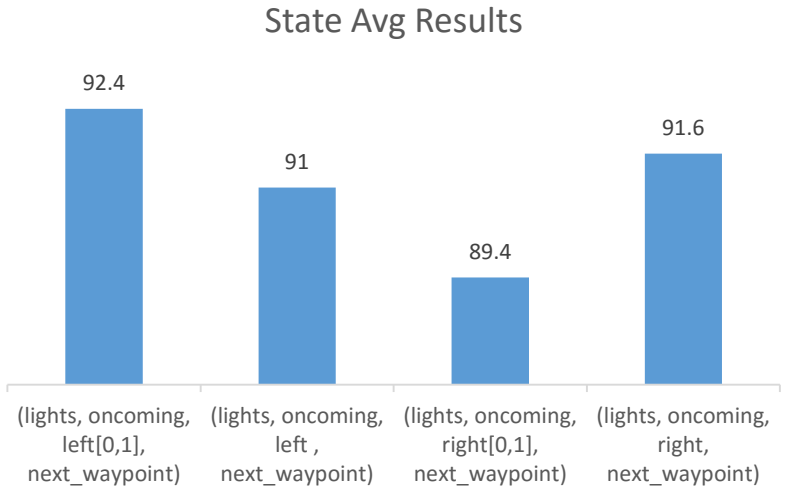
Best initial epsilon value is found to be 0 (99.6). This is rather interesting as a small epsilon value that decays over time is often desirable for the agent to learn better. However, note that the agent is already taking random actions (exploring) at the first 50 steps or so as the Q-table values are all equal at the start, before it gets updated. It could be that the agent has sufficiently learnt the fundamental rules with these 50 iterations in order for it to reach its goal.



Best initial alpha value is among 0.1, 0.9 and 1 (99.6). This is interesting as we get comparable results for a very slow learning and very fast learning agent, but generally poorer results in between. Note that alpha = 0 shows the worst results (21.4). This shows that when there is no update to Q-values, no learning is achieved. Note also that decay is relatively slow (once after every trial, and at x0.9 rate), and thus the agent would have learnt sufficiently within its 1st or 2nd trial, hence the low impact of alpha on the agent's performance.

Improve the Q-Learning Driving Agent

ANSWER Con't: An exploration is also done for varying the state space. Here, the inputs `left` and `right` are compared against, and a further reduction to the state space is performed by defining `left` and `right` in only two options ('forward' == 1 and 0 for other directions) instead of four. Thus, the state space is further reduced from 96 to 48.



The bar graph shows that when using the full space for `left` and `right`, `right` interestingly provides the agent a slightly better performance (91.6). This is rather contradictory to the earlier deduction of preferring `left` in the state space as `left` play a larger usefulness role in modelling the right-of-way rules.

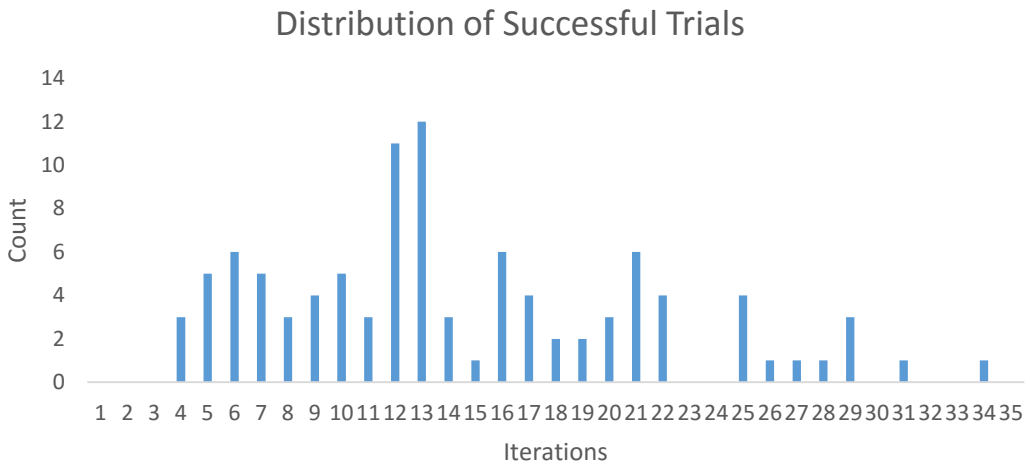
However, when the **reduced** `left` and `right` space is included, `left` performs much better than all other explorations (92.4). Now, this approach seems to conform to the right-of-way rules, and the simplification of the inputs seems to allow for better learning.

Considering the experimentation, the best parameters are `epsilon = 0`, `alpha = 1.0`, `gamma = 0.7`, with `self.state` expressed as `(lights, oncoming, left[0, 1], next_waypoint)`. This gives us a driving agent which performs on average of 99.6/100 trials (i.e. the agent reaches its target **99.6%** of the time within the deadline.)

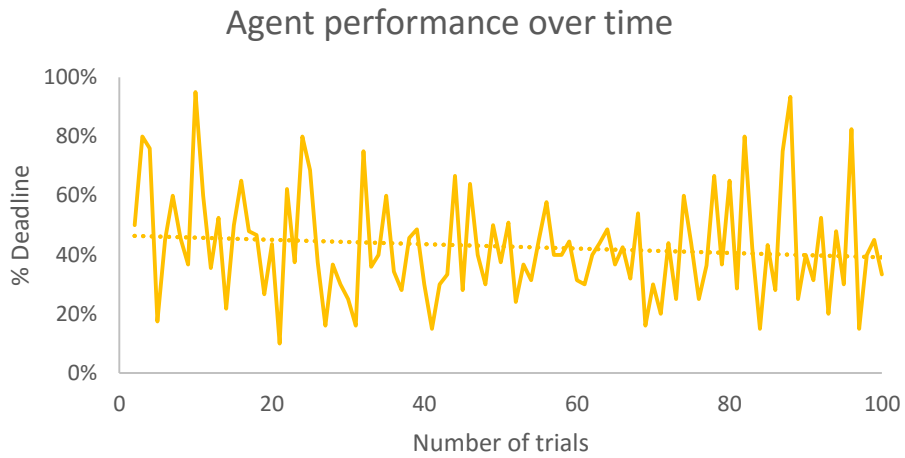
	A	B	C	D	E	F	G	H	I	J	K	Q
1	run_nu	test	epsil	alp	gamn	Initial	alpha dec	epsilon dec	reset mechanis	decay mechanis	inputs	Avg resu
23	22	c	0	0.9	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99.6
24	23	d	0	1	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99.6
33	31	d	0	0.1	0.7	2	0.9	0.95	none	after every trial	(lights, oncoming, left[0,1], next_waypoint)	99.6

Improve the Q-Learning Driving Agent

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 histogram above shows the most frequent Iteration number for each agent success. (i.e. in which Iteration does the agent successfully reached its target?) The mean = 14.5 and median = 13. Generally the agent reached its target well before the shortest deadline, 20.



Here, the % Deadline metric is used to gauge agent successes over time. % Deadline means how many percent of the deadline remained when the agent had reached its destination. Lower values mean better: if the agent is able to reach its target with only 10% into the deadline timer, the agent is performing really well. From the graph, it can be noted that the variance is high initially, then proceeds to converge to a pinch at the ~65th trial, then diverges again until the 100th trial. Upon inspection of the rewards data (rewards_data.txt), the alpha value had decayed close to 0 at the inflection point. This means, beyond the 65th trial, the agent is no longer updating the Q-table and learning has stopped.

*Results from one run of best parameters.

Improve the Q-Learning Driving Agent

Also, it is worth nothing that beyond the 65th trial, the agent tends to avoid choosing actions which results in a penalty, and hence prefers to not take an action (None) and receive 0 rewards.

rewards_data.txt

```
1436  Iter6   2.95126654307e-05   0.0 15  ('red', None, 0, 'forward') None    0.0
1437  Iter7   2.95126654307e-05   0.0 14  ('red', None, 0, 'forward') None    0.0
1438  Iter8   2.95126654307e-05   0.0 13  ('red', None, 0, 'forward') None    0.0
1439  Iter9   2.95126654307e-05   0.0 12  ('green', None, 0, 'forward')   forward 2.0
1440  Iter10  2.95126654307e-05   0.0 11  ('green', None, 0, 'forward')   forward 12.0
1441  Iter1   2.65613988876e-05   0.0 45  ('green', None, 0, 'right') right   2.0
1442  Iter2   2.65613988876e-05   0.0 44  ('green', None, 0, 'forward')   forward 2.0
1443  Iter3   2.65613988876e-05   0.0 43  ('red', None, 0, 'forward') None    0.0
1444  Iter4   2.65613988876e-05   0.0 42  ('red', None, 0, 'forward') None    0.0
1445  Iter5   2.65613988876e-05   0.0 41  ('red', None, 0, 'forward') None    0.0
1446  Iter6   2.65613988876e-05   0.0 40  ('green', None, 0, 'forward')   forward 2.0
1447  Iter7   2.65613988876e-05   0.0 39  ('green', None, 0, 'forward')   forward 2.0
1448  Iter8   2.65613988876e-05   0.0 38  ('red', None, 0, 'forward') None    0.0
1449  Iter9   2.65613988876e-05   0.0 37  ('green', None, 0, 'forward')   forward 2.0
1450  Iter10  2.65613988876e-05   0.0 36  ('green', None, 0, 'forward')   forward 2.0
1451  Iter11  2.65613988876e-05   0.0 35  ('green', None, 0, 'right') right   2.0
1452  Iter12  2.65613988876e-05   0.0 34  ('red', None, 0, 'forward') None    0.0
1453  Iter13  2.65613988876e-05   0.0 33  ('green', None, 0, 'forward')   forward 2.0
1454  Iter14  2.65613988876e-05   0.0 32  ('red', None, 0, 'forward') None    0.0
1455  Iter15  2.65613988876e-05   0.0 31  ('red', None, 0, 'forward') None    0.0
1456  Iter16  2.65613988876e-05   0.0 30  ('green', None, 0, 'forward')   forward 12.0
```

The optimal policy would have a few key attributes:

1) Maximizes the number of times the agent reaches its destination in 100 trials within the deadline. (Most hits)
2) Minimizes the time taken by agent to reach its destination as a percentage of the allotted deadline. (Least steps)
3) Minimizes the penalties received by the agent. (Most safe)