Smartcab P-4 Report by Andrey Shkabko

Implement a Basic Driving Agent

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?*

Smatrcab eventually makes to destination even with a random choice of action. If agent reaches destination it gets 10 rewards, 0 if agent stays without moving, -1 if the Action is invalid.

Inform the Driving Agent

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?*
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?*

At each position the states considered to be the most important are: light (red, green), presence of oncoming vehicles, vehicles from left (None, left, right, forward) and 4 actions to the next waypoint (None, left, right, forward). Each of these states are important because otherwise the Agent will not be able to follow the rules(high bias) on the intersections as well as not able to learn proper actions from the knowledge of rewards for the possible next waypoints. All in all 2*4*4*4*4=512 states

Implement a Q-Learning Driving Agent

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?

With each trial the Agent learns by updating the Q matrix and following the maximum Q update strategy on each local waypoint. However, policy agent strategy should still should be implemented.

Improve the Q-Learning Driving Agent

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?

If I take the score metric as amount of trials the Agent reaches the final destination within the time, than it changes from 0% for the first trials and reaches about 99% for the trials starting from 100 for the best parameters.

Please note that #Trials is amount of trials calculated each time for a new experiment.

#Trials	Discount	Learning	Penalty_per_trial	Succesfull trials
1	0.05	0.5	9.000000	1
2	0.05	0.5	5.500000	2
4	0.05	0.5	2.500000	3
8	0.05	0.5	2.000000	8
16	0.05	0.5	0.875000	16
32	0.05	0.5	0.531250	31
64	0.05	0.5	0.250000	64
128	0.05	0.5	0.320312	127
256	0.05	0.5	0.160156	255
512	0.05	0.5	0.095703	510
1024	0.05	0.5	0.064453	1024
2048	0.05	0.5	0.037109	2041
4096	0.05	0.5	0.020020	4089

#Trials	Discount	Learning	Penalty_per_trial	Succesfull trials
1	0.5	0.5	10.000000	1
2	0.5	0.5	5.000000	1
4	0.5	0.5	2.750000	3
8	0.5	0.5	4.875000	6
16	0.5	0.5	4.437500	12
32	0.5	0.5	5.125000	27
64	0.5	0.5	5.312500	54
128	0.5	0.5	0.710938	127
256	0.5	0.5	1.332031	246
512	0.5	0.5	0.585938	503
1024	0.5	0.5	0.328125	1017
2048	0.5	0.5	0.165039	2031
4096	0.5	0.5	0.201172	4073

#Trials	Discount	Learning	Penalty_per_trial	Succesfull trials
1	0.05	0.9	6.000000	1
2	0.05	0.9	5.000000	2
4	0.05	0.9	4.000000	3
8	0.05	0.9	1.625000	8
16	0.05	0.9	0.937500	15
32	0.05	0.9	0.375000	31
64	0.05	0.9	0.390625	63
128	0.05	0.9	0.281250	128
256	0.05	0.9	0.191406	255
512	0.05	0.9	0.121094	511
1024	0.05	0.9	0.058594	1023
2048	0.05	0.9	0.037598	2043
4096	0.05	0.9	0.022949	4088

#Trials	Discount	Learning	Penalty_per_trial	Succesfull trials
1	0.5	0.9	10.000000	1
2	0.5	0.9	3.500000	2
4	0.5	0.9	5.000000	3
8	0.5	0.9	1.500000	8
16	0.5	0.9	0.937500	15
32	0.5	0.9	0.906250	31
64	0.5	0.9	0.500000	62
128	0.5	0.9	0.406250	124
256	0.5	0.9	0.312500	252
512	0.5	0.9	0.294922	505
1024	0.5	0.9	0.205078	1019
2048	0.5	0.9	0.088379	2039
4096	0.5	0.9	0.057373	4083

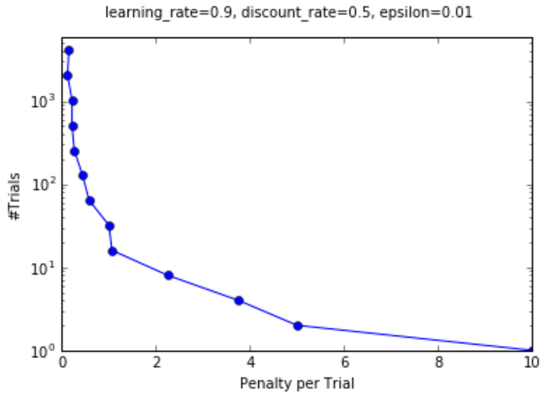
Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties? How would you describe an optimal policy for this problem?

I used 2 parameters(+epsilon=0.01) to roughly analyse the performance of the Agent. An important question is how good is the Agent with time. One of the metrics used to analize is how good Agent does in a last 10 trials out of 100. Another one used in this report is to analize the penalties (times the Agent receives negative rewards) over time. I used to increase amount of trials up to 4096 for each set of parameters: learning_rate and discount_rate. As can be seen from the graph the best parameters are with low discount rate and higher learning rate.

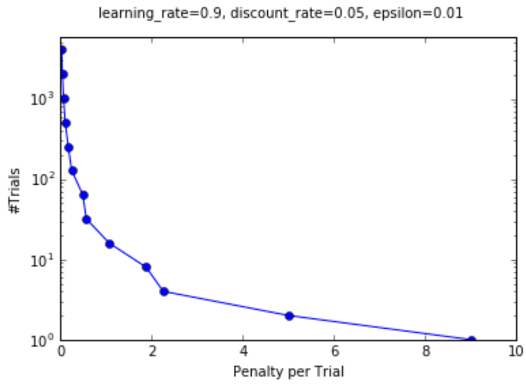
For choosing the correct policy which is in my case value iteration + greedy exploration/explotation the amount of penalties are significantly decreased if I use special for of epsilon/exp(t) function which allow Agent to explore less and less as time passes. This lets the Agent to explore more at the beginning of the trial and less afterwards.

The penalty per trial should approach 0 as the Agent learns more and more. Some of the events have very low probability over 100 trials like when cars are encounter at the intersection. Such events require more trials to update Q matrix and follow right strategy.

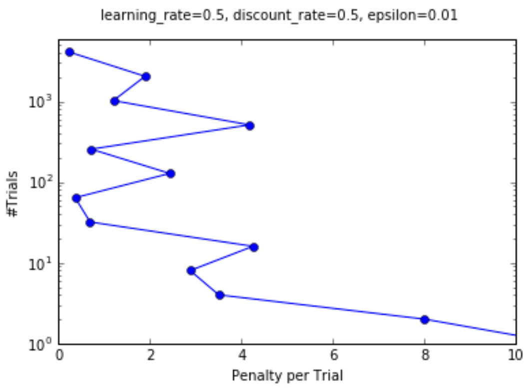
	#Trials	Discount	Learning	Penalty_per_trial
0	1	0.5	0.9	10.000000
1	2	0.5	0.9	5.000000
2	4	0.5	0.9	3.750000
3	8	0.5	0.9	2.250000
4	16	0.5	0.9	1.062500
5	32	0.5	0.9	1.000000
6	64	0.5	0.9	0.578125
7	128	0.5	0.9	0.437500
8	256	0.5	0.9	0.261719
9	512	0.5	0.9	0.212891
10	1024	0.5	0.9	0.214844
11	2048	0.5	0.9	0.118164
12	4096	0.5	0.9	0.146973



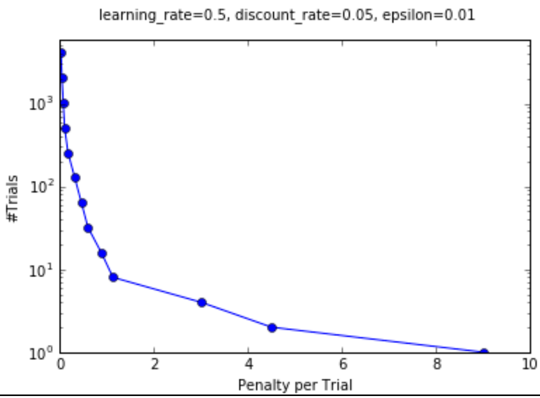
	#Trials	Discount	Learning	Penalty_per_trial
0	1	0.05	0.9	9.000000
1	2	0.05	0.9	5.000000
2	4	0.05	0.9	2.250000
3	8	0.05	0.9	1.875000
4	16	0.05	0.9	1.062500
5	32	0.05	0.9	0.562500
6	64	0.05	0.9	0.484375
7	128	0.05	0.9	0.242188
8	256	0.05	0.9	0.164062
9	512	0.05	0.9	0.099609
10	1024	0.05	0.9	0.062500
11	2048	0.05	0.9	0.037598
12	4096	0.05	0.9	0.020996



	#Trials	Discount	Learning	Penalty_per_trial
0	1	0.5	0.5	11.000000
1	2	0.5	0.5	8.000000
2	4	0.5	0.5	3.500000
3	8	0.5	0.5	2.875000
4	16	0.5	0.5	4.250000
5	32	0.5	0.5	0.687500
6	64	0.5	0.5	0.359375
7	128	0.5	0.5	2.429688
8	256	0.5	0.5	0.710938
9	512	0.5	0.5	4.173828
10	1024	0.5	0.5	1.196289
11	2048	0.5	0.5	1.888672
12	4096	0.5	0.5	0.228760



	#Trials	Discount	Learning	Penalty_per_trial
0	1	0.05	0.5	9.000000
1	2	0.05	0.5	4.500000
2	4	0.05	0.5	3.000000
3	8	0.05	0.5	1.125000
4	16	0.05	0.5	0.875000
5	32	0.05	0.5	0.593750
6	64	0.05	0.5	0.453125
7	128	0.05	0.5	0.312500
8	256	0.05	0.5	0.167969
9	512	0.05	0.5	0.097656
10	1024	0.05	0.5	0.067383
11	2048	0.05	0.5	0.040527
12	4096	0.05	0.5	0.020996



General note: Time to calculate: about 3mins on a PC/Mac.

Code modifications in the environment.py

2 parameters are added inti Environment.__init__ learning_rate=None, discount_rate=None line 34

self.learning_rate = learning_rate lines 44,45
self.discount_rate = discount_rate
state.sucess = 0

and success counter is added self.sucess += 1 for state['location'] == state['destination'] line 222

NOTES:

Idea: effectively reach new destinations in the allotted time

1. Investigate the environment the agent operates in by constructing a very basic driving implementation
2. identify each possible state the agent can be in when considering such things as traffic lights and oncoming traffic at each intersection
3. implement a Q-Learning algorithm for the self-driving agent to guide the agent towards its destination within the allotted time
4. improve upon the Q-Learning algorithm to find the best configuration of learning and exploration factors to ensure the self-driving agent is reaching its destinations with consistently positive results

The smartcab operates in an ideal, grid-like city (similar to New York City), with roads going in the North-South and East-West directions.

- On a green light, a left turn is permitted if there is no oncoming traffic making a right turn or coming straight through the intersection.
- On a red light, a right turn is permitted if no oncoming traffic is approaching from your left through the intersection.
- The route is split at each intersection into waypoints, and its assumed that the smartcab, at any instant, is at some intersection
- Smartcab **can determine** the state of the *traffic light* for its direction of movement, and whether there is a *vehicle at the intersection* for each of the oncoming directions
- For each action, the smartcab may either idle at the intersection, or drive to the next intersection to the left, right, or ahead of it
- Each trip has a time to reach the destination which decreases for each action taken

Task 1. Get the **smartcab** to move around in the environment *without* optimal driving policy

The driving agent is given the following information at each intersection (set of possible actions (None, 'forward', 'left', 'right') at each intersection:

- The next waypoint location relative to its current location and heading.
- The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions.

- The current time left from the allotted deadline.

Some of the Evironment class variables
valid_headings = [(1, 0), (0, -1), (-1, 0), (0, 1)] # ENWS
Basically the heading car is in (1,0) – east
(0,-1) – north
(-1,0) – west
(0,1) – south
position.

valid_actions = [None, 'forward', 'left', 'right']
Action for the car to go.

valid_inputs = {'light': TrafficLight.valid_states, 'oncoming': valid_actions, 'left': valid_actions, 'right': valid_actions}
The inputs at each position. Note: oncoming left and right have valid actions as [None, 'forward', 'left', 'right']

Environment.valid_inputs
Out[79]: {'left': [None, 'forward', 'left', 'right'], 'light': [True, False], 'oncoming': [None, 'forward', 'left', 'right'], 'right': [None, 'forward', 'left', 'right']}
Instance variable for valid inputs

TrafficLight.valid_states
Out[80]: [True, False]
The light has 2 states

start_heading = random.choice(self.valid_headings)
one of the (1, 0), (0, -1), (-1, 0), (0, 1) directions

heading = state['heading'] its one of the (1, 0), (0, -1), (-1, 0), (0, 1)

For heading(1,0)
heading[0]=1 and heading[1]=0
Be aware: If the heading to the East (1,0) for example, your car position is on the left of the crossroad and if you turn left you go to the North (0, 1)

If (x,y) if your current heading

For turning left:
```
x' = y
y' = -x
```

For turning right:
```
x' = -y
y' = x
```

The problem of headings is addressed here https://discussions.udacity.com/t/headings-left-turn-and-right-turn/164468/4

valid_actions = [None, 'forward', 'left', 'right']
valid_inputs = {'light': TrafficLight.valid_states, 'oncoming': valid_actions, 'left': valid_actions, 'right': valid_actions}
valid_headings = [(1, 0), (0, -1), (-1, 0), (0, 1)] # ENWS

There are 4 'actions' and 2 input 'light' states, 4 'oncoming' input states, 4 'left' states, 4 'direction' = 4*2*4*4*4 = 512 states

The road network is
self.grid_size = (8, 6) # (cols, rows)
self.bounds = (1, 1, self.grid_size[0], self.grid_size[1]