Project Report

You will be required to submit a project report along with your modified agent code as part of your submission. As you complete the tasks below, include thorough, detailed answers to each question provided in italics.

Implement a Basic Driving Agent

To begin, your only task is to get the **smartcab** to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information 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.

To complete this task, simply have your driving agent choose a random action from the set of possible actions (`None`, `'forward'`, `'left'`, `'right'`) at each intersection, disregarding the input information above. Set the simulation deadline enforcement, `enforce_deadline` to `False` and observe how it performs.

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

Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

- About 30%, random action smartcab can arrive at destination.
- random action
  - average distance : 4.7423032
  - average dealine : 30.15
  - average steps : 25.41
  - average total reward : 1.595

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the **smartcab** and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the `self.state` variable. Continue with the simulation deadline enforcement `enforce_deadline` being set to `False`, and observe how your driving agent now reports the change in state as the simulation progresses.

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?*

ANSWER

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

- available environment values
  - light = ['red', 'green']
  - oncoming = [None, 'forward', 'left', 'right']
  - right = [None, 'forward', 'left', 'right']
  - left = [None, 'forward', 'left', 'right']
  - next_waypoint = ['forward', 'left', 'right', None]
- all available states count is 2 * 4 * 4 * 4 * 4 = 512
- But we do not need all 512 states. I think following 16 states are appropriate for this problem.
- canMove states
  - light = 'green'
    - GREEN_CAN_LEFT
      - None, 'forward', 'right', 'left'
    - GREEN_CANT_LEFT
      - None, 'forward', 'right'
  - light = 'red'
    - RED_CAN_RIGHT
      - None, 'right'
    - RED_CANT_RIGHT
      - None
- Append next_waypoints.
- SO, available_states = [

```
        'GREEN_CAN_LEFT_NEXT_NONE'
        , 'GREEN_CAN_LEFT_NEXT_FORWARD'
        , 'GREEN_CAN_LEFT_NEXT_LEFT'
        , 'GREEN_CAN_LEFT_NEXT_RIGHT'

        , 'GREEN_CANT_LEFT_NEXT_NONE'
        , 'GREEN_CANT_LEFT_NEXT_FORWARD'
        , 'GREEN_CANT_LEFT_NEXT_LEFT'
        , 'GREEN_CANT_LEFT_NEXT_RIGHT'

        , 'RED_CAN_RIGHT_NEXT_NONE'
        , 'RED_CAN_RIGHT_NEXT_FORWARD'
        , 'RED_CAN_RIGHT_NEXT_LEFT'
        , 'RED_CAN_RIGHT_NEXT_RIGHT'

        , 'RED_CANT_RIGHT_NEXT_NONE'
        , 'RED_CANT_RIGHT_NEXT_FORWARD'
        , 'RED_CANT_RIGHT_NEXT_LEFT'
        , 'RED_CANT_RIGHT_NEXT_RIGHT'
    ]
```

Why do you believe each of these states to be appropriate for this problem?

- I think above 16 states can represent all possible actions.

How many states in total exist for the smartcab in this environment?

- 512

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?

- No. too many redundant states.

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the **smartcab** will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement `enforce_deadline` to `True`. Run the simulation and observe how the **smartcab** moves about the environment in each trial.

The formulas for updating Q-values can be found in this (https://classroom.udacity.com/nanodegrees/nd009/parts/0091345409/modules/e64f9a65-fdb5-4e60-81a9-72813beebb7e/lessons/5446820041/concepts/6348990570923) video.

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

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken?

- random action
  - average distance : 4.7423032
  - average dealine : 30.15
  - average steps : 25.41
  - average total reward : 1.595
- using q-learning
  - average distance : 4.66289817
  - average dealine : 29.9
  - average steps : 13.21
  - average total reward : 22.105

Why is this behavior occurring?

- Smartcab learned by q-learning. And select appropriate way rather than dumb random way.

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the **smartcab** is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (`alpha`), the discount factor (`gamma`) and the exploration rate (`epsilon`) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your smartcab:

- Set the number of trials, `n_trials`, in the simulation to 100.
- Run the simulation with the deadline enforcement `enforce_deadline` set to `True` (you will need to reduce the update delay `update_delay` and set the `display` to `False`).
- Observe the driving agent's learning and **smartcab's** success rate, particularly during the later trials.
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

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?*

ANSWER

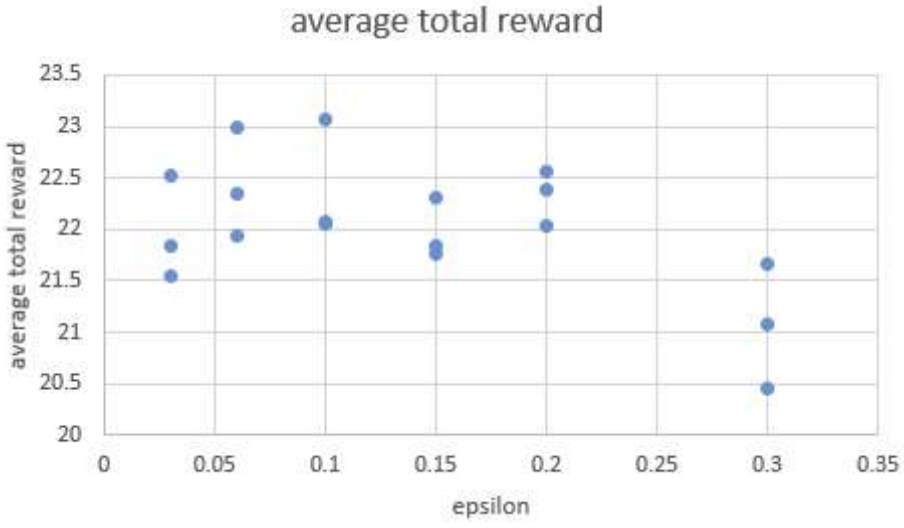
Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best?

- I justify q-learning function like this.
  - alpha = 1 / (learning_delta * n)
  - Q' = (1 - alpha) * Q + alpha * (reward + gamma * max_q_value_in_state)
- And find learning_delta, gamma, epsilon values which maximize average total reward.
- test result

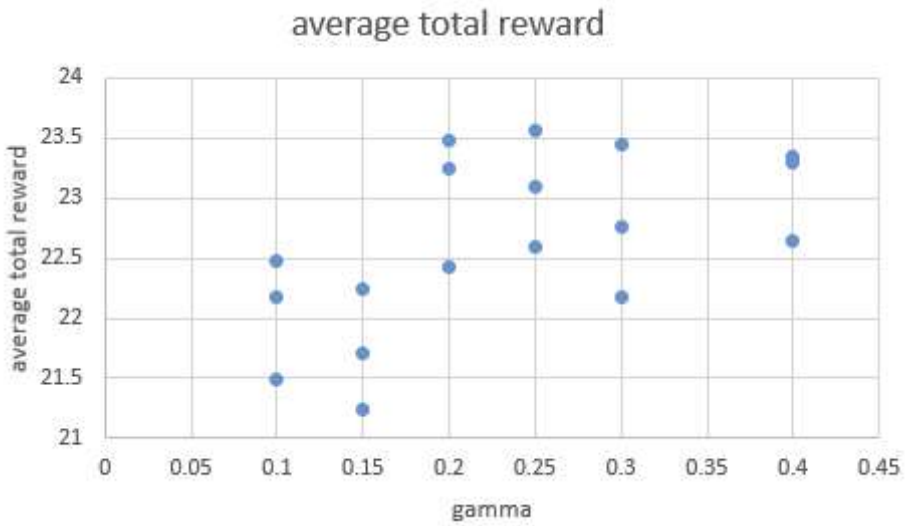
learning delta	gamma	epsilon	average distance	average deadline	average used step	average total reward
1	0.25	0.1	4.445029405	28.5	13.4	22.17
1	0.25	0.1	4.823567238	31	14.33	23.165
1	0.25	0.1	4.661012498	30.2	13.1	22.425
2	0.25	0.1	4.684603366	30.15	12.94	21.815
2	0.25	0.1	4.917794468	31.2	13.86	22.085
2	0.25	0.1	4.587266461	29.6	12.7	21.44
0.3	0.25	0.1	4.891841038	30.85	13.94	22.72
0.3	0.25	0.1	4.529582568	28.3	13.45	22.58
0.3	0.25	0.1	4.686030945	30.1	13.62	22.41
0.6	0.25	0.1	4.719580003	29.95	13.67	22.135
0.6	0.25	0.1	4.654595872	29.8	13	23.02
0.6	0.25	0.1	4.676575377	29.85	14.07	22.38
1.5	0.25	0.1	4.700350577	29.9	13.5	22.145
1.5	0.25	0.1	4.661494478	29.95	13.93	21.94
1.5	0.25	0.1	4.752578692	30.45	13.12	22.74
1	0.1	0.1	4.695436678	29.9	12.7	22.47
1	0.1	0.1	4.613651683	29.65	13.46	22.175
1	0.1	0.1	4.453366429	28.1	12.02	21.485
1	0.2	0.1	4.879043884	31.7	14.64	23.25
1	0.2	0.1	4.726260245	29.95	13.68	22.42
1	0.2	0.1	4.888714757	31.45	15.24	23.475
1	0.3	0.1	4.719841964	30.55	14.24	23.445
1	0.3	0.1	4.590227426	29.45	13.19	22.175
1	0.3	0.1	4.615549215	29.3	13.54	22.755
1	0.4	0.1	4.496128809	29.05	18.23	23.34
1	0.4	0.1	4.724104037	30.15	14.22	22.64

1	0.4	0.1	4.612747589	29.9	13.46	23.295
1	0.15	0.1	4.49978608	28.45	13.07	21.705
1	0.15	0.1	4.408157503	27.85	13.28	21.245
1	0.15	0.1	4.86151975	31.1	13.27	22.24
1	0.25	0.1	4.72827664	30.25	13.78	23.57
1	0.25	0.1	4.666418334	30.05	14.52	23.095
1	0.25	0.1	4.818859577	30.8	13.09	22.595
1	0.1	0.1	4.493096762	28.85	12.75	22.08
1	0.1	0.1	4.645103686	29.95	12.86	23.06
1	0.1	0.1	4.579446009	28.65	12.84	22.06
1	0.1	0.2	4.677	29.85	15.2	22.385
1	0.1	0.2	4.835052856	30.05	15.62	22.035
1	0.1	0.2	4.717859314	30.35	15.32	22.57
1	0.1	0.3	4.694416221	29.55	15.16	20.45
1	0.1	0.3	4.753798412	30.7	17.72	21.085
1	0.1	0.3	4.578724389	29.55	15.91	21.67
1	0.1	0.03	4.648674809	30.25	12.8	22.53
1	0.1	0.03	4.384893795	27.8	11	21.555
1	0.1	0.03	4.393938535	28.7	10.78	21.845
1	0.1	0.06	4.711212475	30.55	13.73	22.995
1	0.1	0.06	4.86973656	31.35	11.91	22.355
1	0.1	0.06	4.657328285	29.8	12.97	21.93
1	0.1	0.15	4.483369755	28.7	12.35	21.77
1	0.1	0.15	4.751669527	30	13.7	21.85
1	0.1	0.15	4.637391645	29.45	13.99	22.305

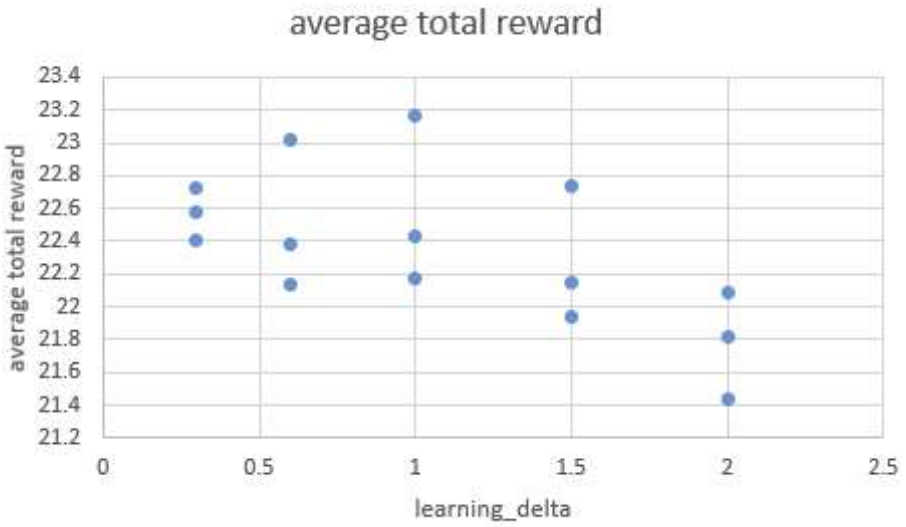
- epsilon : 0.1 is best.



- gamma : 0.25 is best.



- learning_delta : 1 is best.



- best parameter set is
  - alpha : 1/(1 * n)
  - gamma : 0.25
  - epsilon : 0.1

How well does the final driving agent perform?

- Maximum average total reward point : 23.57

How would you describe an optimal policy for this problem?

- Optimal policy should be like this.
  - Not incur any penalites.
  - Average total reward is high.

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?

- Q-value's table is like this.

state	action : None	action : forward	action : left	action : right
GREEN_CAN_LEFT_NEXT_NONE	0.000000	0.000000	0.000000	0.000000
GREEN_CAN_LEFT_NEXT_FORWARD	0.021873	3.007901	-0.026798	-0.183984

GREEN_CAN_LEFT_NEXT_LEFT	0.001054	-0.001949	0.982606	-0.127843
GREEN_CAN_LEFT_NEXT_RIGHT	0.053981	0.000000	-0.005937	2.223150
GREEN_CANT_LEFT_NEXT_NONE	0.000000	0.000000	0.000000	0.000000
GREEN_CANT_LEFT_NEXT_FORWARD	0.000000	0.000000	-0.000853	-0.000539
GREEN_CANT_LEFT_NEXT_LEFT	0.000000	0.000000	0.000000	0.008247
GREEN_CANT_LEFT_NEXT_RIGHT	0.000000	0.000000	0.000000	0.000000
RED_CAN_RIGHT_NEXT_NONE	0.000000	0.000000	0.000000	0.000000
RED_CAN_RIGHT_NEXT_FORWARD	0.000000	-0.106408	-0.096744	-0.076966
RED_CAN_RIGHT_NEXT_LEFT	0.000000	-0.043063	-0.031250	-0.020292
RED_CAN_RIGHT_NEXT_RIGHT	0.001979	-0.002189	-0.003727	1.767729
RED_CANT_RIGHT_NEXT_NONE	0.000000	0.000000	0.000000	0.000000
RED_CANT_RIGHT_NEXT_FORWARD	0.000000	0.000000	-0.000881	-0.000907
RED_CANT_RIGHT_NEXT_LEFT	0.000000	0.000000	-0.001025	-0.004082
RED_CANT_RIGHT_NEXT_RIGHT	0.000000	0.000000	0.000000	0.000000

- Our smartcab always select action that not incur any penalties.
- Average total reward is relatively high(compared to random selection).

In [ ]: