report

September 29, 2016

Project 4: Train a smartcab to drive

Version 1 from Shitao Wang

Task 1: 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?

The Basic driving agent takes random actions at each time and makes no attempt to learn. The resulting agent moves randomly observed by the pygame GUI. The agent reaches the destination 20 our of 100 trials with 2451 times breaking the traffic rules. Even if some of the trials evetually reach the destination, the agent takes many steps. The following table (Out[7]) shows some basic statistics of this basic agent.

We expect the behaviour of the basic agent since there are a lot of combinations from this 48 intersections' environment. It should be noted that there are only 4 cars including our agent on the road. Our basic driving agent will cause more crashes on the road if we have more cars.

see agent_without_q_learn.py for detailed implementations.

```
In [7]: import pandas as pd
        task1 = pd.read_csv('random.csv')
        task1.describe()
```

```
Out[7]:        total_reward  negative_reward  tria_length
        count    100.000000       100.000000   100.000000
        mean      -0.165000       -11.645000    27.200000
```

```
std         8.112235          4.817526     9.859621
min       -17.500000        -23.500000     5.000000
25%        -5.500000        -14.125000    21.000000
50%        -2.500000        -11.000000    26.000000
75%         6.125000         -9.000000    35.250000
max        20.500000         -1.000000    51.000000
```

Task 2: 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?

I select light, oncoming, left, next_waypoint as the set of states.

light = {‘red’,‘green’} *(Determine whether the smartcab can move forward)*

oncoming = {‘forward’,‘left’,‘right’,None} *(If the smartcab wants to turn left, it should yield the oncoming traffic)*

left = {‘forward’,‘left’,‘right’,None} *(If the smartcab wants to turn right, it should yield the left forward traffic)*

next_waypoint{‘forward’,‘left’,‘right’} *(Information from the planner to guide the smartcab to the destination)*

With only the information of next_waypoint, the smartcab can reach the destination within all trials with 610 times breaking the traffic rules. Basic statistics shown in Out[8]. When applying the inputs information and contraining the action with the traffic rules, the smartcab can reach the destination within all trials with 0 times breaking the traffic rules as expected. Statistics shown in Out[9].

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?

There are total $2 * 4 * 4 * 3 = 96$ states.

I added another inputs information ‘right’ in my Q learning algorithm, so the total should be $2 * 4 * 4 * 4 * 3 = 384$ states. It should be noted that we do not need to know the information from the right in order to follow the traffic rules. But I added it anyway in order to use it to model a full environment for the agent.

The deadline is another potential variable that we could consider. If we want to build the most comprehensive state we could include deadline in our state. However, we will greatly increase the size of our state since the dealine equals to the L1 distance between two points multiplied by a factor of 5 (check environment.py line 97). We thus choose not to include the deadline in the state.

With only three other agents in this exercise, the Q matrix is likely to be sparse. The Q matrix is relative large for only 100 trials to learn. But the fact that only three other agents are in the play makes it possible to learn the optimal policy within a small number of trials.

```
In [8]: task2 = pd.read_csv('planner_only.csv')
        task2.describe()
```

```
Out[8]:        total_reward  negative_reward  trial_length
        count    100.000000       100.000000    100.000000
        mean      16.300000        -6.100000     12.300000
        std        4.520436         4.556048      5.549593
        min        4.000000       -22.000000      4.000000
        25%       13.000000        -9.000000      8.000000
        50%       16.000000        -5.500000     11.000000
        75%       19.000000        -2.750000     16.000000
        max       28.000000         0.000000     31.000000
```

```
In [9]: task22 = pd.read_csv('planner_and_traffic_rule.csv')
        task22.describe()
```

```
Out[9]:        total_reward  negative_reward  trial_length
        count    100.000000            100.0    100.000000
        mean      22.400000              0.0     12.300000
        std        3.629286              0.0      5.549593
        min       18.000000              0.0      4.000000
        25%       19.500000              0.0      8.000000
        50%       22.000000              0.0     11.000000
        75%       24.500000              0.0     16.000000
        max       32.000000              0.0     31.000000
```

Task 3: 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.

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?

In the Q learning algorithm, the agent improves its performance by learning. Several tunable parameters need to be set for the Q learning algorithm to work. For the first try, I use learning rate alpha = 0.5, exploration rate epsilon = 0.1 and discount factor gamma = 0.9.

In terms of agent behaviours, the Q learning agent does much less random action than the basic driving agent. It learns quickly to avoid violating the traffic laws and it reaches the destination much faster than the basic driving agent. The main reason for this is that Q learning algorithm encourages the actions that lead to some rewards while discourages the actions that lead to some penalties. A basic observation at the beginning of the trials is that the agent receives a lot of none actions. In order to improve this, I could propose three approaches: 1) increase the exploration rate. 2) use a higher initial values. (refers to "optimism in the face of uncertainty" suggested by the reviewer) 3) add some penalty such that every action does not guide the agent to the destination is discouraged.

In terms of agent statistics, at the first few trials, the total reward is very small than the later trials shown in Out[12]. After several iterations, the agent performs better. 71 out of 100 trials the smartcab reaches the destination with only 67 times breaking traffic rules which is much better than the basic drving agent, at the same time, the basic statistics of the q learning agent is also better than that of the basic driving agent shown in Out[10].

see the detailed implementation in agent.py

```
In [10]: task3 = pd.read_csv('Q_learning_gamma0.9.csv')
         task3.describe()
```

```
Out[10]:        total_reward  negative_reward  trial_length
         count    100.000000       100.000000    100.000000
         mean      23.685000        -3.205000     20.500000
         std       10.479816         2.618982     11.734005
         min        0.000000       -16.000000      1.000000
         25%       17.875000        -4.500000     11.750000
         50%       23.000000        -2.500000     19.000000
         75%       28.500000        -1.500000     26.000000
         max       65.500000         0.000000     61.000000
```

```
In [12]: task3
```

```
Out[12]:     total_reward  negative_reward  trial_length
         0            0.0             -2.0            26
         1            3.0             -1.0            26
         2            4.5             -1.5            41
         3            8.5             -1.5            26
         4           28.0             -2.0            22
         5           24.5             -1.5            11
         6           19.0             -1.0             7
         7           12.0              0.0            36
         8           26.5             -1.5            17
         9           14.0             -4.0            26
         10          25.5             -2.5            26
         11          28.5             -3.5            19
         12          23.0             -1.0             9
         13          65.5            -10.5            61
         14          44.0             -6.0            33
         15          22.5             -1.5             9
         16          21.0             -1.0             8
         17          24.0             -2.0            15
         18           9.5              0.0             1
         19          28.0             -2.0            15
         20          33.5             -2.5            19
         21          25.5             -2.5            19
         22          25.5             -0.5             9
         23          21.5             -2.5            13
         24          29.5             -2.5            16
         25          39.0             -5.0            61
         26          14.0             -2.0            11
         27          19.5             -4.5            21
```

```
28          31.5             -4.5             31
29          26.0             -8.0             34
..           ...              ...            ...
70          51.5             -6.5             41
71          15.0             -1.0              5
72          55.0             -7.0             45
73          29.5             -2.5             16
74          28.5             -1.5             13
75          26.0             -2.0             16
76          21.5             -0.5              7
77          22.0              0.0              8
78          17.0             -1.0              5
79          41.5             -4.5             32
80          25.0             -5.0             31
81          20.0             -6.0             26
82          33.0             -5.0             25
83          19.0             -1.0              7
84          19.0             -1.0              7
85          23.5             -2.5             16
86          15.5             -0.5              5
87          24.0              0.0              8
88          27.5             -4.5             22
89          19.5             -2.5             15
90          18.0             -6.0             31
91          31.0             -9.0             51
92          12.5             -5.5             21
93          28.5             -1.5             18
94          27.0             -1.0             12
95          23.5             -0.5             12
96          15.0             -7.0             26
97          17.0             -5.0             21
98          11.5             -4.5             31
99          31.0             -3.0             19

[100 rows x 3 columns]
```

Task 4: 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?

The tuning parameter I select is the discount factor gamma from [0.1,0.3,0.5,0.7,0.9]. The other two parameters, alpha and epsilon are fixed at 0.5 and 0.1, respectively. The gamma parameter should range from 0 to 1 and the larger this value is, the more influence of the future states will be included.

gamma = 0.1 provides the best performance, in which the smartcab reaches the destination 90 out of 100 trials with 36 times breaking the traffic rules. The fact that the best performance comes from the small gamma indicats weighting too much future reward may degrade the performance of the agent/ Summary of performance for all different values of gamma is shown in the following table. The basic statistics of gamma = 0.1 are shown in Out[11].

gamma	destination reached	crashes	averaged trial length
0.1	90	36	15.66
0.3	90	42	15.55
0.5	86	41	17.40
0.7	78	43	18.83
0.9	71	67	20.50

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 optimal policy for this problem can be described as that the agent reaches the destination within a very small number of steps and obey the traffic rules at the same time. Although there are still failures, crashes, rule breakings, the final policy is much better compared with the basic agent. The Q learning algorithm amazingly learn from the traffic rules and other information to improve the performance of the agent. For the future work, more parameters can be tuned in order to obtain a even better policy.

```
In [11]: task4 = pd.read_csv('Q_learning_gamma0.1.csv')
         task4.describe()
```

```
Out[11]:        total_reward  negative_reward  trial_length
         count    100.000000       100.000000    100.000000
         mean      20.625000        -0.630000     15.660000
         std        6.561433         0.799684      9.536977
         min        0.000000        -5.000000      1.000000
         25%       18.000000        -1.000000      9.000000
         50%       21.500000        -0.500000     13.000000
         75%       24.000000         0.000000     21.000000
         max       37.000000         0.000000     49.000000
```

```
In [ ]:
```