Smartcab

Ryosuke Honda

2016/06/23

1 Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading.
- Intersection state(traffic light and presence of cars) and
- Current deadline value(time steps remining)

And produces some random move/action [None,'forward','left','right']

2 Identify and update state

3 Implement Q-learning

One of the most important breakthroughs in reinforcement learning was the development of Q-learning.Its simplest form,one step Q-learning, is defined by

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r(s,a) + \gamma \max_{a'}(Q'(s',a'))) \tag{1}$$

α:Learning rate
γ:Discount rate
s stands for state, **a** stands for action, **s'** stands for the next state, **a'** stands for the all actions

First, set the parameters(α,γ) and environmental reward.Then Initialize the Q table to zero. For each episode, select a random initial state and do the following things until the agent has reached the goal.

- Select one among all possible actions for the current state.
- Using this possible action, consider going to the next state.
- Get maximum Q value for this next state based on all possible actions

- Compute Q value
- Set the next state as the current state

The Gamma and Alpha parameters has a range of 0 to 1. If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Alpha is closer to zero, the agent will tend to consider only past experience(don't learn).

The reward and penalty for this task is as follows.

```
# TODO: Select action according to your policy
epsilon=0.1
#epsilon=1.0
rand=random.random()
if rand<epsilon:
    action=random.choice(Environment.valid_actions[0:])
    print "RANDOM ACTION"
else:
    if max(self.q_table[self.state].values())==0:
        action=random.choice(Environment.valid_actions[0:])
        print self.q_table[self.state]

    else:
        action = max(self.q_table[self.state],
            key=self.q_table[self.state].get)

    print action
```

Figure 1: Python code snippet for deciding action policy

In this experiment, I use ϵ-greedy and set ϵ to be 0.1. The agent chooses random action with the probability of 0.1. If the maximum of q-table is 0, the agent also chooses random action. If the maximum q-table is more than 0, the agent chooses the maximum value and moves toward it.

4 Enhance the driving agent

Report what changes you make to your basic implementation of Q-learning to achieve the final version of the agent.How well does it perform?
When the agent just started, it has no q-value since the q-table is set to be 0. Therefore, at first I need to set the move randomly(ϵ-greedy). With the probability of ϵ, the agent moves randomly. This prevents the agent from falling into the wrong choice.

First, I implemented random action.The agent just moves random action("None","Forward","Left","Right") and doesn't learn anything from experience.
In the agent.py file, I set ϵ to be 1(The agents moves randomly). I define "Success Rate" to see the agent's performance.

[Success Rate]=[No.trials achieve goal before deadline]/[No. trials]
For these trials, the number of trials are 100.

In this experiment,I set the conditions as follows.

Success rate in this trial is 0.20.

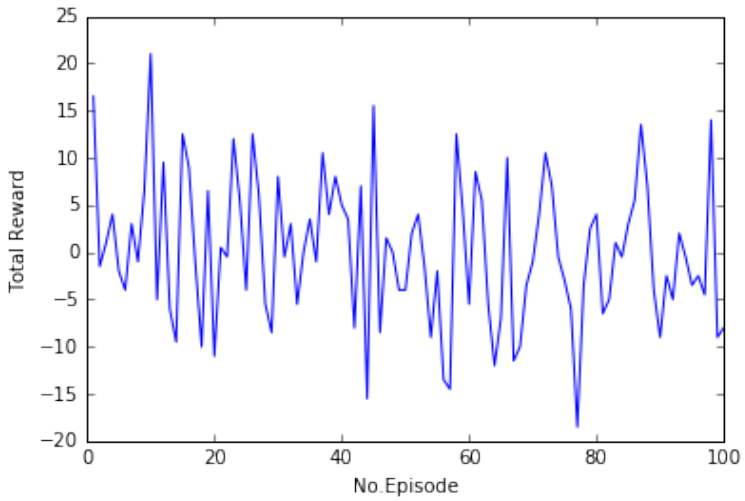

Figure 2: Random Action

As the Fig2 shows,the total rewards with this trial contains negative values. Without learning from their experience, the agent won't get high reward.

Second, the parameters of α and γ are 0.5 and 0.9 respectively. With this condition, the success rate is 0.72 which is higher than the random action.

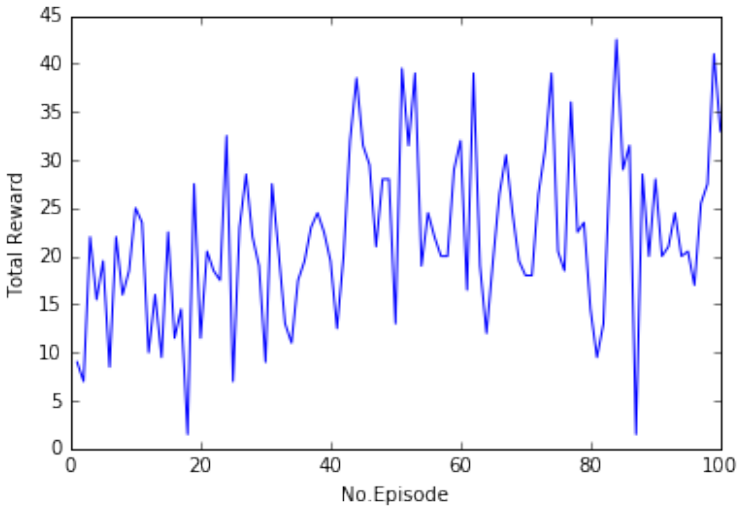

Figure 3: Constant α and γ

The Fig3 shows,the total rewards for each episode are all positive and the cumulative sum of total reward of all episodes are 2195.5.

Third, the parameters of α and γ are as follows.

$$\alpha = \frac{1.0}{1.0 + time} \tag{2}$$

$$\gamma = \frac{1.0}{1.0 + deadline} \tag{3}$$

With this condition, the success rate is 0.86 which is higher than the trials before.

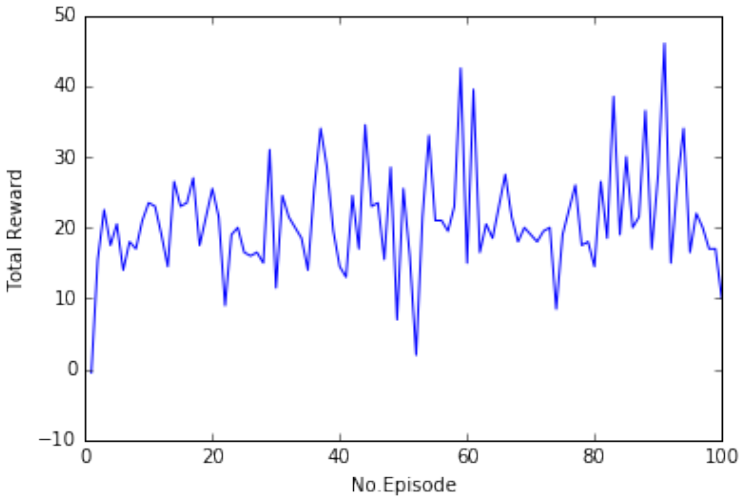

Figure 4: Fig

The Fig4 shows,the total rewards for each episode are all positive and the cumulative sum of total reward of all episodes are 2099.0

Final version

The condition in this implementation is as follows. α is equal to the fomula (2) and γ sets to be 0.9.

With this condition, the success rate is 0.86 which is the same value as the trial 3.

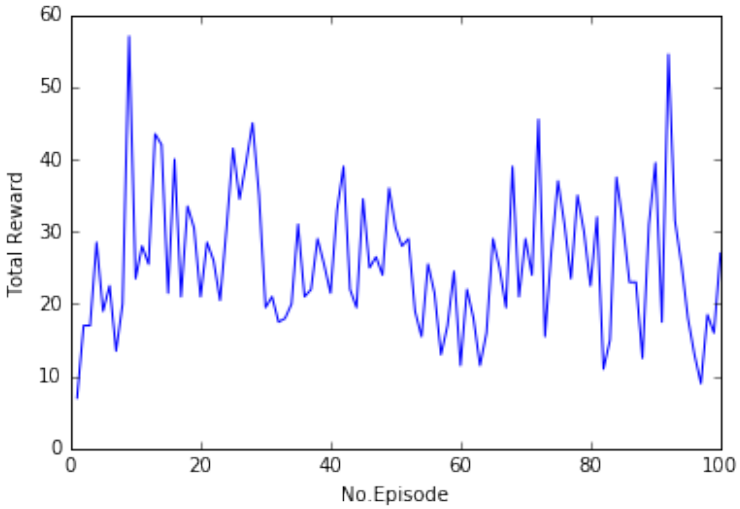

Figure 5: Fig

The Fig5 shows,the total rewards for each episode are all positive and the cumulative sum of total reward of all episodes are 2580.5. The cumulative sum of total reward of all episodes are conspicuously higher than the other trials.