17044633_RL_hw2

February 26, 2018

By: Yuan Zhang
SN: 17044633

1 RL homework 2

Due date: 26 February 2018, 23:55am

1.0.1 Policy iteration

We used TD to do policy evaluation for the random policy on this problem. Consider doing policy improvement, by taking the greedy policy with respect to a one-step look-ahead. For this, you may assume we have a true model, so for each state the policy would for each action look at the value of the resulting state, and would then pick the action with the highest state value. You do **not** have to implement this, just answer the following questions.

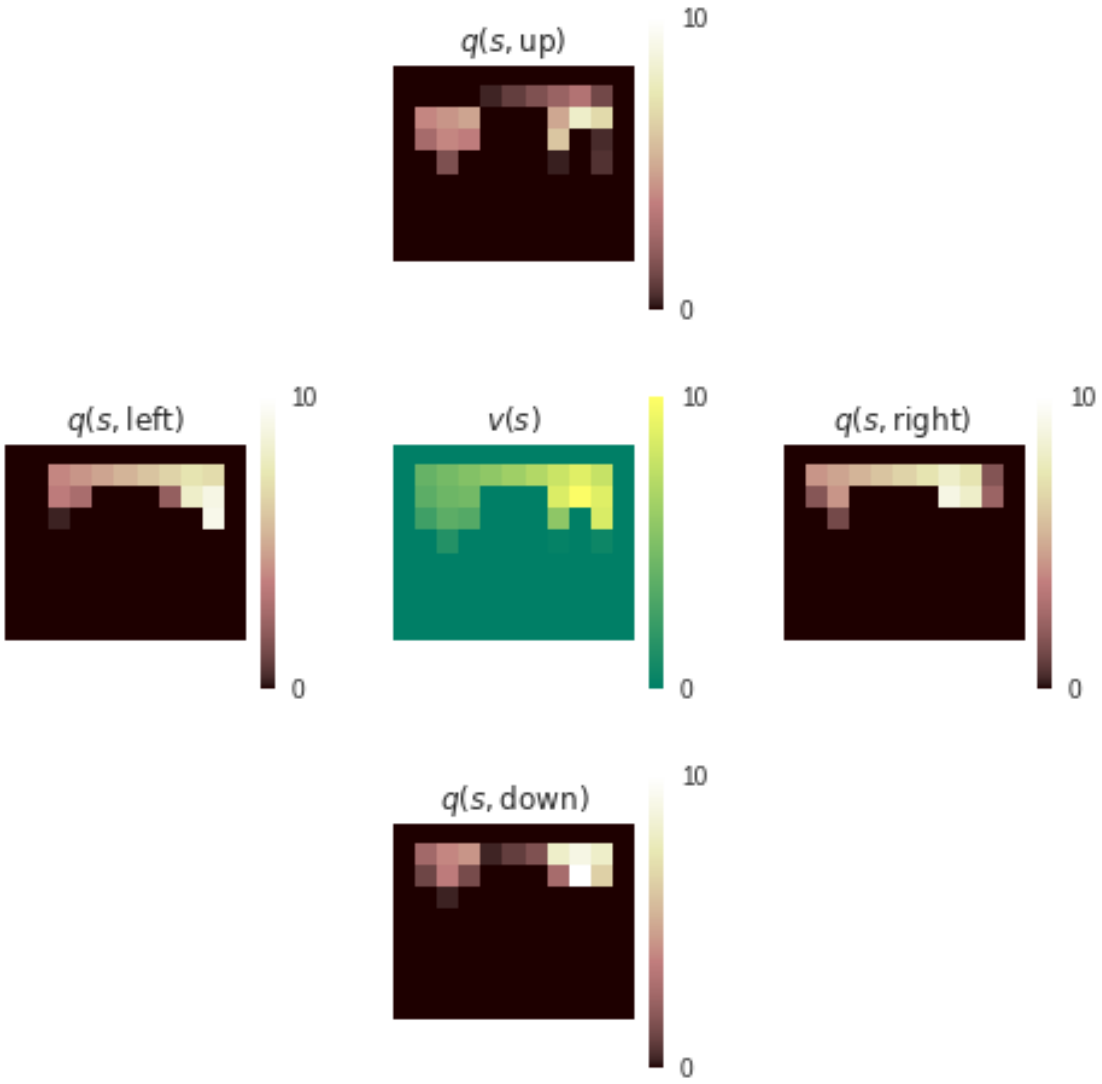
[5 pts] Would the greedy after one such iteration of policy evaluation and policy improvement be optimal on this problem? Explain (in one or two sentences) why or why not.

No. Because iteration of evaluation and improvement only look at the adjacent state and it takes time to spread the reward of the goal to every state.

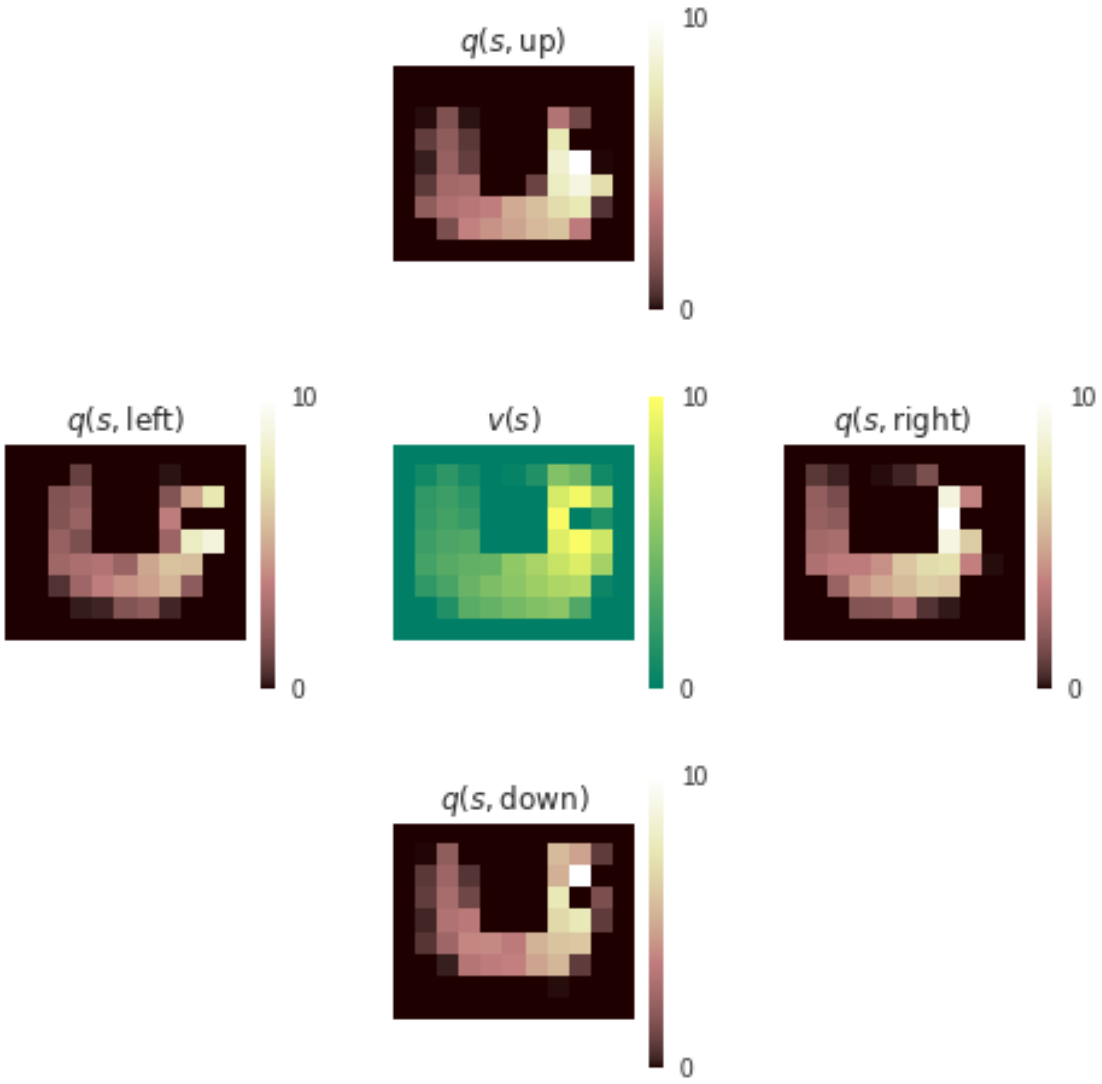
[5 pts] If we repeat the process over and over again, and repeatedly evaluate the greedy policy and then perform another improvement step, would then the policy eventually become optimal? Explain (in one or two sentences) why of why not.

Yes. After enough iterations every state will be influenced by the goal and the policy will converge to the optimal policy even if the state value has not converged yet.

```
In [21]: grid = Grid()
         def target_policy(q, a):
           return np.eye(len(q))[np.argmax(q)]
         def behaviour_policy(q):
           return epsilon_greedy(q, 0.1)
         agent = GeneralQ(grid._layout.size, 4, grid.get_obs(),
                          target_policy, behaviour_policy, double=False)
         run_experiment(grid, agent, int(1e5))
         q = agent.q_values.reshape(grid._layout.shape + (4,))
         plot_action_values(q)
```




```
In [22]: grid = Grid()
         def target_policy(q, a):
           return np.eye(len(q))[a]
         def behaviour_policy(q):
           return epsilon_greedy(q, 0.1)
         agent = GeneralQ(grid._layout.size, 4, grid.get_obs(),
                          target_policy, behaviour_policy, double=False)
         run_experiment(grid, agent, int(1e5))
         q = agent.q_values.reshape(grid._layout.shape + (4,))
         plot_action_values(q)
```



1.1 Questions

Consider the greedy policy with respect to the estimated values

[10 pts] *How* do the policies found by Q-learning and Sarsa differ? (Explain qualitatively how the behaviour differs in one or two sentences.)

For Q-learning, it works mainly on the upper path, while Sarsa works on the downner path, which is further from the start point, but much safer.

[10 pts] *Why* do the policies differ in this way?

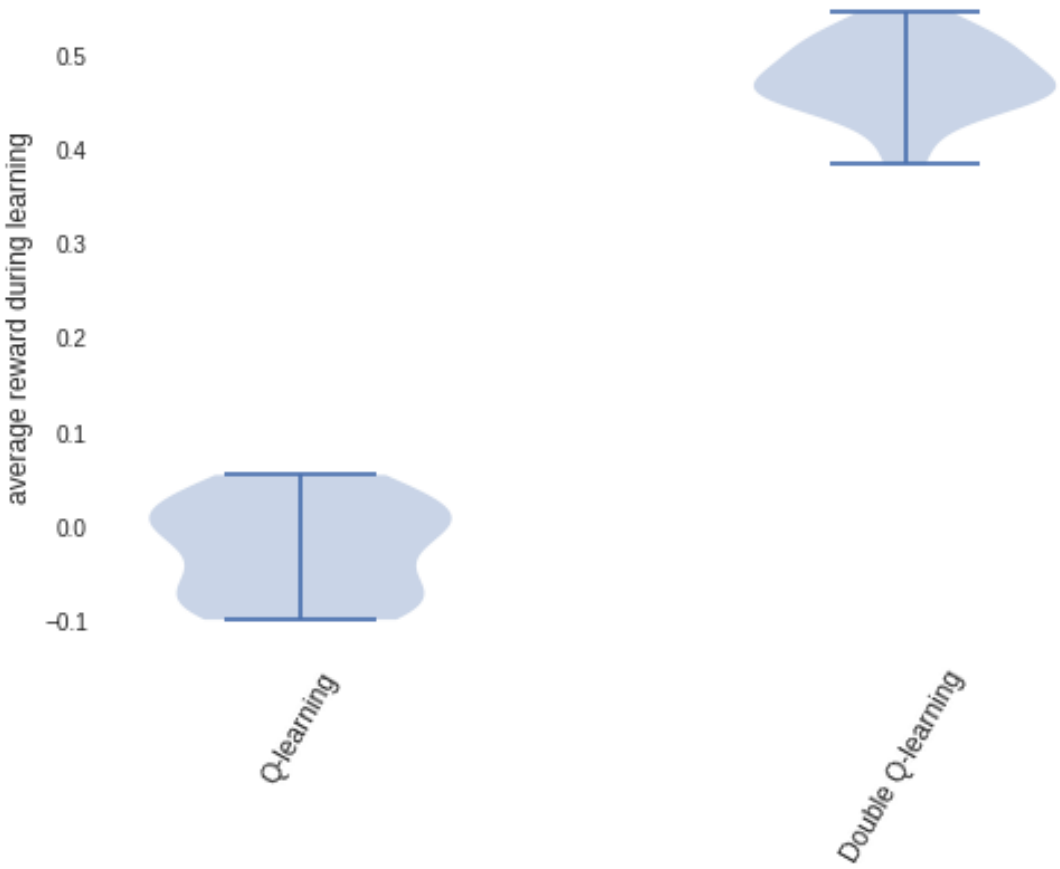
Because the target policy for Q-learning is greedy, which makes it explore less than Sarsa. Sarsa uses epsilon-greedy for target policy.

[10 pts] Which greedy policy is better, in terms of actual value?

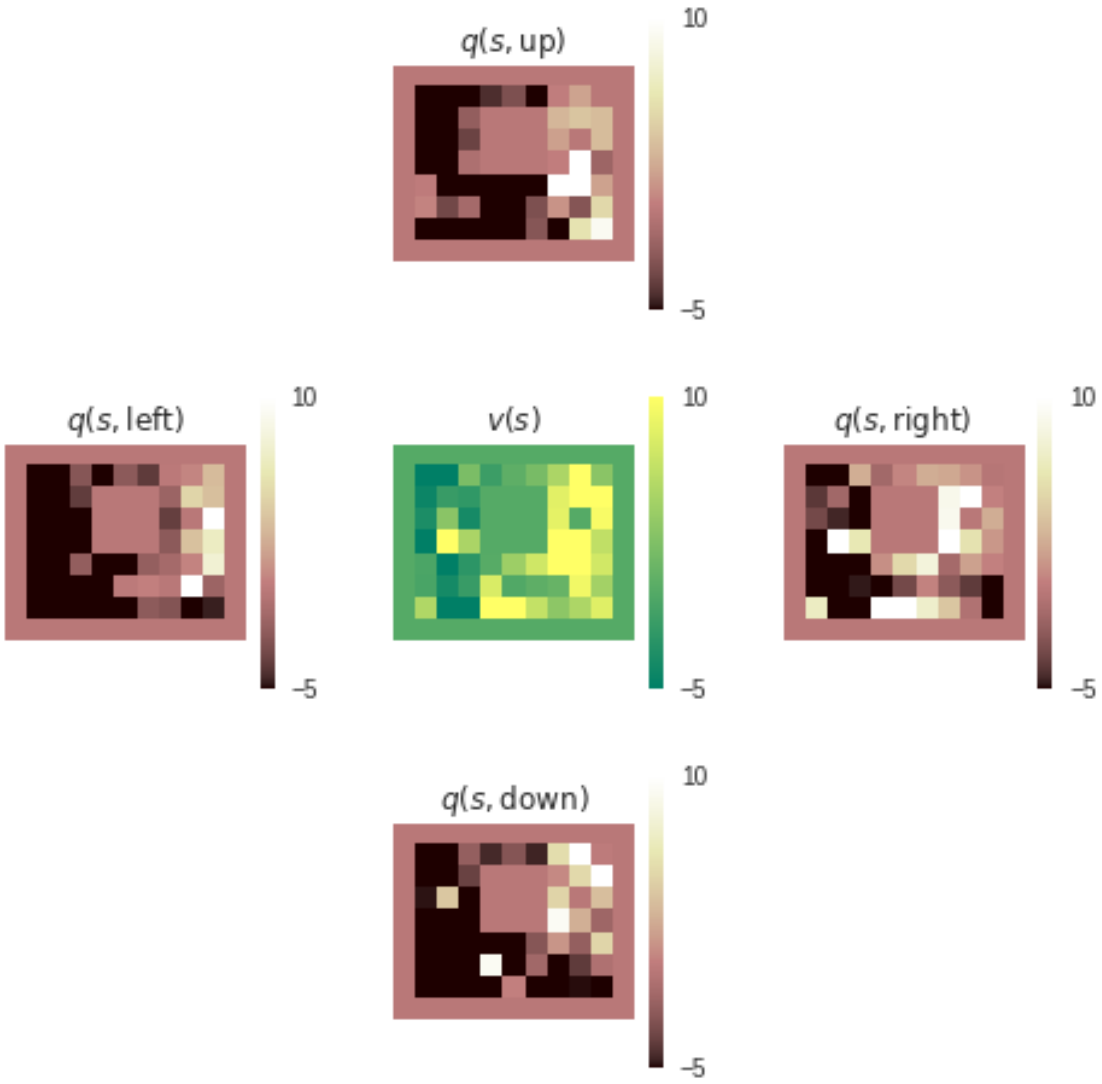
For this problem, Sarsa is better. Because it finds a safer path to the goal further from the wall.

In [23]:
```
def target_policy(q, a):
  max_q = np.max(q)
  pi = np.array([1. if qi == max_q else 0. for qi in q])
  return pi / sum(pi)
def behaviour_policy(q):
  return epsilon_greedy(q, 0.1)
mean_reward_q_learning = []
mean_reward_double_q_learning = []
for _ in range(20):
  grid = Grid(noisy=True)
  q_agent = GeneralQ(grid._layout.size, 4, grid.get_obs(),
                     target_policy, behaviour_policy, double=False, step_size=0.1)
  dq_agent = GeneralQ(grid._layout.size, 4, grid.get_obs(),
                      target_policy, behaviour_policy, double=True, step_size=0.1)
  mean_reward_q_learning.append(run_experiment(grid, q_agent, int(2e5)))
  mean_reward_double_q_learning.append(run_experiment(grid, dq_agent, int(2e5)))
plt.violinplot([mean_reward_q_learning, mean_reward_double_q_learning])
plt.xticks([1, 2], ["Q-learning", "Double Q-learning"], rotation=60, size=12)
plt.ylabel("average reward during learning", size=12)
ax = plt.gca()
ax.set_axis_bgcolor('white')
ax.grid(0)
```

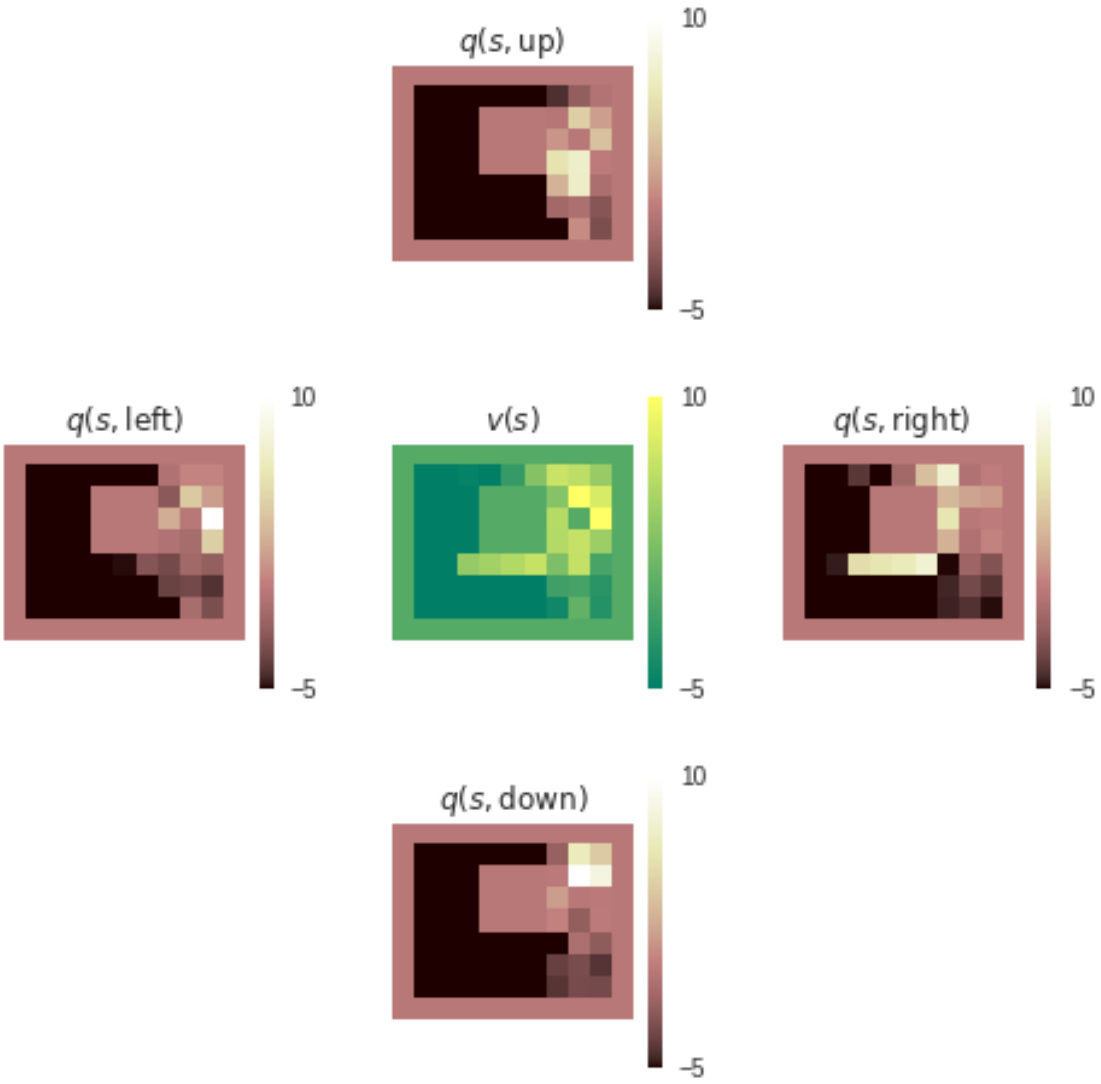
```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:21: MatplotlibDeprecationWarning:
```



In [26]:
```
q = q_agent.q_values.reshape(grid._layout.shape + (4,))
plot_action_values(q, vmin=-5)
```




```
In [25]: q = dq_agent.q_values.reshape(grid._layout.shape + (4,))
         plot_action_values(q, vmin=-5)
```



The plots above show 1) the distributions of average rewards (over all learning steps) over the 20 experiments per algorithm, 2) the action values for Q-learning, and 3) the action values for Double Q-learning.

[10 pts] Explain why Double Q-learning has a higher average reward. Use at most four sentences, and discuss at least a) the dynamics of the algorithm, b) how this affects behaviour, and c) why the resulting behaviour yields higher rewards for Double Q-learning than for Q-learning.

Because Double Q-learning dynamically update Q1 and Q2, and these two values estimates are independent. It avoids the bias of maximisation so that the value estimate is more real and leads to good path to the goal.

(Feel free to experiment to gain more intuition, if this is helpful. Especially the action value plots can be quite noisy, and therefore hard to interpret.)