SYS 6582: Reinforcement Learning **Spring 2018**

Homework 2

Handed Out: February 12, 2018 *Due: February* 26, 2018

Instructions: Solve each of the following exercises. Give quantitative answers where required, and always explain your reasoning. Finally, pledge your solutions.

1. (Exercise 4.1, S & B) In Example 4.1, if π is the equiprobable random policy, what is $q_\pi(11, down)$? What is $q_\pi(7, down)$? (5%)

2. (Exercise 4.2, S & B) In Example 4.1, suppose a new state 15 is added to the gridworld just below state 13, and its actions, left, up, right, and down, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions from the original states are unchanged. What, then, is $v_\pi(15)$ for the equiprobable random policy? Now suppose the dynamics of state 13 are also changed, such that action down from state 13 takes the agent to the new state 15. What is $v_\pi(15)$ for the equiprobable random policy in this case? (10%)

3. (Exercise 4.3, S & B) What are the equations analogous to (4.3), (4.4), and (4.5) for the action-value function q_π and its successive approximation by a sequence of functions $q_0, q_1, q_2, \ldots$? (10%)

4. (Exercise 4.6, S & B) How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to that on page 65 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book. (10%)

5. (Exercise 4.8, S & B) Why does the optimal policy for the gambler's problem have such a curious form? In particular, for capital of 50 it bets it all on one flip, but for capital of 51 it does not. Why is this a good policy? (5%)

6. (Exercise 4.10, S &B) What is the analog of the value iteration backup (4.10) for action values, $q_{k+1}(s, a)$? (5%)

7. In this problem you will implement the policy iteration algorithm, which includes two components: policy evaluation and policy improvement.

    a. Complete function *policyEval* in *dynamicProgramming.py*. Then test your code on the gridworld example (Example 4.1). Run *test_policyEval.py*. You should get the same values as the ones on the left panel of Figure 4.1 in the textbook. Append results as comment to the test file. Submit *test_policyEval.py* on collab. (10%)

    b. Complete function *policyImprv* in *dynamicProgramming.py*. Now you can call function *policyIteration* to find the optimal policy. Run *test_policyIteration.py* , then append results as comment following the code in this test file. Submit *test_policyIteration* on collab. (10%)

8. In this problem you will implement the value iteration algorithm and apply it to the gambler's problem (Example 4.3).

   a. Complete *valueIteration* in *dynamicProgramming.py*. (You can test your code on the gridworld example before moving on to the next part.) Submit the **complete** *dynamicProgramming.py* on collab. (10%)

   b. Set up the transition matrix $\mathbf{P}$ and the reward matrix $\mathbf{R}$ for the gambler's problem (Example 4.3) in *gambler_example.py*. Run the script with $p_h = 0.4$, Your results should be similar to Figure 4.3. Then run the script with $p_h = 0.25$ and 0.5. Briefly discuss the optimal policy when p_h is 0.25 and 0.5, respectively. Submit *gambler_example.py* and figures from these three runs on collab. (10%)

9. As a further test, we will apply the value iteration algorithm (of course, we can also use the policy iteration algorithm) on the example of Jack's car rental (Example 4.2). First, run *carRental.py*. The optimal policy should be similar to the the one in Figure 4.2. Then modify *carRental_setup.py* according to the changes specified in Exercise 4.5 and run *carRental.py* again. Briefly discuss the optimal policy in this case. Submit figures of the optimal policy under both the original and the modified scenarios on collab. (15%)