Last name:_____LI_____ First name:__xianhang__ SID#:__1465904__

Collaborators:________________________________

CMPUT 366Assignment 5: Planning and learning

Due: Wed Nov 21st by gradescope

There are a total of 98 points available on this assignment, as well as 10 bonus points

The first 3 questions are exercises are from the Sutton and Barto textbook, 2nd edition:

Question 1: Exercise 6.2 [6 points] (*TD vs MC; driving home example*)

Question 2: Exercise 6.3 [6 points] (*first episode in chain problem*)

Question 3. Exercise 8.2 [6 points] (*Dyna-Q+ vs Dyna-Q*)

Question 4. **Programming Exercise** [**80 points, three parts**].

Part 1 [20 points]. Implement the 1000 state random walk described in Example 9.1. You will make an RL-glue environment program that implements the 1000 state random walk described in chapter 9, example 9.1.

Please submit your environment program for this part.

Part 2 [40 points]. Implement two prediction agents based on TD(0); each using a different function approximation schemes in RL-glue.

Tabular feature encoding: agent one will implement Semi-gradient TD(0) (described on page 203) with a tabular or "one-hot" state encoding. A tabular encoding of the current state uses a unique feature vector for each state, where the number of components in the feature vector is equal the number of states. For example imagine an MDP with 4 states. The feature vectors corresponding to each state would be: [1,0,0,0], [0,1,0,0],[0,0,1,0],[0,0,0,1] for states 1,2,3,4 respectively. Test your semi-gradient TD(0) agent with alpha=0.5, tabular features on your 1000 state random walk environment.

Tile coding features: agent two will implement Semi-gradient TD(0) with tile coding. You will use the supplied tiling coding software (tiles3.py). In this case we treat the state number as a continuous number and tile code it producing a list of active features equal to the number of tilings. You will use number of tilings = 50, and tile width equal to 0.2. For TD(0) use alpha = 0.01/50.

Tile coder documentation can be found here: **http://www.incompleteideas.net/tiles/tiles3.html**

Please submit your two agents for this part. They can be in one agent program with programatic way to switch between them, or two different agent program files. Either is fine.

Part 3 [20 points]. Plot the learning curve of your two agents. For this part you will write an RL-glue experiment program and produce a plot similar to Figure 9.10 (page 218), except you will plot two performance lines corresponding to the Average RMS error over 1000 states for: (1) Semi-gradient TD(0) with tabular features, and (2) Semi-gradient TD(0) with Tile coding features. For each of your two agents you will plot the RMS error averaged over 30 runs, over 2000 episodes, plotting performance every 10 episodes. The RMS error is the squared error between the value function learned by your agent after *k* episodes and the true value function, averaged over all 1000 states, then rooted. The formula is given below:

$$\text{RMSE}_k = \sqrt{\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}(v_\pi(s) - v_k(s))^2}$$

The RMS error should be computed after each episode is completed, and averaged over 30 independent runs of the experiment. The error is the unweighted average over states, unlike the RMSVE (discussed in class), which is weighted by the on-policy distribution. The RMS error was used in Example 9.2 in the textbook.

Please take care to control the random seed in your experiment program to ensure both the agents learn from the same trajectories of experience. The RMSVE requires knowledge of the true value for each of the 1000 states—thus we need to know $v_\pi(s)$ for all S. You can do this yourself using dynamic programming or using the function provided in the assignment folder. You can do this computation inside the experiment program and use RL_agent_message to access the value function learned by your agents, similar to the Monte Carlo agent you implemented for the Gambler's problem.

Our implementation takes 4 minutes per agent to finish the experiment. When you are testing and debugging set number of runs to something small like 2, and number of episodes to something smaller like 500. That will allow you to debug your code without having to wait a long time for it to run. For your final submission makes sure you set runs equal to 30 and number of episodes equal to 2000.

Above we have given suggestions for a good alpha values for each of the two settings. Better alpha values may be possible. Feel free to experiment with different alpha values to improve the performance.

Please submit your experiment program and any additional code used for plotting and computing the RMSE [10 points for the code]. Submit a single plot [10 points for the plot] with two lines, one for each agent.

Bonus Programming exercise. **[10 bonus points].** Implement Dyna-Q on the grid world described in Example 8.1 of the SB textbook. You can modify your windy grid world environment program from the previous assignment. Your task is to recreate something similar to Figure 8.2 (SB) (you dont need to include the picture of the gridworld). Your learning curves should be averaged over 10 runs, with the random seed controlled appropriately such that the number of steps per episode during the first episode is the same for all three parameterizations of Dyna-Q. Note your results are not guaranteed to be exactly the same. You must use the RL-glue software to complete this question. Submit all code and all plots.

Exercise 6.2.

TD is always more efficient than MC since TD will learn the environment. In driving model, TD analysis for second part will not generate noisy analyze like it did in the first part. However, in MC model, there will always be noisy analyze generated since variations for how long you take for the same segment.
MC doesn't know the environment. When you are driving on a new route, the error will accumulate each run step. For example, you drive from home to highway, and there is an error occurred, then another error from highway to work. The error would accumulate to the end. However, in TD, it learned from environment before. It already know how much time should I spend from home to highway is optimal; if I spend more time on this fragment, it will not update result from this attempts. When you driving on the same route(same state) to a new destination, TD has more advantage before it has been learned on this part before, so that you will get your result more sooner and more accurate than MC.

Exercise 6.3

In first episode, it terminated on the left because the only V(A) was changed.
All predictions were the same, all rewards were zeros, All other state transitions has a zero as errors.
By exactly,

$$\alpha[\underset{\substack{\downarrow \\ \text{reward}=0}}{R} + \underbrace{\underset{\substack{\downarrow \\ \text{termination}}}{V(t)} - \underset{\substack{\downarrow \\ VA}}{V(A)}}_{\text{predicted diff should be } 0.5\text{. and terminal state has value } 0\text{.}}] = 0.1[0 + 0 - 0.5] = -0.05$$

Exercise 8.2.

Dyna-Q+ performed better than Dyna-Q because agent of Dyna-Q performed exploratory, and it may miss the optimal action in early time. Dyna-Q+ search the unsearched area more positive, so it could find optimal choice in early time more efficient.