Last name:_____ First name:_____ SID#:_____

Collaborators:_____

CMPUT 366/609 Assignment 6: Prediction with linear function approximation

Due: Tuesday Nov 21st by gradescope

There are a total of 100 points available on this assignment, as well as 15 bonus points

The first question is an exercise from the Sutton and Barto textbook, 2nd edition:

Question 1. [10 points] Exercise 9.4 (*tile coding two dimensions*)

Question 2. **Programming Exercise** [**90 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 [50 points]. Implement three 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 166) 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/sutton/tiles/tiles3.html**

State aggregation: agent three will implement Semi-gradient TD(0) with state aggregation. You can implement state aggregation, as described in Example 9.1. "For the state aggregation, the 1000 states were partitioned into 10 groups of 100 states each (i.e., states 1–100 were one group, states 101–200 were another, and so on)." Or you can use the tile coder with num tilings equal to one and a tile width of 0.1 to achieve state aggregation. For semi-gradient TD(0) use alpha = 0.1.

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

Part 3 [20 points]. Plot the learning curve of your three agents. For this part you will write an experiment program and produce a plot similar to Figure 9.10, except your will plot three performance lines corresponding to the Average RMS error over 1000 states for: (1) Semi-gradient TD(0) with tabular features, (2) Semi-gradient TD(0) with Tile coding features, and (3) Semi-gradient TD(0) with state aggregation. For each of your three agents you will plot the RMS error averaged over 10 runs, over 5000 episodes just like in the figure. 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 10 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 all three 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.

Above we have given suggestions for a good alpha values for each of the three 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 RMSVE [10 points for the code]. Submit a single plot [10 points for the plot] with three lines, one for each agent.

Bonus Question. **Programming exercise**. **[15 bonus points].** Add one more line to the graph produced in part 3, question 2. Implement one of: polynomial basis, radial basis functions, or Fourier basis described in chapter 9. You can use the parameter settings from the book as a starting place (see the polynomial or Fourier representations experiment on page 175). Feel free to test different parameter settings to see if you can get this 4th approach to perform well compared to the tile coding representation. Discuss and report your experiences with this additional representation. Support your discussion with performance plots. Please submit your agent program and plot(s).