CMPUT 366Assignment 6: Control with Function Approximation

Due: Friday Dec 7th by Gradescope

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

Question 1. Programming Exercise—Solving Mountain Car in RL-Glue[**100 points**].

Part 1 [60 points]: You will implement the Sarsa Control algorithm, with replacing eligibility traces and tile coding function approximation. This algorithm is described on the lecture slides (traces.pdf, slide 65) in the course folder. You must use RL-glue.

We have provided an experiment program (“exp_hw6.py”), and the environment program(“env_hw6.py”) for this assignment (check the assignments directory for “A7_code.zip”). **Do not modify the environment program given.** You will implement episodic semi-gradient Sarsa(lambda) with tile coding, replacing traces, and ε-greedy action selection to solve the mountain-car problem. It is a good idea to start with your existing code for semi-gradient TD with tile coding that you wrote in A5. exp_hw6.py assumes your agent will be called “agent_hw6.py”, if you use a different name for you agent make sure you change the experiment program. Please keep max_eps_steps equal to 100,000.

The mountain car problem is described in Example 10.1 page 244 of the textbook. The three actions (decelerate, coast, and accelerate) are represented by the integers 0, 1, and 2. The states are represented by a numpy array of doubles corresponding to the position and velocity of the car. Please look over env_hw6.py to better understand the task.

Your agent program should use the following parameter settings:

memorySize = 2048 (for the tile coder)
num tilings = 8
shape/size of tilings = 8x8
α = 0.1/(numTilings)
λ = 0.9
ϵ = 0.0
initial weights = random numbers between 0 and -0.001
γ = 1

In your last assignment you used the tile coder to convert a one dimensional state into a binary state-feature vector. In this assignment you will use the tile coder to convert a two-dimensional state vector into a binary state-action feature vector (“tiles3.py” can be found inside a6.zip). To do this properly you must: (1) account for the shape of the tilings and ranges of the position and velocity before calling the tile coder, and (2) incorporate the action into the call to the tile coder.

What to submit for Part 1: your agent program (implemented in RL-glue).

Part 2 [10 points]: Produce a learning curve for your Sarsa agent. In this part you will use your agent and the provided experiment program and environment program to produce a learning curve, similar to Figure 10.2 in the book (except your plot will only contain 1 line, for the performance of your Sarsa agent). Plot number of steps per episode (y-axis), over 200 episodes (x-axis), averaged over 50 independent runs. If you wish you may run for more episodes like figure 10.2, but that is not required.

You may use the plotting code in "plot.py", or some other means of plotting.

What to submit for Part 2: your learning curve, and the experiment program your used to produce it (even if you did not modify it), and any other plotting code you used (e.g., plot.py or your own plotting code).

Part 3 [30 points]: In this part you will produce a 3D plot of the cost-to-go function learned by your agent after 1000 episodes like the bottom right subplot in Figure 10.1. Make a 3D plot of minus the state-values used by your agent after 1 run of 1000 episodes. That is plot:

$$-\max_a \hat{q}(s, a, \mathbf{w}) = -\max_a \mathbf{w}^T \mathbf{x}(s, a)$$

as a function of the state, over the range of allowed positions and velocities as given in the book. That is for 50 equally spaced allowed values of the position, for 50 equally spaced allowed values of the velocity query the tile coder for the corresponding state-action features, then use those features to compute the negative of the best action-value according to your agent's learned action-value function after 1000 episodes. For example something like the following python-style pseudo code:

```
fout = open('value', 'w')
steps = 50
for i in range(steps):
   for j in range(steps):
      values = []
      for a in range(numActions):
           # tilecode ([pos = -1.2 + (i * 1.7 / steps), vel = -0.07 + (j * 0.14 / steps)], action = [a]) => inds
           # values.append(Q(inds,w))
       # height = max of values
      fout.write(repr(-height) + ' ')
   fout.write('\n')
fout.close()
```

For this plot we only do a single run (no averaging), and compute the data for the plot inside the agent program, after the completion of episode 1000. You may modify the experiment program given to you.

What to submit for Part 3: the experiment program, your agent program (where you compute the values for the 3D plot), your plot of the cost-to-go function, and any additional code you used to produce the 3D plot.

Bonus Question [10 points]: Find a better setting of the parameters for your Sarsa agent on the Mountain Car problem. Experiment with the parameters listed above, try finding a better setting (except γ, γ must be equal to 1.0). You can also change the kind of traces used and the tile-coding strategy (number of tilings, size and shape of the tiles). Your job is to outperform the performance obtained with the original parameter settings.

As an overall measure of performance on a run, use the sum of all the rewards received in the first 200 episodes, averaged over runs (you will need to modify exp_hw6.py a little). Note the RL_return() function returns the total reward achieved by the agent during an episode. Find a set of parameters that improves this performance measure by two standard errors. To show the improvement, you must do many runs with the standard parameters and then many runs with your parameters, and measure the mean performance and standard error in each case (a standard error is the standard deviation of the performance numbers divided by the square root of the number of runs). If the difference between the means is greater than 2.5 times the larger of the two standard errors, then you have shown that your parameters are significantly better. It is permissible to use any number of runs greater or equal to 50 (note that larger numbers of runs will tend to make the standard errors smaller).

What to submit: (1) Report the alternate parameter settings (or other variations) that you found, the means and standard errors you obtained for the two sets of parameters, and the number of runs you used in each case. (2) Once you have found your favorite set of parameters, make a learning curve based on 50 runs of 200 episodes with them, like you did in Q1 Part 2. Please submit that plot.