CMPSCI 687 Homework 3

Due November 9, 2017, 11pm Eastern Time

Instructions: This homework assignment consists of only a programming portion. You may discuss concepts with other students, but should not discuss implementation details or results. The assignment should be submitted as a single .zip (not .tar or .tar.gz) file on Moodle. The automated system will not accept assignments after 11:55pm on the due date specified above.

Programming Portion (100 Points Total)

For this assignment you will implement q-learning and Sarsa using linear function approximation (and the Fourier basis) on the mountain car domain. You should use the mountain car domain as defined on pages 214 and 215 of Sutton and Barto's book (first edition, which is available free online), except where the initial state always has position -0.5 and velocity 0. You should modify the domain to terminate episodes after 20,000 time steps (do not give the current time step to the agent—it should still only observe the position and velocity). You may use any language that you want. Part 3 below took 3 CPU hours on my desktop using a C++ implementation. Make sure to finish your code early enough to be able to generate the plot for part 3 before the deadline. You must write the environment, agent, and the code interfacing the two entirely on your own and from scratch (do not use any RL libraries, and do not look at code in existing RL libraries for inspiration).

Recall that the q-learning update with linear function approximation is:

$$\delta_t = r_t + \gamma \max_{a'} w^\intercal \phi(s_{t+1}, a') - w^\intercal \phi(s_t, a_t)$$
$$w_{t+1} = w_t + \alpha \delta_t \phi(s_t, a_t),$$

and the Sarsa update has the same update for w_t, but uses the alternate equation for the TD-error:

$$\delta_t = r_t + \gamma w^\intercal \phi(s_{t+1}, a_{t+1}) - w^\intercal \phi(s_t, a_t),$$

where α is a step size parameter and $\phi(s_t, a_t)$ is a feature vector. Recall also that the q-learning update should be applied before a_{t+1} is generated, while the Sarsa update should be applied after a_{t+1} is generated.

Use a tabular representation over the actions, and linear function approximation over the states. For the linear approximation component, use the Fourier basis (http://psthomas.com/papers/Konidaris2011a.pdf). That is, let $\psi(s) \in \mathbb{R}^n$ denote the features generated by the Fourier basis for state s. Think of the actions as integers, $\mathcal{A} = \{1, 2, 3, \ldots, m\}$. Then $\phi(s, a) \in \mathbb{R}^{nm}$ and all entries in $\phi(s, a)$ are zero, except for the elements $(a-1)n$ through $an - 1$, which are $\psi(s)$ (assuming array indices start with zero, not one). A careful implementation can avoid constructing this entire sparse feature vector (when

checking run times in my reference implementation, I did **not** use this more efficient careful implementation). Do **not** use different step sizes for each feature as described in the section "Scaling Gradient Descent Parameters" of the linked paper.

Use initial weight vector $w_{-1} = 0$. Use ϵ-greedy action selection. That is, in state s, with probability $1 - \epsilon$ select an action uniformly randomly from the set of actions, a, that maximize $w^\intercal \phi(s, a)$, and with probability ϵ select an action uniformly randomly from $\mathcal{A}$. Also, the value of terminal states is always zero. This must be included in your implementation, and can be implemented by replacing $w^\intercal \phi(s, a)$ with zero if s is a terminal state (or the absorbing state). Although our goal is to maximize the expected return with $\gamma = 1$, you should treat γ as a hyperparameter of the algorithm, but plot your results using $\gamma = 1$.

There should be no ambiguity in this assignment: your code should produce exactly the same results that our reference code produces (except for the random numbers and differences in floating point computations). If you reach a point where you are uncertain about how something should work, please ask.

1. (30 Points) Include your code, along with compilation instructions in the .zip file. Although we do not have the resources to compile and verify every student's code, we will randomly select a sub-sample of submissions and will verify that they compile, run, and produce the reported results. Submitting code that does not compile, run, and produce the results that you report below, or code that is similar to code found online or a peer's code, will be reviewed as a possible case of academic dishonesty.

2. (40 Points) Find hyperparameters that work well for each method (possibly different hyperparameters for each method). The hyperparameters are α, γ, ϵ, and the order of the Fourier basis. For the best hyperparameters that you find, include a plot showing the mean undiscounted return on the vertical axis and the number of training episodes on the horizontal axis. Include one curve for each method (two curves total), average the mean undiscounted returns over at least 500 trials, and include error bars.

3. (30 Points) Run both methods with the parameters $\alpha = 0.05$, $\gamma = 1.0$, $\epsilon = 0.5$, and the first-order Fourier basis. Run 10,000 trials and plot the performance the same way that you did for the previous section, where the horizontal axis spans episodes $0 - 200$ and the vertical axis shows returns from -1000 to 0. If your plot does not precisely match our reference plot, you will not get partial credit for this portion. Report how long it took for your code to run.