CMPSCI 687 Homework 4

Due December 7, 2017, 11pm Eastern Time

75 Points Total

Instructions: 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 .zip should include a .pdf file with your response to all of the questions and with the results plots included. The .zip should also include your source code. The automated system will not accept assignments after 11:55pm on the due date specified above.

If you do not include your source code, you will get a 0/75 on this assignment. If the code that you include does not reproduce the results that you report, it will be handled as a possible academic honesty violation.

For this assignment you should (but need not necessarily[1]) use the provided code. When run the code allows you to select an RL algorithm, an environment, and then asks you to specify hyperparameters. You then must specify the number of trials (we recommend more than 100 to get consistent results) and how many episodes to run in each trial (we leave this to you to determine in this assignment). After running, the code produces an output .csv file that starts with `out_`, which contains the mean return during each episode and the sample standard deviation of the returns, which should be used to create error bars. You should use your own desired plotting method to translate this .csv file into a plot.[2]

For this assignment you will implement Q(λ), and Actor-Critic algorithm, and REINFORCE with a constant baseline.

1. (25 Points - Q(λ)) You will implement ϵ-greedy Q(λ) using linear function approximation and the Fourier basis. The code that you must add is in QLambda.cpp. This implementation uses the "more efficient" form of linear function approximation with discrete actions, as mentioned in the previous assignment. That is, the feature vector, $\phi(s, a)$, is sparse and is not computed. Instead, we only compute features given the state, i.e., $\phi(s)$, and take the dot product of these features with a segment of the weights, w, which depends on the action. See the `getAction` function in `QLambda.cpp` for more details about how the weights are used to compute q-values. You may wish to review the `segment` function of

[1] You may use any language that you want, but you must code the agents entirely from scratch (you may not use or reference existing agent code). You may use existing code for the environments, but you must ensure that they are equivalent to the versions that we provide (e.g., the gridworld that we use is one that we made up). We strongly recommend using the provided code and C++ because we provide much of the code and the implementation is computationally efficient and threaded.

[2] For example, if you open the .csv in excel and select the first column and make a line plot, this will produce the standard learning curve with horizontal axis "Episode" and vertical axis "Average Discounted Return", which you can then add error bars to using the second column.

vectors documented here: https://eigen.tuxfamily.org/dox/group__TutorialBlockOperations.html.

(A) Find parameters that work well on the provided Gridworld domain. Report your parameters along with a plot showing the resulting performance.

(B) Find parameters that work well on the provided Mountain Car domain. Report your parameters along with a plot showing the resulting performance.

(C) Find parameters that work well on the provided Cart Pole domain. Report your parameters along with a plot showing the resulting performance.

(D) Find parameters that work well on the provided Acrobot domain. Report your parameters along with a plot showing the resulting performance.

(E) Comment on the difficulty of finding optimal hyper-parameters for this algorithm. Is it getting easier as you have more experience with RL algorithms? Which parameters did the algorithm appear to be most and least sensitive to? Did any hyperparameter values surprise you?

2. (25 Points - Actor-Critic) You will implement an actor-critic using linear function approximation and eligibility traces for both the actor and the critic. The actor will use softmax action selection with linear function approximation over states and the Fourier basis. See the `getAction` function in `ActorCritic.cpp` for more details about how the policy is parameterized. The update that you should implement is:

$$\begin{aligned}
\delta &\leftarrow R_t + \gamma v^\intercal \phi(S_{t+1}) - v^\intercal \phi(S_t) \\
e_v &\leftarrow \gamma \lambda e_v + \phi(S_t) \\
v &\leftarrow v + \alpha_{\text{critic}} \delta e_v \\
e_\theta &\leftarrow \gamma \lambda e_\theta + \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta} \\
\theta &\leftarrow \theta + \alpha_{\text{actor}} e_\theta,
\end{aligned}$$

where α_{actor} and α_{critic} are two (possibly different) step sizes, the eligibility traces, e_v and e_θ are initialized to zero at the start of each episode, and $\phi(S_{t+1})$ is the zero-vector if S_{t+1} is a terminal state. We recommend that you use a separate function, `dlnpi` to implement $\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}$, and then use this function in the train functions.

Notice that this algorithm is the actor-critic presented on page 277 of the November 5'th draft of Sutton and Barto's new book. It is the same as the actor-critic algorithm that we wrote on the board in class, but with eligibility traces added to the actor.

(A) Show a derivation of $\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}$ (this is something that you must have done in order to be able to code the method).

(B) Find parameters that work well on the provided Gridworld domain. Report your parameters along with a plot showing the resulting performance.

(C) Find parameters that work well on the provided Mountain Car domain. Report your parameters along with a plot showing the resulting performance.

(D) Find parameters that work well on the provided Cart Pole domain. Report your parameters along with a plot showing the resulting performance.

(E) Find parameters that work well on the provided Acrobot domain. Report your parameters along with a plot showing the resulting performance.

(F) Comment on the difficulty of finding optimal hyper-parameters for this algorithm. How does this algorithm compare to Q(λ)?

3. (25 Points - REINFORCE with Constant Baseline) Implement the REINFORCE algorithm using a baseline, $b(s)$, that is a constant, i.e., that does not depend on the state. The code that you must fill in is in `REINFORCE.cpp`. To simplify this portion, we have implemented the baseline update for you—you need only implement the computation of the returns and the policy update, along with the same `dlnpi` function from the actor-critic algorithm.

   Recall that the REINFORCE update for time step t is:

$$\theta \leftarrow \theta + \alpha(G_t - b)\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta},$$

   where b is the baseline. In the code **1)** we store the entire episode and then perform the updates for each time step after the episode has completed, so that we can compute the returns G_t, and **2)** we define an array, `errors`, which stores $G_t - b$ for each time step, t.

(A) Find parameters that work well on the provided Gridworld domain. Report your parameters along with a plot showing the resulting performance.

(B) Attempt to find parameters that work well on the provided Mountain Car domain. Report the best parameters that you find. Comment on why you think that REINFORCE does not perform well on Mountain Car relative to the other methods.

(C) Find parameters that work well on the provided Cart Pole domain. Report your parameters along with a plot showing the resulting performance.

(D) Find parameters that work well on the provided Acrobot domain. Report your parameters along with a plot showing the resulting performance.

(E) Comment on the difficulty of finding optimal hyper-parameters for this algorithm. How does this algorithm compare to Q(λ) and the actor-critic algorithm?