6140: Programming Assignment 4

You will implement *Q-learning* and Sarsa(λ) on the gridworld problem described below with ϵ-greedy as policy.

Please view this assignment as a chance to code, analyze the methods and subsequently document the observations. This assignment is to be completed individually (i.e., no teams). You have to create all deliverables yourself from scratch. In particular, you are not allowed to look at or copy someone else's code/text and modify it. If you use publicly available code/text, you need to cite the source in your report! Cheating and other acts of academic dishonesty will be referred to OSCCR (office of student conduct and conflict resolution) and the College of Computer Science.

All programming assignments must be completed in **Python**. You need to make sure your code is **runnable** before you submit it. We will grade mainly based on your document submission, so make sure you have a clear explanation in the .pdf file including plots, running results, and explanations.

To submit your solutions,

1. Use your CCIS Github account and access the repository named ml-<your_name>
2. Create a new folder for this assignment: (e.g. PA4)
3. Push your solutions for this assignment to that folder.
4. Your submission should include one .py file and one .pdf file. Please name these files with your name (e.g. Kechen_Qin_PA4.pdf).

Please submit your solution by the due date shown online (usually two weeks from release). Late assignments will be penalized by 10% for each day late. For example, if you turned in a perfect programming assignment two days late, you would receive an 80% instead of 100%.

1 Domain

Our domain is the gridworld and is specified just by it's size s. It is a 2-dimensional world of $s \times s$, where the agent starts in the lower left corner, and the right upper corner is the only terminal state.

The reward for being in any of the states is -0.1, except for being in the terminal state, which yields a reward of 5. The possible actions are *left, right, up, & down*, which all have a 0.8 probability of *succeeding*. If the action fails, any action is chosen uniformly(including the *succeeding* action). If the action causes the agent to 'bump into the wall', the **no** extra cost is involved (the system still generates the usual reward of -0.1, however), and the agent will stay in the same state. A good policy will move the agent to the terminal state, the optimal policy should do so in the fastest way possible.

2 I/O & parameters

The program takes the following parameters (note that the defaults are not necessarily the optimal values):

- alg: either 'q' or 's', to determine the algorithm used
- size(default 5): size of gridworld
- gamma(default 0.99): the discount factor
- exps(default 500): amount of experiments to run
- eps(default 500): amount of learning episodes per experiment
- epsilon(default 0.1): the epsilon in ϵ-greedy policy execution
- alpha(default 0.1): the learning rate
- lambda(default 0): the parameter for $Sarsa(\lambda)$

Example usage of the program is: **your_python_script.py -alg q -size 3 -exps 100 -gamma 0.8 -epsilon 0.1**. I assume it is possible to leave out flags wherever necessary, so please make sure you do not assume some sort of order in the flags or that all will be set.

3 Experiments

In the deliverables you are asked multiple times to provide an analysis on some range of values for a specific parameter. Provide the following graphs and **discuss** them whenever an analysis is being asked. **If a flag or parameter is not mentioned, then use its default value.**

- The average amount of time steps necessary to reach the goal for the learned policy, per amount of learning episodes (x-axis)

- The maximum *q-value* for the start state over 500 episodes for each of the values for *alpha*, with the number of episodes on the x-axis (so a graph where the learning of the $Q(s_{start}, a_{up})$ value is plotted for each episode)

4 Analysis

The report should include the following:

1. A brief list of parts of your program that are not working as desired, and anything that does work but you would like to share about your approach or code

2. Run and analyze *Q-learning* on a 5×5 grid with *learning rate alpha*:

- 0
- 0.01
- 0.05
- 0.1
- 0.5
- 1

3. Repeat 2, but compare the results of various values for *epsilon*:

- 0
- 0.1
- 0.25
- 0.5
- 1

4. Repeat the experiments in 2 and 3 for $Sarsa(\lambda)$. Additionally, do the same analysis on $Sarsa(\lambda)$ with the following values for *lambda*:

- 0
- 0.25
- 0.5
- 0.75

- 1

5. Given the analysis on *Q-learning*, pick the best parameter values for ϵ & α to solve a 10×10 instance of Gridworld. Show how many episodes it takes to converge to the optimal policy, and **explain** why you have chosen these values.

6. Given the analysis on $\mathtt{Sarsa}(\lambda)$, pick the best parameter values for ϵ, α & λ to solve a 10×10 instance of Gridworld. Show how many episodes it takes to converge to the optimal policy, and **explain** why you have chosen these values.

7. How would one in general choose ϵ, α & λ? How do they affect *Q-learning* and $\mathtt{Sarsa}(\lambda)$?

5 Advice

- Start with implementing *Q-learning*
- Debug on small domains
- Begin by writing some sort of plotting or display function of the *q-values*.Try to reason on what you expect to happen to those values and verify(i.e.: do the converged values correspond to a reasonable policy).
- Grading will not only be based on the quality of code, but on the quality of there search as well. Unless copied from somewhere else, its beauty or style will not be graded. It is important to be efficient up to some point, it must finish running the grading script.
- Plotting is a large part of this assignment, try to automate them to avoid repeating manual work.