VRIJE UNIVERSITEIT BRUSSEL

Computational Game Theory
Assignment 3

Deadline: 23 December 2018

Submission Instructions

- We have organized a Github classroom for this assignment. Please create a Github account (if you do not have one already) and use this link: `https://classroom.github.com/a/yib4TPxI` to accept the assignment and to create inside the classroom your own private repository. Commit there your code and report (preferably in PDF form), before the deadline.
- Do not forget to add your name, student number and affiliation in your report.
- Late submissions will be penalized.
- If you have any questions send an email to: `roxana@ai.vub.ac.be`.

1 N-Armed Bandit

For this exercise you will have to implement the N-Armed bandit problem using each of the following action selection methods:

- Random
- ϵ-Greedy with parameter ϵ
- Softmax with parameter τ

All Q_{a_i} are initialized to 0. The rewards are subject to noise according to a normal probability distribution with mean $Q^*_{a_i}$ and standard deviation σ_i, $\forall i = 1, \ldots, N$.

You are provided with 5 instances of reward distributions (Tables 1-5) for the N-Armed bandit problem. You only have to select **one** of the tables, according to the following criteria:

- Identify the last number of your VUB/ULB enrolment number
- If the number is 0 or 9, select Table 1
- If the number is 1 or 8, select Table 2
- If the number is 2 or 7, select Table 3
- If the number is 3 or 6, select Table 4
- If the number is 4 or 5, select Table 5

Let $N = 4$, $Q^*_{a_i}$ and σ_i as defined in your selected table.

Action	$Q^*_{a_i}$	σ_i
action #1	2.3	0.9
action #2	2.1	0.6
action #3	1.5	0.4
action #4	1.3	2

Table 1: Last number is 0 or 9.

Action	$Q^*_{a_i}$	σ_i
action #1	1.2	1.1
action #2	0.4	0.6
action #3	1	0.8
action #4	0.3	2

Table 2: Last number is 1 or 8.

Action	$Q^*_{a_i}$	σ_i
action #1	1.3	0.9
action #2	1.1	0.6
action #3	0.5	0.4
action #4	0.3	2

Table 3: Last number is 2 or 7.

Action	$Q^*_{a_i}$	σ_i
action #1	2.4	0.9
action #2	1.3	2
action #3	1	0.4
action #4	2.2	0.6

Table 4: Last number is 3 or 6.

Action	$Q^*_{a_i}$	σ_i
action #1	2.1	1.2
action #2	1.1	0.8
action #3	0.7	2
action #4	1.9	0.9

Table 5: Last number is 4 or 5.

Plotting

For each of the following exercises, run your simulation for 1000 time steps and plot the following graphs:

- One combined plot of the average reward for each algorithm (One graph)
- One plot per arm showing the $Q^*_{a_i}$ of that action along with the *actual* Q_{a_i} estimate over time (all action selection methods should be depicted on the same plot!) (One graph per arm).
- For each algorithm, plot a histogram showing the number of times each action is selected (One graph per action selection strategy).

See Section 2.2 of "Reinforcement Learning, An Introduction" by Sutton and Barto for a more comprehensive explanation of the problem[1].

Remarks:

- You can run each experiment (i.e., the 1000 time steps for each action selection method) for multiple trials and then average over the results (plotting the obtained mean together with an error measurement, such as standard deviation or standard error).
- **Provide comments and explanations for each of the obtained plots/results! Provide answers to all the questions explicitly asked in the following exercises!**

1.1 Exercise 1

After selecting the appropriate table, run the following algorithms and provide the requested plots:

- Random
- ϵ-Greedy with parameter $\epsilon = 0$
- ϵ-Greedy with parameter $\epsilon = 0.1$
- ϵ-Greedy with parameter $\epsilon = 0.2$
- Softmax with parameter $\tau = 1$
- Softmax with parameter $\tau = 0.1$

Comment the results. Which algorithms arrive close to the best arm after 1000 iterations? Which one arrives there faster? Explain your findings.

1.2 Exercise 2

Re-run Exercise 1 doubling the standard deviations of each arm, and comment the plots and results. **Discuss the results.** Is this problem harder or easier to learn? Does the performance of the algorithms change significantly? Which of the above performs best now?

1.3 Exercise 3

Re-run exercise 1 with two additional algorithms, and plot their results. The new algorithms have a time-varying parameter, which depends on the iteration number $t = 1, \ldots, 1000$ as follows:

- ϵ-Greedy with parameter $\epsilon(t) = 1/\sqrt{t}$
- Softmax with parameter $\tau = 4 * \frac{1000-t}{1000}$

Discuss the results. Are these algorithms better than the ones with a fixed parameter?

[1] http://incompleteideas.net/book/bookdraft2017nov5.pdf

2 Windy Gridworld

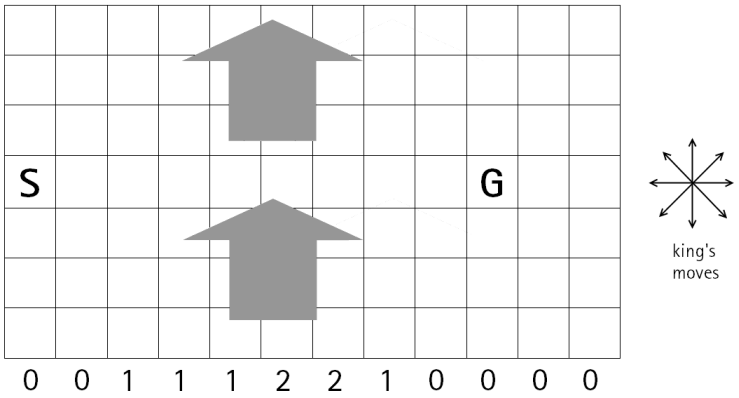

Figure 1: Gridworld in which movement is altered by a location-dependent, upward stochastic "wind".

Figure 1 shows a standard gridworld, with start (S) and goal (G) cells, but with one difference: there is a crosswind upward through the middle of the grid. The available actions in each cell are the king's moves — 8 actions in total for each cell. If any action would bring you outside the gridworld, you end up in the nearest cell (e.g. going northeast in the top left cell will bring you one cell to the right). In the middle region the resultant next cells are shifted upward by a stochastic "wind", the mean strength of which varies column by column. The mean strength of the wind is given below each column, in number of cells shifted upward. Due to stochasticity, the wind sometimes varies by 1 from the mean values given for each column. That is, a third of the time you are shifted upwards exactly according to the values indicated below the column, a third of the time you are shifted one cell above that, and another third of the time you are shifted one cell below that. For example, if you are two cells to the left of the goal (wind = 1) and you move *right*, then one-third of the time you end up one row north-east of that cell, one-third of the time you end up two rows north-east of that cell, and one-third of the time you end up at the same row east of that cell (left of the goal). A crosswind of 0 also implies there is no stochasticity as well when choosing an action (e.g., if you are in the cell under the goal, with wind = 0, and decide to move upwards, you will arrive at the goal 100% of the time). Note that the wind will only affect the cell you are in, and not the one you will arrive at.

Implement the Q-learning algorithm[2] in the above problem with $\alpha = 0.1$, $\gamma = 0.9$ and initial $Q(s, a) = 0$ for all s, a. Each action generates a reward of $r_s = -1$, except for the actions that lead immediately to the goal cell ($r_g = 10$). Use the ϵ-Greedy action selection method with $\epsilon = 0.2$.

2.1 Plotting

Run the Q-learning algorithm for 5000 episodes and at the end of the learning plot the following *on a single graph*:

- the grid of the Windy gridworld
- arrow(s) in each cell indicating the best action(s)[3]
- one run from start to goal with the Greedy action selection (no exploration) on the learned policy showing the action selected in each cell (use here a different colour)
- for the same run above, plot a line displaying the path taken (i.e. all the visited cells) during that run

Figure 2 shows an example of a learned policy together with a sample run following that policy (applied to a slightly modified scenario).

In addition, plot the following two graphs:

[2]The pseudo-code can be found at http://incompleteideas.net/book/bookdraft2017nov5.pdf
[3]In Matlab you can use the *quiver* function for this.

- Total collected reward per episode versus the episode number
- Number of steps to reach the goal per episode versus the episode number

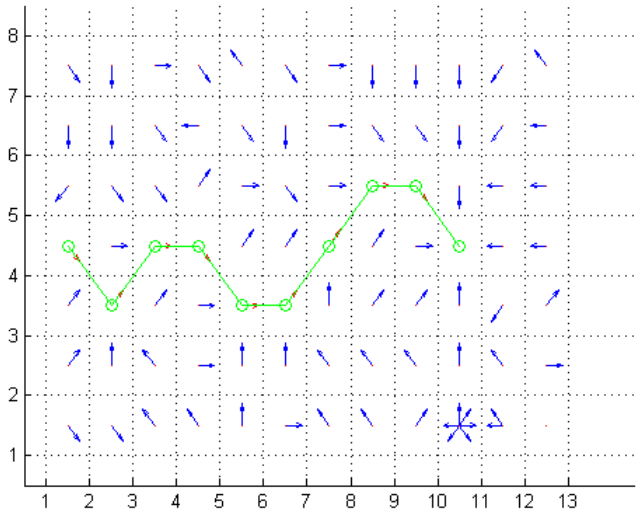

Figure 2: An example of a learned policy (arrows = best action) and a sample run following that policy (green path = visited cells). Note that in this example the scenario has been modified.

2.2 Discussion

Discuss and interpret all your results and plots!

Additionally, discuss the following questions:

1. How will the learning process change if we make $\epsilon = 0$, while keeping the other parameters the same?
2. How will the learning process change if we make $\gamma = 1$, while keeping the other parameters the same?

3 Graphical Coordination Game

For this exercise you will have to implement the Win-Stay-Lose-probabilistic-Shift algorithm [1] in a graphical coordination game (presented in the lecture on December 13, 2018) with the following characteristics:

- Use 26 agents arranged in a ring topology (such that each agent will have 2 neighbours)
- For each round use pairwise interactions (i.e., select at random an agent from the existing 26, then select randomly one of his 2 neighbours to perform an interaction)
- Each agent starts with a randomly initialized action. After an interaction, the actions of both agents should be updated.

Remember that in the case of a coordination game the convergence state is reached when all agents have selected the same action (regardless which one that is). To check if the algorithm has converged you can always have a check at the end of an iteration and see if all the 26 agents have selected the same action. If this is the case for the last 10 iterations, then you can consider that the algorithm has converged.

3.1 Exercise 1

For this exercise you will have to perform a study of how the convergence time varies depending on the value of the α parameter and the number of actions. Run the algorithm for $\alpha \in \{0, 0.1, 0.2, \ldots, 0.8, 0.9, 1\}$ and $num\_actions \in \{2, 3\}$. Run each experiment (i.e., for each combination of α and $num\_actions$ value) for 200 trials in order to be able to average over your results and record for each run, **if it converges**, the convergence time.

Write down the payoff matrices for the coordination games with 2 agents − 2 actions and 2 agents − 3 actions.

Comment the results. What is the role of α for the Win-Stay-Lose-probabilistic-Shift algorithm? How is the convergence time affected by the value of α? Which α value(s) result in the fastest convergence time for your setting? For which α values the algorithm fails to converge? How is the convergence time affected by the number of actions? Explain any other findings you deem necessary.

Plotting

Plot a single figure with the convergence time on the y-axis and the α values on the x-axis. Display for every α the mean of the recorded convergence time together with error bars representing the 90% confidence interval of the mean or the standard deviation. The plot should contain 2 lines, one for each studied number of actions. Consider using a logarithmic scale, if it improves the readability of the plot.

NOTE: The choice of the programming language is free. Save yourself work by creating an abstraction for the number of actions and all parameters, so you can re-use your code. **Do not forget to always explain your results!**

References

[1] M. Mihaylov, K. Tuyls, and A. Nowé. A decentralized approach for convention emergence in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 28(5):749–778, 2014.