COMP-597: Reinforcement Learning - Assignment 1

Posted Thursday January 11, 2024
Due Thursday, January 25, 2024

The assignment can be undertaken individually or in teams of two. Alongside this document, you have been provided with a partially completed .IPYNB (Jupyter Notebook) file. Your task is to complete the .IPYNB file by adding the necessary code, displaying results, and providing explanations for questions. While following the notation of predefined classes and functions is not mandatory, it is recommended. Also, you may adapt notations to better align with your thought process and coding style.

Within the .IPYNB file, you will find some pre-existing graphs. These are included not for result comparison, but rather to guide you on the expected format and content of the plots you are required to generate. DO NO COMPARE THEM WITH YOUR RESULTS!

Ensure that you only submit the completed .IPYNB file for evaluation. Your submission should include a comprehensive set of code, results, and explanations. If you have any questions or require clarification, feel free to reach out for assistance on the Ed. We will try to answer your questions within 24 hours. Good luck with your assignment!

Bandit algorithms [100 points]

For this assignment, you will carry out some experimentation with bandit algorithms, in order to help you understand what we discussed in class, and to get used to the way in which we will run experiments for other assignments as well.

1. [5 points] Write a small simulator for a Bernoulli bandit with k arms. The probability of success p_i for each arm $i \in \{1, ...k\}$ should be provided as an input. The bandit should have a function called "sample" which takes as input the index of an action and provides a reward sample. Recall that a Bernoulli bandit outputs either 1 or 0, drawn from a binomial distribution of parameter p_k. Test your code with 3 arms of parameters $q_* = [0.5 + \delta, 0.5, 0.5 - \delta]$, with $\delta = 0.1$. Generate and save a set of 50 samples for each action. For the test, plot one graph for each action, containing the reward values obtained over the 50 draws, the empirical mean of the values, and the true q_* for each arm. Each graph will have an x-axis that goes to 50, two horizontal lines (true value and estimated value) and a set of points of value 0 and 1.

2. [5 points] Code the rule for estimating action values with a fixed learning rate α, in a function called `update`, and using the incremental computation of the mean presented, in a function called `updateAvg`. Using the previous data, plot for each action a graph showing the estimated q value as a function of the number of samples, using averaging as well as $\alpha = 0.01$ and $\alpha = 0.1$, and the true value. Each graph should have three curves and a horizontal line.

3. [10 points] Repeat the above experiment 100 times, starting with action value estimates of 0. Each run will contain 100 samples for each action. Plot the same graph as above, but where the curves have the average and standard error over the 100 runs. Explain in 1-2 sentences what you observe. Which of the α values is better? How do they compare to averaging? If you wanted to optimize further, in what range of α would you look for better values?

4. [20 points] Code the ϵ-greedy algorithm discussed in class, with averaging updates, with ϵ provided as an input. Note that if multiple actions have maximum value, you should choose randomly among them (and not always pick eg the one with the lowest index). You will run 100 independent runs, each consisting of 1000 time steps. Plot the following graphs:

   (a) The reward received over time, averaged at each time step over the 100 independent runs (with no smoothing over the time steps), and the standard error over the 100 runs

   (b) The fraction of runs (out of 100) in which the first action (which truly is best) is also estimated best based on the action values

   (c) The instantaneous regret l_t (as discussed in lecture 3) (averaged over the 100 runs)

   (d) The total regret L_t up to time step t (as discussed in lecture 3) (averaged over the 100 runs)

   Generate this set of graphs, for the following values of ϵ: 0, 1/8, 1/4, 1/2, 1. Note that the x-axis for the graph will go up to 1000, and you should have on each graph a set of 5 curves, one for each value of ϵ. Explain what you observe in the graphs and discuss the effect of ϵ you observe.

5. [5 points] For $\epsilon = 1/4$ and $\epsilon = 1/8$, plot the same graphs for $\alpha = 0.1$, $\alpha = 0.01$, $\alpha = 0.001$ and averaging. Explain in 2 sentences what you observe.

6. [20 points] Write a function that implements the UCB algorithm discussed in class. Set $c = 2$. Plot the same graphs as above for $\alpha = 0.1$, $\alpha = 0.01$, $\alpha = 0.001$ and averaging. Explain briefly the behavior you observe.

7. [20 points] Write a function that implements the Thompson sampling to be discussed in class. Plot the same graphs as above. Explain briefly the behavior you observe.

8. [5 points] For each of the algorithms, pick the best hyper-parameter combination you have observed (explain how you decided what "best" means). Plot together the curves for this setting. Comment on the relative behavior of the different algorithms.

9. [10 points] Let us now consider a non-stationary problem. Let $\delta = 0.1$ and imagine that after 500 time steps, the parameter of actions 2 and 3 become $0.5 + 2\delta$ and $0.5 + 3\delta$ respectively. Run ϵ-greedy and UCB for a fixed value of $\alpha = 0.1$ and the averaging value estimation. For ϵ use values $1/4$ and $1/8$. Plot *only the reward graph* as above (you should have 2 lines for each of the ϵ values, two lines for UCB and one for Thompson sampling). Explain what you see in the graph. Based on these results, which algorithm is best suited to cope with non-stationarity?