# CS 747 (Autumn 2020): End-semester Examination

Instructor: Shivaram Kalyanakrishnan

To be submitted by 11.55 p.m., December 2, 2020

**Note.** Provide justifications/calculations/steps along with each answer to illustrate how you arrived at the answer. You will not receive credit for giving an answer without sufficient explanation.

**Submission.** Write down your answer by hand, then scan and upload to Moodle. Write clearly and legibly. Be sure to mention your roll number.

**Question 0.** Have you read the instructor's message with subject "End-semester Examination", announced through Moodle on November 16, 2020? Have you followed the rules laid out in that message, in letter and in spirit? Specify related observations or comments, if any. [It is mandatory for you to answer this question.]

**Question 1.** Consider a stochastic $n$-armed bandit, $n \geq 2$, in which the arms give 0–1 (Bernoulli) rewards. We restrict our attention to instances $I$ in which the means of the arms all lie in $(0, 1)$, and moreover, no two arms have the same mean. In any such instance $I$, let $a_2$ be the arm with the *second* highest mean, and let $u_2^T$ be a random variable denoting the number of pulls of $a_2$ over a horizon $T \geq 1$.

Describe a deterministic algorithm $L$, which, for every qualifying bandit instance $I$, achieves

$$\lim_{T \to \infty} \frac{\mathbb{E}_{L,I}[u_2^T]}{T} = 1.$$

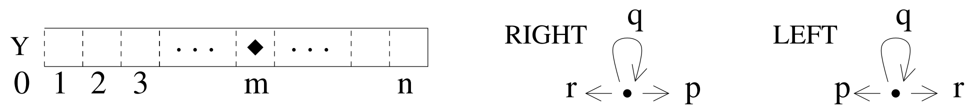
In other words, the number of pulls of arms other than $a_2$ under $L$ must be a vanishing fraction of the horizon. Provide a proof sketch that $L$ satisfies this property; no need for a detailed mathematical working. [4 marks]

**Question 2.** Consider the same family of bandit instances as in Question 1: $n$ arms, Bernoulli rewards, distinct means drawn from $(0, 1)$. In this question, we consider a generalisation of the setting presented in the lectures. Specifically, we consider algorithms $L$ that must pull $k$ distinct arms at each time step, where $1 \leq k < n$ is a fixed parameter (in the lectures we took $k = 1$). The reward for any time step is taken as the sum of the rewards returned by the $k$ arms pulled.

Define the (expected cumulative) regret $R(L, I, T)$ of algorithm $L$ on instance $I$ at horizon $T \geq 1$ as the difference between the maximum expected reward that can be achieved in $T$ steps on $I$ and the expected reward obtained by $L$ on $I$ in $T$ steps.

Does there exist an algorithm $L$ that for every instance $I$, satisfies $R(L, I, T) \leq C_I \ln(T)$ for $T \geq 1$, with $C_I$ a constant completely determined by $I$? We know that the answer is affirmative for $k = 1$, so you must specifically address $k \geq 2$. Either prove the existence of $L$ achieving logarithmic regret as defined above, or explain why such an algorithm does not exist. [4 marks]

**Question 3.** You are right outside the entrance to a cave, as seen in the figure below. The cave has $n > 1$ chambers numbered $1, 2, \ldots, n$. You (denoted "Y" in the figure) are in chamber "0", a convenient name for "just outside". The cave is linear, with each chamber connected to one on the left and one on the right (except that the last chamber, $n$, has no right neighbour).



You have arrived at the cave to go in and retrieve a diamond, which known to be in chamber $m \in \{1, 2, \ldots, n\}$. Since caves are dangerous places, you would like to be done with your mission as quickly as possible. Unfortunately the cave has a slippery interior, and so your two available actions—RIGHT and LEFT—can have unintended consequences. When you take the RIGHT action from chamber $i \in \{1, 2, \ldots, n-1\}$, you move to chamber $i+1$ with probability $p$, you remain in chamber $i$ with probability $q$, and you go to chamber $i-1$ with probability $r$, where $p, q, r \in [0, 1]$, $p + q + r = 1$, and $p > r$ (so displacement in the intended direction is more probable than displacement in the opposite direction). Similarly, when you take the LEFT action from chamber $i \in \{1, 2, \ldots, n-1\}$, you move to chamber $i-1$ with probability $p$, you remain in chamber $i$ with probability $q$, and you go to chamber $i+1$ with probability $r$. From chamber 0, taking LEFT keeps you (deterministically) in chamber 0; action RIGHT moves you to chamber 1 with probability $p$ and keeps you in chamber 0 with probability $1 - p$. From chamber $n$, taking LEFT and RIGHT take you to chamber $n-1$ with probabilities $p$ and $r$, respectively, but keep you in chamber $n$ with the remaining probabilities. You can assume that as soon as you enter chamber $m$ for the first time, the diamond (instantly) drops into your pocket and stays there (no additional action or time needed to collect it). Hence, your task can be viewed as that of visiting chamber $m$ (at least once) and thereafter returning to chamber 0.

3a. You have enough time to formulate your plan before entering the cave. However, due to the slippery surface inside, the actual number of steps taken by your plan is bound to be random. How would you plan so that the *expected* number of steps to retrieve the diamond and return to chamber 0 is minimised? Prove that your plan is *optimal*, in the sense that you cannot finish in fewer (expected) steps. We are using the informal word "plan" here in place of the more usual "policy" because *we* have not formalised the task in the language of MDPs. It is up to you to do so (and then, as usual, you can think in terms of policies). [4 marks]

3b. For the special case of $r = 0$ (that is, no backward slides), calculate the expected number of steps to complete the task using your optimal plan from 3a. Your answer must be a function $n$, $m$, $p$, and $q$ (although you might be able to eliminate dependencies on some of these parameters). [3 marks]

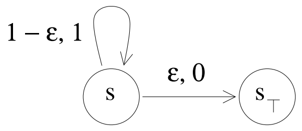
3c. Your friends are worried about your expedition, and have decided that if you are not back to chamber 0 with the diamond even after $T$ steps (for some $T \geq 2m$), they will send a search party to find you. Write down pseudocode to compute the probability that they will send a search party. Inputs to your code will be $n$, $m$, $p$, $q$, $r$, and $T$; the output must be the required probability. Only code is required; no need for a closed-form expression. The running time of your code must be polynomial in $n$ and $T$. [5 marks]

**Question 4.** Consider a run of value iteration on MDP $M = (S, A, T, R, \gamma)$. The initial value function guess is $V^0 : S \to \mathbb{R}$, and for $t \geq 0$, we set $V^{t+1} = B^\star(V^t)$, where $B^\star$ is the Bellman optimality operator. Prove or disprove each of the following statements. Proof of truth must hold for every MDP $M$, whereas a single (counterexample) MDP can establish the falsity of a statement.

4a. If $V^\star \succ V^0$, then $V^5 \succeq V^0$. [2 marks]

4b. If $V^\star \succeq V^0$, then $V^\star \succeq V^5$. [2 marks]

**Question 5.** Consider the prediction problem on the MDP shown below, with transitions according to policy $\pi$. The sole non-terminal state $s$ has a self-loop with probability $1 - \epsilon$, yielding reward 1. With probability $\epsilon$, the episode terminates with a 0-reward. Assume $\epsilon \in (0, 1)$ and no discounting.



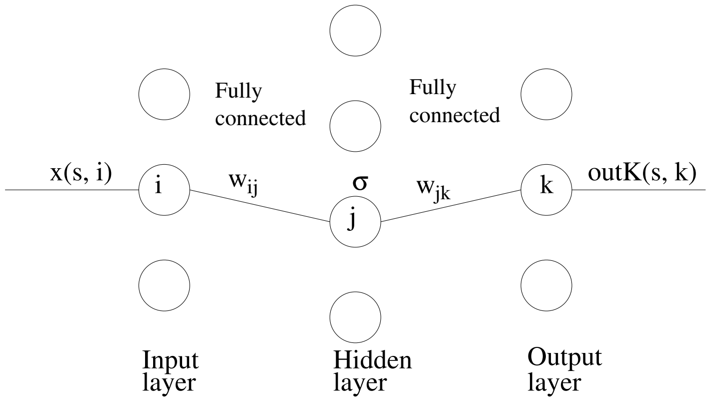
Suppose at some time step $t \geq 0$, we are in state $s$. Let our current estimate of $V^\pi(s)$ be $V^t \in \mathbb{R}$. This question examines the *variance* of 1-step and Monte Carlo returns from $s$. Recall that for a real-valued random variable $X$, $Var[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

5a. What is $Var[G_{t:t+1}]$, where $G_{t:t+1}$ is the 1-step return? [2 marks]

5b. What is $Var[G_{t:\infty}]$, where $G_{t:\infty}$ is the Monte Carlo return? [2 marks]

5c. Does $V^t$ play a role in determining which among these two returns is preferable? If so, how?; if not, why not? [1 mark]

**Question 6.** We consider the use of a single-hidden-layer neural network for representing a stochastic policy or a value function in RL. The input to the neural network are features of state. The weights of the neural network are the parameters to be updated. There is one output corresponding to each action in the control setting, and a single output in the prediction setting. For reference, a pictorial representation of the neural network is shown below, and the notation explained thereafter.



- Let there be $I$, $J$, and $K$ nodes in the input, hidden, and output layers, respectively. For convenience, define $[I] = \{0, 1, \ldots, I-1\}$, $[J] = \{0, 1, \ldots, J-1\}$, and $[K] = \{0, 1, \ldots, K-1\}$. In the figure, $I = 3$, $J = 4$, $K = 3$.

- For state $s$ and $i \in [I]$, let $x(s, i)$ denote the $i$-th feature of state. The input layer merely passes on its input to output. Hence, for state $s$ and $i \in [I]$, we have $outI(s, i) = x(s, i)$.
- Every $j$-th node in the hidden layer linearly combines the outputs of the input layer, and passes the weighted sum through the sigmoid function $\sigma(\beta) = \frac{1}{1+e^{-\beta}}$ for $\beta \in \mathbb{R}$. Thus, for state $s$ and $j \in [J]$, $outJ(s, j) = \sigma(\sum_{i \in [I]} w_{ij} outI(s, i))$. Observe that there is a weight $w_{ij}$ connecting *each* input node $i \in [I]$ with *each* hidden node $j \in [J]$.
- Every $k$-th node in the output layer corresponds to an action (hence take the set of actions as $[K]$). Node $k \in [K]$ linearly combines the outputs of the hidden layer. Thus, for state $s$ and $k \in [K]$, $outK(s, k) = \sum_{j \in [J]} w_{jk} outJ(s, j)$. Again, there is a weight $w_{jk}$ connecting *each* hidden node $j \in [J]$ with *each* output node $k \in [K]$. For uniformity of notation, think of the prediction task as having a single action (implemented by taking $K = 1$).

The parameters of the representation are the weights $w_{ij}$ for $i \in [I], j \in [J]$ and the weights $w_{jk}$ for $j \in [J], k \in [K]$; in pseudocode you are asked to provide for this question (see below), store these parameters in 2-$d$ arrays $wIJ[\ ][\ ]$ and $wJK[\ ][\ ]$, respectively. You can also assume that functions $x(\cdot, \cdot)$, $outI(\cdot, \cdot)$, $outJ(\cdot, \cdot)$, $outK(\cdot, \cdot)$, and $\sigma(\cdot)$ are already implemented.

6a. A stochastic policy is implemented using the "soft-max" operator on the outputs of the neural network. Thus, for state $s$, the probability of selecting action $k \in [K]$ is

$$\frac{e^{outK(s,k)}}{\sum_{k' \in [K]} e^{outK(s,k')}}.$$

Suppose the current policy $\pi$ is parameterised by weights $wIJ[\ ][\ ]$ and $wJK[\ ][\ ]$. Say an episode $s[0], a[0], r[0], s[1], a[1], r[1], s[2], \ldots, s[T]$ is generated by following $\pi$, where $s[T]$ is a terminal state. Write down pseudocode to perform a REINFORCE update with step size $\alpha$. You can assume $wIJ[\ ][\ ]$, $wJK[\ ][\ ]$, $T$, $s[\ ]$, $a[\ ]$, $r[\ ]$, and $\alpha$ are already populated. Your code must terminate with $wIJ[\ ][\ ]$ and $wJK[\ ][\ ]$ updated as per the REINFORCE rule at the end of the episode. Since you will need to compute a gradient for making the update, show the steps to work out the actual form of the gradient before presenting the pseudocode for the update. [6 marks].

6b. Suppose the same neural network is used to approximate a value function for a prediction task. In this case we have a single output (that is, $K = 1$). For each state $s$, $outK(s, 0)$ is interpreted as $\hat{V}(s)$. The aim is to drive $\hat{V}$ towards $V^\pi$ by making TD(0) updates, where $\pi$ is the policy being followed. Suppose the current approximation of $\hat{V}$ uses weights $wIJ[\ ][\ ]$ and $wJK[\ ][\ ]$. Say the transition $s, r, s'$ is observed. Write down pseudocode for the TD(0) update performed upon reaching $s'$, with learning rate $\alpha$ and discount factor $\gamma$. Assume $wIJ[\ ][\ ]$, $wJK[\ ][\ ]$, $s$, $r$, $s'$, $\alpha$, and $\gamma$ are already populated. Your code must terminate with $wIJ[\ ][\ ]$ and $wJK[\ ][\ ]$ updated correctly. Here, too, show the steps to obtain the form of the gradient before using it in your pseudocode as a part of the learning update. Since $K = 1$, you can use $wJK[\ ]$ instead of $wJK[\ ][\ ]$ if you would like, but the 2-dimensional variant is also okay, keeping the second index 0. [3 marks].

**Question 7.** In the lectures, both in the planning and learning settings, we only considered MDPs in which the set of actions is discrete and enumerable. In practice one often finds tasks that have a continuous action space (with an infinite number of actions). Take Sarsa as an illustrative method. What major changes would be needed in practice to apply the method on a task with a continuous action space? Your answer should take no more than 5–6 lines. [2 marks]