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

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

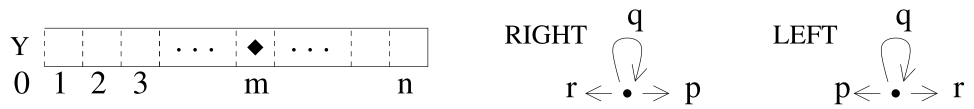
In other words, the number of pulls of arms other than a_2 under L must be a vanishing fraction of the horizon. Provide a proof sketch that L satisfies this property; no need for a detailed mathematical working. [4 marks]

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]

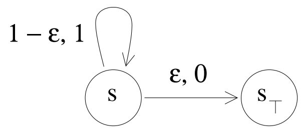
3c. Your friends are worried about your expedition, and have decided that if you are not back to chamber 0 with the diamond even after T steps (for some $T \geq 2m$), they will send a search party to find you. Write down pseudocode to compute the probability that they will send a search party. Inputs 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 π. The sole non-terminal state s has a self-loop with probability $1 - \epsilon$, yielding reward 1. With probability ϵ, the episode terminates with a 0-reward. Assume $\epsilon \in (0, 1)$ and no discounting.



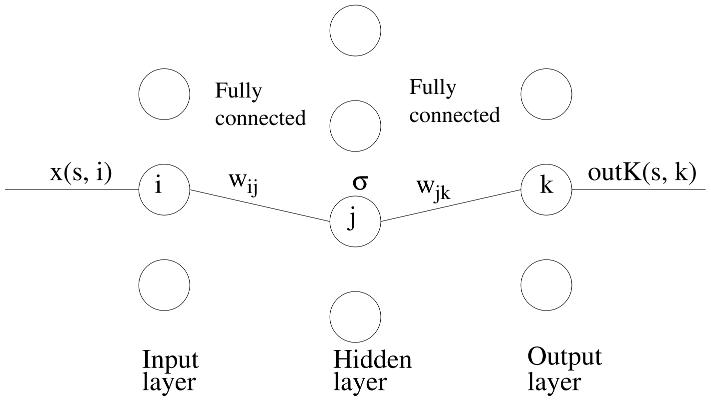
Suppose at some time step $t \geq 0$, we are in state s. Let our current estimate of $V^\pi(s)$ be $V^t \in \mathbb{R}$. This question examines the *variance* of 1-step and Monte Carlo returns from s. Recall that for a real-valued random variable X, $Var[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

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 π is parameterised by weights $wIJ[\,][\,]$ and $wJK[\,][\,]$. Say an episode $s[0], a[0], r[0], s[1], a[1], r[1], s[2], \dots, s[T]$ is generated by following π, where $s[T]$ is a terminal state. Write down pseudocode to perform a REINFORCE update with step size α. You can assume $wIJ[\,][\,]$, $wJK[\,][\,]$, T, $s[\,]$, $a[\,]$, $r[\,]$, and α 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^{π} by making TD(0) updates, where π 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 α and discount factor γ. Assume $wIJ[\,][\,]$, $wJK[\,][\,]$, s, r, s', α, and γ 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]

Solutions

1. L needs to perform an infinite amount of exploration so that a_2 can be reliably identified. Indeed if L pulls each arm an infinite number of times, then the fraction of time steps on which the arms are "sorted" (that is, having empirical means in the same order as their true means) will approach 1. If the fraction of times L pulls the empirically-second-best arm approaches 1, then the fraction of pulls of a_2 will also approach 1. In short, L needs to be greedy in the limit (albeit with respect to the empirically-second-best arm) while performing an infinite amount of exploration. This combination is easily arranged, say by selecting the arm to pull in a round-robin fashion on steps $t \geq 0$ that are perfect squares, but on other steps, pulling the empirically-second-best arm (invoking any deterministic tie-breaking rule).

2. Yes: for $k \geq 2$, too, we can achieve logarithmic regret. Perhaps the conceptually-simplest solution is as follows. With each n-armed bandit instance I, associate an $\binom{n}{k}$-armed bandit $f(I) = I'$, whose each arm a' corresponds to a distinct k-sized subset of arms $a_1, a_2, \dots, a_k$ in I. The reward from a' is a random variable that is the sum of the corresponding reward random variables of $a_1, a_2, \dots, a_k$. Observe that the expected reward of a' is $p_{a_1} + p_{a_2} + \cdots + p_{a_k}$; its rewards take values $0, 1, \dots, k$. Interacting with I'—pulling one arm at a time—can be simulated by pulling k arms at a time on I and taking the sum of the rewards as feedback.

We already know that in the regular setting (with one arm pulled at a time), we can achieve $O(C \log T)$ regret with constant C determined by the bandit instance. Hence we can achieve $O(C_{I'} \log T)$ regret on instances of the form I' described above. For instance I from the original family, the regret bound is therefore $O(C_{f(I)} \log T)$, which is in the required form.

While it might appear from the description above that the proposed approach needs $\theta(n^k)$ space (to store the statistics of $\binom{n}{k}$ arms), once can simple take the UCB of arm a' in I' as the sum of the UCBs of its constituent arms $a_1, a_2, \dots, a_k$ on I (with possible modifications to the exact UCB formula). This approach would reduce to pulling k arms in I with the highest UCBs at each time step.

3a. In modeling the task using the MDP framework, the first step is to note that having or not having the diamond with you (equivalently, having or not having visited chamber m) is a part of state. From the same chamber, the optimal action to take should depend on this information. One way to proceed is to formally define the pair "(chamber, possess diamond?)" as state. Alternatively, we can just solve two separate tasks in sequence: T_1 being the task of reaching chamber m from the entrance, and T_2 the task of getting out with the diamond. For each of these tasks there is a state corresponding to each chamber. Since completing the task requires both T_1 and T_2 to be completed in sequence, we can focus on each task independently, aiming to complete it as quickly as possible.

Since $p > r$, intuition suggests that we must consistently go RIGHT under T_1, and consistently LEFT under T_2. We shall show that this intuition is justified for each of the tasks.

$\mathbf{T_1}$. For $i \in \{0, 1, \dots, m-1, \}$, let X_i denote the expected number of steps to reach chamber m by always (that is, from every chamber) taking RIGHT, given we start from chamber i. By invoking the transition probabilities, we observe that $X_0 = 1 + pX_1 + (q + r)X_0$, and for $i \in \{1, 2, \dots, m-1, \}$, $X_i = 1 + pX_{i+1} + qX_i + rX_{i-1}$. In order to show that our policy is optimal, we will show that there is no improvable state. For $i \in \{0, 1, \dots, m-1\}$, let Y_i denote the expected number of steps to reach chamber m, starting from chamber i, by taking LEFT for the very first time step, and thereafter taking RIGHT. We shall show that $Y_i > X_i$ for all $i \in \{0, 1, \ldots, m-1\}$ (note that we are *minimising* the aggregate number of steps, hence $>$ instead of $<$).

- Simplifying the Bellman equation of X, we get

$$X_0 = \frac{1 + pX_1}{p}; X_i = \frac{1 + pX_{i+1} + rX_{i-1}}{1-q} \text{ for } i \in \{1, 2, \ldots, m-1\}.$$

- First, it is clear that $X_0 = (1/p) + X_1 > X_0$; moreover

$$X_i = \frac{1 + pX_{i+1} + rX_{i-1}}{1-q} > \frac{1 + pX_{i+1} + rX_i}{1-q} \implies X_i(1-q-r) > 1+pX_{i+1} \implies X_i > \frac{1}{p}+X_{i+1}.$$

  In other words, the expected number of steps to m decreases monotonically with i. This observation should not be surprising: an alternative way to argue the result is that for $i < m-1$, $X_i = X_{i \to i+1} + X_{i+1}$, wherein $X_{i \to i+1}$ is the expected number of steps to go from chamber i to chamber $i+1$. Note that all these expectations are positive quantities.

- We are ready to complete the proof, which uses the monotonicity of X along with the fact that $p > r$.

$$\begin{aligned} Y_0 &= 1 + (p+q)X_0 + rX_1 \\ &> 1 + (r+q)X_0 + pX_1 \\ &= X_0. \end{aligned}$$

  For $i \in \{1, 2, \ldots, m-2\}$,

$$\begin{aligned} Y_i &= 1 + pX_{i-1} + qX_i + rX_{i+1} \\ &> 1 + pX_{i+1} + qX_i + rX_{i-1} \\ &= X_i \end{aligned}$$

  Finally,

$$\begin{aligned} Y_{m-1} &= 1 + pX_{m-2} + qX_{m-1} \\ &> 1 + rX_{m-2} + qX_{m-1} \\ &= X_{m-1}. \end{aligned}$$

$\mathbf{T_2}$. The process of getting back to chamber 0 with the diamond by always taking LEFT is identical to the one to reach chamber m from chamber 0 in T_1 by always taking RIGHT—only we start at an intermediate chamber rather than the extreme one. The reasoning given above for T_1 applies in this case, too, and established that "always LEFT" is the sole optimal policy for getting out.

3b. By setting $r = 0$ in 3a, we get $X_i = \frac{1}{p} + X_{i+1}$ for $i \in \{1, 2, \ldots, m-1\}$ which gives $X_0 = \frac{m}{p}$. Since there are no back slides, the expected number of steps for returning to chamber 0 with the diamond is the same—giving $\frac{2m}{p}$ expected steps in aggregate.

3c. By fixing our policy, we fix transition probabilities among states. If we have a probability distribution over states at any step (just a single state to begin), we can obtain the distribution at the next step by performing a matrix multiplication with the transition probabilities. Of interest is the state distribution after T steps (specifically the probability of being in a target state).

To proceed formally, let us begin with task T_1. Let $x_1(t, i)$ be the probability of being in chamber i at time step t for $i \in \{0, 1, \ldots, m-1\}$ and $t \geq 0$. We have

$$x_1(0, i) = \begin{cases} 1 & i = 0, \\ 0 & 1 \leq i \leq m-1, \end{cases}$$

and for $t \geq 0$,

$$x_1(t+1, i) = p x_1(t, i-1) + q x_1(t, i) + r x_1(t, i+1),$$

adopting the convention that $x_1(t, m) = 0$ (we are ignoring the probability of being in chamber m; we can get it from the probabilities of being in the other chambers). Now let us do the same for task T_2.

$$x_2(0, i) = \begin{cases} 1 & i = m, \\ 0 & 1 \leq i \leq m-1, \end{cases}$$

and for $t \geq 0$,

$$x_2(t+1, i) = p x_2(t, i+1) + q x_2(t, i) + r x_2(t, i-1),$$

adopting the convention that $x_2(t, 0) = 0$. We have reset time upon completing T_1. The probability that we will not exit with the diamond within T steps is

$$\sum_{t_1=m}^{T-m} \left(\sum_{i=0}^{m-1} x_1(t_1, i) \right) \sum_{t_2=m}^{T-t_1} \left(\sum_{j=1}^{m} x_2(t_2, j) \right),$$

which can be computed in $\theta(nT)$ time since the t_2, j sums inside can first be computed independently and stored for reuse with each outer t_1, i sum.

4a. We disprove the statement using a counterexample. Take the episodic, undiscounted MDP shown below, in which all transitions are deterministic. States are numbered s_1 through s_6. Each non-terminal state has a single action that yields a reward of 1 (shown on arrow) and goes into a successor state (except it terminates from s_6).

$$s_1 \xrightarrow{1} s_2 \xrightarrow{1} s_3 \xrightarrow{1} s_4 \xrightarrow{1} s_5 \xrightarrow{1} s_6 \xrightarrow{1} s_\top$$

The table below shows $V^\star$ and the iterates of value iteration for a particular choice of V^0. Observe that $V^\star \succ V^0$, yet V^5 and V^0 are incomparable.

V	s_1	s_2	s_3	s_4	s_5	s_6
$V^\star$	6	5	4	3	2	1
V^0	5.5	0	0	0	0	0
V^1	1	1	1	1	1	1
V^2	2	2	2	2	2	1
V^3	3	3	3	3	2	1
V^4	4	4	4	3	2	1
V^5	5	5	4	3	2	1

The example can easily be converted to one involving a continuing task with discounting. The principle we have exploited is that $V^{t+1}(s_1)$ does not depend on $V^t(s_1)$ (hence $V^0(s_1)$ does not affect subsequent iterates). Also, the long horizon delays the approach of $V^t(s_1)$ towards $V^\star(s_1)$.

4b. We have already shown that for $X : S \to \mathbb{R}$ and $Y : S \to \mathbb{R}$, if $X \succeq Y$, then $B^\star(X) \succeq B^\star(Y)$. It follows that if $V^\star \succeq V^0$, then $(B^\star)^5(V^\star) \succeq (B^\star)^5(V^0)$: in other words, $V^\star \succeq V^5$.

5a.

$$\begin{aligned} G_{t:t+1} &= \begin{cases} 0 & \text{with probability } \epsilon, \\ 1 + V^t & \text{with probability } 1 - \epsilon. \end{cases} \\ \mathbb{E}[G_{t:t+1}] &= (1-\epsilon)(1+V^t). \\ Var[G_{t:t+1}] &= (1-\epsilon)(1+V^t)^2 - (1-\epsilon)^2(1+V^t)^2 = \epsilon(1-\epsilon)(1+V^t)^2. \end{aligned}$$

5b.

$$\begin{aligned} G_{t:\infty} &= \begin{cases} 0 & \text{with probability } \epsilon, \\ 1 & \text{with probability } (1-\epsilon)\epsilon, \\ 2 & \text{with probability } (1-\epsilon)^2\epsilon, \\ 3 & \text{with probability } (1-\epsilon)^3\epsilon, \\ \vdots \end{cases}. \\ \mathbb{E}[G_{t:\infty}] &= \sum_{k=0}^{\infty} k(1-\epsilon)^k \epsilon = \frac{1-\epsilon}{\epsilon}. \\ Var[G_{t:\infty}] &= \sum_{k=0}^{\infty} k^2(1-\epsilon)^k \epsilon - \frac{(1-\epsilon)^2}{\epsilon^2} = \frac{(1-\epsilon)(2-\epsilon)}{\epsilon^2} - \frac{(1-\epsilon)^2}{\epsilon^2} = \frac{1-\epsilon}{\epsilon^2}. \end{aligned}$$

5c. The ideal return is one that has no bias (that is, expected value $V^\pi(s)$) and low variance, but unfortunately we cannot always have both properties satisfied. The expected value of the bootstrapping return depends on V^t (it is biased), but the Monte Carlo return is unbiased. On the other hand, when V^t approaches $V^\pi(s)$ (as we might expect it to over the course of learning), the variance of the bootstrapping return is approximately $\frac{1-\epsilon}{\epsilon}$, only an ϵ-fraction of the variance of the Monte Carlo return. Surely the bootstrapping return is preferable when $V^t \approx V^\pi(s)$.

6a. Recall that for state s and action $k \in [K]$,

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

In order to perform the REINFORCE update, we need partial derivatives of $\ln \pi(s,a)$ with respect to each weight in the neural network. First we write down these partial derivatives; thereafter we specify the update that is performed at the end of the episode. Some of the derivatives given below are also used in 6b. The derivatives hold for $i \in [I], j \in [J], k, k' \in [K]$.

$$\frac{\partial}{\partial w_{jk}} outK(s,k') = \begin{cases} outJ(s,j) & k = k', \\ 0 & k \neq k'. \end{cases}$$
$$\frac{\partial}{\partial w_{ij}} outK(s,k) = outI(s,i)\sigma'\left(\sum_{i' \in [I]} w_{i'j} outI(s,i')\right) w_{jk},$$

where $\sigma'(x)$ can be seen to evaluate to $\sigma(x)(1-\sigma(x))$ for $x \in \mathbb{R}$.

$$\frac{\partial}{\partial w_{jk}} \ln \pi(s,k') = \begin{cases} (1-\pi(s,k))outJ(s,j) & k = k', \\ -\pi(s,k)outJ(s,j) & k \neq k'. \end{cases}$$
$$\frac{\partial}{\partial w_{ij}} \ln \pi(s,k) = outI(s,i)\sigma'\left(\sum_{i' \in [I]} w_{i'j} outI(s,i')\right)\left(w_{jk} - \sum_{k' \in [K]} w_{jk'}\pi(s,k')\right).$$

Given weights $wIJ[\,][\,]$ and $wJK[\,][\,]$ that define a neural network, let $pIJ^{sa}[\,][\,]$ and $pJK^{sa}[\,][\,]$ hold the corresponding partial derivatives of $\ln \pi(s,a)$ for state s and action a. After observing episode $s[0], a[0], r[0], s[1], a[1], r[1], s[2], \ldots, s[T]$, the updates given by REINFORCE are:

$$wIJ[i][j] \leftarrow wIJ[i][j] + \alpha \sum_{t=0}^{T-1} pIJ^{s[t]a[t]}[i][j]\left(\sum_{u=t}^{T-1} r[u]\right),$$
$$wJK[j][k] \leftarrow wJK[j][k] + \alpha \sum_{t=0}^{T-1} pJK^{s[t]a[t]}[j][k]\left(\sum_{u=t}^{T-1} r[u]\right).$$

6b. Using a similar convention in the context of TD(0), let $pIJ^{s}[\,][\,]$ and $pJK^{s}[\,][\,]$ hold the corresponding partial derivatives of $\hat{V}(s) = outK(s,0)$. The updates to perform, based on semi-gradient TD(0), upon encountering transition (s, r, s'), are as follows.

$$wIJ[i][j] \leftarrow wIJ[i][j] + \alpha(r + \gamma outK(s',0) - outK(s,0))pIJ^{s}[i][j].$$
$$wJK[j][k] \leftarrow wJK[j][k] + \alpha(r + \gamma outK(s',0) - outK(s,0))pJK^{s}[j][k].$$

7. With a continuous action space, it becomes necessary to generalise over actions: that is, enforcing the principle that similar actions will have similar values. Suppose actions can be taken as real-valued vectors. One way to implement generalisation is to feed in action as input to the function approximator (such as tile coding or a neural network). Given state features and action as input, the function approximator will compute the corresponding action value.

A second major issue is the actual computation of a greedy action for a particular state, which is needed if the agent must go along rewarding trajectories. In the enumerable case we can just compare all action values, but the computation is non-trivial if there are infinitely many actions. Approaches such as random sampling, and local search can yield a reasonable approximation of the greedy action in practice.

Discretisation of continuous actions is yet another easy way to bridge the continuous case with the usual enumerable setting. It can be effective especially if the action is low-dimensional.