CS 747 (Autumn 2018): End-semester Examination

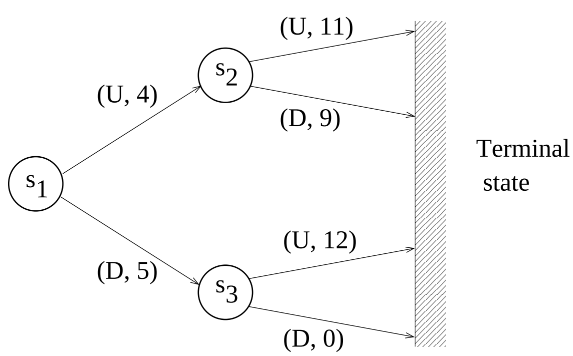
Instructor: Shivaram Kalyanakrishnan

2.00 p.m. – 5.00 p.m., November 14, 2018, SOM IC3/IC4

Total marks: 25

Note. Provide justifications and/or calculations along with each answer to illustrate how you arrived at the answer.

Question 1. The episodic MDP shown below as a state diagram has a set of non-terminal states $S = \{s_1, s_2, s_3\}$ and a set of actions $A = \{U, D\}$ (U for "up", D for "down", as in the figure). All transitions are deterministic. Each episode starts at s_1 and reaches a terminal state after exactly two transitions. Rewards are shown in the diagram. No discounting is used.



Consider an agent that is learning an action value function Q while following an ϵ-greedy policy, with $\epsilon = \frac{1}{2}$ (hence, a suboptimal action will be picked with probability 25%.). The agent goes along a trajectory $s^0, a^0, r^0, s^1, a^1, r^1, s^2, \ldots$, and uses a learning rate of $\frac{1}{t+1}$ for the t-th learning update, $t \geq 0$. You do not have to substitute numbers from the MDP for 1a, but you must for 1b and 1c.

1a. Write down the formula for updating Q after each transition (i) if the agent applies Q-learning and (ii) if the agent applies Sarsa. [1 mark]

1b. If the agent performs Q-learning updates, what will be the entries of Q at convergence? To what policy will behaviour converge? [3 marks]

1c. If the agent performs Sarsa updates, what will be the entries of Q at convergence? To what policy will behaviour converge? [3 marks]

Question 2. Suppose, on the MDP from Question 1, an agent takes actions uniformly at random from each state. It uses Monte Carlo policy evaluation to estimate the value function of this random policy—but is constrained to use just a single parameter, V, to serve as the estimate of each state's value. It is as though each state has a feature value of 1, which gets multiplied by the learned parameter V to approximate the state's value.

What is V at convergence? [2 marks]

Question 3. This question relates to the implementation of transition functions for MDPs.

3a. Assume the states of the MDP to implement are $1, 2, \ldots, n$ and the actions are $1, 2, \ldots, k$. You are given the transition function as a real-valued array $T[\ ][\ ][\ ]$, wherein the first index gives the start state, the second index the action, and the third index the next state. The entry contains the corresponding transition probability. Your only access to random numbers is through the function $random(0, 1)$, which returns a real number drawn uniformly at random from $[0, 1)$. You can only make a single call to this function.

Provide pseudocode for $getNextState()$, which should take in state s and action a as input, and return next state s' as output. The probability that $s' \in \{1, 2, \ldots, n\}$ is the output must be exactly $T[s][a][s']$. [2 marks]

3b. For fixed s and a, what is the expected number of times that $getNextState(s, a)$ will be called before a particular state s' is returned as the output? Your answer can be in terms of the entries of $T[\ ][\ ][\ ]$. [2 marks]

Question 4. The following code snippet prints a sequence $(w_t)_{t=0}^{\infty}$. Recall that $random(0, 1)$ returns a real number drawn uniformly at random from $[0, 1)$.

```
w_0 ← 0.
For t = 1, 2, ...
        Print w_{t-1}.
        x_t ← random(0, 1).
        y_t ← random(0, 1).
        z_t ← 0.
        If ((x_t)^2 + (y_t)^2 < 1)
                z_t ← 1.
        w_t ← w_{t-1} + (1/t)(z_t − w_{t-1}).
```

What is the logic being implemented by the code? Does the sequence $(w_t)_{t=0}^{\infty}$ converge (if so, to what value)? If it does not converge, what behaviour does the sequence exhibit? [4 marks]

Question 5. We studied REINFORCE and other policy gradient algorithms in class in the context of *learning*. Recall that we derived an expression for the gradient of the objective (such as the value of the start state) with respect to the parameters of the policy, and showed how an unbiased estimate of the gradient can be obtained by sequentially sampling the MDP. The idea was to then perform stochastic gradient ascent.

Now consider the *planning* setting, which we considered in the first half of the course. With access to the MDP's transition and reward functions in a compact form, it is conceivable to compute the gradient described above *exactly*. Why, then, are policy gradient methods typically *not* used in place of planning methods such as value iteration, policy iteration, and linear programming? Can you think of any use that policy gradient methods might still have in the planning setting? [2 marks]

Question 6. Describe the main advances over AlphaGo that were demonstrated in the AlphaGo Zero program. (Recall that both of these programs were designed to play Go. Do not confuse AlphaGo Zero with AlphaZero, which was a general-purpose game-playing program also applied to games other than Go.) [2 marks]

Question 7. MDPs $M_1 = (S, A, T, R_1, \gamma)$ and $M_2 = (S, A, T, R_2, \gamma)$ are identical except for their reward functions (notations are as usual). It so happens that there is a policy $\pi : S \to A$ that is *optimal* both for M_1 and for M_2.

Now consider the MDP $M_3 = (S, A, T, R_1 + R_2, \gamma)$. In other words, M_3 is identical to M_1 and M_2 except for its reward function. The reward for each transition under M_3 is the sum of the rewards obtained for the same transition under M_1 and M_2. Is π guaranteed to be an optimal policy for M_3? Prove that your answer is correct. Assume that all three MDPs implement continuing tasks, with $\gamma < 1$. [4 marks]

Solutions

1a. If the agent goes along trajectory $s^0, a^0, r^0, s^1, a^1, r^1, s^2, \ldots$, then the t-th update under Q-learning is:

$$Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \frac{1}{t}(r^t + \max_{a \in A} Q(s^{t+1}, a) - Q(s^t, a^t)).$$

The update under Sarsa is:

$$Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \frac{1}{t}(r^t + Q(s^{t+1}, a^{t+1}) - Q(s^t, a^t)).$$

1b. Q-learning converges to the optimal action value function:

$$Q(s_1, U) = 15, Q(s_1, D) = 17, Q(s_2, U) = 11, Q(s_2, D) = 9, Q(s_3, U) = 12, Q(s_3, D) = 0.$$

The policy π followed at convergence is ϵ-greedy with respect to Q.

$$\pi(s_1, U) = 0.25, \pi(s_1, D) = 0.75, \pi(s_2, U) = 0.75, \pi(s_2, D) = 0.25, \pi(s_3, U) = 0.75, \pi(s_3, D) = 0.25.$$

1c. Sarsa converges such that Q is the action value function of the policy being followed, which is itself ϵ-greedy with respect to Q.

$$Q(s_1, U) = 14.5, Q(s_1, D) = 14, Q(s_2, U) = 11, Q(s_2, D) = 9, Q(s_3, U) = 12, Q(s_3, D) = 0.$$

$$\pi(s_1, U) = 0.75, \pi(s_1, D) = 0.25, \pi(s_2, U) = 0.75, \pi(s_2, D) = 0.25, \pi(s_3, U) = 0.75, \pi(s_3, D) = 0.25.$$

2. The stationary distribution D^π of the random policy π evaluates to:

$$D^\pi(s_1) = \frac{1}{2}, D^\pi(s_2) = \frac{1}{4}, D^\pi(s_3) = \frac{1}{4}.$$

We also have

$$V^\pi(s_1) = 12.5, V^\pi(s_2) = 10, V^\pi(s_3) = 6.$$

Thus,

$$V = \operatorname*{argmin}_x \left(\frac{1}{2}(x - 12.5)^2 + \frac{1}{4}(x - 10)^2 + \frac{1}{4}(x - 6)^2 \right) = 10.25.$$

3a. Here is one possible implementation.

```
getNextState(s, a):
    r ← random(0, 1).
    For i = 1, 2, ..., n − 1
        r ← r − T[s][a][i].
        If r ≤ 0
            Return i.
    Return n.
```

3b. If $T[s][a][s'] = 0$, then naturally s' will never get returned. If not, the expected number of calls is given by

$$(T[s][a][s'])(1) + (1 - T[s][a][s'])(T[s][a][s'])(2) + (1 - T[s][a][s'])^2(T[s][a][s'])(3) + \cdots = \frac{1}{T[s][a][s']}.$$

4. Observe that w_t for $t \geq 1$ is merely the arithmetic mean of $z_1, z_2, \dots, z_t$. Also observe that $z_1, z_2, , z_3, \dots$ are generated i.i.d. from the same process, which can be thought of as implementing a Bernoulli distribution. Hence, the sequence $(w_t)_{t=0}^{\infty}$ will converge to the mean of this Bernoulli distribution. It is seen that the mean of the distribution (equal to the probability of emitting 1) is the probability that a point picked uniformly at random from the area of a square with vertices $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ falls within the circle $x^2 + y^2 = 1$. This probability is $\frac{\pi}{4}$.

5. Planning methods such as value iteration, policy iteration, and linear programming are usually applied when it is feasible to compute an optimal policy for the MDP *exactly* (or to arbitrary precision). And indeed the computation time is polynomial in associated parameters such as the number of states and actions and the horizon. By contrast, policy gradient methods only assure convergence to a *local* optimum. Note that they necessarily operate on stochastic policies, although, in general, MDPs need not have stochastic policies that are optimal. In short: policy gradient methods are not the method choice when exact solution is feasible.

Naturally, in the planning setting too one could encounter problems which are intractable to solve exactly, such as when the state space is extremely large. One can imagine creating a parameterised policy and optimising it in such a scenario. In fact policy gradient methods are often used in conjunction with *POMDP* planning, in which exact computation can be prohibitively expensive even for small state spaces.

6. Unlike AlphaGo, which was bootstrapped using supervised learning on a human expert database, AlphaGo Zero was trained completely based on self-play, starting with a random policy. The latter program (1) only used a raw encoding of the board; (2) shared weights between the value and policy networks; and (3) used the learned values during game play with no Monte Carlo rollouts. By contrast, AlphaGo used processed features; employed separate value and policy networks; and performed rollouts while playing the game.

7. For every policy π', we have, for each state $s \in S$,

$$
\begin{aligned}
V_{M_3}^{\pi'}(s) &= \mathbb{E}_{\pi'}[r_{M_3}^0 + \gamma r_{M_3}^1 + \gamma^2 r_{M_3}^2 + \dots | s^0 = s] \\
&= \mathbb{E}_{\pi'}[(r_{M_1}^0 + r_{M_2}^0) + \gamma(r_{M_1}^1 + r_{M_2}^1) + \gamma^2(r_{M_1}^2 + r_{M_2}^2) + \dots | s^0 = s] \\
&= \mathbb{E}_{\pi'}[r_{M_1}^0 + \gamma r_{M_1}^1 + \gamma^2 r_{M_1}^2 + \dots | s^0 = s] + \mathbb{E}_{\pi'}[r_{M_2}^0 + \gamma r_{M_2}^1 + \gamma^2 r_{M_2}^2 + \dots | s^0 = s] \\
&= V_{M_1}^{\pi'}(s) + V_{M_2}^{\pi'}(s).
\end{aligned}
$$

Now, suppose that there is a policy π_x such that for some state $s \in S$, $V_{M_3}^{\pi_x}(s) > V_{M_3}^{\pi}(s)$. Hence, $V_{M_1}^{\pi_x}(s) + V_{M_2}^{\pi_x}(s) > V_{M_1}^{\pi}(s) + V_{M_2}^{\pi}(s)$, which implies $V_{M_1}^{\pi_x}(s) > V_{M_1}^{\pi}(s)$ or $V_{M_2}^{\pi_x}(s) > V_{M_2}^{\pi}(s)$—neither of which is possible since π is optimal both for M_1 and for M_2. Thus, π must be an optimal policy for M_3.