CS 747 (Autumn 2019): End-semester Examination

Instructor: Shivaram Kalyanakrishnan

9.30 a.m. – 12.30 p.m., November 13, 2019, LA 201/202

Total marks: 25

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

Question 1. A learning agent A_1 interacts with an MDP with the aim of eventually converging to optimal behaviour. Assume that every state in the MDP (S, A, T, R, γ) is reachable with positive probability from every other state under every policy; otherwise the MDP is arbitrary.

The agent implements Q-learning and exercises the discipline in its learning and exploration rates required for convergence (take $\epsilon_t = \alpha_t = 1/(t+1)$ for $t \geq 0$). However, the agent adopts a relatively unusual method to update Q-values.

In particular, the agent keeps two Q-tables, Q_X and Q_Y, which are initialised arbitrarily (and possibly differently). As it goes through its life gathering experience, the agent dynamically binds each time step with either Q_X or Q_Y, making this decision uniformly at random. The binding at time step t is used for action-selection at t, and also for the TD update to the state at t once the state at $t+1$ becomes known. Here is a precise description of the procedure followed by A_1.

> Initialise Q_X and Q_Y.
> Be born in state s.
> $$Q_{\text{now}} \leftarrow \begin{cases} Q_X \text{ with probability } 1/2, \\ Q_Y \text{ with probability } 1/2. \end{cases}$$
> For $t = 0, 1, \ldots$:
>
> $a \rightarrow \epsilon_t$-greedy(Q_{now}, s).
>   Take action a, get reward r and next state s'.
> $$Q_{\text{next}} \leftarrow \begin{cases} Q_X \text{ with probability } 1/2, \\ Q_Y \text{ with probability } 1/2. \end{cases}$$
> $Q_{\text{now}}(s, a) \leftarrow Q_{\text{now}}(s, a)(1 - \alpha_t) + \alpha_t\{r + \gamma \max_{a' \in A} Q_{\text{next}}(s', a')\}$.
> $s \leftarrow s'$.
> $Q_{\text{now}} \leftarrow Q_{\text{next}}$.

By following the learning strategy described above, is A_1 guaranteed to eventually start acting optimally? Explain why or why not. A proof sketch will suffice. [4 marks]

Question 2. This question is about an agent A_2 that interacts with an MDP (S, A, T, R, γ), which is also such that every state is reachable from every state under every policy.

In class we showed that an agent A_{class} that uses ϵ_t-greedy exploration, where $t = 0, 1, 2, \dots$ denotes the number of interactions with the MDP and $\epsilon_t = \frac{1}{t+1}$, can be made to eventually learn optimal action values. For instance, one could do so by applying Q-learning with a suitably-designed learning rate.

The agent in this question, A_2 uses a different action selection strategy. If t is a power of 2 (that is, $1, 2, 4, 8, \dots$), A_2 picks an action uniformly at random. If not, A_2 selects an action greedily with respect to action value function Q, which is updated using Q-learning with harmonic annealing of the learning rate. Assume that Q is initially 0 for all state action pairs, and any ties encountered are broken uniformly at random.

2a. Is Q guaranteed to converge to the optimal action value function $Q^\star$ based on the learning algorithm of A_2? Justify your answer. [3 marks]

2b. Observe that the algorithm implemented by A_2 above is randomised. In general, is there a *deterministic* learning algorithm that guarantees Q will converge to $Q^\star$? Provide a brief justification. [1 mark]

Question 3. This question considers the expressive power of 1-dimensional tile coding: that is, tile coding that uses a separate set of tilings for each dimension. We examine the complexity of functions over two variables that can be represented using 1-dimensional tile coding.

Let $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ be $m \geq 1$ distinct point(s) in $\mathbb{R}^2$, and let these points be associated with function values $f(x_1, y_1), f(x_2, y_2), \dots, f(x_m, y_m) \in \mathbb{R}$, respectively. In short, we have described a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ defined on m points.

We say that f can be 1-*tile-coded* if there exists a 1-dimensional tile coding scheme $T : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that for all $i \in \{1, 2, \dots, m\}, T(x_i, y_i) = f(x_i, y_i)$. Recall that for point $(x, y) \in \mathbb{R}^2$, $T(x, y)$ is the sum of the weights of the tiles that are active for (x, y) in each dimension. Along each dimension, T may employ any number of regularly-spaced tilings, with any tile width (common to all the tiles in that dimension), and any origin. The real-valued weights assigned by T to the individual tiles in the x and y dimensions can be arbitrary.

Consider the following statement:

For every set of m distinct point(s), every function f over the points can be 1-tile-coded.

For which values of $m \in \{1, 2, \dots\}$ is this statement true, and for which ones is it false? Justify your answer. [4 marks]

Question 4. Which factors combine into the "deadly triad" presented by Sutton and Barto (2018)? What is the phenomenon associated with this triad? [2 marks]

Question 5. An agent gets advice from a group of m experts on the action it should take at each state. Assume the agent interacts with MDP (S, A, T, R, γ). Each expert $i \in \{1, 2, \ldots, m\}$ advises action-selection according to a policy π_i that can either be deterministic or stochastic. The agent associates a parameter w_i with each expert $i \in \{1, 2, \ldots, m\}$, and takes action $a \in A$ from state $s \in S$ with probability

$$\pi_{\mathbf{w}}(s, a) = \frac{\sum_{i=1}^{m} e^{w_i} \pi_i(s, a)}{\sum_{j=1}^{m} e^{w_j}},$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ is the vector of parameters for combining the experts' advice. The agent intends to tune these parameters by applying REINFORCE.

5a. What is $\nabla_{\mathbf{w}} \pi_{\mathbf{w}}(s, a)$? Give the formula for its i-th element. [3 marks]

5b. In order for $\nabla_{\mathbf{w}} \pi_{\mathbf{w}}(s, a)$ to be well-defined for all $s \in S, a \in A$, do we need to make any assumption regarding the experts' policies? [1 mark]

5c. How is $\nabla_{\mathbf{w}} \pi_{\mathbf{w}}(s, a)$ used in the REINFORCE update? [1 mark]

Question 6. Write down pseudocode to describe how an agent uses tree search with Monte Carlo roll-outs for action selection. Assume the task is episodic, and also assume access to a sample model that stochastically returns a next state s' (possibly terminal) and reward r when passed state s and action a.

The tree is built up to depth $d \geq 1$. Transition probabilities at internal nodes are estimated by making $M \geq 1$ calls to the sample model. Each leaf is evaluated based on $N \geq 1$ Monte Carlo roll-outs using a policy π. Thus d, M, N, and π are the parameters to your code.

Assume that the set of actions is small and enumerable, but the set of states might be too large to enumerate (although only a small number of states will be reachable in one step from any given state). [4 marks]

Question 7. The AlphaGo program employs a value function v_θ and policies p_σ, p_ρ, and p_π in its construction. Briefly describe the role of each of these four components in the working of the program. [2 marks]

Solutions

1. We observe that the learning process on the given MDP $M = (S, A, T, R, \gamma)$ is identical to the application of the usual form of Q-learning (with a single table!) on an induced MDP $\bar{M} = (\bar{S}, \bar{A}, \bar{T}, \bar{R}, \bar{\gamma})$.

Our construction of $\bar{M}$ is as follows. For every state s in M, the MDP $\bar{M}$ has a corresponding pair of states (s, X) and (s, Y). When the agent is in state s in M, we let it be in either (s, X) or (s, Y) in $\bar{M}$, with equal probability. When a transition happens from s to s' in M, it happens both from (s, X) and (s, Y) to one of (s', X) and (s', Y) in $\bar{M}$, again with equal probability. Rewards are identical, as is the discount factor. Here is a full specification of $\bar{M} = (\bar{S}, \bar{A}, \bar{T}, \bar{R}, \bar{\gamma})$.

- $\bar{S} = S \times \{X, Y\}$.
- $\bar{A} = A$.
- For all $s, s' \in S, a \in A$:
$$\bar{T}((s, X), a, (s', X)) = \bar{T}((s, X), a, (s', Y)) = \bar{T}((s, Y), a, (s', X)) = \bar{T}((s', Y), a, (s, Y)) = \frac{T(s, a, s')}{2}.$$
- For all $s, s' \in S, a \in A$:
$$\bar{R}((s, X), a, (s', X)) = \bar{R}((s, X), a, (s', Y)) = \bar{R}((s, Y), a, (s', X)) = \bar{R}((s', Y), a, (s, Y)) = R(s, a, s').$$
- $\bar{\gamma} = \gamma$.

By writing down Bellman's Optimality Equations, we can verify that for all $s \in S, a \in A$, $\bar{Q}^\star((s, X), a) = \bar{Q}^\star((s, Y), a) = Q^\star(s, a)$, where $\bar{Q}^\star$ is the optimal action value function for $\bar{M}$.

Now consider the learning process described in the question. Take a moment to convince yourself that every run of learning on M, with the initialisation of Q_X and Q_Y simulates a valid, random run of regular Q-learning on $\bar{M}$ with a single table $\bar{Q}$, provided we initialise action-values for states of the form s_X with corresponding ones in Q_X and for states of the form s_Y with corresponding ones in Q_Y. It is easily verified that at every point of time, for all states $s \in S, a \in A$,

$$Q_X(s, a) = \bar{Q}(s_X, a) \text{ and } Q_Y(s, a) = \bar{Q}(s_Y, a).$$

Since learning and exploration rates are annealed harmonically, we know that $\bar{Q}$ will eventually converge to $\bar{Q}^\star$. The implication is that Q_X and Q_Y both converge to $Q^\star$, which induces optimal action selection in the limit.

2a. Consider an MDP with two states s_1 and s_2 and in which the transitions are deterministic. Every action from s_1 leads to s_2, and every action from s_2 leads to s_1. There are a sufficiently large number of actions. Transitions all have different rewards.

Suppose we start at s_1 at $t = 0$, it is easy to see by our construction that s_1 will be visited at $t = 0, 2, 4, \ldots$, and s_2 will be visited at $t = 1, 3, 5, \ldots$. Since we only explore when t is a power of 2, we only explore actions from s_1 infinitely often. As for s_2, one action gets explored at $t = 1$; thereafter only an action with the highest Q estimate is picked. Hence, it is possible that some actions, including optimal ones, will never get picked from s_2.

In general, a successful algorithm must ensure that *every* state-action pair gets picked infinitely often in the limit. The algorithm in this question does not do so, and so will not converge to $Q^\star$ on some MDPs.

2b. Yes, there do exist deterministic algorithms that can converge to $Q^\star$. Such algorithms could, for instance, take exploratory actions at each state in a round-robin manner, and decide whether to explore or exploit on a particular visit to the state depending on the number of times the state has been visited thus far. Exploring if the visit number is a power of 2 and exploiting otherwise would be one way to proceed. Note that deterministic algorithms must use a deterministic tie-breaking strategy.

3. We show that the claim is true for $m = 1, 2, 3$ and false for $m \geq 4$.

First we present our argument below for $m = 1, 2, 3$, taking the set of points and f to be arbitrary and showing that f can be 1-tile-coded.

If $m = 1$—that is, there is only a single point (x_1, y_1)—we can make do with any tile coding scheme. T simply gives any one tile activated by (x_1, y_1) the weight $f(x_1, y_1)$, and gives every other tile zero weight. Clearly $T(x_1, y_1) = f(x_1, y_1)$.

If we have $m = 2$ distinct points (x_1, y_1) and (x_2, y_2), they must differ in at least one coordinate. Let us assume, without loss of generality, that $x_1 \neq x_2$. In T, we use sufficiently "thin" tiles along the x dimension such that there is one tile t_1 activated by (x_1, y_1) and not by (x_2, y_2), and one tile t_2 activated by (x_2, y_2) and not by (x_1, y_1). We give t_1 a weight of $f(x_1, y_1)$ and t_2 a weight of $f(x_2, y_2)$. All other tiles receive zero weight. This scheme ensures $T(x_1, y_1) = f(x_1, y_1)$ and $T(x_2, y_2) = f(x_2, y_2)$.

If we have $m = 3$ distinct points, there must exist at least one point either whose x or y coordinate is unique: that is, not shared with the other points. Without loss of generality, assume the points are (x_1, y_1), (x_2, y_2), and (x_3, y_3) with $x_1 \neq x_2$ and $x_1 \neq x_3$. We can use the construction given above for $m = 2$ to first match the f values of (x_2, y_2) and (x_3, y_3). In so doing, let us assume that the tile coding scheme gives (x_1, y_1) a value of α. Now, since x_1 is different from both x_2 and y_2, we can identify a tile in the x-dimension that is activated by (x_1, y_1) and not by the other points. We set the weight of this tile to $f(x_1, y_1) - \alpha$, thereby matching values for (x_1, y_1). Note that this tile in not active for the other points, and has no effect on their tile-coded value. Thus, f is 1-tile-coded.

Now, we provide a set of $m = 4$ points and a function f on them that cannot be 1-tile-coded. Naturally this negative result can be extended to larger values of m by adding more points to the set provided in the proof. Hence, the statement in the question is true for $m = 1, 2, 3$ and false for $m = 4, 5, \ldots$.

The $m = 4$ points we consider are (x_1, y_1), (x_1, y_2), (x_2, y_1), and (x_2, y_2), with $x_1 \neq x_2$ and $y_1 \neq y_2$—these points are the four corners of a rectangle. Consider an arbitrary tile coding scheme T. For each point P let $T_x(P)$ denote the set of tiles in the x dimension activated by P, and let $T_y(P)$ denote the set of tiles in the y dimension activated by P. For a given tile t, let $weight(t)$ denote the weight assigned by T to t. If, indeed, f is tile-coded by T, we have

$$
\begin{aligned}
&f(x_1, y_1) + f(x_2, y_2) \\
=& \left(\sum_{t \in T_x(x_1, y_1)} weight(t) + \sum_{t \in T_y(x_1, y_1)} weight(t) \right) + \left(\sum_{t \in T_x(x_2, y_2)} weight(t) + \sum_{t \in T_y(x_2, y_2)} weight(t) \right) \\
=& \left(\sum_{t \in T_x(x_1, y_1)} weight(t) + \sum_{t \in T_y(x_2, y_2)} weight(t) \right) + \left(\sum_{t \in T_x(x_2, y_2)} weight(t) + \sum_{t \in T_y(x_1, y_1)} weight(t) \right) \\
=& \left(\sum_{t \in T_x(x_1, y_2)} weight(t) + \sum_{t \in T_y(x_1, y_2)} weight(t) \right) + \left(\sum_{t \in T_x(x_2, y_1)} weight(t) + \sum_{t \in T_y(x_2, y_1)} weight(t) \right) \\
=& f(x_1, y_2) + f(x_2, y_1).
\end{aligned}
$$

For any choice of f such that $f(x_1, y_1) + f(x_2, y_2) \neq f(x_1, y_2) + f(x_2, y_1)$, 1-tile-coding is not possible. Our proof is done.

4. When function approximation (factor 1) is used in reinforcement learning with bootstrapping (factor 2) and off-policy updates (factor 3), it often leads to an unstable learning process—there are even examples of divergence. Any two of these factors alone are usually not enough to create instability; the triad is particularly disruptive.

5a.

$$\begin{aligned}
\frac{\partial}{\partial w_i}\left(\pi_{\mathbf{w}}(s,a)\right) &= \frac{\partial}{\partial w_i}\left(\frac{\sum_{k=1}^{m} e^{w_k}\pi_k(s,a)}{\sum_{j=1}^{m} e^{w_j}}\right)\\
&= \frac{e^{w_i}\pi_i(s,a)}{\sum_{j=1}^{m} e^{w_j}} - \frac{e^{w_i}\sum_{k=1}^{m} e^{w_k}\pi_k(s,a)}{(\sum_{j=1}^{m} e^{w_j})^2}\\
&= \frac{e^{w_i}}{\sum_{j=1}^{m} e^{w_j}}(\pi_i(s,a) - \pi_{\mathbf{w}}(s,a)).
\end{aligned}$$

5b. There is no requirement on the individual expert policies; in particular, they need not be stochastic. This fact does not contradict the requirement we specified out in class that policy gradient methods must have a positive probability of taking every action from every state. Here the agent is constrained to combine the expert policies rather than the atomic actions. From a particular state s, notice that if none of the expert policies takes some action a, there is no way the combined policy will take a. In such an event, we see from 5a that $\nabla_{\mathbf{w}}\pi_{\mathbf{w}}(s,a)$ will be $\mathbf{0}$. The only requirement is that the expert policies be combined "softly", as it is in this soft-max approach.

5c. Under REINFORCE, an episode $s^0, a^0, r^0, s^1, a^1, r^1, \ldots, r^{T-1}, s^T$ is generated by following $\pi_{\mathbf{w}}$, where s^T is a terminal state. Using this data, $\mathbf{w}$ is incremented (in expectation) along the direction given by $\nabla_{\mathbf{w}} V^{\pi_{\mathbf{w}}}(s^0)$. For each (s,a)-pair visited during the episode, a term depending on $\nabla_{\mathbf{w}}\pi_{\mathbf{w}}(s,a)$ arises in the calculation of the gradient of $V^{\pi_{\mathbf{w}}}(s^0)$.

6. The crux of this code is the expectimax calculation of values, which is best coded up recursively. At each internal node, the action selected is one with the largest Q-value; Q-values themselves are calculated by taking an expectation over state values (V). The sample model is used to generate next states. At leaf nodes, values are calculated by taking an average of multiple roll-outs using the given policy. Below is a typical implementation of the tree search procedure.

```
SelectionAction(s, d, M, N, π)
    Return argmax_{a∈A} ActionValue(s, a, d − 1, M, N, π).

ActionValue(s, a, d, M, N, π)
    Call Model(s, a) M times and generate samples (s'_i, r_i)_{i=1}^M.
    - Let state[] contain the names of the states visited;
    - Let probability[] contain the empirical fraction of the visits;
    - Let reward[] contain the corresponding empirical average reward.
    Q ← 0.
    For i ∈ {1, 2, ..., length(state)}:
        Q ← Q + probability[i] × (reward[i] + γ × StateValue(state[i], d, M, N, π)).
    Return Q.

StateValue(s, d, M, N, π)
    If s is terminal:
        Return 0.
    If d = 0:
        Return Roll-outValueEstimate(s, N, π).
    V ← −∞.
    For each a ∈ A:
        Q ← ActionValue(s, a, d − 1, M, N, π).
        If Q > V:
            V ← Q.
    Return V.

Roll-outValueEstimate(s, N, π)
    V ← 0.
    Repeat N times:
        R ← 0; s_now ← s; discount ← 1.
        While s_now is not terminal:
            (s', r) ← Model(s_now, π(s_now)).
            R ← R + discount × r; discount ← discount × γ.; s_now ← s'.
        V ← V + R.
    V ← V/N.
    Return V.
```

7. p_σ is a policy network trained using supervised learning to mimic human expert moves. It is used as the initialisation for p_ρ, a policy trained using REINFORCE and self-play to achieve much higher performance. v_θ is an approximation of the value function of p_ρ, and is used in part to evaluate leaves during tree search while playing. The other part of leaf-evaluation comes from doing roll-outs. p_π, which is based on a linear architecture also trained on human expert moves, is the roll-out policy used; it is much quicker to execute than neural network-based policies.