CSE257 Fall 2021: Assignment 3

Deadline: Dec-5 11:59pm. Submit a PDF of your answers on Gradescope. Upload a zip file of all your source code via `https://www.dropbox.com/request/cjWxmaOciQw6uu9jWdI6` by the same deadline and use the file name "A3_[Your Name]_[Your PID].zip". All of the code should be in Python and you can use standard libraries such as Numpy, Scipy, Matplotlib, etc. Finish all the answers and implementation completely by yourself. Feel free to search online and in textbooks for help with proofs, but you are not allowed to copy any code from other sources.

1 Tic-Tac-Toe (Implementation Required)

Consider the Tic-Tac-Toe game (if you type this in Google, it'll let you play with it) against a purely random opponent. That is, when the opponent needs to make a move, it will choose uniformly randomly from the empty spots, and put its mark in it. The opponent plays first, i.e., from the empty 9 spots, it first picks an arbitrary one and mark it, as the beginning of the game, and then it is your turn. Naturally, the game can be modeled as an MDP, with you taking actions from relevant states in the game, and the opponent acting as the environment.

Use the following reward function: For any game state s that is not terminal (i.e., no winner and there is still some empty spot), you receive $R(s) = 1$. For any game state s that you are the winner, $R(s) = 10$, and if you are the loser, then $R(s) = -10$. If it is a draw state (all spots are occupied and there is no winner), then $R(s) = 0$. We will always use the discount factor $\gamma = 0.9$.

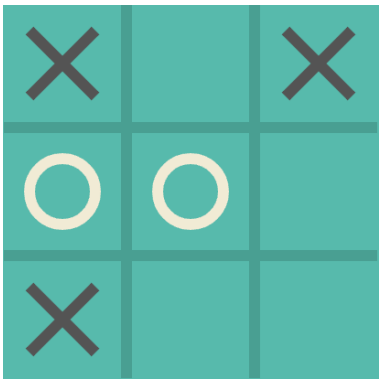

Figure 1: A game state. Your play circle, and it is your turn now.

Question 1 (1 Points)**.** Define mathematically the states and the actions of the MDP (you do not need to specify with state has what actions, just define these two sets). You can choose any representation you like, anything that you find easy to implement later. Using your definition, list all terminal states of the MDP where you win, i.e., any state s that satisfies $R(s) = 10$ as mentioned above.

Question 2 (3 Points)**.** Implement the MDP so that it can generate all feasible trajectories of the game (you don't need to list these, of course). For this you need to specify the probabilistic transitions $P(s'|s, a)$ for arbitrary state s, and action a that is feasible from s for the player. Show 5 arbitrary full trajectories of the game (from some initial state to some terminal state, and you can choose arbitrary actions in each step) and show the reward-to-go for each state in the trajectory (check slides for definition).

Question 3 (3 Points)**.** Implement value iteration for the MDP you defined, starting from arbitrary initial values until the value of any state is at most 0.1 away from the optimal values (i.e., $\|V - V^*\|_\infty \leq 0.1$). Explain the justification for why the values that you compute are no more than 0.1 different from the optimal values on any state.

For the 5 trajectories you showed above, show here the state value for each of the state that have been visited in these trajectories.

Question 4 (3 Points)**.** Recall that the optimal Q-value for any feasible state-action pair (s, a) is defined as

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \cdot \max_{a'} Q(s',a') = R(s) + \gamma \sum_{s'} P(s'|s,a) \cdot V^*(s').$$

where $V^*(s')$ is the value of the state s' under the optimal policy. Choose two different initial states (i.e., the opponent picks a different spot in each case) and show the Q-value of all actions for your agent for each of these two states.

Question 5 (4 Points)**.** Consider the game state shown in Figure 1 (you play with circle and it is now your turn). We will redefine the reward function as follows. For any game state s that is not terminal, set $R(s) = c$ for some fixed constant c. The rewards for all other states (win, lose, and draw states) stay the same as defined above.

Now, come up with two different numbers, $c_1 < 0$ and $c_2 > 0$, for this constant c used by all non-terminal states, such that the optimal actions at the game state of Figure 1 are different under these two different reward functions. Show the Q values of all actions at this state for each of these two reward functions (i.e., under c_1 and under c_2).

Question 6 (2 Points)**.** Also draw (by hand, unless you code faster than that) the expectimax tree with the game state in Figure 1 at the root, which means you should ignore the rewards on all non-terminal nodes, since in expectimax only terminal nodes can have rewards. What is the optimal action calculated by expectimax for this? Intuitively, why does some definition of the rewards above change this optimal action? (just explain the reason in a sentence or two, no need to be precise)

2 Function Approximation in RL (No Implementation)

Question 7 (6 Points)**.** Consider the following MDP: $S = \{s_0, s_1, s_2, s_3\}$, $A = \{a_1, a_2, a_3\}$. The transition probabilities are $P(s_1|s_0, a_1) = 0.5$, $P(s_2|s_0, a_1) = 0.5$, $P(s_2|s_0, a_2) = 1$, $P(s_3|s_0, a_3) = 1$, and all other combinations have probability 0. (So s_1, s_2, s_3 are terminal states.) Let the discount factor be $\gamma = 0.5$.

Suppose we want to represent the Q-value function as a linear function

$$Q(s,a) = \alpha s + \beta a \tag{1}$$

where $\alpha, \beta \in \mathbb{R}$ are the weight parameters that can be arbitrary chosen. When evaluating the function, we fix the encoding for the states and actions as $s_i = i + 1$ and $a_i = i$ (e.g., $s_0 = 1$ and $a_1 = 1$). For instance, if $\alpha = 2$ and $\beta = 3$, then plugging them in with the encoding for the state and action we have $Q(s_0, a_1) = 2 \cdot 1 + 3 \cdot 1 = 5$.

1. (2 Points) Design a reward function such that the linear Q function in (1) can correctly represent all Q values at s_0 (yes, you need to calculate the Q values under the reward function you designed first). Show both the reward function and the corresponding linear Q-value function.

2. (2 Points) Design a reward function such that the linear Q function does not work. Namely, the function in (1) can not represent all Q values at s_0 for any choice of α and β (i.e. the difference between the function value and the true Q value is always large for some inputs, regardless of what α, β are.). Explain the reasoning.

3. (2 Points) Take the reward function you designed for the previous question (which fails to be captured by the linear function) and design a different function approximation scheme (different from (1)), such that all Q-values at s_0 can now be correctly represented.

Note that for all these questions we are only asking the function representation to capture the Q values at state s_0, i.e., it does not need to work for the other states (in general it should, just to simplify things here).

I hope these questions make you think about the complications that using function approximators can introduce and why expressive nonlinear function approximators such as neural networks are necessary.

3 Blackjack (Implementation Required, Extra Credits)

Since towards the end of the quarter you will likely get busy, this implementation exercise is for extra credits only. We encourage you to do it if you have time and are reasonably comfortable with Python as it should improve your practical understanding of the relevant RL algorithms.

Implement Monte Carlo policy evaluation, Temporal-difference policy evaluation, and Q-learning for Blackjack in the simulator here `https://github.com/ucsdcsegaoclasses/blackjack`. All details of the tasks and all parameters needed are given in the README file or in the starter code. Read them carefully.

Question 8 (3 Extra Points)**.** Select two game states, A and B: In State A the player's sum of cards is 10, and in State B the sum of cards is 20. Plot how the value estimate of the each state changes over the number of visits to the state until the values convergence, under Monte Carlo policy evaluation and Temporal-Difference policy evaluation, respectively; so 4 plots in total.

Question 9 (3 Extra Points)**.** Perform Q-learning and plot how the Q-value changes over the number of visits to each action for the same two game states you selected above, until you have run Q-learning for long enough so that the Q-value converges at least on some action for each state (note: a very bad action may receive a small number of visits, so this requirement is saying you only need to wait till the better action has been visited enough times so that the Q-value of it stabilizes).

Also plot the cumulative winning rate over the number of plays in the game: for every n number of plays (x-axis), show the ratio w/n (y-axis) where w is the total number of wins out of the n plays.