CSE257 Winter 2025: Assignment 3

Deadline: Mar-9 Sunday 11:59pm. You will need to make the following two submissions.

1. Submit a pdf file of your (written or typed) answers on Gradescope (if you are enrolled, you should have already been added to the course on Gradescope).
2. Submit your Python code or Jupyter notebook files for problems that have the "Implementation Involved" label, via the Dropbox link: `https://www.dropbox.com/request/XzChzoa7FtsJ7x4e0yR5` . You can import any library that does not already implement the algorithms we ask for. Submit all the code in one zip file. For these questions that have implementation involved, you need to still put whatever is asked in the questions, such as plots and explanations, in the pdf file that you submit to Gradescope. Otherwise the question will be considered unanswered for grading purposes.
3. Again, use of ChatGPT or other language models is strictly forbidden for anything you write in the answer pdf file (even for editing grammar), but they are allowed for the implementation and the code you submit.

Question 1 (Implementation Required)**.** Implement value iteration for the MDP $M = \langle S, A, P, R, \gamma \rangle$ with

- $S = \{s_1, s_2\}$
- $A = \{a_1, a_2, a_3\}$
- $P(s_1|s_1, a_1) = 0.2, P(s_2|s_1, a_1) = 0.8, P(s_1|s_1, a_2) = 0.4, P(s_2|s_1, a_2) = 0.6, P(s_2|s_1, a_3) = 1,$
  $P(s_1|s_2, a_1) = 0.1, P(s_2|s_2, a_1) = 0.9, P(s_1|s_2, a_3) = 0.5, P(s_2|s_2, a_3) = 0.5$
  The transition probability for any other state-action combination is 0.
- $R(s_1) = -10, R(s_2) = 10$
- $\gamma = 0.6$

1. (2 Points) Start from two different initial value vectors: $V_A = (0, 0)$ and $V_B = (100, 100)$. Make 2D plots of the trajectory of Bellman updates by following value iteration from each of these two initializations, until the value converges to be within 0.1 from the optimal value vector in max norm. (So two plots, one for value iteration from V_A till convergence and the other from V_B.)

   made a change here. V_B was $(-10, 10)$, now changed to $(100, 100)$

2. (1 Point) Justify why the values you stopped at are within 0.1 from the optimal values by showing the following general result. Suppose in one Bellman update $V' \leftarrow B(V)$, the distance of movement $\|V - V'\|$ is δ. With just that information, how do we know at most how far V' is from the optimal V^*?

Question 2 (No Implementation)**.** Consider the blackjack game as modeled by the starter code here:

`https://github.com/ucsdcsegao/blackjack`

You don't need to implement anything for these questions. The game engine is completely specified by the code, so all the following questions should be addressed by first reading the code.

1. (1 Point) Suppose you are following a fixed policy to do temporal-difference policy evaluation for it. Let s be an arbitrary *terminal* state with reward r. Assume that all values of all states are initialized to be 0. Write down $V^\pi(s)$ after the first two visits to this terminal state. The answer should be a symbolic one and does *not* need any concrete numerical value. (Explain all the parameters you need to specify it.) When the number of visits increases, what does this $V^\pi(s)$ converge to? Explain why.

2. (1 Point) Consider state s with sum of cards being 20, and the action a of "hit" i.e. asking for a new card. What should $Q(s, a)$ converge to? Let's assume you always get a card from $\{1, ..., 10, J, Q, K\}$ at equal probability for each type of card. Give a numerical answer and explain. You can declare the numerical value of any parameter you may need. You can assume the winner always directly wins whenever they get 21 (so no need to wait for Dealer's outcome).

3. (1 Point) How does visitation affect convergence? Suppose you always start the game at random, under a limited number of sampled trajectories, what states are easy to learn values for, and what states are harder to learn? Just informally explain.

Question 3 (Implementation Required)**.** Consider the safe driving problem in Figure 1. The car is the agent that you control. The car can only move in the two vertical lanes, and the pedestrians only in the two yellow lanes. They only make discrete moves, going from cell to cell in each step. Their paths intersect at the four cells in the middle, labeled as the D1-D4, where the car needs to drive cautiously, hopefully.

The car can choose from 4 actions each time: go downward by 1 or 2 cells (i.e., different speeds), or cross over to the other lane and downward by 1 cell, or not move. When the car goes downward by 2 cells, both cells are considered occupied by it. The goal of the the car is to park at either the P1 or P2 cell.

The two pedestrians can only move in the row they are in, following the corresponding direction indicated by the arrows, i.e., only moving left in L-Row and right in R-Row. When pedestrians go out of bounds, they just reappear from the other end (like in a classical Atari game). At initialization, the two pedestrians can appear in any yellow cell in their corresponding rows. The initial state of the car is in either of the two cells at the top row (labeled as "start").

In all questions, the car does not have access to the transition probabilities/rules of the environment and can only optimize its behavior through experience, i.e., following the setting of Q-learning. Also note that the transition can either use probabilities or deterministic rules, since the latter just means some transitions happen with probability 1.

Always use discount factor $\gamma = 0.9$.

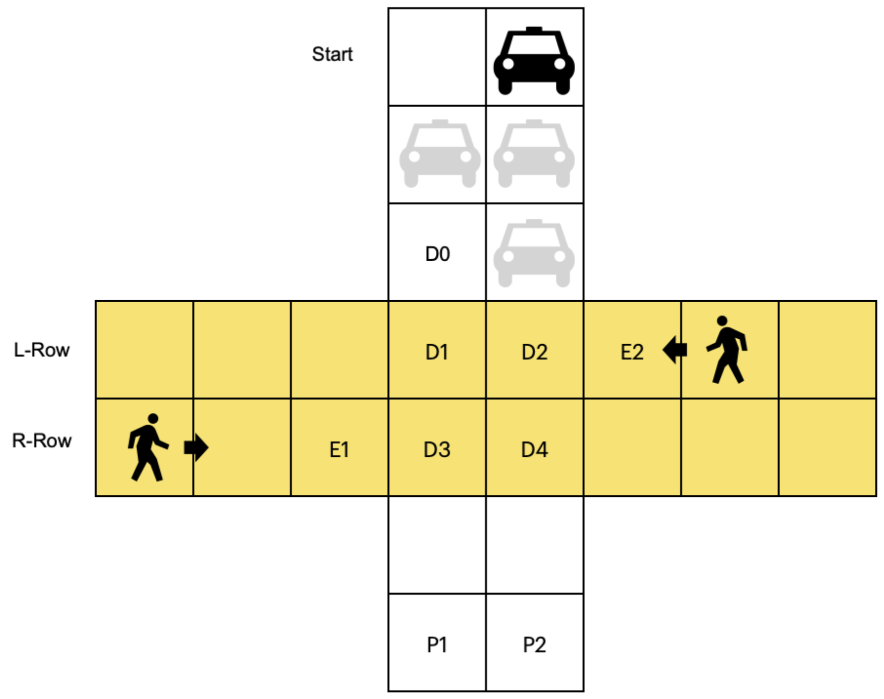

Figure 1: Illustration of the avoid-pedestrians-and-park problem.

1. (1 Point) Let's first assume that there are no pedestrians, so the car does not need to consider pedestrians.

   Roughly describe an appropriate definition of an MDP (states, actions, rewards, transitions, etc.) that models that task for the car agent to achieve the goal of "reaching P1 or P2 as quickly as possible" which means it should try to reach the goal with the least possible number of time steps. That is, the optimal policy (which you do not need to compute) should correspond to a policy that achieves this goal. Note that the transitions are not probabilistic in this case.

2. (2 Points) Implement Q-learning to find the optimal policy for the car for the MDP you defined in the previous question. In the answer pdf, give the Q values for the car at the cell labeled as D0, for each action that is available to the car at that position. (Since there are four actions possible there, you should show four $Q(s, a)$ values.)

   Plot how the Q-values at this state got updated through the iterations (2D plot for the Q-value of each state-action pair at this state, x-axis for number of visits to the pair, y-axis for Q-value).

   Use learning rate $\alpha_k(s, a) = \frac{1}{k}$ in the TD update for the k-th visit on (s, a) ($k \geq 1$).

3. (1 Point) Now consider the pedestrians. They can suddenly show up at any cell in their own row, and can move either 1 or 2 cells each time, with 50% chance for each option. Write down the definition of a new MDP that gives a large penalty of your choice and terminate the trajectory when the car hits pedestrians, i.e., when the car and any of the two pedestrians occupy the same cell at the same time. (Hint: you may need to change the state definition of the car now that the pedestrian needs to be observed.) The goal of the car in this new MDP is still to reach P1 or P2 with the least amount of steps.

4. (1 Point) Perform Q-learning for this new MDP you defined in the previous question. Show the Q values for the car at the cell labeled as D0 with the two pedestrians at the positions as shown in the plot, for each action that is available to the car at that position.

   Plot how the Q-values at this state got updated through the iterations (2D plot for the Q-value of each state-action pair at this state, x-axis for number of visits to the pair, y-axis for Q-value).

Question 4 (No Implementation)**.** 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. (1 Point) 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. (1 Point) 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. (1 Point) 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.

Question 5 (Implementation Required)**.** Visualize the concept of concentration bounds as follows. Let X be a random variable taking values of either 0 or 1, with probability $P(X=0)=0.3$ and $P(X=1)=0.7$. Write Python code to plot the following.

1. (1 Point) Draw samples of X as $X_1,...,X_N$ and plot the histogram of the average $\bar{X}=\frac{1}{N}\sum_{i=1}^{N}X_i$ with 500 trials. That is, each trial computes an $\bar{X}$ for N samples of X, and you need to draw enough samples to compute 500 independent instances of $\bar{X}$. Do this for three different values of $N=1,50,1000$.

   For the histogram you can divide $[0,1]$ into 10 bins ($[0,0.1]$, $[0.1,0.2]$, etc.). The x-axis should be the bins of the value for $\bar{X}_N$, and the y-axis should be the frequency of occurrence of $\bar{X}_N$ in each bin.

   Since there are three N, you should show three plots, each for a different N, and in each one you need do 500 trials and show how the $\bar{X}_N$ is distributed as histogram.

2. (1 Point) Let μ be the true expectation of the random variable X specified above. Plot $P(|\bar{X}_N-\mu|\geq\varepsilon)$ as a function as N grows and compare it with the Hoeffding bound in the following way. Let $\varepsilon=0.1$.

   Let the x-axis be N going from 0 to 100. For each N, do 10 trials of $\bar{X}_N$, and show the frequency of $|\bar{X}_N-\mu|\geq\varepsilon$ happening on the y-axis (for instance, if you see 3 times of that happening out of 10 trials when $N=5$ then the y-axis should show 0.3 when $N=5$). On the same plot, show how the Hoeffding bound changes over time as well. That is, plot the righthand side of the Hoeffding inequality in the slides as a function of N.

3. (Extra 1 Point) Now let X be a random variable that takes continuous values between $[0,1]$, and you can define its distribution (as long as it only takes values in that range). Design a plot that studies how the probability of $P(|\bar{X}_N-\mu|\geq\varepsilon)$ changes as the inputs N and ε change. Namely, you should think about $P(|\bar{X}_N-\mu|\geq\varepsilon)$ as a function over N and ε and make some 3D plot. Here the probability should be evaluated as the frequency from experiments of random samples, not directly using Hoefdding bounds, and then check if the results from Hoefdding bounds are consistent with your plots.

Question 6 (2 Points, Implementation Required)**.** Consider two coins, and write the random variable for the payoff from each coin as $X^{(1)}$ and $X^{(2)}$. The ground truth distribution for each coin is $P(X^{(1)}=0)=0.2$, $P(X^{(1)}=1)=0.8$, $P(X^{(2)}=0)=0.6$ with $P(X^{(2)}=1)=0.4$. Plot the total reward of T rounds of play

$$J(T)=\sum_{i=1}^{T}X^{(c_i)}$$

where $c_i\in\{1,2\}$ is the coin choice, as the number of plays T goes from 10 to 1000 for each of the following strategies. Keep in mind that $J(T)$ is a random variable. The x-axis should be T and y-axis is $J(T)$, and plot everything on the same graph so that the curves can be compared.

1. Explore-then-commit with $N=\lceil 0.2T\rceil$ ($\lceil\cdot\rceil$ is the ceiling function that gives you an integer).
2. Explore-then-commit with $N=\lceil\frac{1}{2}T^{\frac{2}{3}}(\log T)^{\frac{1}{3}}\rceil$, where $\log$ is the natural logarithm.
3. The ε-Greedy strategy with $\varepsilon=0.2$. (With $1-\varepsilon$ probability play the currently-best coin, and with ε probability the other).
4. The Upper Confidence Bound strategy, which plays the coin $i\in\{1,2\}$ that maximizes $\bar{X}^{(i)}+\sqrt{\frac{2\log T}{N^{(i)}}}$ in each round T.