Comprehensive Practice Problem Set

AIMLCZG512: Deep Reinforcement Learning

(Sessions 1 – 8)

Based on Course Handout & Exam Patterns

Contents

1 Session 1: Introduction to RL

Q1. RL vs Supervised Learning (Concept)

Explain the fundamental difference between Reinforcement Learning (RL) and Supervised Learning in terms of feedback and data structure.

Solution:

- **Supervised Learning:** The agent learns from a labeled dataset provided by a knowledgeable external supervisor. Each example consists of a situation and the correct action (label). The goal is to minimize the error between prediction and the true label.
- **Reinforcement Learning:** The agent learns by interacting with an environment. It receives feedback in the form of a scalar **reward signal**, which may be delayed. There is no external supervisor to tell the agent the "correct" action; it must discover optimal actions by trial and error.

Q2. Elements of RL (Definition)

List and briefly define the four main sub-elements of a Reinforcement Learning system.

Solution: 1. **Policy (π):** The agent's behavior function; a mapping from states to actions (or probabilities of actions). 2. **Reward Signal (R):** A scalar feedback signal from the environment indicating how well the agent is doing at a specific time step. 3. **Value Function (V):** A prediction of the total expected future reward from a state. It indicates the long-term desirability of states. 4. **Model of the Environment:** An optional component that mimics the behavior of the environment (predicting the next state and reward), used for planning.

Q3. Tic-Tac-Toe Value Function (Design)

In a Tic-Tac-Toe RL agent, how would you define the value $V(s)$ of a state s? What is the value of a state where the agent has won?

Solution: $V(s)$ represents the probability of winning from state s.

- If s is a state where the agent has won (3 in a row), $V(s) = 1$.
- If s is a state where the agent has lost, $V(s) = 0$.
- If s is a draw, $V(s) = 0.5$ (or 0 depending on formulation).
- For intermediate states, $V(s)$ is estimated based on experience.

Q4. Tic-Tac-Toe Update Rule (Formulation)

Write the temporal difference update rule used to train the Tic-Tac-Toe agent discussed in Sutton & Barto.

Solution:

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

Where:

- S_t is the state before the move.
- S_{t+1} is the state after the move.
- α is the step-size parameter (learning rate).
- The term $[V(S_{t+1}) - V(S_t)]$ is the temporal difference error.

Q5. Exploration vs Exploitation (Concept)

Define Exploration and Exploitation. Why is there a trade-off?

Solution:

- **Exploitation:** Choosing the action that the agent currently believes to be the best (greedy) to maximize immediate reward.
- **Exploration:** Choosing a non-greedy action to gather more information about the environment, potentially discovering better long-term strategies.

Trade-off: You cannot explore and exploit simultaneously. Exploiting maximizes current performance but risks stagnation. Exploring sacrifices short-term reward for potential long-term gain.

Q6. Reward Hypothesis (Theory)

State the Reward Hypothesis.

Solution: "That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)."

Q7. Delayed Reward (Concept)

Give an example of a scenario where an action yields a low immediate reward but a high long-term value.

Solution: Chess: Sacrificing a Queen (high immediate penalty/loss of material) to force a Checkmate sequence (winning the game). The immediate reward is negative, but the value of the state leading to victory is maximal.

Q8. Evolutionary Methods vs RL (Comparison)

How do evolutionary methods (like Genetic Algorithms) differ from Value Function-based RL methods?

Solution: Evolutionary methods search the space of *policies* directly. They evaluate a policy over an entire lifetime/episode and keep the successful ones. They typically do not use the structure of the individual states or value functions to learn *during* the episode. RL methods (Value-based) learn the value of specific states/actions to guide decisions at every step.

Q9. Agent-Environment Boundary (Concept)

Where is the boundary drawn between the Agent and the Environment?

Solution: The boundary is defined by control, not physical location. Anything that the agent cannot arbitrarily control is considered part of the environment. For example, a robot's physical motors and sensors are part of the environment; only the decision-making unit is the agent.

Q10. Tic-Tac-Toe Symmetries (Optimization)

How can symmetries be used to improve learning efficiency in Tic-Tac-Toe?

Solution: Many states in Tic-Tac-Toe are rotations or reflections of each other. By mapping all symmetric configurations to a single canonical state representation, the state space size is reduced. Learning about one state automatically updates the value for all its symmetric equivalents, speeding up convergence.

2 Session 2: Multi-Armed Bandits (MAB)

Q11. k-Armed Bandit Problem (Definition)

Define the k-armed bandit problem formally.

Solution: You are faced with k different options (actions). After each choice, you receive a numerical reward chosen from a stationary probability distribution depending on the action selected. Your objective is to maximize the expected total reward over some time period (e.g., 1000 time steps).

Q12. Action-Value Estimation (Calculation)

Given action a has been selected 3 times with rewards $\{5, 10, 0\}$. Calculate the sample-average estimate $Q_4(a)$.

Solution:

$$Q_t(a) = \frac{\text{Sum of rewards taken with action } a}{\text{Number of times action } a \text{ taken}}$$

$$Q_4(a) = \frac{5 + 10 + 0}{3} = \frac{15}{3} = 5.0$$

Q13. Incremental Implementation (Derivation)

Derive the incremental update rule $Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$.

Solution:

$$\begin{aligned} Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\ &= \frac{1}{n}(R_n + \sum_{i=1}^{n-1} R_i) \\ &= \frac{1}{n}(R_n + (n-1)Q_n) \\ &= \frac{1}{n}(R_n + nQ_n - Q_n) \\ &= Q_n + \frac{1}{n}(R_n - Q_n) \end{aligned}$$

Q14. Epsilon-Greedy (Calculation)

Consider a 2-armed bandit. $Q(1) = 2, Q(2) = 5$. $\epsilon = 0.1$. What is the probability of selecting Action 1?

Solution: Action 2 is greedy. Action 1 is non-greedy.

- Probability of exploring (random action): $\epsilon = 0.1$.
- Random action can pick Action 1 or 2 with prob 0.5.
- Total Prob(Action 1) $= \frac{\epsilon}{2} = \frac{0.1}{2} = 0.05$.
- Total Prob(Action 2) $= (1 - \epsilon) + \frac{\epsilon}{2} = 0.9 + 0.05 = 0.95$.

Answer: 0.05 (or 5%).

Q15. Optimistic Initial Values (Concept)

How does initializing $Q_1(a) = +5$ (where true rewards are ≈ 1) encourage exploration?

Solution: If $Q_1(a) = +5$, the agent is "optimistic". When it tries an action and gets a reward (e.g., 1), the reward is *lower* than the estimate. The estimate $Q(a)$ decreases. The agent then switches to other actions that still have the high initial estimate (+5). This forces the agent to try all actions at least once before settling.

Q16. UCB Formula (Calculation)

Calculate the Upper Confidence Bound (UCB) value for Action 1 at time $t = 10$. Given: $Q_t(1) = 2.0$, action 1 has been selected $N_t(1) = 4$ times. Exploration parameter $c = 2$. use ln.

Solution:

$$A_t = \text{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

$$\text{UCB} = 2.0 + 2\sqrt{\frac{\ln 10}{4}}$$

$\ln 10 \approx 2.302$.

$$\text{UCB} = 2.0 + 2\sqrt{\frac{2.302}{4}} = 2.0 + 2\sqrt{0.5755}$$

$$\text{UCB} = 2.0 + 2(0.758) = 2.0 + 1.517 = 3.517$$

Q17. Non-Stationary Problem (Concept)

Why is the sample-average method poor for non-stationary bandit problems? What is the fix?

Solution: Sample-average gives equal weight to all rewards, even those from the distant past. In non-stationary problems, true reward probabilities change over time, so old rewards are irrelevant. **Fix:** Use a constant step-size parameter $\alpha \in (0, 1]$ (exponential recency-weighted average) instead of $1/n$.

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n]$$

Q18. Gradient Bandit Algorithm (Concept)

In Gradient Bandits, we learn "preferences" $H_t(a)$ instead of values. How are probabilities derived from preferences?

Solution: Using the **Softmax** distribution:

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}}$$

Q19. Regret (Definition)

Define Total Regret in the context of the MAB problem.

Solution: Regret is the difference between the maximum possible expected reward and the actual reward collected.

$$L_T = \sum_{t=1}^{T} (V^* - \mathbb{E}[R_t])$$

Where V^* is the true value of the best arm. Minimizing regret is equivalent to maximizing cumulative reward.

Q20. Contextual Bandits (Concept)

How does a Contextual Bandit differ from a standard Multi-Armed Bandit?

Solution: In a standard MAB, the problem is a single state (or no state). In Contextual Bandits, the agent observes a "context" (state/feature vector) before choosing an action. The goal is to learn a mapping from context to the best action (Policy), rather than just finding the single best action globally. This is intermediate between MAB and full RL.

3 Session 3: Markov Decision Processes

Q21. Markov Property (Definition)

Mathematically define the Markov Property.

Solution: A state S_t has the Markov property if and only if:

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$$

This means the future dynamics depend *only* on the current state and action, not on the history of previous states or actions.

Q22. MDP Tuple (Definition)

List the 5 components of a finite MDP tuple.

Solution: (S, A, P, R, γ)

- S: Set of States.
- A: Set of Actions.
- P: State Transition Probability Function $P(s'|s, a)$.
- R: Reward Function $R(s, a, s')$.
- γ: Discount factor $\in [0, 1]$.

Q23. Return Calculation (Numerical)

An agent receives the reward sequence: $R_1 = 2, R_2 = 2, R_3 = 2 \ldots$ indefinitely. Calculate the Return G_0 if $\gamma = 0.5$.

Solution: This is a geometric series sum.

$$G_0 = \sum_{k=0}^{\infty} \gamma^k R_{k+1} = 2 + 0.5(2) + 0.25(2) + \ldots$$

$$G_0 = 2(1 + 0.5 + 0.25 + \ldots)$$

Sum of infinite geometric series $S = \frac{a}{1-r}$. Here $a = 1, r = 0.5$.

$$\text{Sum} = \frac{1}{1 - 0.5} = 2$$

$$G_0 = 2 \times 2 = 4$$

Q24. Bellman Expectation Equation for v_π (Derivation)

Derive the Bellman Expectation Equation for $v_\pi(s)$ in terms of $v_\pi(s')$.

Solution:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

Q25. Optimal Value Function (Definition)

Define $v_*(s)$ and $q_*(s, a)$.

Solution:

- $v_*(s) = \max_\pi v_\pi(s)$ for all $s \in S$. It is the maximum expected return achievable from state s by following any policy.
- $q_*(s, a) = \max_\pi q_\pi(s, a)$ for all $s \in S, a \in A$. It is the expected return of taking action a in state s and thereafter following an optimal policy.

Q26. Gridworld Formulation (Design)

Formulate a 3×3 Gridworld.

- States: Cells (0,0) to (2,2).
- Goal: (2,2) with terminal reward +10.
- Obstacle: (1,1) (cannot enter).
- Actions: Up, Down, Left, Right.
- Noise: 80% intended move, 20% random.

Write the transition prob $p(s'|s, a)$ for $s = (0, 0), a =$ Right.

Solution: Target square: (0,1).

- With 0.8 prob: Agent moves to (0,1).
- With 0.2 prob: Random move. Let's assume "random" means uniform over 4 directions (0.05 each).
- Effect of Random:
  - Right (0,1): $0.8 + 0.05 = 0.85$.
  - Down (1,0): 0.05.
  - Up (0,0 - wall): Stays at (0,0) with prob 0.05.
  - Left (0,0 - wall): Stays at (0,0) with prob 0.05.

Transitions: $P((0, 1)|(0, 0), R) = 0.85$ $P((1, 0)|(0, 0), R) = 0.05$ $P((0, 0)|(0, 0), R) = 0.10$

Q27. Recycling Robot (Dynamics)

In the Recycling Robot example, describe the transition dynamics for the action "Search" from state "High".

Solution: From state **High**, Action **Search**:

- With probability α: State remains **High**. Reward = R_{search}.
- With probability $1 - \alpha$: State becomes **Low**. Reward = R_{search}.

Q28. Episodic vs Continuing Tasks (Concept)

Differentiate between Episodic and Continuing tasks.

Solution:

- **Episodic:** Interaction breaks naturally into subsequences called episodes (e.g., a game of Chess). Each episode ends in a terminal state. The state resets.
- **Continuing:** Interaction goes on continually without a limit (e.g., a thermostat or robot balancing). There is no terminal state. Discounting $\gamma < 1$ is essential to keep returns finite.

Q29. Bellman Optimality Equation for v_* (Equation)

Write the Bellman Optimality Equation for $v_*(s)$ (max form).

Solution:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

Q30. Policy from Value (Calculation)

Given $v_*(s)$ for all states, how do you determine the optimal policy $\pi_*(s)$?

Solution: You perform a one-step greedy lookahead:

$$\pi_*(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

No search or further iteration is required if v_* is known accurately.

4 Session 4: Dynamic Programming (Basics)

Q31. DP Prerequisites (Concept)

What are the two main assumptions required to apply Dynamic Programming for solving MDPs?

Solution: 1. The environment acts as a perfect Markov Decision Process (Markov property holds). 2. We have access to a perfect **Model** of the environment (the transition probabilities $p(s'|s,a)$ and rewards $r(s,a,s')$ are known).

Q32. Policy Evaluation (Concept)

What is the goal of Policy Evaluation (Prediction)?

Solution: To compute the state-value function $v_\pi(s)$ for a specific arbitrary policy π. It answers "How good is this specific policy?"

Q33. Policy Evaluation Update (Calculation)

State A has two actions: Left (0.5 prob) and Right (0.5 prob). Left $\rightarrow$ State B (Reward 0). Right $\rightarrow$ State C (Reward 2). Given current estimates $V_k(B) = 10, V_k(C) = 20$. $\gamma = 0.9$. Calculate $V_{k+1}(A)$.

Solution:

$$V_{k+1}(A) = \sum_a \pi(a|A)[R + \gamma V_k(s')]$$

- Left: $0.5 \times [0 + 0.9(10)] = 0.5 \times 9 = 4.5$
- Right: $0.5 \times [2 + 0.9(20)] = 0.5 \times [2 + 18] = 0.5 \times 20 = 10$

$V_{k+1}(A) = 4.5 + 10 = 14.5$.

Q34. Policy Improvement Theorem (Theory)

State the Policy Improvement Theorem.

Solution: Let π and π' be any pair of deterministic policies such that, for all $s \in S$, $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. Then the policy π' must be as good as, or better than, π. That is, $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in S$.

Q35. Policy Iteration Steps (Algorithm)

List the steps of the Policy Iteration Algorithm.

Solution: 1. **Initialization:** Randomly initialize $V(s)$ and $\pi(s)$. 2. **Policy Evaluation:** Iteratively compute $v_\pi(s)$ until convergence ($\Delta < \theta$). 3. **Policy Improvement:** For each state s, update $\pi(s) \leftarrow \text{argmax}_a \sum p(s', r|s, a)[r + \gamma V(s')]$. 4. **Check Stability:** If $\pi_{new} == \pi_{old}$, stop (Optimal Policy found). Else, go to Step 2.

Q36. Deterministic Policy (Concept)

Why does Policy Improvement typically result in a deterministic policy?

Solution: The improvement step uses the argmax operator. It selects the single action that maximizes the Q-value. Even if the previous policy was stochastic, the greedy update essentially collapses the probabilities onto the single best action (or splits among ties), making it deterministic.

Q37. Initialization in DP (Practice)

Why is the value of the terminal state initialized to 0 and kept fixed?

Solution: The terminal state represents the end of an episode. No future rewards can be obtained from it. The definition of Return G_t implies the sum ends. If we assigned a non-zero value, it would imply infinite future rewards or incorrect returns propagating backwards.

Q38. In-Place DP (Optimization)

What is "In-Place" Dynamic Programming and why is it used?

Solution: Standard DP uses two arrays: V_{old} and V_{new}. In-Place DP uses a single array V. When updating state s, it uses the most recent values of neighbors (some might be from iteration k, some from $k+1$). **Benefit:** It reduces memory usage by half and often converges faster because updates propagate immediately within the same sweep.

Q39. Gridworld Example (Trace)

In a 4×4 gridworld with terminal states at top-left and bottom-right, and reward -1 per step: What is the value v_π of the state adjacent to the terminal state under a random policy? (Qualitative check).

Solution: It will be negative (e.g., -1.0 for optimal, but lower/more negative for random policy). Since the random policy might move away from the goal, the expected return is the negative of the expected number of steps to reach the terminal state. For a random walk, this takes several steps, so $v_\pi \approx -14$ or similar (exact value depends on full calculation).

Q40. Policy Improvement Stability (Concept)

What happens if the policy improvement step yields the same policy as before?

Solution: The algorithm has converged. The current policy π satisfies the Bellman Optimality Equation: $v_\pi(s) = \max_a q_\pi(s, a)$. Thus, π is the optimal policy π_*.

5 Session 5: DP (Value Iteration & Efficiency)

Q41. Value Iteration vs Policy Iteration (Comparison)

How does Value Iteration differ from Policy Iteration structurally?

Solution: Policy Iteration consists of two distinct loops: full Policy Evaluation (until convergence) followed by Policy Improvement. Value Iteration combines these by truncating the Policy Evaluation step to just **one sweep** (one update per state) and then immediately performing the max operation (improvement). It iterates the values directly toward v_* without explicit policy stabilization at each step.

Q42. Value Iteration Update Rule (Equation)

Write the update equation for Value Iteration.

Solution:

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$$

Q43. Generalized Policy Iteration (Concept)

Explain the concept of Generalized Policy Iteration (GPI).

Solution: GPI refers to the general idea of two interacting processes: 1. **Evaluation:** Making the value function consistent with the current policy. 2. **Improvement:** Making the policy greedy with respect to the current value function. In GPI, we don't need to wait for full completion of one process before switching to the other. Most RL methods are instances of GPI.

Q44. Asynchronous DP (Concept)

What is Asynchronous Dynamic Programming?

Solution: Standard DP sweeps through the entire state set S in a fixed order (Synchronous). Asynchronous DP updates states in any order, using whatever values of other states are currently available. Some states may be updated many times before others are updated once. This is useful for large state spaces where a full sweep is computationally prohibitive.

Q45. Efficiency of DP (Complexity)

What is the time complexity of one sweep of Value Iteration for an MDP with $|S|$ states and $|A|$ actions?

Solution: For each state ($|S|$), we iterate over all actions ($|A|$). For each action, we sum over possible next states ($|S|$). Complexity: $O(|S|^2|A|)$. (Note: If transitions are sparse with branching factor $b \ll |S|$, it is $O(|S| \cdot |A| \cdot b)$).

Q46. Bootstrapping (Definition)

What does "Bootstrapping" mean in the context of DP?

Solution: Bootstrapping means updating an estimate based on other estimates. In DP, $V(s)$ is updated using the existing values of successor states $V(s')$. The method does not wait for a final real outcome; it builds a guess upon a guess.

Q47. Convergence of Value Iteration (Theory)

Does Value Iteration guarantee convergence to v_*?

Solution: Yes, for finite MDPs with bounded rewards and $\gamma < 1$, the Bellman Optimality operator is a **contraction mapping**. Applying it repeatedly brings the value function closer to v_* and guarantees convergence to the unique fixed point v_*.

Q48. Gambler's Problem (Application)

In the Gambler's Problem (betting on coin flips to reach \$100), why does the optimal value function structure look fractal/hierarchical?

Solution: The optimal policy depends on the binary representation of the capital. With $p_h = 0.4$, it is better to make bold bets to reach the goal quickly before the house edge drains the capital. The value spikes at 50, 25, 75, etc., because those amounts allow the gambler to reach 100 with a specific small number of consecutive wins.

Q49. Prioritized Sweeping (Concept)

How does Prioritized Sweeping improve DP efficiency?

Solution: Instead of sweeping states in a fixed order, Prioritized Sweeping maintains a priority queue of states based on the magnitude of their **Bellman Error** (how much the value would change if updated). States with large potential updates are processed first, propagating rewards through the state space much faster.

Q50. Curse of Dimensionality (Problem)

Explain the "Curse of Dimensionality" in DP.

Solution: The number of states often grows exponentially with the number of state variables. For example, a robot with n degrees of freedom might have k^n discrete states. DP requires iterating over all states, making it intractable for large complex problems (millions/billions of states).

6 Session 6: Monte Carlo Methods

Q51. Monte Carlo Idea (Concept)

How do Monte Carlo methods estimate value functions?

Solution: MC methods estimate $V(s)$ by averaging the returns observed after visiting state s across many episodes.

$$V(s) \approx \frac{\text{Sum of Returns from } s}{\text{Number of visits to } s}$$

They require no model of the environment dynamics (Model-Free).

Q52. First-Visit vs Every-Visit MC (Distinction)

Difference between First-Visit MC and Every-Visit MC.

Solution:

- **First-Visit MC:** In a single episode, if state s is visited multiple times, only the return following the *first* visit is included in the average.
- **Every-Visit MC:** All visits to state s in an episode are treated as separate start points, and their following returns are averaged.

Both converge to $v_\pi(s)$ as the number of episodes goes to infinity.

Q53. MC Calculation (Numerical)

Episode: $S_1, +1, S_2, +2, S_1, +3, T$. No discount ($\gamma = 1$). Calculate the First-Visit MC target for S_1.

Solution: First visit to S_1 occurs at $t = 0$. Return $G_0 = R_1 + R_2 + R_3 = 1 + 2 + 3 = 6$. Note: The second visit to S_1 (at $t = 2$) is ignored for First-Visit calculation. **Target:** 6.

Q54. Exploring Starts (Assumption)

What is the assumption of "Exploring Starts" and why is it needed for MC Control?

Solution: Assumption: Every state-action pair (s, a) has a non-zero probability of being selected as the *start* of an episode. **Reason:** MC methods only learn about states/actions they visit. If a deterministic policy is followed, many actions will never be taken. Exploring starts ensures all state-action pairs are visited infinitely often to ensure convergence to the global optimum.

Q55. On-Policy vs Off-Policy MC (Definition)

Define On-Policy and Off-Policy learning.

Solution:

- **On-Policy:** The agent evaluates and improves the same policy (π) that it uses to generate the data (behavior). E.g., Epsilon-Greedy MC.
- **Off-Policy:** The agent evaluates a target policy (π) while following a different behavior policy (b) to generate data. E.g., Using Importance Sampling.

Q56. Importance Sampling Ratio (Calculation)

Target policy π: Always Right ($\pi(R) = 1.0, \pi(L) = 0$). Behavior policy b: 50% Right, 50% Left. Episode observed: $S_{start} \xrightarrow{R} S_1 \xrightarrow{R} S_{term}$. Calculate the Importance Sampling Ratio ρ.

Solution:

$$\rho = \prod_{t=0}^{T-1} \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Step 1: Action R. $\frac{\pi(R)}{b(R)} = \frac{1.0}{0.5} = 2$. Step 2: Action R. $\frac{\pi(R)}{b(R)} = \frac{1.0}{0.5} = 2$. Total $\rho = 2 \times 2 = 4$.

Q57. MC Control Algorithm (Structure)

Describe the basic structure of On-Policy MC Control.

Solution: 1. Initialize $Q(s, a)$ arbitrarily, π to be ϵ-greedy w.r.t Q. 2. Generate an episode using π: $S_0, A_0, R_1, \ldots, S_T$. 3. For each step in episode: $G \leftarrow$ return from step. $Q(S, A) \leftarrow$ Average(G). 4. Update π: For visited states, set $\pi(s)$ to be ϵ-greedy w.r.t updated Q.

Q58. Ordinary vs Weighted Importance Sampling (Concept)

Why is Weighted Importance Sampling generally preferred over Ordinary IS?

Solution:

- **Ordinary IS:** Unbiased but has infinite variance (can produce unstable, massive updates).
- **Weighted IS:** Biased (bias goes to 0 asymptotically) but has bounded variance. It generally results in more stable and efficient learning.

Q59. MC Advantages (Concept)

State two advantages of MC methods over DP.

Solution: 1. **Model-Free:** Does not require knowledge of transition probabilities $p(s'|s, a)$. 2. **Focus:** Can focus on estimating values for a single state or subset of states without updating the entire state space (just run episodes starting there).

Q60. Application Scenario (Design)

Would you use MC or DP for playing Blackjack? Why?

Solution: **MC.** Blackjack transition probabilities are complex (depend on the deck composition, dealer rules) and not easily available as a closed-form matrix. However, it is easy to simulate or play many hands of Blackjack. Episodic nature fits MC perfectly.

7 Session 7: Temporal Difference Learning

Q61. TD(0) Update Rule (Equation)

Write the update rule for TD(0) value estimation.

Solution:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Q62. TD Error (Calculation)

$V(A) = 10, V(B) = 12$. Transition $A \to B$ yields reward $R = 2$. $\gamma = 0.9$. Calculate the TD error δ.

Solution:

$$\delta = R + \gamma V(B) - V(A)$$

$$\delta = 2 + 0.9(12) - 10$$

$$\delta = 2 + 10.8 - 10 = 2.8$$

Q63. SARSA Update (Algorithm)

Explain the components of the SARSA tuple and its update rule.

Solution: Tuple: $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. It represents the transition from one state-action pair to the next state-action pair. Update Rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

It is an **On-Policy** control method.

Q64. Q-Learning Update (Algorithm)

Write the Q-Learning update rule. Why is it Off-Policy?

Solution:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

It is **Off-Policy** because the update target uses the max Q-value (greedy action), regardless of the actual action A_{t+1} selected by the agent's exploration policy (e.g., ϵ-greedy). It learns the optimal policy while behaving sub-optimally.

Q65. TD vs MC (Comparison)

Give one reason to prefer TD over MC.

Solution: **TD can learn online during an episode.** MC must wait until the episode ends to calculate the return G. TD updates immediately after a single step. This is crucial for continuing tasks or very long episodes.

Q66. Backup Diagram TD(0) (Concept)

Describe the backup diagram for TD(0).

Solution: The diagram starts at a state node s, goes down to an action node (via policy), then to a reward r and next state node s' (via environment). The backup creates a link back up to s. Unlike DP, it samples only **one** specific transition (s'), not all possible next states. Unlike MC, it stops at one step (s'), utilizing bootstrapping.

Q67. Batch TD (Concept)

What happens if you run TD(0) on a finite batch of data repeatedly until convergence?

Solution: It converges to the **Maximum Likelihood Estimate** of the Markov model. It effectively fits the transition probabilities and rewards consistent with the observed data and calculates the value function for that empirical model. (MC converges to minimum squared error on returns).

Q68. Maximization Bias (Problem)

What is Maximization Bias in Q-Learning?

Solution: Q-learning uses the max operator to estimate the value of the next state. If the Q-value estimates have random noise, max tends to pick the overestimated values, leading to a positive bias. For example, if true values are 0 but estimates are $-0.1, 0.1$, max is 0.1. This bias can degrade performance.

Q69. Double Q-Learning (Solution)

How does Double Q-Learning fix maximization bias?

Solution: It maintains two independent Q-tables, Q_1 and Q_2. To update Q_1, use Q_1 to *select* the best action ($a^* = \text{argmax}Q_1(s, a)$) but use Q_2 to *evaluate* its value: Target $= R + \gamma Q_2(s', a^*)$. This decouples selection from evaluation, removing the bias.

Q70. SARSA vs Q-Learning Cliff Walk (Application)

In the Cliff Walking example, why does SARSA take the "safer" path while Q-Learning takes the "optimal" (risky) path?

Solution:

- **Q-Learning** assumes optimal behavior in the future. It learns the path right along the cliff edge because that is the shortest path, assuming no mistakes.
- **SARSA** accounts for the current exploration policy (e.g., ϵ-greedy). It "knows" that walking on the edge carries a risk of randomly falling off due to ϵ. Thus, it learns a path further inland (safer) to avoid the penalty during training.

8 Session 8: Classification & Concepts

Q71. Model-Based vs Model-Free (Classification)

Classify the following algorithms as Model-Based or Model-Free: (a) Q-Learning, (b) Value Iteration, (c) Monte Carlo, (d) Dyna-Q.

Solution: (a) **Q-Learning:** Model-Free (learns from experience). (b) **Value Iteration:** Model-Based (requires P and R functions). (c) **Monte Carlo:** Model-Free (learns from experience). (d) **Dyna-Q:** Hybrid/Model-Based (learns a model from experience and plans with it).

Q72. Prediction vs Control (Definition)

What is the difference between the Prediction problem and the Control problem in RL?

Solution:

- **Prediction:** Given a fixed policy π, calculate the value function $v_\pi(s)$. (Policy Evaluation).
- **Control:** Find the optimal policy π_* that maximizes rewards. (Policy Improvement/Optimization).

Q73. Bootstrapping vs Sampling (Matrix)

Create a 2×2 matrix classifying DP, MC, and TD based on "Bootstrapping" and "Sampling".

Solution:

	No Bootstrapping	**Bootstrapping**
No Sampling	(Exhaustive Search)	**Dynamic Programming**
Sampling	**Monte Carlo**	**Temporal Difference**

- DP: Bootstraps (uses $V(s')$), No Sampling (uses expectation).
- MC: No Bootstraps (uses full G), Samples (experience).
- TD: Bootstraps (uses $V(s')$), Samples (experience).

Q74. On-Policy vs Off-Policy (General)

Is Q-learning On-Policy or Off-Policy? Why?

Solution: Off-Policy. Because it updates the Q-values based on the max estimate of the next state (Greedy Policy), while the agent actually moves using an exploratory policy (e.g., Epsilon-Greedy). The behavior policy differs from the target policy.

Q75. Tabular vs Function Approximation (Concept)

What is the limitation of Tabular RL methods?

Solution: Tabular methods store a value for every state-action pair in a table. This works for small state spaces. For real-world problems (images, robotics), the state space is huge or continuous. Tables run out of memory and cannot **generalize** to unseen states.

Q76. Convergence Speeds (Comparison)

Generally, which converges faster: TD or MC? Why?

Solution: TD generally converges faster. TD updates after every step, while MC must wait for the end of an episode. Also, TD often has lower variance because it depends on only one transition of randomness plus a stable estimate, whereas MC depends on the randomness of an entire trajectory.

Q77. Afterstates (Concept)

What is an "Afterstate" and when is it useful?

Solution: An afterstate is the state reached *after* the agent makes a move but *before* the environment responds (or if the environment is deterministic part of the move). Example: Tic-Tac-Toe. Placing an X leads to a specific board configuration. It is more efficient to learn the value of the "board position" (Afterstate) than the pair (State, Action), because different actions in different states might lead to the same board position.

Q78. Sample Efficiency (Metric)

Define Sample Efficiency. Which class of algorithms is generally more sample efficient: Model-Based or Model-Free?

Solution: Sample Efficiency measures how much data (environment interactions) an algorithm needs to reach a certain performance level. **Model-Based** methods are generally more sample efficient because they learn a model of the dynamics and can "simulate" millions of steps (Planning) for every real step taken, extracting more value from each piece of real data.

Q79. n-step TD (Concept)

What is n-step TD learning? Where does it sit between TD(0) and MC?

Solution: TD(0) updates based on 1 step ($R_{t+1} + \gamma V_t(S_{t+1})$). MC updates based on full episode (∞ steps). **n-step TD** updates based on n steps of real reward plus the estimated value of the state n steps later.

$$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

It bridges the gap between TD(0) (bias) and MC (variance).

Q80. Deadlay Triad (Theory)

What is the "Deadly Triad" in RL instability?

Solution: RL algorithms are prone to divergence (instability) when three elements are combined: 1. **Function Approximation** (e.g., Deep Networks). 2. **Bootstrapping** (TD updates). 3. **Off-Policy Training** (Training on data different from current policy). When all three are present (as in DQN), special care (Experience Replay, Target Networks) is needed to ensure stability.