Assignment 3.1: The Agent's Mind

MDPs, Bellman Equations, and the Nature of Reward

Objective

This assignment contains no Python code. It is a "Pen and Paper" exercise. Reinforcement Learning is mathematically grounded. If you cannot solve these simple environments by hand, you will not deeply understand how Monte Carlo control works.

Part I: The Essentials

Question 1: The "Cliff Walker" (Manual Calculation)

Consider a tiny 1D GridWorld with 4 states: S_0, S_1, S_2, S_{Term}.

- **Transitions:** From any state S_i, you can move **Right** (S_{i+1}) or **Left** (S_{i-1}).
- **Boundaries:** Moving Left from S_0 keeps you in S_0. Moving Right from S_2 takes you to S_{Term} (Game Over).
- **Rewards:**
  - Transition to S_{Term}: **+10**
  - Any other transition (Left or Right): **-1** (Living penalty)
- **Discount Factor (γ):** 0.9

The Tasks:

1. **Calculate Return (G_t):** An agent starts at S_1. It takes the path: $S_1 \xrightarrow{Right} S_2 \xrightarrow{Left} S_1 \xrightarrow{Right} S_2 \xrightarrow{Right} S_{Term}$.
   Calculate the total discounted return G_0 for this episode. Show your working using powers of γ.
2. **Value Function (v_π):** Consider a "Random Drunk" policy π where the agent chooses Left or Right with probability 0.5. Write down the **Bellman Expectation Equation** for the value of state S_2 (i.e., $v_\pi(S_2)$).
   Hint: Express $v_\pi(S_2)$ in terms of the immediate rewards and the values of its neighbors ($v_\pi(S_1)$ and $v_\pi(S_{Term})$).

Question 2: The Philosophy of Reward (Design)

One of the hardest parts of RL is designing the reward function. A poorly designed reward leads to "Reward Hacking."

Scenario: You are training a robot to clean a messy room.

- **State:** Camera image of the room.
- **Action:** Move, Suck, Idle.
- **Your Reward Design:** You give the agent a reward of **+1** every time its sensors detect that "Dust" has been "Sucked Up."

The "What If":

The agent eventually learns a policy that maximizes total reward, but the room never stays clean. In fact, the room gets dirtier over time. **Explain exactly what behavior the agent has likely learned to exploit your reward function.**
(Hint: Think about where the dust goes after it is sucked up).

Question 3: The Discount Factor (Concept)

The discount factor γ represents how much the agent cares about the future.

Part A: The Math

Why do we mathematically *need* $\gamma < 1$ for continuous (infinite horizon) tasks? What would happen to the value function $v_\pi(s)$ if the task never ends, rewards are always $+1$, and $\gamma = 1$?

Part B: The Intuition

Imagine a generic MDP.

- **Case 1:** $\gamma = 0$.
- **Case 2:** $\gamma = 0.99$.

Explain in plain English how the behavior of the agent differs in these two cases. Which agent is "Impulsive" and which is "Strategic"?

Question 4: The Brain Teaser (Nuance)

This question tests your understanding of the **Reward Hypothesis**.

Suppose we have an environment where the agent's goal is to reach a Goal State in the shortest number of steps.

- **Original Setup:** Every step gives a reward of $R = -1$. Reaching the goal ends the episode. $\gamma = 1$. The optimal policy finds the shortest path.

The Modification: A developer decides to "boost" the agent's morale by adding a constant $C = +2$ to every single reward.
Now, every step gives a reward of $R_{new} = (-1 + 2) = +1$.

The Question:

Does the optimal policy π_* change?

- If yes, describe what the new optimal agent will do.
- If no, explain why.

Hint: Think about what the agent wants to maximize. Does it want to finish the game, or keep collecting rewards?

Part II: Advanced Mechanics

The following questions are rigorous. They connect the abstract math of Week 3 to the computational reality of building an AI.

Question 5: Rigorous Derivation (Proof)

The Bellman Expectation Equation is often stated intuitively. Derive it formally from the definition of Return.

Let $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$.
Using the definition of return $G_t = R_{t+1} + \gamma G_{t+1}$ and the **Law of Iterated Expectations**, formally prove that:

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

Requirement: Show every step of the summation splitting. Do not skip the conditional probability expansions.

Question 6: The Linear Algebra of RL (Systems)

For a **fixed** policy π, the MDP collapses into a Markov Reward Process (MRP). The Bellman Equation becomes a system of linear equations.

$$v_\pi(s) = \sum_{s'} \mathcal{P}^\pi_{ss'}[\mathcal{R}^\pi_{ss'} + \gamma v_\pi(s')]$$

Let $\mathbf{v}_\pi$ be the column vector of state values. Let $\mathbf{P}^\pi$ be the $|\mathcal{S}| \times |\mathcal{S}|$ transition matrix. Let $\mathbf{R}^\pi$ be the reward vector.

1. **Matrix Form:** Rearrange the Bellman Equation into the form $A\mathbf{v}_\pi = b$ and solve for $\mathbf{v}_\pi$ symbolically using matrix inversion.
2. **The Computational Wall:** The game of Backgammon has roughly 10^{20} states. The time complexity of Matrix Inversion for an $N \times N$ matrix is roughly $O(N^3)$. Estimate how many years it would take a supercomputer (doing 10^{18} FLOPs/sec) to solve Backgammon analytically.
3. **Conclusion:** What does this imply about the necessity of methods like Monte Carlo?

Question 7: The "Model-Free" Dilemma (Design)

This question mathematically motivates why we use **Action-Values** $q(s, a)$ instead of **State-Values** $v(s)$ in Monte Carlo Control(will be covered in detail next week).

The Setup: You want to improve your policy. You act greedily. The greedy policy $\pi'(s)$ is the action that maximizes the expected return.

1. **Using $v(s)$:** Write down the formula for the greedy action $\pi'(s)$ using **only** the optimal state-value function $v_*(s)$ and the environment dynamics $p(s', r|s, a)$.

$$\pi'(s) = \arg\max_a \dots$$

2. **Using $q(s, a)$:** Write down the formula for the greedy action $\pi'(s)$ using **only** the optimal action-value function $q_*(s, a)$.

3. **The Comparison:** In a "Model-Free" environment (like Blackjack or the Real World), we do **not** know the transition probabilities $p(s', r|s, a)$ or the reward function beforehand. Based on your formulas above, explain why it is impossible to construct an optimal agent using only $v_*(s)$, but possible if we learn $q_*(s, a)$.

Submission Instructions

- Submit a PDF with your answers.
- Handwritten (scanned) or LaTeX submissions are both accepted.
- **Freshers:** Q1-Q4 are mandatory. Q5-Q7 are optional but recommended.
- **Sophomores:** Q1-Q7 are expected.