Markov Decision Processes

Minor in AI - IIT ROPAR

16th April, 2025

Algorithm Concept

Imagine you're an engineer working on an intelligent agent — perhaps a self-driving car or a trading bot — that must make decisions in an uncertain environment. You have a simulator at your disposal, which you've trained using historical data, assumptions, and approximations about how the real world behaves.

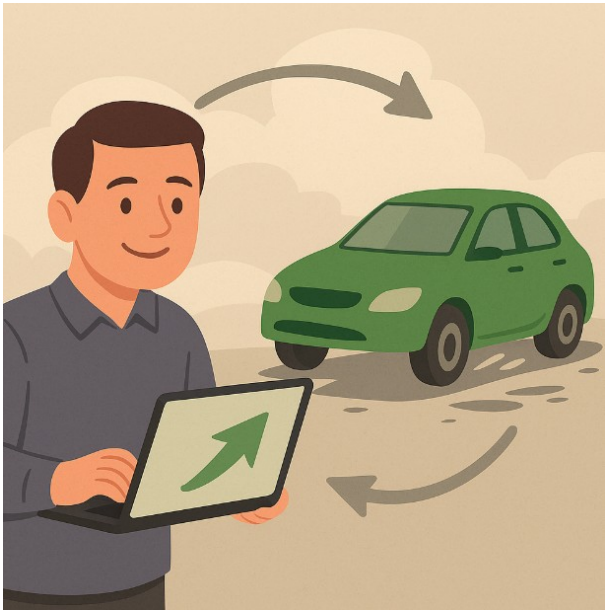
Now, picture yourself standing at a decision point. Your simulator suggests that among several possible actions, the **green action** is the best one — the one most likely to give you a good outcome. Trusting this recommendation, you execute the green action in the real world.

However, the real environment is messy, unpredictable, and often deviates from your idealized model. Instead of the high reward you anticipated, the actual result is disappointing — perhaps the car slightly veers off due to a slippery road, or the trade results in a loss due to market volatility. The reward you receive is lower than expected.

This discrepancy highlights a fundamental truth: **simulators are imperfect approximations**. What follows is a critical moment in the learning process — you measure the **error** between the predicted and actual outcomes. This error tells you something powerful: your beliefs need updating.

So, you return to your simulator and **adjust your internal belief** — specifically, you modify the probability distribution that guided your choice of the green action. You nudge it in the direction of what reality just taught you. Over time, as you keep taking actions, observing results, and adjusting beliefs, your simulator becomes smarter. It begins to mirror the real environment more closely.

This continuous loop of belief, action, observation, and belief revision is the essence of learning under uncertainty. It's the heartbeat of algorithms like Thompson Sampling, which we'll simplify and explore through mathematical tools like the Beta distribution. This brings us to a natural question: how do we formally capture such decision-making under uncertainty? That's where Markov Decision Processes come in.



Binary Reward Simplification and Beta Distribution

In certain reinforcement learning scenarios, particularly those with binary feedback, each action results in either a success (reward = 1) or a failure (reward = 0). A common example is a trading bot, where each trade yields either a profit (success) or a loss (failure).

To model such binary outcomes probabilistically, we use the **Beta distribution**, a conjugate prior for the Bernoulli and Binomial distributions. The Beta distribution is defined over the interval $[0, 1]$ and parameterized by two positive real numbers:

- α: Represents the number of observed successes (plus one if using a non-informative prior).
- β: Represents the number of observed failures (plus one if using a non-informative prior).

The probability density function of the Beta distribution is:

$$\text{Beta}(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$$

where $B(\alpha, \beta)$ is the Beta function, acting as a normalizing constant.

Interpretation:

- The mean of the Beta distribution is: $\mathbb{E}[X] = \frac{\alpha}{\alpha+\beta}$.
- The variance is: $\text{Var}[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$.

Initial State (Uninformed Prior): We often begin with $\alpha = 1, \beta = 1$, which corresponds to a uniform distribution over $[0, 1]$. This reflects complete uncertainty—every probability is equally likely.

Posterior Updating: As interactions occur:

- If the action results in a reward: increment α by 1.
- If the action results in no reward: increment β by 1.

This results in a posterior distribution that reflects updated beliefs based on empirical evidence.

Prior and Posterior

- **Prior:** The belief about the success probability of an action before observing any data (e.g., Beta(1,1)).
- **Posterior:** The updated distribution after incorporating observed outcomes (e.g., Beta(7,3) after 6 successes and 2 failures beyond the initial prior).

Simulation-Based Algorithm Summary (Thompson Sampling)

This section outlines a simple yet powerful algorithm used in multi-armed bandit settings and RL: **Thompson Sampling**, which leverages the Beta distribution.

1. **Initialization:** For each action a_i, initialize its Beta distribution as $\text{Beta}(1, 1)$.
2. **Repeat for each time step:**
   - **Sample** a random value $\theta_i \sim \text{Beta}(\alpha_i, \beta_i)$ for each action.
   - **Select** the action with the highest sampled value: $a_t = \arg\max_i \theta_i$.
   - **Execute** the action a_t in the environment.
   - **Observe** the binary reward $r_t \in \{0, 1\}$.
   - **Update** the Beta distribution of the selected action:
     - If $r_t = 1$: $\alpha_{a_t} \leftarrow \alpha_{a_t} + 1$
     - If $r_t = 0$: $\beta_{a_t} \leftarrow \beta_{a_t} + 1$

Outcome: Over time, the Beta distributions converge, and the agent increasingly favors actions with higher empirical success probabilities, balancing exploration and exploitation.

Reinforcement Learning Cycle Recap

Reinforcement Learning involves a continuous interaction between an agent and its environment:

- The **agent** observes the current **state** of the environment.
- It selects an **action** based on a policy.
- It receives a **reward** and transitions to a new **state**.
- It uses this information to update its strategy or policy.

This iterative loop is what allows the agent to learn optimal behavior over time.

Markov Decision Processes

To rigorously formulate this cycle, we use the concept of a **Markov Decision Process (MDP)**.

Definition

An MDP is formally defined as a 5-tuple:

$$(S, A, P, R, \gamma)$$

- S: Set of possible states
- A: Set of possible actions
- $P(s'|s, a)$: Transition probability function — the probability of reaching state s' given that the agent was in state s and took action a.
- $R(s, a, s')$: Reward function — the expected immediate reward received after transitioning from s to s' via action a.
- $\gamma \in [0, 1]$: Discount factor — determines the importance of future rewards compared to immediate ones.

Markov Property

The **Markov property** assumes that the future state depends only on the current state and action, not the full history:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_1, a_1, \ldots, s_t, a_t)$$

This memoryless assumption simplifies the modeling of decision processes and makes algorithms computationally feasible.

Chessboard Example

To intuitively understand the **Markov property**, let us consider a strategic game environment: the game of **chess**. Chess is a turn-based, fully observable, deterministic environment with a finite set of states and actions. It serves as an excellent analogy to explain why the current state is sufficient to make decisions and how historical states become redundant under the Markov assumption.

Scenario: Suppose you are a chess-playing agent tasked with making the next move. At a given time step t, you are presented with the current board configuration—this includes the positions of all pieces, castling rights, en passant possibility, and turn information (whose move it is).

Key Observation: Even though the current board configuration may have been reached via many different sequences of moves (i.e., different game histories), **none of that history is required** to determine the best move now. All the relevant information for decision-making is captured by the current configuration itself.

Markov Property in Action: Mathematically, this implies:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_1, a_1, \ldots, s_t, a_t)$$

In chess:

- s_t: The current board configuration.
- a_t: The move you choose to make (e.g., Knight to f3).
- s_{t+1}: The resulting configuration after making the move.

The probability of transitioning to s_{t+1} depends solely on s_t and a_t. Previous configurations $(s_1, \ldots, s_{t-1})$ and actions are not needed to calculate s_{t+1}, because:

- The rules of chess are deterministic.
- All legal information about the game (such as if castling is possible) is encoded in s_t.
- The next state is fully determined once the move a_t is applied to s_t.

Why This Matters: The Markov property allows reinforcement learning algorithms to model the chess environment more efficiently:

- It avoids the need to store and process full action histories.
- It allows value functions and policies to be defined purely over the current state.
- It reduces the computational complexity and memory overhead.

Components of an MDP

Component	**Description**
States (S)	All distinct situations or configurations the agent can find itself in.
Actions (A)	Choices or decisions the agent can make when in a given state.
Transition Probability (P)	Probability of ending up in state s' when taking action a from state s: $P(s'\|s, a)$.
Reward Function (R)	Immediate feedback from the environment in response to an action: $R(s, a, s')$.
Discount Factor (γ)	A factor between 0 and 1 that determines how future rewards are weighted relative to immediate ones.

Goal of an MDP

The primary objective in an MDP is to determine a policy π that maximizes the expected return from each state.

Given the tuple (S, A, P, R, γ), the return is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

And the optimal policy π^* is the one that yields the highest expected return from any state:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[G_t|s_t = s]$$

Understanding Policy and Value Functions

Policy: The Agent's Decision Rule

In reinforcement learning, a **policy** defines the agent's behavior — what action it takes in each state.

- **Deterministic Policy:** A direct mapping from state to action:

$$\pi : S \to A$$

  For any state $s \in S$, the agent always chooses action $a = \pi(s)$.

- **Stochastic Policy:** A probability distribution over actions for each state:

$$\pi(a \mid s)$$

  In this case, the agent chooses action a with probability $\pi(a \mid s)$ when in state s.

Goal: Find an *optimal policy* π^* that maximizes the expected cumulative reward over time.

Value Functions: Quantifying Long-Term Reward

Value functions measure how good it is to be in a state, or to take an action in a state, under a given policy π.

1. **State-Value Function $V^\pi(s)$:**

   This function gives the expected return when starting in state s and following policy π thereafter:

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right]$$

   - $\gamma \in [0, 1]$ is the **discount factor** for future rewards.
   - R_{t+1} is the reward received at time step $t + 1$.
   - $\mathbb{E}^\pi$ denotes the expected value assuming actions are chosen according to policy π.

2. **Action-Value Function $Q^\pi(s, a)$:**

   This function gives the expected return starting from state s, taking action a, and thereafter following policy π:

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right]$$

   - This tells us how good it is to take action a in state s, before resuming policy π.
   - Used extensively in Q-learning and other value-based methods.

Key Idea: The optimal policy π^* can be derived from the optimal value functions V^* or Q^*, which maximize expected returns.

Bellman Expectation Equation for $V^\pi(s)$

The Bellman Expectation Equation provides a recursive definition for the state-value function under a policy π. It relates the value of a state to the expected immediate reward and the value of subsequent states.

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s, a) \left[R(s, a, s') + \gamma V^\pi(s') \right]$$

Interpretation of each component:

- $V^\pi(s)$: Expected cumulative return starting from state s and following policy π.
- $\pi(a|s)$: Probability of choosing action a in state s under policy π.
- $P(s'|s,a)$: Probability that action a in state s leads to next state s'.
- $R(s,a,s')$: Expected immediate reward after transitioning from s to s' via a.
- γ: Discount factor (controls preference for immediate vs. future rewards).
- $V^\pi(s')$: Value of the next state s', assuming policy π is followed.

In essence: The value of a state is the expected sum of immediate reward and discounted future rewards, weighted by the policy and transition probabilities.

This equation enables iterative computation of value functions and forms the foundation for algorithms such as:

- Policy Evaluation
- Policy Iteration
- Value Iteration

Key Takeaways

- Simulators provide approximations; real-world feedback is essential for refining decisions.
- Learning involves updating beliefs based on discrepancies between expected and observed outcomes.
- The Beta distribution is a flexible tool for modeling binary outcomes using Bayesian updating.
- Thompson Sampling balances exploration and exploitation through probabilistic action selection.
- Reinforcement learning is driven by interaction: observe, act, receive reward, and update.
- Markov Decision Processes (MDPs) formally define decision-making under uncertainty.
- The Markov property implies the future depends only on the current state and action.
- Chess illustrates the Markov property—decisions depend only on the current board state.
- MDPs are defined by states, actions, transition probabilities, rewards, and discount factors.
- A policy defines the agent's decision-making strategy; it can be deterministic or stochastic.
- Value functions estimate long-term return; action-value functions include the current action.
- The Bellman equation relates a state's value to immediate and future rewards recursively.
- Optimal policies can be derived from value functions using dynamic programming.