Monte Carlo in Reinforcement Learning

Minor in AI - IIT ROPAR

21st April, 2025

The Tale of Explorer Bot A7: Learning in the Unknown

Imagine a futuristic world where Earth has discovered a new habitable planet—**Zeburon-9**. It's rich in rare resources, full of unpredictable terrains, and governed by physics unlike any found on Earth. The problem? It's uncharted territory. There's no map, no guide, and definitely no model that explains how things work. The best minds on Earth can only guess what might happen when you enter a forest of floating rocks or walk across a glowing energy lake.

Enter **Explorer Bot A7**, a sophisticated AI-powered robot, built to survive, explore, and learn. But here's the kicker: A7 cannot be pre-programmed with rules about this alien world. Why? Because no one knows the rules. No one knows what stepping on a crystal might do, or whether following a strange humming noise leads to treasure or danger. The agency back on Earth has only given the bot one guiding principle: *Learn from your experience.* Every journey you make, every path you follow, every strange anomaly you encounter—record it, and try to get better next time.

How does Explorer Bot A7 make sense of this chaotic, uncertain world? This is where **Monte Carlo methods in Reinforcement Learning (RL)** come into play.

What is Monte Carlo in Reinforcement Learning?

In reinforcement learning, we often think of an agent—like Explorer Bot A7—that interacts with an environment in order to maximize some cumulative notion of reward. In ideal scenarios, we might have full access to how the world works: we might know the probability that stepping on a glowing platform leads to a lava trap, or the exact reward from reaching a certain destination. This is the world of **Dynamic Programming**—where the rules are known, and planning can be done like solving a math puzzle.

But Explorer Bot A7 doesn't have this luxury. The alien world of Zeburon-9 is a **black box**. It doesn't expose its rules. The bot can only learn by trying, observing the outcomes, and remembering what worked. This kind of scenario calls for **model-free reinforcement learning**, and Monte Carlo methods are among the most foundational tools in that toolbox.

Monte Carlo methods rely on one elegant, intuitive idea: *learn from complete experiences.* The bot goes on a journey—an **episode**—starting from some initial position, making decisions, facing consequences, and eventually reaching some terminal state (perhaps finding a rare gem, or being zapped by a security drone). Once the episode is over, the bot looks back at the entire experience and calculates the total reward it received. This is called the **return**.

Instead of trying to guess what happens at every single step (which would require knowing the environment's dynamics), Monte Carlo methods just say: *"Let's wait until the end of the journey, and then look at the whole picture."*

By averaging the returns from many such episodes, the bot begins to estimate **value functions**—that is, it starts to understand which states are generally good (leading to high returns) and which ones are bad (leading to failure or little reward). Over time, with enough episodes, this averaging approach becomes remarkably powerful.

Why Monte Carlo Methods? The Case for Simplicity and Power

Why are Monte Carlo methods so important—especially in alien or unknown environments like Zeburon-9?

- **Simplicity:** Monte Carlo methods don't require any knowledge of how the world transitions from one state to another. There's no need for probabilities or models. You just sample real experience, the way any human or animal would: trial and error, memory, and reflection.
- **Realism:** In many real-world environments—whether it's an autonomous car driving in city traffic, or a finance bot navigating volatile markets—we rarely have perfect models. Monte Carlo methods learn from the world directly.
- **Episodic Suitability:** Monte Carlo methods work beautifully in **episodic settings**, where each task or experience has a clear beginning and end—like a chess game or an exploration mission.

Learning to Predict: Monte Carlo for Value Estimation

One major use of Monte Carlo methods is in **prediction**. Suppose the agency back on Earth gives Explorer Bot A7 a fixed policy—a way of behaving—and asks: *"Can you tell us how good this policy is?"*

Using Monte Carlo prediction, the bot runs many episodes following the fixed policy. After each episode, it records the return (the total cumulative reward from start to finish). Over time, it keeps an average of these returns for each state it visits. This average becomes the estimate of the **value function** under the current policy.

For instance, starting in a misty canyon and following the fixed policy may lead to an average return of +20. The bot learns that this state is valuable under the current strategy. If another location yields -5 on average, it marks that as a less desirable state.

Learning to Control: Monte Carlo for Policy Improvement

Prediction is only half the game. Explorer Bot A7 doesn't just want to understand the world—it wants to **do better**.

This brings us to the second use of Monte Carlo methods: **control**. Here, the goal is to *improve* the policy. The bot might start with a basic strategy—choosing actions at random. But after gathering data on which actions lead to better returns, it can start **favoring** those actions more often.

This creates a loop:

1. Use Monte Carlo to evaluate the current policy.
2. Use the value estimates to improve the policy.
3. Repeat the process to converge toward an optimal policy.

One important concept here is **exploration vs. exploitation**. The bot must sometimes try new actions to discover better paths (exploration), while at other times it chooses the best-known action (exploitation). Monte Carlo methods often use strategies like ϵ-greedy policies to maintain this balance.

Fundamental Concepts and Notation in Reinforcement Learning

In reinforcement learning, we model problems where an agent interacts with an environment over time to maximize some notion of cumulative reward. Below are the core concepts and mathematical notations used to define this process.

1. Return

The **return** G_t is the total accumulated reward from time step t onward, possibly discounted by a factor γ:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

Where:

- R_{t+k+1} is the reward received $k+1$ steps after time t
- $\gamma \in [0, 1]$ is the *discount factor*, controlling the importance of future rewards
- T is the time step at which the episode ends (for episodic tasks)

Interpretation: The agent values immediate rewards more than future ones if $\gamma < 1$, which encourages short-term gains. If $\gamma = 1$, the agent values future rewards equally, useful in undiscounted tasks.

2. Recursive Form of Return

The return can be expressed recursively as:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Interpretation: The return at time t equals the immediate reward plus the discounted return from the next time step onward. This recursive formulation is fundamental to many RL algorithms, including Monte Carlo and Temporal Difference (TD) methods.

3. State-Value Function

The **state-value function** under a policy π is the expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Where:

- $V^\pi(s)$ is the expected return when starting in state s and following policy π
- $\mathbb{E}_\pi[\cdot]$ denotes the expectation under policy π

Interpretation: It tells us how good it is to be in a particular state, assuming the agent follows policy π thereafter.

4. Action-Value Function

The **action-value function** under a policy π is the expected return starting from state s, taking action a, and then following policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Where:

- $Q^\pi(s, a)$ represents the expected return after taking action a in state s and then behaving according to policy π

Interpretation: It evaluates the usefulness of an action in a given state under a specific policy.

5. Policy

A **policy** $\pi(a \mid s)$ is a mapping from states to probabilities of selecting each action:

$$\pi(a \mid s) = \Pr(A_t = a \mid S_t = s)$$

Where:

- π can be deterministic (one action per state) or stochastic (a probability distribution over actions)
- A_t is the action chosen at time t, and S_t is the current state

Interpretation: The policy is the agent's strategy—it defines how the agent chooses actions based on the current state.

6. Discount Factor

The **discount factor** γ determines the present value of future rewards:

$$0 \leq \gamma \leq 1$$

Where:

- $\gamma = 0$ makes the agent myopic (only cares about immediate rewards)
- $\gamma = 1$ considers future rewards as important as immediate ones (used in undiscounted tasks)

Interpretation: A smaller γ leads to short-term planning, while a higher γ encourages long-term strategies.

Monte Carlo Methods Overview

Monte Carlo (MC) methods are a class of model-free reinforcement learning techniques. They are used for learning value functions and making decisions based solely on experience, without requiring a model of the environment.

Main Categories

Monte Carlo algorithms can be broadly categorized into:

1. **Prediction (Policy Evaluation)**: Estimate the value function $V^\pi(s)$ for a fixed policy π.
   - **First-Visit MC Prediction**: Considers only the first occurrence of a state in each episode.
   - **Every-Visit MC Prediction**: Considers all occurrences of the state in each episode.
2. **Control (Policy Improvement)**: Improve the policy based on value estimates.
   - **MC with Exploring Starts**: Requires starting from random state-action pairs.
   - **MC Control with ϵ-Greedy**: Uses ϵ-greedy exploration for improved learning without the need for random starts.

First-Visit Monte Carlo Prediction

Goal

Estimate the state-value function $V^\pi(s)$ under a fixed policy π, by averaging the returns following the first visit to each state.

Key Idea

- Sample complete episodes using policy π.
- For each state s, consider only the **first occurrence** of s in the episode.
- Compute the return G_t from that point onward and update the estimate of $V(s)$.

Return

The return G_t from time step t onward is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

Update Rule

Let $G^{(i)}(s)$ be the return following the first visit of state s in the i-th episode. Then:

$$V(s) \leftarrow \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s)$$

Where:

- $N(s)$ is the number of episodes in which state s was visited **first**.

Algorithm (Pseudo-code Summary)

1. Initialize $Returns(s) \leftarrow 0$, $N(s) \leftarrow 0$ for all states s
2. For each episode:
   - Generate an episode $(s_0, a_0, r_0, \ldots, s_T)$
   - For each time step $t = T - 1$ down to 0:
     - If s_t is the first occurrence of state s in the episode:
       * Compute G_t
       * Update: $Returns(s_t) \leftarrow Returns(s_t) + G_t$
       * Increment: $N(s_t) \leftarrow N(s_t) + 1$
3. Compute $V(s) = \frac{Returns(s)}{N(s)}$

Every-Visit Monte Carlo Prediction

Goal

Estimate the state-value function $V^\pi(s)$ under a fixed policy π, by averaging the returns for **every occurrence** of a state in an episode.

Key Idea

- Sample full episodes using policy π
- For each occurrence of each state s, compute the return from that time step onward.
- Update $V(s)$ using all such returns.

Return

Same as First-Visit:

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

Update Rule

Let $G_j^{(i)}(s)$ be the return from the j-th occurrence of state s in episode i. Then:

$$V(s) \leftarrow \frac{1}{N(s)} \sum_{i,j} G_j^{(i)}(s)$$

Where:

- $N(s)$ is the total number of **times** state s is visited across all episodes.

Algorithm (Pseudo-code Summary)

1. Initialize $Returns(s) \leftarrow 0$, $N(s) \leftarrow 0$ for all states s
2. For each episode:
   - Generate an episode $(s_0, a_0, r_0, \ldots, s_T)$
   - For each time step $t = T - 1$ down to 0:
     - Compute G_t
     - Update: $Returns(s_t) \leftarrow Returns(s_t) + G_t$
     - Increment: $N(s_t) \leftarrow N(s_t) + 1$
3. Compute $V(s) = \frac{Returns(s)}{N(s)}$

Comparison: First-Visit vs Every-Visit

- **First-Visit MC:**
  - Updates based only on the first occurrence of each state per episode
  - Lower variance but slower updates
  - Good for theoretical convergence
- **Every-Visit MC:**
  - Updates based on all occurrences of a state in an episode
  - Faster learning but slightly higher variance

Monte Carlo Control with Exploring Starts

Goal

To find the optimal policy π^* by learning the optimal action-value function $Q^*(s, a)$, and improving the policy iteratively through experience.

Key Idea

- Generate episodes that start from randomly chosen **state-action pairs** — known as **Exploring Starts**.
- Estimate the action-value function $Q(s, a)$ using sampled returns.
- Improve the policy π by acting greedily with respect to $Q(s, a)$.

Return

The return G_t at time step t is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

Update Rule

Let $G^{(i)}(s, a)$ be the return following the first occurrence of pair (s, a) in episode i. Then:

$$Q(s, a) \leftarrow \frac{1}{N(s, a)} \sum_{i=1}^{N(s,a)} G^{(i)}(s, a)$$

$$\pi(s) \leftarrow \arg\max_a Q(s, a)$$

Requirements

- Every state-action pair must be visited infinitely often (Exploring Starts guarantees this).
- The policy must continually improve towards the optimal policy.

Pseudocode

Input: Set of states S, actions A, discount factor γ, number of episodes n_{ep}
Output: Optimal policy π^*, and corresponding action-value function $Q(s, a)$

1. Initialize:
   - $Q(s, a) \leftarrow 0$ for all $s \in S, a \in A$
   - $N(s, a) \leftarrow 0$ for all $s \in S, a \in A$
   - $\pi(s) \leftarrow \text{random}(A)$
2. For each episode $i = 1$ to n_{ep}:
   (a) Choose random state-action pair (s_0, a_0) as starting point (Exploring Start)
   (b) Generate an episode:
   $$(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots, (s_{T-1}, a_{T-1}, r_{T-1})$$
   using policy π
   (c) For each time step $t = T - 1$ down to 0:
   - Compute return:
   $$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$
   - Let (s_t, a_t) be the state-action pair at time t
   - Update:
   $$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$
   $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)} (G_t - Q(s_t, a_t))$$
   - Improve policy:
   $$\pi(s_t) \leftarrow \arg\max_a Q(s_t, a)$$

Remarks

- This method is guaranteed to converge to the optimal policy π^* and action-value function $Q^*(s, a)$, given infinite episodes and proper exploring starts.
- However, in practice, true exploring starts are often infeasible — which motivates alternatives like ϵ-greedy exploration in MC Control.

Monte Carlo Control with ε-Greedy Policy

Goal

To learn the optimal policy π^* using sampled episodes generated under an ε-greedy exploration strategy — without requiring exploring starts.

Key Idea

- Instead of starting episodes from arbitrary state-action pairs, encourage exploration using an ε-greedy policy.
- With probability ε, take a random action (exploration).
- With probability $1 - \varepsilon$, take the greedy action (exploitation).
- Estimate the action-value function $Q(s, a)$ from episodes.
- Improve the policy iteratively to become more greedy over time.

Exploration Policy Definition

Let $A(s)$ be the set of available actions in state s. The ε-greedy policy $\pi(a|s)$ is defined as:

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A(s)|}, & \text{if } a = \arg\max_{a'} Q(s, a') \\ \dfrac{\varepsilon}{|A(s)|}, & \text{otherwise} \end{cases}$$

This ensures that:

- All actions are explored with non-zero probability.
- The action with the highest current value estimate is preferred.

Update Rule

For a given state-action pair (s_t, a_t) at time t, we compute the return G_t, and then use an incremental update for $Q(s_t, a_t)$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[G_t - Q(s_t, a_t)\right]$$

where:

- G_t is the return following (s_t, a_t)
- $\alpha \in (0, 1]$ is the learning rate

Advantages

- Does not require exploring starts.
- Easily implemented in continuous, stochastic environments.
- Balances exploration and exploitation.

Pseudocode

Input: States S, Actions A, Discount factor γ, Exploration rate ε, Learning rate α, Number of episodes n_{ep}
Output: Approximated optimal action-value function $Q(s, a)$, and greedy policy π

1. Initialize:
   - $Q(s, a) \leftarrow 0$ for all $s \in S, a \in A$
2. For each episode $i = 1$ to n_{ep}:
   (a) Initialize s_0
   (b) Generate an episode $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T)$ using ε-greedy policy π derived from Q
   (c) For each time step $t = T - 1$ down to 0:
   - Compute return:
   $$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$
   - Update:
   $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[G_t - Q(s_t, a_t)\right]$$

Remarks

- If $\varepsilon \to 0$ over time, the policy becomes greedy, and this method can converge to the optimal policy π^*.
- A decaying ε schedule is often used in practice to balance exploration and convergence.

Method	Task	Sampling Type	Output
First-Visit MC	Prediction	First occurrence of s	$V(s)$
Every-Visit MC	Prediction	Every occurrence of s	$V(s)$
MC with Exploring Starts	Control	Random (s, a) starts	$Q(s, a), \pi$
MC with ε-Greedy Policy	Control	ε-greedy policy over time	$Q(s, a), \pi$

Prediction methods estimate value functions for a fixed policy; control methods aim to improve the policy toward optimality.

Key Takeaways

- Monte Carlo methods learn from complete episodes, using the total reward from each episode to estimate the value of states or actions by summing rewards from a point to the end of the episode.
- MC is model-free, meaning it does not require knowledge of the environment's transition probabilities or reward function. It learns directly from the experience of the agent interacting with the environment.
- Value estimates are based on averaging actual returns observed during episodes. As more episodes are collected, the value estimates become more accurate, reflecting real rewards.
- In First-visit MC, only the first occurrence of a state in an episode is used to update its value, while Every-visit MC uses all occurrences of a state in an episode to calculate the average return.
- Exploring starts ensures that every state-action pair is visited by starting episodes from randomly selected states, guaranteeing that the agent explores diverse situations.
- On-policy MC control uses ϵ-soft policies, where the agent takes random actions with a small probability to ensure sufficient exploration of all actions while still improving the policy.
- MC methods are ideal for episodic tasks where episodes always terminate, as they rely on complete episodes to calculate returns and learn from them.