



## DEPARTMENT OF COMPUTER SCIENCE ST. FRANCIS XAVIER UNIVERSITY

St. Francis Xavier University  
Department of Computer Science  
CSCI-531 - Reinforcement Learning  
Practice Exercises: Monte Carlo Methods

### Part I: Monte Carlo Fundamentals

#### 1. Model-Free vs Model-Based Learning

(a) Compare Dynamic Programming and Monte Carlo methods by filling in the table:

Aspect	Dynamic Programming	Monte Carlo Methods
Model Requirements		
Learning Source		
Episode Requirements		
Convergence Guarantee		

#### Solution:

<b>Model Requirements</b>	Full transition model $p(s' s, a)$ required	No model required
<b>Learning Source</b>	Mathematical computation from model	Experience from episodes
<b>Episode Requirements</b>	No episodes needed	Must have episodic tasks
<b>Convergence Guarantee</b>	Exact solution with sufficient iterations	Converges to true values by Law of Large Numbers

(b) A robot is learning to navigate a maze. For each scenario, determine whether Dynamic Programming or Monte Carlo would be more appropriate:

1. The maze layout is fully known, including all wall positions and the goal location
2. The robot must learn by exploring an unknown maze
3. The maze has moving obstacles with unknown movement patterns
4. The maze is deterministic but the transition probabilities are too complex to model

**Solution:**

1. **Dynamic Programming** - Full knowledge of maze layout provides complete model
2. **Monte Carlo** - Unknown environment requires learning from experience
3. **Monte Carlo** - Moving obstacles make modeling impractical
4. **Monte Carlo** - Complex transitions make model-based approach difficult

**2. Episode Generation and Returns**

Consider a simple 3-state MDP where an agent can be in states  $\{A, B, C\}$  with  $C$  being the terminal state. An episode generates the following trajectory with  $\gamma = 0.9$ :

$$A \xrightarrow{r=2} B \xrightarrow{r=1} A \xrightarrow{r=3} C$$

- (a) Calculate the return  $G_0$  (from the first visit to state  $A$ ).

**Solution:**  $G_0 = r_1 + \gamma r_2 + \gamma^2 r_3 = 2 + 0.9(1) + 0.9^2(3) = 2 + 0.9 + 2.43 = 5.33$

- (b) Calculate the return  $G_2$  (from the second visit to state  $A$ ).

**Solution:**  $G_2 = r_3 = 3$

From the second visit to  $A$ , only the immediate reward to terminal state  $C$  contributes to the return.

- (c) If this is the only episode collected so far, what would be the First-Visit Monte Carlo estimate for  $V(A)$ ?

**Solution:** First-Visit MC only considers the first occurrence of state  $A$  in each episode.

$$V(A) = G_0 = 5.33$$

The second visit to  $A$  is ignored in First-Visit MC.

- (d) What would be the Every-Visit Monte Carlo estimate for  $V(A)$ ?

**Solution:** Every-Visit MC considers all occurrences of state  $A$ .

$$V(A) = \frac{G_0 + G_2}{2} = \frac{5.33 + 3}{2} = 4.165$$

**3. Blackjack Value Estimation**

A Monte Carlo agent is learning to play Blackjack using a simple policy: "Hit if hand value  $< 17$ , Stand otherwise." After collecting episodes, the agent has the following returns for the state "Hand = 16":

Episodes:  $G_1 = -1, G_2 = +1, G_3 = -1, G_4 = -1, G_5 = +1, G_6 = -1$

- (a) Calculate the Monte Carlo estimate  $V(\text{Hand} = 16)$  after all 6 episodes.

**Solution:**  $V(\text{Hand} = 16) = \frac{-1+1+(-1)+(-1)+1+(-1)}{6} = \frac{-2}{6} = -0.333$

- (b) If the true value is  $V^*(\text{Hand} = 16) = -0.4$ , what does this suggest about the current estimate?

**Solution:** The estimate (-0.333) is close to the true value (-0.4), suggesting that:

- The agent is learning the correct value function
- More episodes would likely improve the estimate
- The current policy may be reasonable for this state

The small difference indicates the estimate is converging toward the true value.

- (c) How would the estimate change if we collected 1000 more episodes with the same win/loss ratio?

**Solution:** If the win/loss ratio remains the same (2 wins out of 6 episodes = 1/3 win rate):

- Expected value would be approximately:  $\frac{1}{3}(+1) + \frac{2}{3}(-1) = \frac{1-2}{3} = -0.333$
- With 1000 more episodes, the estimate would remain close to -0.333
- However, the variance would decrease significantly due to Law of Large Numbers
- The estimate would become more reliable and stable

## Part II: Monte Carlo Prediction

### 4. First-Visit vs Every-Visit Monte Carlo

Consider the following episode with  $\gamma = 1$ :

$$S_1 \xrightarrow{r=3} S_2 \xrightarrow{r=1} S_1 \xrightarrow{r=2} S_3 \xrightarrow{r=0} \text{Terminal}$$

- (a) For state  $S_1$ , calculate the returns for each visit and determine the estimates using both First-Visit and Every-Visit Monte Carlo.

**Solution: Visits to  $S_1$ :**

- First visit (time 0):  $G_0 = 3 + 1 + 2 + 0 = 6$
- Second visit (time 2):  $G_2 = 2 + 0 = 2$

**First-Visit MC:**  $V(S_1) = G_0 = 6$

**Every-Visit MC:**  $V(S_1) = \frac{G_0 + G_2}{2} = \frac{6+2}{2} = 4$

- (b) Explain why First-Visit and Every-Visit give different estimates and discuss the trade-offs.

**Solution: Why different:**

- First-Visit uses only the first occurrence, treating each episode as one independent sample
- Every-Visit uses all occurrences, providing more data but with correlated samples

**Trade-offs:**

- **First-Visit:** Simpler analysis, proven convergence, independent samples
- **Every-Visit:** More data per episode, faster convergence in practice, but correlated samples
- **Bias:** Both are unbiased estimators of the true value function
- **Variance:** Every-Visit may have lower variance due to more samples

**5. Monte Carlo Algorithm Implementation**

- (a) Explain why the algorithm processes episodes backwards (from terminal state to initial state).

**Solution: Reasons for backward processing:**

- **Efficiency:** Returns are sums of future rewards. By going backwards, we can incrementally build the return using  $G = \gamma G + R_{t+1}$
- **Correctness:** When we reach time  $t$ , we already have the complete return  $G_t$  calculated
- **Memory:** We don't need to store all future rewards - just update the running sum
- **Implementation:** Much simpler than recalculating returns from scratch for each time step

**Part III: Action Values and Exploration****6. Why Action Values Are Essential**

- (a) Explain why state values  $V^\pi(s)$  are insufficient for policy improvement in model-free settings.

**Solution: The problem with state values alone:**

- Policy improvement in DP requires:  $\pi'(s) = \arg \max_a \sum_{s'} p(s'|s, a)[r + \gamma V(s')]$
- This requires transition probabilities  $p(s'|s, a)$  which are unknown in model-free settings
- State values tell us how good states are, but not which actions lead to the best states
- Without the model, we cannot evaluate action consequences

**Why action values solve this:**

- $Q(s, a)$  directly tells us the value of taking action  $a$  in state  $s$
- Policy improvement becomes:  $\pi'(s) = \arg \max_a Q(s, a)$  (no model needed)
- Action values encode both immediate rewards and future consequences

- (b) Consider a state with three actions where  $Q(s, a_1) = 5.2$ ,  $Q(s, a_2) = 3.8$ , and  $Q(s, a_3) = 4.1$ . What would a greedy policy select, and what potential problem does this create?

**Solution: Greedy policy selection:**  $\pi(s) = a_1$  (highest Q-value = 5.2)

**Potential problems:**

- **Exploration problem:** If we always select  $a_1$ , we never get new samples for  $a_2$  and  $a_3$

- **Poor estimates:**  $Q(s, a_2)$  and  $Q(s, a_3)$  may be based on very few samples and could be inaccurate
- **Missed opportunities:** The true optimal action might actually be  $a_2$  or  $a_3$ , but we'll never discover this with pure greedy selection
- **Stagnation:** Learning stops for non-selected actions

### 7. $\epsilon$ -Greedy Policy Analysis

- (a) For a state with 4 actions and  $\epsilon = 0.2$ , calculate the  $\epsilon$ -greedy policy probabilities if  $Q(s, a_1) = 7$ ,  $Q(s, a_2) = 4$ ,  $Q(s, a_3) = 5$ ,  $Q(s, a_4) = 2$ .

**Solution: Identify greedy action:**  $a^* = \arg \max_a Q(s, a) = a_1$  (Q-value = 7)

**Calculate probabilities:**

- $\pi(a_1|s) = 1 - \epsilon + \frac{\epsilon}{|A|} = 1 - 0.2 + \frac{0.2}{4} = 0.8 + 0.05 = 0.85$
- $\pi(a_2|s) = \frac{\epsilon}{|A|} = \frac{0.2}{4} = 0.05$
- $\pi(a_3|s) = \frac{\epsilon}{|A|} = \frac{0.2}{4} = 0.05$
- $\pi(a_4|s) = \frac{\epsilon}{|A|} = \frac{0.2}{4} = 0.05$

**Verification:**  $0.85 + 0.05 + 0.05 + 0.05 = 1.0$

- (b) How would the policy probabilities change if after more episodes, the Q-values become  $Q(s, a_1) = 7$ ,  $Q(s, a_2) = 7.5$ ,  $Q(s, a_3) = 5$ ,  $Q(s, a_4) = 2$ ?

**Solution: New greedy action:**  $a^* = a_2$  (Q-value = 7.5)

**New probabilities:**

- $\pi(a_1|s) = \frac{\epsilon}{|A|} = \frac{0.2}{4} = 0.05$
- $\pi(a_2|s) = 1 - \epsilon + \frac{\epsilon}{|A|} = 0.8 + 0.05 = 0.85$
- $\pi(a_3|s) = \frac{\epsilon}{|A|} = 0.05$
- $\pi(a_4|s) = \frac{\epsilon}{|A|} = 0.05$

**Key insight:** The policy automatically adapts as Q-values improve, shifting probability mass to the new best action while maintaining exploration.

## Part IV: Monte Carlo Control

### 8. On-Policy Monte Carlo Control

Consider a simple 2-state MDP with states  $\{A, B\}$  where state  $B$  is terminal. From state  $A$ , the agent can take actions  $\{left, right\}$ . After several episodes using  $\epsilon$ -greedy policy with  $\epsilon = 0.1$ , the current Q-values are:

$$Q(A, left) = 3.2, Q(A, right) = 2.8$$

- (a) Calculate the current  $\epsilon$ -greedy policy probabilities for state  $A$ .

**Solution: Greedy action:** *left* (Q-value = 3.2)

**Policy probabilities:**

- $\pi(\text{left}|A) = 1 - \epsilon + \frac{\epsilon}{|A|} = 1 - 0.1 + \frac{0.1}{2} = 0.9 + 0.05 = 0.95$
- $\pi(\text{right}|A) = \frac{\epsilon}{|A|} = \frac{0.1}{2} = 0.05$

- (b) If the next episode follows the trajectory  $A \xrightarrow{\text{left}, r=4} B$  and this is the 4th time we've visited state-action pair  $(A, \text{left})$  with previous returns  $\{3.0, 3.1, 3.5\}$ , what is the updated Q-value?

**Solution: New return:**  $G = 4$  (immediate reward to terminal state)

**All returns for  $(A, \text{left})$ :**  $\{3.0, 3.1, 3.5, 4.0\}$

**Updated Q-value:**  $Q(A, \text{left}) = \frac{3.0+3.1+3.5+4.0}{4} = \frac{13.6}{4} = 3.4$

- (c) How does the policy change after this update?

**Solution: Updated Q-values:**  $Q(A, \text{left}) = 3.4$ ,  $Q(A, \text{right}) = 2.8$

**Greedy action:** Still *left* (higher Q-value)

**New policy probabilities:**

- $\pi(\text{left}|A) = 0.95$  (unchanged)
- $\pi(\text{right}|A) = 0.05$  (unchanged)

**Result:** Policy remains the same since *left* is still the greedy action, but the Q-value estimate became more accurate.

## 9. Off-Policy Monte Carlo: Importance Sampling

- (a) Explain the key difference between on-policy and off-policy Monte Carlo methods.

**Solution: On-Policy Monte Carlo:**

- Uses the **same policy** for generating episodes and learning
- Policy being evaluated = Policy being followed
- Simpler but limited to learning about the current exploration policy
- Can only find optimal  $\epsilon$ -soft policies, not truly optimal deterministic policies

**Off-Policy Monte Carlo:**

- Uses **two different policies**:
  - **Behavior policy**  $b$ : generates episodes (usually exploratory)
  - **Target policy**  $\pi$ : the policy we want to learn about (can be deterministic)
- More complex (requires importance sampling) but more powerful
- Can learn optimal deterministic policies while maintaining exploration
- Requires coverage assumption:  $\pi(a|s) > 0 \Rightarrow b(a|s) > 0$

(b) Given a simple episode  $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2$  where:

- Behavior policy:  $b(A_0|S_0) = 0.4$ ,  $b(A_1|S_1) = 0.3$
- Target policy:  $\pi(A_0|S_0) = 0.8$ ,  $\pi(A_1|S_1) = 0.1$

Calculate the importance sampling ratio for this episode.

**Solution: Importance sampling ratio:**  $\rho = \prod_{t=0}^{T-1} \frac{\pi(A_t|S_t)}{b(A_t|S_t)} = \frac{\pi(A_0|S_0)}{b(A_0|S_0)} \times \frac{\pi(A_1|S_1)}{b(A_1|S_1)}$

$$\rho = \frac{0.8}{0.4} \times \frac{0.1}{0.3} = 2.0 \times 0.333 = 0.667$$

**Interpretation:** This episode is less likely under the target policy than the behavior policy, so it gets down-weighted in the learning update.

(c) Why is the coverage assumption ( $\pi(a|s) > 0 \Rightarrow b(a|s) > 0$ ) necessary for off-policy learning?

**Solution: Coverage assumption necessity:**

- If  $\pi(a|s) > 0$  but  $b(a|s) = 0$ , then action  $a$  in state  $s$  will **never be taken** during episode generation
- We cannot learn about the value of actions that are never experienced
- The importance sampling ratio  $\frac{\pi(a|s)}{b(a|s)}$  would be undefined (division by zero)
- Without coverage, our Q-value estimates for important actions could remain forever inaccurate

**Practical implication:** The behavior policy must be "exploratory enough" to visit all state-action pairs that the target policy might want to use.

## Part V: Comparative Analysis and Applications

### 10. Method Comparison

(a) Fill in the comparison table for different reinforcement learning methods:

Method	Dynamic Programming	Pro-	On-Policy MC	Off-Policy MC
Model Required				
Episode Requirement				
Policy Flexibility				
Exploration Handling				
Computational Complexity				

**Solution:**

<b>Model Required</b>	Yes, full model	No	No
<b>Episode Requirement</b>	No episodes	Complete episodes	Complete episodes
<b>Policy Flexibility</b>	Any policy	$\epsilon$ -soft only	Any target policy
<b>Exploration Handling</b>	N/A (no exploration needed)	Built into policy	Separate behavior policy
<b>Computational Complexity</b>	$O( S ^2 A )$ per iteration	$O(\text{episode length})$	$O(\text{episode length})$ + importance sampling

(b) For each scenario, recommend the most appropriate method and justify your choice:

1. Chess AI with complete game rules known
2. Robot learning to walk in unknown terrain
3. Stock trading bot learning from historical data while developing a new strategy
4. Game AI that must learn while playing against human opponents

**Solution:**

1. **Dynamic Programming** - Complete game rules provide full model; no need for sample-based learning; can compute exact optimal policy
2. **On-Policy Monte Carlo** - Unknown terrain requires experience-based learning; real-time learning during exploration is natural fit for on-policy methods
3. **Off-Policy Monte Carlo** - Historical data represents behavior policy; want to learn new strategy (target policy) without being constrained by historical trading patterns
4. **On-Policy Monte Carlo** - Must learn while playing; cannot separate behavior from target policy;  $\epsilon$ -greedy provides reasonable exploration against human opponents

## 11. Practical Implementation Considerations

(a) What are the main challenges in implementing Monte Carlo methods for real-world applications?

**Solution: Main challenges:**

- **Sample efficiency:** Requires many episodes to get good estimates, especially for rare states
- **Episodic requirement:** Many real-world problems are naturally continuing, not episodic
- **High variance:** Returns can have high variance, leading to slow convergence
- **Exploration vs exploitation:** Balancing learning and performance, especially in safety-critical applications
- **Long episodes:** Returns from long episodes may have high variance and delayed learning
- **Memory requirements:** Storing returns for all state-action pairs can be memory-intensive

- **Non-stationary environments:** If environment changes, historical data becomes less relevant

(b) Suggest three strategies to improve the practical performance of Monte Carlo methods.

**Solution: Improvement strategies:**

**1. Incremental Updates:**

- Use  $Q(s, a) \leftarrow Q(s, a) + \alpha[G - Q(s, a)]$  instead of storing all returns
- Reduces memory requirements and allows faster adaptation
- Learning rate  $\alpha$  can be adjusted for different convergence properties

**2. Function Approximation:**

- Use neural networks or other function approximators instead of tabular representation
- Enables learning in large/continuous state spaces
- Allows generalization between similar states

**3. Variance Reduction Techniques:**

- Baseline subtraction:  $G - V(s)$  instead of just  $G$
- Control variates to reduce return variance
- Importance sampling with control variates in off-policy methods