Outline

- Introduction
- MC Policy Evaluation
  - Blackjack example
- MC Estimation of Action-Value function
- MC Control
- Incremental Implementation
Introduction

Monte Carlo methods solve the RL problem **without requiring a model of the environment**
- Often a model is not available
- Storage requirements are substantially relaxed

Require only experience - sample sequences of states, actions and rewards

Experience can come in two general forms:
- **Online interaction** with an environment - no prior knowledge needed to yield optimal behavior
- **Simulated interactions** - require model but rely on samples
  - Many times it is easy to generate samples but not obtain the probability distribution of the state transitions

Monte Carlo methods are ways of solving the RL problem based on **averaging sample returns**
Introduction (cont.)

We consider mainly episodic (terminal) tasks

- Experience is divided into episodes that terminate regardless of the actions taken
- MC methods are incremental in an episode-by-episode manner
- Different than the step-by-step scheme employed by DP

Otherwise, same as DP - policy evaluation, policy improvement and generalized policy iteration

We start with policy evaluation - the computation of a value function corresponding to a given policy
Monte Carlo Policy Evaluation

- We begin with learning the state-value function for a given policy

An obvious way to estimate it from experience: average all returns observed after visiting a given state
  - Main idea behind MC methods

Each occurrence of state \( s \) in an episode is called a visit to \( s \)

First-visit MC method averages just the returns following first visits to \( s \)

Every-visit MC method averages the returns following all the visits to \( s \) in a set of episodes

---

Initialize:
\[
\begin{align*}
\pi & \leftarrow \text{policy to be evaluated} \\
V & \leftarrow \text{an arbitrary state-value function} \\
Returns(s) & \leftarrow \text{an empty list, for all } s \in S
\end{align*}
\]

Repeat forever:
(a) Generate an episode using \( \pi \)
(b) For each state \( s \) appearing in the episode:
   \[
   \begin{align*}
   R & \leftarrow \text{return following the first occurrence of } s \\
   \text{Append } R \text{ to } Returns(s) \\
   V(s) & \leftarrow \text{average}(Returns(s))
   \end{align*}
   \]
Blackjack Example

The object is to obtain cards the sum of which is \( \leq 21 \)
- All face cards count as 10
- Ace can count either as 1 or as 11
- Player competes independently against the dealer

Begins where two cards are dealt to both dealer and agent

Outline of rules:
- If the player has 21 immediately (e.g. an ace and a 10-card) he wins, unless the dealer has the same, in which case the game is a draw.
- Otherwise, player can request additional cards, one by one until he either stops or exceeds 21 (loses).
- If he stops, it becomes the dealer's turn. The dealer's strategy is simple and fixed: stops on sum of 17 or greater, and continues otherwise.
- If the dealer exceeds 21, then the player wins; otherwise, the outcome--win, lose, or draw--is determined by whose final sum is closer to 21.
Blackjack Example (cont.)

- Playing blackjack is naturally formulated as an episodic finite MDP
- Each game of blackjack is an episode
  - Rewards of +1, -1, and 0 are given for winning, losing, and drawing, respectively
- We do not discount rewards (i.e. $\gamma=1$), and since all intermediate rewards are zero - the terminal reward == the return
- Action set = {hit (get another card), stick (stop)}
- States depend on the player’s card and the visible card of the dealer
- We assume cards are dealt from an infinite deck
  - No advantage to keeping track of cards dealt
- If the player gets an ace that is usable, i.e. $\leq 21$, he uses it
- Decisions are based on three variable: current sum (12-21), the dealer’s displayed card (ace-10) and whether the player has a usable ace (total of 200 states)
Blackjack Example (cont.)

- Consider a policy that stops if the player's sum is 20 or 21
  - Assume many blackjack games are played
  - We're looking for the value function that fits this policy
- Note that here the same state is never revisited (why?)
Blackjack Example (cont.)

Although we have complete knowledge of the environment, it would not be easy to apply DP policy evaluation.

Very hard to calculate $P^a_{ss'}, R^a_{ss'}$:
- e.g. player’s sum is 14 - what is the expected reward as a function of the dealer’s displayed card?

Since all of these calculations must be done prior to running DP - it is often an impractical approach.

In contrast, generating sample games (for MC) is much easier to do.

Surprising insight: even if the environment’s dynamics are known, MC is often a more efficient method to apply:
- Estimating values are independent (no “bootstrapping”)
- Optimal policy trajectory corresponds to a small state subset.
Can the backup diagram be applied to MC methods?
- Recall that backup diagrams show top node to be updated and below all the transitions and leaf nodes that contribute to the update.

For MC, the root is a state node and below are all the nodes visited until terminal node is reached.
- Shows only transitions sampled on the one episode.

DP focuses on one-step transitions, whereas MC goes all the way to the end of the episode.

Updating values for states is independent in MC.
- Computational complexity of updating one node is independent of $|S|$.
- An attractive feature, since one can estimate only a subset of the node values (not have to do all).
Monte Carlo estimation of action values

- If a model is not available, then it is particularly useful to estimate action values rather than state values
  - Value function alone is not enough to establish policy
- We’ll need to estimate $Q^\pi(s,a)$
- The same MC approach will be employed: the agent records the rewards received after taking action $a$ at state $s$
  - However, here we must exercise exploration, otherwise some actions will never be evaluated
- One way to achieve this: exploring starts - each state-action pair at the beginning of an episode has non-zero probability
  - We’ll consider the general stochastic approach later
Monte Carlo Control

Next, we’ll outline how MC methods can be used in control - i.e. to approximate an optimal policy

- The idea is to follow generalized policy iteration (GPI)
- Approximated policy and value functions are maintained

Let’s consider the basic policy iteration method for MC

- Policy evaluation is achieved by recording the outcomes of many episodes
- Policy improvement is done by selecting greedy actions, i.e. \( \pi(s) = \text{argmax}_a Q(s,a) \)
Monte Carlo Control (cont.)

- Policy improvement is achieved by constructing $\pi_{k+1}$ as the greedy policy with respect to $Q^{\pi_k}$.
- The policy improvement theorem holds since for all $s$:

$$Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, \arg \max_a Q^{\pi_k}(s, a))$$

$$= \max_a Q^{\pi_k}(s, a)$$

$$\geq Q^{\pi_k}(s, \pi_k(s))$$

$$= V^{\pi_k}(s).$$

- If the two policies are equal, they are both optimal:
  - This way MC can lead to optimal policies with no model.
  - This assumes exploring starts and infinite number of episodes for MC policy evaluation.
- To remove the latter:
  - Update only to a given level of performance
  - Alternate between evaluation and improvement per episode.
Monte Carlo ES (Exploring Starts)

Initialize, for all \( s \in S, a \in A(s) \):
\[
Q(s, a) \leftarrow \text{arbitrary}
\]
\[
\pi(s) \leftarrow \text{arbitrary}
\]
\[
\text{Returns}(s, a) \leftarrow \text{empty list}
\]

Repeat forever:
(a) Generate an episode using exploring starts and \( \pi \)
(b) For each pair \( s, a \) appearing in the episode:
\[
R \leftarrow \text{return following the first occurrence of } s, a
\]
Append \( R \) to \( \text{Returns}(s, a) \)
\[
Q(s, a) \leftarrow \text{average}(\text{Returns}(s, a))
\]
(c) For each \( s \) in the episode:
\[
\pi(s) \leftarrow \arg \max_a Q(s, a)
\]
On-Policy MC Control

- How can we avoid the unlikely assumption of exploring starts?
- Can't simply improve the policy by following a greedy policy, since no exploration will take place
- We employ an $\varepsilon$-greedy policy instead
  - With “high” probability we choose the greedy action
  - Otherwise - explore uniformly

**Solution**: generate soft policies, i.e. $\pi(s,a)>0$ for all $s$ and $a$

- e.g. $\varepsilon$-soft policy - moves policy toward greedy

\[
\begin{align*}
\varepsilon & \quad \text{for greedy (max)} \\
\frac{1 - \varepsilon}{|A(s)|} & \quad \text{for non-max} \\
\frac{\varepsilon}{|A(s)|} & 
\end{align*}
\]
On-Policy vs. Off-Policy methods

Two methods for ensuring that the agent selects all actions infinitely often (as number of episodes goes to infinity)

On-Policy Methods
- Evaluate/improve the policy that is used to make decisions
- Policy is generally soft (e.g. $\varepsilon$-soft policy)

Off-Policy Methods
- Value estimation and policy evaluation are separated
- Two policies are considered:
  - Behavior policy – the policy used to generate the actions
  - Estimation policy – the policy evaluated and improved
- As we will see, this has some key advantages...
Let $\pi'$ denote the $\varepsilon$-greedy policy

- We want to show that its value function is higher than $\pi$'s

The condition of the policy improvement theorem to hold for all states is ...

$$Q^\pi(s, \pi'(s)) = \sum_a \pi'(s, a)Q^\pi(s, a)$$

$$= \frac{\varepsilon}{|A(s)|} \sum_a Q^\pi(s, a) + (1 - \varepsilon) \max_a Q^\pi(s, a)$$

$$\geq \frac{\varepsilon}{|A(s)|} \sum_a Q^\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(s, a) - |A(s)|}{1 - \varepsilon} Q^\pi(s, a)$$

$$= \frac{\varepsilon}{|A(s)|} \sum_a Q^\pi(s, a) - \frac{\varepsilon}{|A(s)|} \sum_a Q^\pi(s, a) + \sum_a \pi(s, a)Q^\pi(s, a)$$

$$= V^\pi(s).$$
On-Policy MC Control (cont.)

Initialize, for all \( s \in \mathcal{S}, a \in \mathcal{A}(s) \):
\[
Q(s, a) \leftarrow \text{arbitrary} \\
Returns(s, a) \leftarrow \text{empty list} \\
\pi \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}
\]

Repeat forever:
(a) Generate an episode using \( \pi \)
(b) For each pair \( s, a \) appearing in the episode:
\[
R \leftarrow \text{return following the first occurrence of } s, a \\
\text{Append } R \text{ to } Returns(s, a) \\
Q(s, a) \leftarrow \text{average}(Returns(s, a))
\]
(c) For each \( s \) in the episode:
\[
a^* \leftarrow \text{arg max}_a Q(s, a) \\
\text{For all } a \in \mathcal{A}(s):
\]
\[
\pi(s, a) \leftarrow \begin{cases} 
1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\
\varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^*
\end{cases}
\]
Evaluating one policy while following another (cont.)

Suppose we wanted to evaluate a policy by viewing episodes generated from a different policy.

We require that $\pi(s,a) > 0$ implies $\pi'(s,a) > 0$.

Assumptions and definitions:

- We view episodes generated under policy $\pi'$.
- Consider the $i^{th}$ first visit to state $s$ and the complete sequence of states and actions following that visit.
- Let $p_i(s)$ and $p'_i(s)$ denote the probabilities of that complete sequence occurring under policies $\pi$ and $\pi'$, respectively.
- Let $R_i(s)$ denote the $i^{th}$ observed return from state $s$.

To average these returns to obtain an unbiased estimate of $V^\pi(s)$, we need only weight each return by its relative probability of occurring under each policy.
The desired MC estimate after observing \( n_s \) returns from state \( s \) is given by

\[
V_\pi(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)}}.
\]

- The above assumes knowledge of \( p_i(s) \) and \( p'_i(s) \), which are usually unknown.
- However, their ratio can be determined without a model since

\[
p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}
\]

from which we have

\[
\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}.
\]
Recall that in on-policy methods, the value of a policy is estimated while it is used for control.

In off-policy methods - the two functions are separated:
- **Behavior** policy - policy used to generate behavior
- **Estimation** policy - policy evaluated and improved

Off-policy MC methods employ the technique shown in the previous couple of slides:
- They follow the behavior policy while learning about and improving the estimation policy
- Requires the behavior policy to be soft (e.g. $\varepsilon$-soft)
- Advantage: estimation policy can be deterministic (e.g. greedy)

We will next look at an off-policy method for computing $Q^*$:
- The estimation policy is greedy with respect to $Q$
- The behavior policy is $\varepsilon$-soft
Off-Policy Monte Carlo Control (cont.)

Initialize, for all \( s \in S, a \in A(s) \):
- \( Q(s, a) \leftarrow \text{arbitrary} \)
- \( N(s, a) \leftarrow 0 \); Numerator and
- \( D(s, a) \leftarrow 0 \); Denominator of \( Q(s, a) \)
- \( \pi \leftarrow \text{an arbitrary deterministic policy} \)

Repeat forever:
- (a) Select a policy \( \pi' \) and use it to generate an episode:
  \( s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T \)
- (b) \( \tau \leftarrow \text{latest time at which } a_\tau \neq \pi(s_\tau) \)
- (c) For each pair \( s, a \) appearing in the episode after \( \tau \):
  - \( t \leftarrow \text{the time of first occurrence (after } \tau \text{) of } s, a \)
  - \( w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)} \)
  - \( N(s, a) \leftarrow N(s, a) + wR_t \)
  - \( D(s, a) \leftarrow D(s, a) + w \)
  - \( Q(s, a) \leftarrow \frac{N(s, a)}{D(s, a)} \)
- (d) For each \( s \in S \):
  - \( \pi(s) \leftarrow \arg \max_a Q(s, a) \)
Summary

- **MC** has several advantages over DP:
  - Can learn directly from interaction with environment
  - No need for full models
  - No need to learn about ALL states
  - Less harm by Markovian violations (more detail later)

- **MC methods provide an alternate policy evaluation process**

- **One issue to watch for: maintaining sufficient exploration**
  - exploring starts, soft policies

- Introduced distinction between **on-policy** and **off-policy** methods

- No bootstrapping (as opposed to DP)