Reinforcement Learning
04 - Monte Carlo

Elena, Xi
Previous lecture
Markov Decision Processes

Markov decision processes formally describe an environment for reinforcement learning where the environment is fully observable.

A finite MDP is defined by a tuple \( \langle \mathcal{S}, \mathcal{A}, p(\cdot), \mathcal{R} \rangle \)

- \( \mathcal{S} \) is a finite set of possible states
- \( \mathcal{A}(S_t) \) is a finite set of actions in state \( S_t \)
- \( p(s' \mid s, a) \) is a state transition probability matrix, \( p(s' \mid s, a) = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a] \)
- \( \mathcal{R} \) is a final set of all possible rewards
Planning by Dynamic Programming

Dynamic programming assumes that we know the MDP for our problem. It is used for planning in an MDP.

For **prediction**:
- Input: MDP \((S, A, P, R)\) and policy \(\pi\)
- Output: value function \(v_{\downarrow}\pi\)

For **control**:
- Input: MDP \((S, A, P, R)\)
- Output: optimal policy \(\pi_{\downarrow*}\)
  - (optimal value function \(v_{\downarrow*}\))
# Dynamic Programming Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iterative Policy Evaluation</th>
<th>Policy Iteration</th>
<th>Value Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bellman Equation</td>
<td>Bellman Expectation Equation</td>
<td>Bellman Expectation Equation Policy Iteration + Greedy Policy Improvement</td>
<td>Bellman Optimality Equation</td>
</tr>
<tr>
<td>Problem</td>
<td>Prediction</td>
<td>Control</td>
<td>Control</td>
</tr>
</tbody>
</table>

**Iterative Policy Evaluation**
- Calculates the value function $v_{k+1}(s) \leftarrow s$
- For each state $s$, updates the value of the state $v_k(s') \leftarrow s'$

**Policy Iteration**
- Starts with an initial policy $\pi$ and a value function $V^\pi$
- Iterates between policy evaluation and policy improvement
  - Policy Evaluation: $V' = V^\pi$
  - Policy Improvement: $\pi' = \text{greedy}(V')$

**Value Iteration**
- Computes the optimal value function $v_{k+1}(s) \leftarrow s$
- For each state $s$, updates the value of the state $v_k(s') \leftarrow s'$

**Bellman Equation**
- The Bellman equation is used in all algorithms to update the value function.

**Problem**
- Prediction
- Control
This lecture
Like previous
but with blackjack
Model-Free Reinforcement Learning

Previous lecture:
  Planning by dynamic programming
  Solve a known MDP

This lecture:
  Model-free prediction
  Estimate the value function of an unknown MDP using Monte Carlo
  Model-free control
  Optimise the value function of an unknown MDP using Monte Carlo
Monte Carlo Method Introduction

MC method - any method which solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties.

\[ E[X] = \frac{1}{n} \sum_{i=1}^{n} x_i \]

Modern version of MC was named by Stanislaw Ulam in 1946 in honor of his uncle who often borrowed money from relatives to gamble in Monte Carlo Casino (Monaco). S. Ulam came up with this idea while recovering from surgery and playing solitaire. He tried to estimate the probability of winning given the initial state.
Monte Carlo method applied to approximating the value of $\pi$. After placing 30,000 random points, the estimate for $\pi$ is within 0.07% of the actual value.
Monte Carlo Reinforcement Learning

MC methods learn directly from episodes of experience
MC is model-free: no knowledge of MDP transitions / rewards
MC learns from complete episodes: no bootstrapping
MC uses the simplest possible idea: value = mean return
Caveat: can only apply MC to episodic MDPs
    All episodes must terminate
Monte Carlo method introduction

Monte Carlo Prediction

Monte Carlo Control
Monte Carlo method introduction

Monte Carlo Prediction

Monte Carlo Control
Monte Carlo Policy Evaluation

Goal: learn $v_\pi(s)$ from episodes of experience under policy $\pi$

$S_1, A_1, R_2, \ldots, S_k \sim \pi$

Recall that the return is the total discounted reward:

$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$

Recall that the value function is the expected return:

$v_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s]$

MC policy evaluation uses empirical mean return instead of expected return

First-visit MC: average returns only for first time $s$ is visited in an episode

Every-Visit MC: average returns for every time $s$ is visited in an episode

Both converge asymptotically
First-visit Monte Carlo policy evaluation

First-visit MC policy evaluation (returns $V \approx v_\pi$)

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

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

By the law of large numbers, $V(s) \rightarrow v_\pi(s)$ as number of episodes $\rightarrow \infty$
MC policy evaluation EXAMPLE

undiscounted Markov Reward Process
two states A and B
transition matrix and reward function are unknown
observed two sample episodes

\[ A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate} \]

\[ B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate} \]

A+3 → A indicates a transition from state A to state A, with a reward of +3

Using **first-visit**, state-value functions \( V(A), V(B) \) - ?
Using **every-visit**, state-value functions \( V(A), V(B) \) - ?
MC policy evaluation EXAMPLE Solution

\[ A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate} \]
\[ B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate} \]

**first-visit**
\[ V(A) = \frac{1}{2}(2 + 0) = 1 \]
\[ V(B) = \frac{1}{2}(-3 + -2) = -\frac{5}{2} \]

**every-visit**
\[ V(A) = \frac{1}{4}(2 + -1 + 1 + 0) = \frac{1}{2} \]
\[ V(B) = \frac{1}{4}(-3 + -3 + -2 + -3) = -\frac{11}{4} \]
Blackjack Example

States (200 of them):
  - Current sum (12-21)
  - Dealer’s showing card (ace-10)
  - Do I have a “useable” ace? (yes-no)

Action **stick**: Stop receiving cards (and terminate)
Action **hit**: Take another card (no replacement)

Reward for **stick**: +1 if sum of cards > sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards < sum of dealer cards

Reward for **hit**: -1 if sum of cards > 21 (and terminate)
  - 0 otherwise

Transitions: automatically **hit** if sum of cards < 12
Blackjack Value Function after Monte Carlo Learning

Policy: **stick** if sum of cards ≥ 20, otherwise **hit**
Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally

\[
\mu_k = \frac{1}{k} \sum_{j=1}^{k} x_j
\]

\[
= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right)
\]

\[
= \frac{1}{k} (x_k + (k - 1)\mu_{k-1})
\]

\[
= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})
\]
Incremental Monte Carlo Updates

Update $V(s)$ incrementally after episode $S_1, A_1, R_2, ..., S_T$

For each state $S_t$ with return $G_t$

\[
N(S_t) \leftarrow N(S_t) + 1
\]

\[
V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))
\]

In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

\[
V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))
\]
Monte Carlo Backup

\[ V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t)) \]
Dynamic Programming

$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$
Backup diagram for Monte Carlo

Entire episode included

Only one choice at each state (unlike DP)

MC does not bootstrap (update estimates on the basis of other estimates)

Estimates for each state are independent

Time required to estimate one state does not depend on the total number of states
Monte Carlo method introduction

Monte Carlo Prediction

Monte Carlo Control
Monte Carlo method introduction
Monte Carlo Prediction
Monte Carlo Control
Generalised Policy Iteration (Refresher)

Policy evaluation - Estimate $v\downarrow \pi$ e.g.
Iterative policy evaluation
Policy improvement - Generate $\pi' \geq \pi$
e.g. Greedy policy improvement
Generalised Policy Iteration With Monte Carlo Evaluation

Policy evaluation - Monte-Carlo policy evaluation, $V = \nu \downarrow \pi$?
Policy improvement - Greedy policy improvement?
Model-Free Policy Iteration Using Action-Value Function

Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \arg\max_{a \in A} \sum s' \sum r \sum ps' r, s, a [r + \gamma v \downarrow \pi(s')]$$

Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \arg\max_{a \in A} Q(s, a)$$
Generalised Policy Iteration with Action-Value Function

Policy evaluation - Monte Carlo policy evaluation, $Q = q \downarrow \pi$
Policy improvement - Greedy policy improvement?
Example of Greedy Action Selection

There are two doors in front of you.

You open the left door and get reward 0
\[ V(\text{left}) = 0 \]
You open the right door and get reward +1
\[ V(\text{right}) = +1 \]
You open the right door and get reward +3
\[ V(\text{right}) = +2 \]
You open the right door and get reward +2
\[ V(\text{right}) = +2 \]

\[ \ldots \]

Are you sure you’ve chosen the best door?
**ε-Greedy Policy Exploration**

Simplest idea for ensuring continual exploration all $m$ actions are tried with non-zero probability with probability $1 - \varepsilon$ choose the greedy action with probability $\varepsilon$ choose an action at random

$$\pi_{as} = \begin{cases} \varepsilon/m + 1 - \varepsilon, & \text{if } a^* = \underset{a \in A}{\arg \max} Q(s,a) \\ \varepsilon/m, & \text{otherwise} \end{cases}$$
\( \epsilon \)-Greedy Policy Improvement

**Theorem**

For any \( \epsilon \)-greedy policy \( \pi \), the \( \epsilon \)-greedy policy \( \pi' \) with respect to \( q_\pi \) is an improvement, \( \nu_{\pi'}(s) \geq \nu_\pi(s) \)

\[
q_\pi(s, \pi'(s)) = \sum_{a \in A} \pi'(a|s)q_\pi(s, a) \\
= \epsilon/m \sum_{a \in A} q_\pi(s, a) + (1 - \epsilon) \max_{a \in A} q_\pi(s, a) \\
\geq \epsilon/m \sum_{a \in A} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in A} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\
= \sum_{a \in A} \pi(a|s)q_\pi(s, a) = \nu_\pi(s)
\]

Therefore from policy improvement theorem, \( \nu_{\pi'}(s) \geq \nu_\pi(s) \)
Monte Carlo Policy Iteration

Policy evaluation - Monte Carlo policy evaluation, $Q = q_\pi$
Policy improvement - $\epsilon$-greedy policy improvement
Monte Carlo Control

Every episode:
Policy evaluation - Monte Carlo policy evaluation, $Q \approx q_{\pi}$
Policy improvement - $\varepsilon$-greedy policy improvement
Monte Carlo Control in Blackjack

\[ \pi_* \]

- Usable ace
  - STICK
  - HIT

\[ \nu_* \]

- No usable ace
  - STICK
  - HIT

Player sum

Dealer showing
On-policy vs Off-policy
On-policy vs Off-policy

There are two ideas to take away the Exploring Starts assumption:

- **On-policy methods:**
  Learning while doing the job
  Learning policy $\pi$ from the episodes that generated using $\pi$

- **Off-policy methods:**
  Learning while watching other people doing the job
  Learning policy $\pi$ from the episodes generated using another policy $\mu$
On-policy

In On-policy control methods the policy is generally "soft", meaning that:

\[ \pi(a|s) > 0 \quad \forall s \in S, a \in A(s) \]

\(\varepsilon\)-Greedy Policy Improvement:
All policies have a probability to be chosen, but gradually the selected policy is closer and closer to a deterministic optimal policy by controlling the \(\varepsilon\) value.
Other ways of soft policies improvement

- Uniformly random policy: $\pi(s,a) = 1/|A(s)|$

- $\epsilon$-soft policy: $\pi(s,a) \geq \epsilon/|A(s)|$

- $\epsilon$-greedy policy: $\pi(s,a) = \epsilon/|A(s)|$ , and $\pi(s,a) = 1-\epsilon+\epsilon/|A(s)|$ for the greedy action
Off-policy

*Learning policy* \( \pi \) *by following the data generated using policy* \( \mu \)

Why is it important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies
- Learn about optimal policy while following exploratory policy

We call:
- \( \pi \) the **target policy**: the policy being learned about
- \( \mu \) the **behavior policy**: the policy generates the moves
Off-policy

However we need $\mu$ to satisfy a condition:

$$\pi(a, s) > 0 \quad \rightarrow \quad \mu(a, s) > 0$$

Every action which is taken under policy $\pi$ must have a non-zero probability to be taken as well under policy $\mu$. We call this the assumption of coverage.

Typically the target policy $\pi$ would be a greedy policy with respect to the current action-value function.
Off-policy: Importance Sampling

The tool we use for estimation is called importance sampling. It is a general technique for estimating expected values of one distribution given samples from another.

\[
\{S_1, A_1, R_2, ..., S_T\} \sim \mu
\]

\[
\prod_{k=t}^{T-1} \pi A \downarrow k \ S \downarrow k \ p(S \downarrow k+1 \mid S \downarrow k, A \downarrow k)
\]

Where \( p(S \downarrow k+1 \mid S \downarrow k, A \downarrow k) \) is the state-transition probability.
Off-policy: Importance Sampling

The relative probability of the trajectory under the target and behavior policies, or the **importance sampling ratio**, is:

\[
p_
abla_f(T) = \prod_{k=t+1}^{t+T-1} \pi(a|s) p(s'|s,a) / \prod_{k=t+1}^{t+T-1} \mu(a|s) p(s'|s,a)
\]

The state-transition probability depend on the MDP, which are generally unknown, cancel each other out.
Off-policy: Importance Sampling

**Ordinary importance sampling**: scale the returns by the ratios and average the results.

\[
V(s) = \frac{\sum_{t \in \mathcal{I}(s)} \rho_t^{T(t)} G_t}{|\mathcal{I}(s)|}.
\]

- \(\mathcal{I}(s)\): Episodes follow behavior policy
- \(\rho_{t\uparrow T(t)}\): Importance sampling ratio
- \(G_t\): Episode reward

**Weighted importance sampling**: scale the returns use weighted average.

\[
V(s) = \frac{\sum_{t \in \mathcal{I}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathcal{I}(s)} \rho_t^{T(t)}}
\]
Off-policy: Importance Sampling

**Ordinary importance sampling:** scale the returns by the ratios and average the results.

\[
V(s) \doteq \sum_{t \in \mathcal{T}(s)} \frac{\rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.
\]

**Weighted importance sampling:** scale the returns use weighted average.

\[
V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)}}.
\]

\[
V(s) \doteq \rho_t^{T(t)} G_t
\]

\[
V(s) \doteq G_t
\]
Off-policy: Importance Sampling

In practice the weighted estimator has dramatically lower variance and is therefore strongly preferred. Example of a blackjack state.
Pros and cons of MC

MC has several advantages over DP:

- Can learn V and Q directly from interaction with environment (using episodes!)
- No need for full models (using episodes!)
- No need to learn about ALL states (using episodes!)

However, there are some limitations:

- MC only works for episodic (terminating) environments
- MC learns from complete episodes, so no bootstrapping
- MC must wait until the end of an episode before return is known

Solution: Temporal-Difference

- TD works in continuing (non-terminating) environments
- TD can learn online after every step
- TD can learn from incomplete sequences

Next lecture
Assignment: Blackjack

Play Blackjack using Monte Carlo with exploring starts.
- Implement the part for updating $Q(s,a)$ value inside the function `monte_carlo_es(n_iter)`.
- Try different methods to select the start state and action. (in the code it is totally random)
- Play with different reward and iteration number

You should get the similar result to the example in the book.
Assignment: Blackjack

Modify the code and implement “Monte Carlo without exploring starts” using on-policy learning with $\epsilon$-greedy policies.

What is the difference between these two methods?
References


Online lectures:

D. Silver. Reinforcement Learning Course, Lecture 4-5, 2015 [YouTube video] Retrieved from https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PL7-jPKtc4r78-wCZcQn5lqyuWhBZ8fOxT