Lecture 04: Monte Carlo Methods

Oliver Wallscheid
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General Idea and Differences to Dynamic Programming</td>
</tr>
<tr>
<td>2</td>
<td>Basic Monte Carlo Prediction</td>
</tr>
<tr>
<td>3</td>
<td>Basic Monte Carlo Control</td>
</tr>
<tr>
<td>4</td>
<td>Extensions to Monte Carlo On-Policy Control</td>
</tr>
<tr>
<td>5</td>
<td>Monte Carlo Off-Policy Prediction</td>
</tr>
<tr>
<td>6</td>
<td>Monte Carlo Off-Policy Control</td>
</tr>
</tbody>
</table>
Monte Carlo Methods vs. Dynamic Programming

Dynamic Programming:
- Model-based prediction and control
- Planning inside known MDPs

Monte Carlo methods:
- Model-free prediction and control
- Estimating value functions and optimize policies in unknown MDPs
- But: still assuming finite MDP problems (or problems close to that)
- In general: broad class of computational algorithms relying on repeated random sampling to obtain numerical results
General Monte Carlo (MC) Methods’ Characteristics

- Learning from experience, i.e., sequences of samples $\langle x_k, u_k, r_{k+1} \rangle$
- Main concept: Estimation by averaging sample returns
- To guarantee well-defined returns: limited to episodic tasks
- Consequence: Estimation and policy updates only possible in an episode-by-episode way compared to step-by-step (online)

Fig. 4.1: Monte Carlo port
(source: www.flickr.com, by Miguel Mendez CC BY 2.0)
Table of Contents

1. General Idea and Differences to Dynamic Programming

2. Basic Monte Carlo Prediction

3. Basic Monte Carlo Control

4. Extensions to Monte Carlo On-Policy Control

5. Monte Carlo Off-Policy Prediction

6. Monte Carlo Off-Policy Control
Task Description and Basic Solution

MC prediction problem statement

- Estimate state value $v_{\pi}(x)$ for a given policy $\pi$.
- Available are samples $\langle x_{k,j}, u_{k,j}, r_{k+1,j} \rangle$ for episodes $j = 1, \ldots, J$.

MC solution approach:

- Average returns after visiting state $x_k$ over episodes $j = 1, \ldots$

$$v_{\pi}(x_k) \approx \hat{v}_{\pi}(x_k) = \frac{1}{J} \sum_{j=1}^{J} g_{k,j} = \frac{1}{J} \sum_{j=1}^{J} \sum_{i=0}^{T_j} \gamma^i r_{k+i+1,j}.$$  \hspace{1cm} (4.1)

- Above, $T_j$ denotes the terminating time step of each episode $j$.
- First-visit MC: Apply (4.1) only to the first state visit per episode.
- Every-visit MC: Apply (4.1) each time visiting a certain state per episode (if a state is visited more than one time per episode).
input: a policy $\pi$ to be evaluated
output: estimate of $v^\pi_X$ (i.e., value estimate for all states $x \in X$)
init: $\hat{v}(x) \forall x \in X$ arbitrary except $v_0(x) = 0$ if $x$ is terminal
        $l(x) \leftarrow$ an empty list for every $x \in X$
for $j = 1, \ldots, J$ episodes do
    Generate an episode following $\pi$: $x_0, u_0, r_1, \ldots, x_{T_j}, u_{T_j}, r_{T_j+1}$;
    Set $g \leftarrow 0$;
    for $k = T_j - 1, T_j - 2, T_j - 3, \ldots, 0$ time steps do
        $g \leftarrow \gamma g + r_{k+1}$;
        if $x_k \notin \langle x_0, x_1, \ldots, x_{k-1} \rangle$ then
            Append $g$ to list $l(x_k)$;
            $\hat{v}(x_k) \leftarrow \text{average}(l(x_k))$;
    end
end

Algo. 4.1: First-visit MC state-value prediction
Incremental Implementation

- Algo. 4.1 is inefficient due to large memory demand.
- Better: use incremental / recursive implementation.
- The sample mean $\mu_1, \mu_2, \ldots$ of an arbitrary sequence $g_1, g_2, \ldots$ is:

$$\mu_j = \frac{1}{j} \sum_{i=1}^{j} g_i,$$

$$= \frac{1}{j} \left[ g_j + \sum_{i=1}^{j-1} g_i \right],$$

$$= \frac{1}{j} [g_j + (j - 1)\mu_{j-1}],$$

$$= \mu_{j-1} + \frac{1}{j} [g_j - \mu_{j-1}].$$  \hfill (4.2)

- If a given decision problem is non-stationary, using a forgetting factor $\alpha \in \{\mathbb{R}|0 < \alpha < 1\}$ allows for dynamic adaption:

$$\mu_j = \mu_{j-1} + \alpha [g_j - \mu_{j-1}].$$  \hfill (4.3)
Statistical Properties of MC-Based Prediction (1)

First-time visit MC:

- Each return sample $g_j$ is independent from the others since they were drawn from separate episodes.
- One receives i.i.d. data to estimate $\mathbb{E} [\hat{v}_\pi]$ and consequently this is bias-free.
- The estimate’s variance $\text{Var} [\hat{v}_\pi]$ drops with $1/n$ ($n$: available samples).

Every-time visit MC:

- Each return sample $g_j$ is not independent from the others since they might be obtained from same episodes.
- One receives non-i.i.d. data to estimate $\mathbb{E} [\hat{v}_\pi]$ and consequently this is biased for any $n < \infty$.
- Only in the limit $n \to \infty$ one receives $(v_\pi(x) - \mathbb{E} [\hat{v}_\pi(x)]) \to 0$.

Statistical Properties of MC-Based Prediction (2)

- State-value estimates for each state are independent.
- One estimate does not rely on the estimate of other states (no bootstrapping compared to DP).
- Makes MC particularly attractive when one requires state-value knowledge of only one or few states.
  - Hence, generating episodes starting from the state of interest.

Fig. 4.2: Back-up diagrams for DP (left) and MC (right) prediction: shallow one-step back-ups with bootstrapping vs. deep back-ups over full episodes
Let’s reuse the forest tree MDP example with *fifty-fifty policy* and discount factor $\gamma = 0.8$ plus disaster probability $\alpha = 0.2$:

![Forest MDP with fifty-fifty-policy including state values](image)

*Fig. 4.3: Forest MDP with fifty-fifty-policy including state values*
Fig. 4.4: State-value estimate of forest tree MDP initial state using MC-based prediction over the number of episodes being evaluated (mean and standard deviation are calculated based on 2000 independent runs).
MC-Based Prediction Example: Blackjack (1)

- **States** (200 in total)
  - Player’s sum: 12-21 (automatically twist if player’s sum < 12)
  - Usable ace (counts 1/11): yes/no
  - Dealer’s initial card: ace-10

- **Action** **stick**: stop receiving cards (terminate episode)

- **Reward for stick**:
  - +1 if player’s sum > dealer’s sum
  - 0 if player’s sum = dealer’s sum
  - -1 if player’s sum < dealer’s sum

- **Action** **twist/hit**: take another card (assuming no replacements)

- **Reward for twist/hit**:
  - -1 if player’s sum > 21 (’bust’, terminates episode)
  - 0 otherwise

- Dealer always sticks on any sum ≥ 17.
MC-Based Prediction Example: Blackjack (2)

Fig. 5.1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation.

Exercise 5.1 Consider the diagrams on the right in Figure 5.1. Why does the estimated value function jump up for the last two rows in the rear? Why does it drop for the whole last row on the left? Why are the frontmost values higher in the upper diagrams than in the lower?

Exercise 5.2 Suppose every-visit MC was used instead of first-visit MC on the blackjack task. Would you expect the results to be very different? Why or why not?

Although we have complete knowledge of the environment in the blackjack task, it would not be easy to apply DP methods to compute the value function. DP methods require the distribution of next events—in particular, they require the environment's dynamics as given by the four-argument function $p$—and it is not easy to determine this for blackjack. For example, suppose the player's sum is 14 and he chooses to stick. What is his probability of terminating with a reward of +1 as a function of the dealer's showing card? All of the probabilities must be computed before DP can be applied, and such computations are often complex and error-prone. In contrast, generating the sample games required by Monte Carlo methods is easy. This is the case surprisingly often; the ability of Monte Carlo methods to work with sample episodes alone can be a significant advantage even when one has complete knowledge of the environment’s dynamics.

Can we generalize the idea of backup diagrams to Monte Carlo algorithms? The general idea of a backup diagram is to show at the top the root node to be updated and to show below all the transitions and leaf nodes whose rewards and estimated values contribute to the update. For Monte Carlo estimation of $v_\pi$, the root is a state node, and below it is the entire trajectory of transitions along a particular single episode, ending
MC Estimation of Action Values

Is a model available (i.e., tuple $\langle X, U, P, R, \gamma \rangle$)?

- Yes: state values plus one-step prediction deliver optimal policy.
- No: action values are very useful to directly obtain optimal choices.
- Recap policy improvement from last lecture.

Estimating $q_{\pi}(x, u)$ using MC approach:

- Analog to state values summarized in Algo. 4.1.
- Only small extension: a visit refers to a state-action pair $(x, u)$.
- First-visit and every-visit variants exist.

Possible problem when following a deterministic policy $\pi$:

- Certain state-action pairs $(x, u)$ are never visited.
- Missing degree of exploration.
- Workaround: exploring starts, i.e., starting episodes in random state-action pairs $(x, u)$ and thereafter following $\pi$. 
Table of Contents

1. General Idea and Differences to Dynamic Programming
2. Basic Monte Carlo Prediction
3. Basic Monte Carlo Control
4. Extensions to Monte Carlo On-Policy Control
5. Monte Carlo Off-Policy Prediction
6. Monte Carlo Off-Policy Control
Applying Generalized Policy Iteration (GPI) to MC Control

GPI concept is directly applied to MC framework using action values:

\[ \pi_0 \to \hat{q}_{\pi_0} \to \pi_1 \to \hat{q}_{\pi_1} \to \cdots \pi^* \to \hat{q}_{\pi^*}. \]  

\[ (4.4) \]

- Degree of freedom: Choose number of episodes to approximate \( \hat{q}_{\pi_i} \).
- Policy improvement is done by greedy choices:

\[ \pi(x) = \arg \max_u q(x, u) \quad \forall x \in \mathcal{X}. \]  

\[ (4.5) \]

**Fig. 4.6:** Transferring GPI to MC-based control
(source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
Assuming that one is operating in an unknown MDP, the policy improvement theorem Theo. 3.1 is still valid for MC-based control:

\[
q_{\pi_i}(x, \pi_{i+1}(x)) = q_{\pi_i}(x, \arg\max_u q_{\pi_i}(x, u)),
\]
\[
= \max_u q_{\pi_i}(x, u),
\]
\[
\geq q_{\pi_i}(x, \pi_i(x)),
\]
\[
\geq v_{\pi_i}(x).
\] (4.6)

- Each \( \pi_{i+1} \) is uniformly better or just as good (if optimal) as \( \pi_i \).
- Assumption: All state-action pairs are evaluated due to sufficient exploration.
  - For example using exploring starts.
Algorithmic Implementation: MC-Based Control

\textbf{output:} Optimal deterministic policy $\pi^*$

\textbf{init:} $\pi_{i=0}(x) \in \mathcal{U}$ arbitrarily $\forall x \in \mathcal{X}$

- $\hat{q}(x, u)$ arbitrarily $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$
- $n(x, u) \leftarrow$ an empty list for state-action visits $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

\textbf{repeat}

\hspace{1em} $i \leftarrow i + 1$

\hspace{1em} Choose $\{x_0, u_0\}$ randomly such that all pairs have probability $> 0$ ;

\hspace{1em} Generate an episode from $\{x_0, u_0\}$ following $\pi_i$ until termination step $T_i$;

\hspace{1em} Set $g \leftarrow 0$;

\hspace{1em} \textbf{for} $k = T_i - 1, T_i - 2, T_i - 3, \ldots, 0$ \textit{time steps} \textbf{do}

\hspace{2em} $g \leftarrow \gamma g + r_{k+1}$;

\hspace{2em} \textbf{if} $\{x_k, u_k\} \notin \langle \{x_0, u_0\}, \ldots, \{x_{k-1}, u_{k-1}\}\rangle$ \textbf{then}

\hspace{3em} $n(x_k, u_k) \leftarrow n(x_k, u_k) + 1$;

\hspace{3em} $\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + 1/n(x_k, u_k) \cdot (g - \hat{q}(x_k, u_k))$;

\hspace{3em} $\pi_i(x_k) \leftarrow \text{arg max}_u \hat{q}(x_k, u)$;

\hspace{1em} \textbf{until} $\pi_{i+1} = \pi_i$;

\textbf{Algo. 4.2:} MC-based control using exploring starts (first-visit)
Fig. 4.7: The optimal policy and state-value function for blackjack found by MC-based control using exploring starts. (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
Table of Contents

1. General Idea and Differences to Dynamic Programming
2. Basic Monte Carlo Prediction
3. Basic Monte Carlo Control
4. Extensions to Monte Carlo On-Policy Control
5. Monte Carlo Off-Policy Prediction
6. Monte Carlo Off-Policy Control
On-policy learning

- Evaluate or improve the policy used to make decisions.
- Agent picks own actions.
- Exploring starts (ES) is an on-policy method example.
- However: ES is a restrictive assumption and not always applicable (in some cases the starting state-action pair cannot be chosen freely).

Off-policy learning

- Evaluate or improve a policy different from that used to generate data.
- Agent cannot apply own actions.
- Will be focused in the next sections.
$\varepsilon$-Greedy as an On-Policy Alternative

- Exploration requirement:
  - Visit all state-action pairs with probability:
    \[ \pi(u|x) > 0 \quad \forall \{x \in X, u \in U\} \]  
    (4.7)
  - Policies with this characteristic are called: soft.
  - Level of exploration can be tuned during the learning process.

- $\varepsilon$-greedy on-policy learning
  - With probability $\varepsilon$ the agent’s choice (i.e., the policy output) is overwritten with a random action.
  - Probability of all non-greedy actions:
    \[ \frac{\varepsilon}{|U|} \]  
    (4.8)
  - Probability of the greedy action:
    \[ 1 - \varepsilon + \frac{\varepsilon}{|U|} \]  
    (4.9)
  - Above, $|U|$ is the cardinality of the action space.
Algorithmic Implementation $\varepsilon$-Greedy MC-Control

output: Optimal $\varepsilon$-greedy policy $\pi^*(u|x)$
parameter: $\varepsilon \in \mathbb{R} | 0 < \varepsilon << 1$

init: $\pi_{i=0}(u|x)$ arbitrarily soft $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$
    $\hat{q}(x,u)$ arbitrarily $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$
    $n(x,u) \leftarrow$ an empty list counting state-action visits $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

repeat
    Generate an episode following $\pi_i$: $x_0, u_0, r_1, \ldots, x_{T_i}, u_{T_i}, r_{T_i+1}$;
    $i \leftarrow i + 1$;
    Set $g \leftarrow 0$;
    for $k = T_i - 1, T_i - 2, T_i - 3, \ldots, 0$ time steps do
        $g \leftarrow \gamma g + r_{k+1}$;
        if $\{x_k, u_k\} \notin \langle \{x_0, u_0\}, \ldots, \{x_{k-1}, u_{k-1}\} \rangle$ then
            $n(x_k, u_k) \leftarrow n(x_k, u_k) + 1$;
            $\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + 1/n(x_k, u_k) \cdot (g - \hat{q}(x_k, u_k))$;
            $\tilde{u} \leftarrow \arg \max_u \hat{q}(x_k, u)$;
            $\pi_i(u|x_k) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{U}|, & u = \tilde{u} \\ \varepsilon/|\mathcal{U}|, & u \neq \tilde{u} \end{cases}$;
        end
    until $\pi_{i+1} = \pi_i$;

Algo. 4.3: MC-based control using $\varepsilon$-greedy approach
**ε-Greedy Policy Improvement (1)**

**Theorem 4.1: Policy improvement for ε-greedy policy**

Given an MDP, for any ε-greedy policy \( \pi \) the ε-greedy policy \( \pi' \) with respect to \( q_\pi \) is an improvement, i.e., \( v_{\pi'} > v_\pi \) \( \forall x \in \mathcal{X} \).

Small proof:

\[
q_\pi(x, \pi'(x)) = \sum_u \pi'(u|x)q_\pi(x, u),
\]

\[
= \frac{\varepsilon}{|U|} \sum_u q_\pi(x, u) + (1 - \varepsilon) \max_u q_\pi(x, u),
\]

(4.10)

\[
\geq \frac{\varepsilon}{|U|} \sum_u q_\pi(x, u) + (1 - \varepsilon) \sum_u \frac{\pi(u|x) - \varepsilon}{1 - \varepsilon} q_\pi(x, u).
\]

In the inequality line, the second term is the weighted sum over action values given an ε-greedy policy. This weighted sum will be always smaller or equal than \( \max_u q_\pi(x, u) \).
Continuation:

\[ q_\pi(x, \pi'(x)) \geq \frac{\varepsilon}{|U|} \sum_u q_\pi(x, u) + (1 - \varepsilon) \sum_u \frac{\pi(u|x) - \varepsilon}{1 - \varepsilon} q_\pi(x, u), \]

\[ = \frac{\varepsilon}{|U|} \sum_u (q_\pi(x, u) - q_\pi(x, u)) + \sum_u \pi(u|x) q_\pi(x, u), \]

\[ = \sum_u \pi(u|x) q_\pi(x, u), \]

\[ = v_\pi(x). \]

- **Policy improvement theorem is still valid when comparing \( \varepsilon \)-greedy policies against each other.**

- **But:** There might be a non-\( \varepsilon \)-greedy policy which is better.
MC-Based Control Example: Forest Tree MDP (1)

Fig. 4.8: Different estimates of forest tree MDP ($\alpha = 0.2, \gamma = 0.8$) using MC control with $\varepsilon = 0.2$ over the number of episodes. Mean is red and standard deviation is light blue, both calculated based on 2000 independent runs.
Fig. 4.9: Different estimates of forest tree MDP ($\alpha = 0.2, \gamma = 0.8$) using MC control with $\varepsilon = 0.05$ over the number of episodes. Mean is red and standard deviation is light blue, both calculated based on 2000 independent runs.
Observations on forest tree MDP with $\varepsilon$-greedy MC-based control:

- Rather slow convergence rate: quite a number of episodes is required.
- Significant uncertainty present in a single sequence.
- Later states are less often visited and, therefore, more uncertain.
- Exploration is controlled by $\varepsilon$: in a totally greedy policy the state $x = 3$ is not visited at all (cf. Fig. 2.17). With $\varepsilon$-greedy this state is visited occasionally.

Nevertheless, the above figures highlight that MC-based control for the forest tree MDP tend towards the optimal policies discovered by dynamic programming (cf. Tab. 3.3).
Definition 4.1: Greedy in the limit with infinite exploration (GLIE)

A learning policy \( \pi \) is called GLIE if it satisfies the following two properties:

1. If a state is visited infinitely often, then each action is chosen infinitely often:

\[
\lim_{i \to \infty} \pi_i(u|x) = 1 \quad \forall \{x \in X, u \in U\}.
\]

(4.12)

2. In the limit \( i \to \infty \) the learning policy is greedy with respect to the learned action value:

\[
\lim_{i \to \infty} \pi_i(u|x) = \pi(x) = \arg \max_u q(x, u) \quad \forall x \in X.
\]

(4.13)
Theorem 4.2: Optimal decision using MC-control with $\varepsilon$-greedy

MC-based control using $\varepsilon$-greedy exploration is GLIE, if $\varepsilon$ is decreased at rate

$$\varepsilon_i = \frac{1}{i} \quad (4.14)$$

with $i$ being the increasing episode index. In this case,

$$\hat{q}(x, u) = q^*(x, u) \quad (4.15)$$

follows.

Remarks:

- Limited feasibility: infinite number of episodes required.
- $\varepsilon$-greedy is an undirected exploration strategy. Can that be the most efficient way of learning?
Table of Contents

1. General Idea and Differences to Dynamic Programming
2. Basic Monte Carlo Prediction
3. Basic Monte Carlo Control
4. Extensions to Monte Carlo On-Policy Control
5. Monte Carlo Off-Policy Prediction
6. Monte Carlo Off-Policy Control
Off-Policy Learning Background

Drawback of on-policy learning:
▶ Only a compromise: comes with inherent exploration but at the cost of learning action values for a near-optimal policy.

Idea off-policy learning:
▶ Use two separated policies:
  ◀ Behavior policy $b(u|x)$: Explores in order to generate experience.
  ◀ Target policy $\pi(u|x)$: Learns from that experience to become the optimal policy.

▶ Use cases:
  ◀ Learn from observing humans or other agents/controllers.
  ◀ Re-use experience generated from old policies ($\pi_0, \pi_1, \ldots$).
  ◀ Learn about multiple policies while following one policy.
Off-Policy Prediction Problem Statement

MC off-policy prediction problem statement

- Estimate $v_\pi$ and/or $q_\pi$ while following $b(u|x)$.
- Both policies are considered fixed (prediction assumption).

Requirement:

- **Coverage:** Every action taken under $\pi$ must be (at least occasionally) taken under $b$, too. Hence, it follows:

\[
\pi(u|x) > 0 \Rightarrow b(u|x) > 0 \quad \forall \{x \in X, u \in U\}. \tag{4.16}
\]

- Consequences from that:
  - In any state $b$ is not identical to $\pi$, $b$ must be stochastic.
  - Nevertheless: $\pi$ might be deterministic (e.g., control applications) or stochastic.
Importance Sampling

What is the probability of observing a certain trajectory on random variables $U_k, X_{k+1}, U_{k+1}, \ldots, X_T$ starting in $X_k$ while following $\pi$?

$$\mathbb{P} [U_k, X_{k+1}, U_{k+1}, \ldots, X_T | X_k, \pi],$$

$$= \pi(U_k | X_k)p(X_{k+1} | X_k, U_k) \pi(U_{k+1} | X_{k+1}) \cdots,$$

$$= \prod_{k}^{T-1} \pi(U_k | X_k)p(X_{k+1} | X_k, U_k).$$

(4.17)

Above $p$ is the state-transition probability (cf. Def. 2.6).

**Definition 4.2: Importance sampling ratio**

The relative probability of a trajectory under the target and behavior policy, the importance sampling ratio, from sample step $k$ to $T$ is:

$$\rho_{k:T} = \frac{\prod_{k}^{T-1} \pi(U_k | X_k)p(X_{k+1} | X_k, U_k)}{\prod_{k}^{T-1} b(U_k | X_k)p(X_{k+1} | X_k, U_k)} = \frac{\prod_{k}^{T-1} \pi(U_k | X_k)}{\prod_{k}^{T-1} b(U_k | X_k)}. \quad (4.18)$$
Definition 4.3: State-value estimation via Monte Carlo importance sampling

Estimating the state value $v_\pi$ following a behavior policy $b$ using (ordinary) importance sampling (OIS) results in scaling and averaging the sampled returns by the importance sampling ratio per episode:

$$\hat{v}_\pi(x_k) = \frac{\sum_{k \in T(x_k)} \rho_k T(k) g_k}{|T(x_k)|}.$$  \hspace{1cm} (4.19)

Notation remark:

$\triangleright$ $T(x_k)$: set of all time steps in which the state $x_k$ is visited the first or each time per episode.

$\triangleright$ $T(k)$: Termination of a specific episode starting from $k$.

General remark:

$\triangleright$ From (4.18) it can be seen that $\hat{v}$ is bias-free (first-visit assumption).

$\triangleright$ However, if $\rho$ is large (distinctly different policies) the estimate’s variance is large (i.e., uncertain for small numbers of samples).
Definition 4.4: Weighted importance sampling for MC-based prediction

Estimating the state value $v_{\pi}$ following a behavior policy $b$ using weighted importance sampling (WIS) results in a scaling and a weighted averaging of the sampled returns by the importance sampling ratio per episode:

$$\hat{v}_{\pi}(x_k) = \begin{cases} \frac{\sum_{k \in T(x_k)} \rho_{k:T(k)} g_k}{\sum_{k \in T(x_k)} \rho_{k:T(k)}} , & \text{if } \sum_{k \in T(x_k)} \rho_{k:T(k)} \neq 0 \\ 0 , & \text{if } \sum_{k \in T(x_k)} \rho_{k:T(k)} = 0 \end{cases}.$$ (4.20)

Comparison to OIS:

- Weighting introduces bias (which reduces to zero in the limit).
- But: weighting limits variance while the OIS’s variance is unbounded.
  - Largest weight per return sample for WIS is one, while OIS’s weights are unbounded.

- Conclusion: WIS delivers ’better’ estimates for few samples and, therefore, is often preferred in practice.
Importance Sampling Example: Blackjack

- Behavior policy: stick or hit at equal probability.
- Target policy: stick if player has sum of 20 or 21 (cf. Fig. 4.5).
- True state value of target policy is $v = -0.2773$. 

**Fig. 4.10:** Off-policy state-value estimation of single state \{dealer's sum = 2, player's sum = 13 and player has usable ace\} (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
WIS: Incremental Implementation

- For OIS, (4.19) can be directly transformed into an incremental implementation following (4.2).
- For WIS an incremental implementation is also highly desirable to save memory and computational resources.
- Rewrite WIS (4.20) for a sequence of returns $g_1, g_2, \ldots$:

$$\hat{v}_i(x) = \frac{\sum_{k=1}^{i-1} w_k g_k}{\sum_{k=1}^{i-1} w_k} \text{ with } w_k = \rho_k T(k). \quad (4.21)$$

- The WIS recursive/incremental update rule is then:

$$\hat{v}_{i+1}(x) = \hat{v}_i(x) + \frac{w_i}{c_i} (g_i - \hat{v}_i(x)), \quad i \geq 1 \quad (4.22)$$

$$c_{i+1} = c_i + w_{i+1} \text{ with } c_0 = 0. \quad (4.23)$$

- Above, $c_i$ is the cumulative sum of weights over all considered state visits.
**MC-Based Off-Policy Prediction Using WIS**

**input:** a target policy $\pi$ to be evaluated

**init:** $\hat{q}(x, u)$ arbitrarily $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

$c(x, u) \leftarrow 0$ cumulative sum of WIS weights $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

**for** $j = 1, \ldots, J$ **episodes** **do**

Choose an arbitrary behavior policy $b$ with coverage of $\pi$;

Generate an episode following $b$: $x_0, u_0, r_1, \ldots, x_{T_j}, u_{T_j}, r_{T_j+1}$;

Set $g \leftarrow 0$;

Set $w \leftarrow 1$;

**for** $k = T_i - 1, T_i - 2, T_i - 3, \ldots, 0$ **time steps** **while** $w \neq 0$ **do**

$g \leftarrow \gamma g + r_{k+1}$;

$c(x_k, u_k) \leftarrow c(x_k, u_k) + w$;

$\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \frac{w}{c(x_k, u_k)} (g - \hat{q}(x_k, u_k))$;

$w \leftarrow w \frac{\pi(u_k|x_k)}{b(u_k|x_k)}$;

**Algo. 4.4:** MC-based off-policy prediction using WIS
MC-Based Off-Policy Prediction: Forest Tree MDP

- Behavior policy $b$: 'fifty-fifty'
- Target policy $\pi$: optimal policy from Fig. 2.17

**Fig. 4.11:** Action-value estimates of forest tree MDP ($\alpha = 0.2, \gamma = 0.8$) using MC-based off-policy prediction with WIS over the number of episodes. Mean is red and standard deviation is light blue, both calculated based on 2000 independent runs.
Table of Contents

1. General Idea and Differences to Dynamic Programming
2. Basic Monte Carlo Prediction
3. Basic Monte Carlo Control
4. Extensions to Monte Carlo On-Policy Control
5. Monte Carlo Off-Policy Prediction
6. Monte Carlo Off-Policy Control
Off-Policy Monte Carlo Control: Introduction

Just put everything together:

- MC-based control utilizing GPI (cf. Fig. 4.6),
- Off-policy learning based on Algo. 4.4.

Requirement for off-policy MC-based control:

- **Coverage**: behavior policy $b$ has nonzero probability of selecting actions that might be taken by the target policy $\pi$.
- **Consequence**: behavior policy $b$ is soft (e.g., $\varepsilon$-soft).
init: $\hat{q}(x, u)$ arbitrarily $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

$c(x, u) \leftarrow 0$ cumulative sum of WIS weights $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

$\pi(x) \leftarrow \arg \max_u \hat{q}(x, u)$ (with ties broken consistently)

for $j = 1, \ldots, J$ episodes do

Choose an arbitrary soft policy $b$;
Generate an episode following $b$: $x_0, u_0, r_1, \ldots, x_{T_j}, u_{T_j}, r_{T_j+1}$;
Set $g \leftarrow 0$;
Set $w \leftarrow 1$;

for $k = T_i - 1, T_i - 2, T_i - 3, \ldots, 0$ time steps do

$g \leftarrow \gamma g + r_{k+1}$;
$c(x_k, u_k) \leftarrow c(x_k, u_k) + w$;
$\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \frac{w}{c(x_k, u_k)} (g - \hat{q}(x_k, u_k))$;

$\pi(x_k) \leftarrow \arg \max_u \hat{q}(x_k, u_k)$ (with ties broken consistently);

if $u_k \neq \pi(x_k)$ then

break/exit inner loop;

$w \leftarrow \frac{w}{b(u_k|x_k)}$;

Algo. 4.5: MC-based off-policy control using WIS
MC-Based Off-Policy Control: Forest Tree MDP

- Behavior policy $b$: 'fifty-fifty'
- Initial target policy $\pi$: 'fifty-fifty'

![Figure 4.12: Action-value estimates and policy trends of forest tree MDP](image)

\( \hat{q}_\pi(x = 1, u = w) \) \( \hat{q}_\pi(x = 2, u = w) \) \( \hat{q}_\pi(x = 3, u = w) \)

\( \pi(u = w | x = 1) \) \( \pi(u = w | x = 2) \) \( \pi(u = w | x = 3) \)

Number of episodes

**Fig. 4.12:** Action-value estimates and policy trends of forest tree MDP (\( \alpha = 0.2, \gamma = 0.8 \)) using MC-based off-policy control with WIS over the number of episodes. Mean is red and standard deviation is light blue, both calculated based on 2000 independent runs.
Remark on MC-Based Off-Policy Control with WIS

Potential problems with MC-based off-policy control:

▶ Learns only 'from tails' of episodes if the remaining actions generated by $b$ are greedy with respect to $\pi$.
  - See last and inner-most if query in Algo. 4.5.
  - Hence, a lot of samples generated by $b$ remain unused for training $\pi$.

▶ Slows down training.

▶ Adds uncertainty to the training process.

Possible improvements/extensions:

1. Per-decision importance sampling
2. Discounting-aware importance sampling (if discount factor $\gamma < 1$)

▶ Introduce modified value estimators for importance sampling.
▶ Main goal: decrease variance of the learning process.
▶ For details see chapter 5.8 & 5.9 of R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018.
MC methods allow model-free learning of value functions and optimal policies from experience in the form of sampled episodes.

Using deep back-ups over full episodes, MC is largely based on averaging returns.

MC-based control reuses generalized policy iteration (GPI), i.e., mixing policy evaluation and improvement.

Maintaining sufficient exploration is important:

- Exploring starts: not feasible in all applications but simple.
- On-policy $\epsilon$-greedy learning: trade-off between optimality and exploration cannot be resolved easily.
- Off-policy learning: agent learns about a (possibly deterministic) target policy from an exploratory, soft behavior policy.

Importance sampling transforms expectations from the behavior to the target policy.

- This estimation task comes with a bias-variance-dilemma.
- Slow learning can result from ineffective experience usage in MC methods.
The End for Today

Thanks for your attention and have a nice week!