Monte Carlo Methods and Temporal-Difference Learning

Reinforcement Learning : Module 05

Monte Carlo Prediction, Monte Carlo Estimation of Action Values, Monte Carlo Control, TD Prediction, TD control using Q-Learning

The slides contain

- Excerpts from the book Reinforcement Learning : An Introduction, 2nd edition, Richard S. Sutton and Andrew G. Barto
- Content acquired using ChatGPT, Gemini and CoPilot

Monte Carlo Methods : Introduction

- Monte Carlo methods are a class of techniques used in reinforcement learning (RL) for estimating values, making decisions, and finding optimal policies.
- These methods are based on random sampling and are particularly useful when the dynamics of the environment are unknown or complex.
- The term “Monte Carlo” is often used more broadly for any estimation method whose operation involves a significant random component.

Monte Carlo Prediction

Monte Carlo Prediction

- MC Prediction is a method for learning the state-value function for a given policy.
- Recall that the value of a state is the expected return—expected cumulative future discounted reward—starting from that state.
- An obvious way to estimate it from experience is simply to average the returns observed after visits to that state.
- As more returns are observed, the average should converge to the expected value.

Monte Carlo Prediction

- Suppose we wish to estimate $v_\pi(s)$, the value of a state s under policy π, given a set of episodes obtained by following π and passing through s.
- Each occurrence of state s in an episode is called a visit to s.
- s may be visited multiple times in the same episode; let us call the first time it is visited in an episode the first visit to s.
- The ***first-visit MC method*** estimates $v_\pi(s)$ as the average of the returns following first visits to s, whereas the ***every-visit MC method*** averages the returns following all visits to s.
- These two Monte Carlo (MC) methods are very similar but have slightly different theoretical properties.

first-visit Monte Carlo Prediction Method

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Every-visit Monte Carlo Prediction Method

- Every-visit MC would be the same except without the check for S_t having occurred earlier in the episode.

Example 5.1 : Blackjack

Example 5.1: Blackjack The object of the popular casino card game of *blackjack* is to obtain cards the sum of whose numerical values is as great as possible without exceeding 21. All face cards count as 10, and an ace can count either as 1 or as 11. We consider the version in which each player competes independently against the dealer. The game begins with two cards dealt to both dealer and player. One of the dealer's cards is face up and the other is face down. If the player has 21 immediately (an ace and a 10-card), it is called a *natural*. He then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (*hits*), until he either stops (*sticks*) or exceeds 21 (*goes bust*). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn. The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome—win, lose, or draw—is determined by whose final sum is closer to 21.

Example 5.1 : Blackjack

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. All rewards within a game are zero, and we do not discount ($\gamma = 1$); therefore these terminal rewards are also the returns. The player's actions are to hit or to stick. The states depend on the player's cards and the dealer's showing card. We assume that cards are dealt from an infinite deck (i.e., with replacement) so that there is no advantage to keeping track of the cards already dealt. If the player holds an ace that he could count as 11 without going bust, then the ace is said to be *usable*. In this case it is always counted as 11 because counting it as 1 would make the sum 11 or less, in which case there is no decision to be made because, obviously, the player should always hit. Thus, the player makes decisions on the basis of three variables: his current sum (12–21), the dealer's one showing card (ace–10), and whether or not he holds a usable ace. This makes for a total of 200 states.

Example 5.1 : Blackjack

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 environments 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.

Example 5.1 : Blackjack

- Consider the policy that sticks if the player's sum is 20 or 21, and otherwise hits. To find the state-value function for this policy by a Monte Carlo approach, one simulates many blackjack games using the policy and averages the returns following each state.
- In this way, we obtained the estimates of the state-value function shown in Figure 5.1.
- The estimates for states with a usable ace are less certain and less regular because these states are less common. In any event, after 500,000 games the value function is very well approximated.

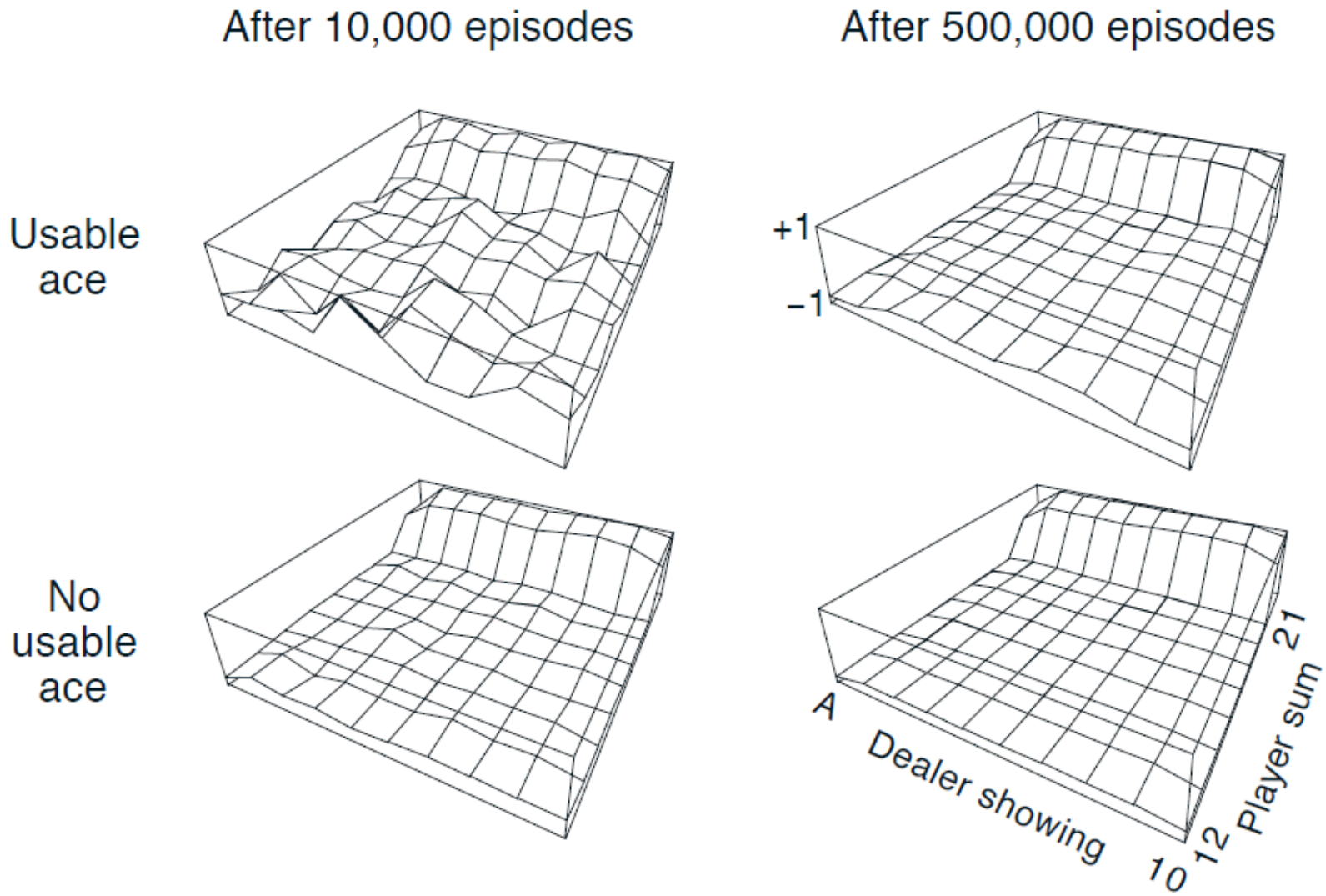


Figure 5.1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation. ■

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 off for the whole last row on the left?
  - Why are the frontmost values higher in the upper diagrams than in the lower?

Exercise 5.1

1. It is due to the strategy that player will not stop until meeting 20 or 21. That indicates player would face the risk of failing by hitting, which results the low value part right before 20 and 21. On the 20 and 21, however, the player stops and has a very high opportunity to win, especially when dealer will stop at 17 or higher.

2. It drop off for the whole last row on the left because if Dealer showing an ACE, It has very high possibility of getting higher score than the player when it counts as 11. Thus, the value of dealer's A has contained the dealer's winning rate of making it usable or not. Other cards have no such condition thus A is a special which makes the gap.

3. Frontmost values are higher in the upper diagrams because A represent dual values of being used as 1 and 11 in the upper diagram. It makes the player better off and is similar with the condition of having drop in the leftmost rows.

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?
- No. Black jack does not contain two duplicate state in any episode, making first-visit and every-visit method essentially the same thing.

Monte Carlo Estimation of Action Values

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.
- With a model, state values alone are sufficient to determine a policy; one simply looks ahead one step and chooses whichever action leads to the best combination of reward and next state.
- Without a model, however, state values alone are not sufficient. One must explicitly estimate the value of each action in order for the values to be useful in suggesting a policy.
- Thus, one of our primary goals for Monte Carlo methods is to estimate q_*.

Why finding action-values is important here?

- Decision Making:
  - In reinforcement learning, the ultimate goal is to determine the best action to take in each state to maximize cumulative rewards over time. While state values indicate the potential long-term rewards achievable from each state, they do not provide information about which action to take in a given state. Action values, on the other hand, explicitly quantify the expected cumulative reward of taking each action from a given state, facilitating decision-making.
- Policy Improvement:
  - Action values are essential for improving the policy of an agent. By estimating the value of each action in each state, an agent can select the action that maximizes the expected cumulative reward, leading to the formulation of a more effective policy. This process is fundamental in methods like Q-learning and Monte Carlo control, where action values are iteratively updated to converge towards optimal policies.

Why finding action-values is important here?

- Exploration vs. Exploitation:
  - Action values play a crucial role in balancing exploration and exploitation. Agents often need to explore uncertain or unexplored regions of the state-action space to discover optimal policies. By estimating action values, agents can select actions that balance the exploration of new states and the exploitation of known information about the environment's rewards.
- Efficiency:
  - Explicitly estimating action values allows agents to focus computational resources on the most promising actions in each state, rather than considering all possible actions indiscriminately. This improves the efficiency of reinforcement learning algorithms, particularly in large state spaces, by guiding the agent towards actions that are more likely to lead to higher cumulative rewards.

Monte Carlo Estimation of Action Values

- The policy evaluation problem for action values is to estimate $q_\pi(s, a)$, the expected return when starting in state s, taking action a, and thereafter following policy π.
- A state– action pair s, a is said to be visited in an episode if ever the state s is visited and action a is taken in it.
- The ***every-visit MC method*** estimates the value of a state–action pair as the average of the returns that have followed all the visits to it.
- The ***first-visit MC method*** averages the returns following the first time in each episode that the state was visited and the action was selected.
- These methods converge to the true expected values as the number of visits to each state–action pair approaches infinity.

Monte Carlo Estimation of Action Values

- The only complication is that many state–action pairs may never be visited. If π is a deterministic policy, then in following π one will observe returns only for one of the actions from each state. With no returns to average, the Monte Carlo estimates of the other actions will not improve with experience.
- To compare alternatives we need to estimate the value of all the actions from each state, not just the one we currently favor.
- For policy evaluation to work for action values, we must assure continual exploration.
- One way to do this is by specifying that the episodes start in a state–action pair, and that every pair has a nonzero probability of being selected as the start. This guarantees that all state–action pairs will be visited an infinite number of times in the limit of an infinite number of episodes.
  - We call this the assumption of exploring starts.

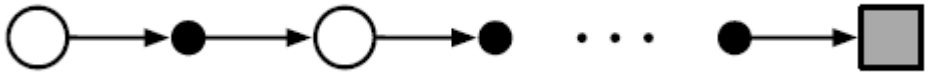
Monte Carlo Estimation of Action Values

- The assumption of exploring starts is sometimes useful, but of course it cannot be relied upon in general, particularly when learning directly from actual interaction with an environment.
- In that case the starting conditions are unlikely to be so helpful.
- The most common alternative approach to assuring that all state–action pairs are encountered is to consider only policies that are stochastic with a nonzero probability of selecting all actions in each state.

We discuss two important variants of this approach in later sections. For now, we retain the assumption of exploring starts and complete the presentation of a full Monte Carlo control method.

Exercise 5.3

- What is the backup diagram for Monte Carlo estimation of q_π?

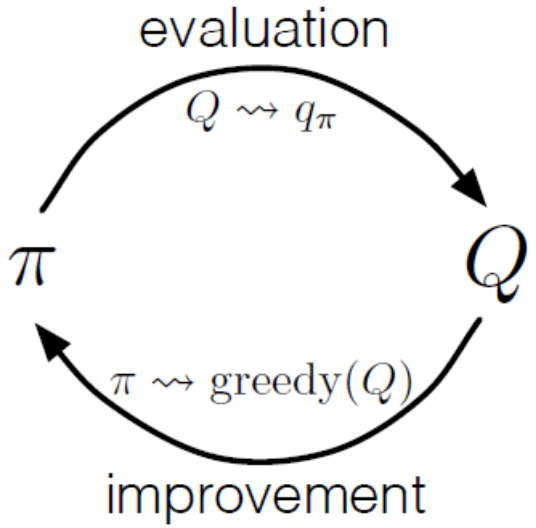


An empty circle and a black filled circle represent a state action pair.

Monte Carlo Control

Monte Carlo Control

- Let us consider how Monte Carlo estimation can be used in control, that is, to approximate optimal policies.
- The overall idea is to proceed according to the idea of generalized policy iteration (GPI).
- In GPI one maintains both an approximate policy and an approximate value function.
- The value function is repeatedly altered to more closely approximate the value function for the current policy, and the policy is repeatedly improved with respect to the current value function, as suggested by the diagram



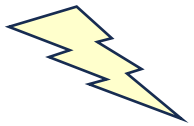
Monte Carlo policy iteration

We perform alternating complete steps of policy evaluation and policy improvement, beginning with an arbitrary policy π_0 and ending with the optimal policy and optimal action-value function:

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*,$$

where $\xrightarrow{\text{E}}$ denotes a complete policy evaluation and $\xrightarrow{\text{I}}$ denotes a complete policy improvement.

Many episodes are experienced, with the approximate action-value function approaching the true function asymptotically. For the moment, let us assume that we do indeed observe an infinite number of episodes and that, in addition, the episodes are generated with exploring starts. Under these assumptions, the Monte Carlo methods will compute each q_{π_k} exactly, for arbitrary π_k.



In the asymptotic limit, as the number of interactions with the environment becomes infinitely large, the action-value function estimates reach a point where they stabilize and closely approximate the true values

Monte Carlo policy improvement

Policy improvement is done by making the policy greedy with respect to the current value function. In this case we have an *action*-value function, and therefore no model is needed to construct the greedy policy. For any action-value function q, the corresponding greedy policy is the one that, for each $s \in \mathcal{S}$, deterministically chooses an action with maximal action-value:

$$\pi(s) \doteq \arg\max_a q(s,a). \tag{5.1}$$

Policy improvement then can be done by constructing each π_{k+1} as the greedy policy with respect to q_{π_k}. The policy improvement theorem (Section 4.2) then applies to π_k and π_{k+1} because, for all $s \in \mathcal{S}$,

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

Monte Carlo method : Assumptions

- We made two unlikely assumptions above in order to easily obtain guarantee of convergence for the Monte Carlo method.
  - One was that the episodes have exploring starts, and
  - the other was that policy evaluation could be done with an infinite number of episodes.
- To obtain a practical algorithm we will have to remove both assumptions.
- We postpone consideration of the first assumption until later.
- For now we focus on the assumption that policy evaluation operates on an infinite number of episodes.

Monte Carlo method : Restricting number of episodes

q_{π_k}

- There are two ways to solve the problem.
- One is to hold firm to the idea of approximating $q_{\pi k}$ in each policy evaluation.
- **Measurements and assumptions** are made to **obtain bounds** on the **magnitude and probability of error** in the estimates, and then sufficient **steps are taken** during each policy evaluation **to assure** that these **bounds are sufficiently small**.
- This approach can probably be made completely satisfactory in the sense of guaranteeing correct convergence up to some level of approximation.
- However, it is also likely to require far too many episodes to be useful in practice on any but the smallest problems.

Measurements and assumptions

- **Measurements**: The performance of a reinforcement learning algorithm is often assessed based on the accuracy of its estimates of value functions or action values. To understand the accuracy of these estimates, measurements are made to quantify the magnitude and probability of error.
- **Assumptions**: Various assumptions may be made about the environment, the learning algorithm, or the data collected during the learning process. These assumptions help in formulating theoretical bounds on the error in the estimates.

Monte Carlo method : Restricting number of episodes

There is a second approach to avoiding the infinite number of episodes nominally required for policy evaluation, in which we give up trying to complete policy evaluation before returning to policy improvement. On each evaluation step we move the value function *toward* q_{π_k}, but we do not expect to actually get close except over many steps. We used this idea when we first introduced the idea of GPI in Section 4.6. One extreme form of the idea is value iteration, in which only one iteration of iterative policy evaluation is performed between each step of policy improvement. The in-place version of value iteration is even more extreme; there we alternate between improvement and evaluation steps for single states.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Monte Carlo Control without Exploring Starts

- How to avoid the unlikely assumption of exploring starts?
- The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them.
- There are two approaches to ensuring this, resulting in what we call on-policy methods and off-policy methods.
- On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data.
- The Monte Carlo ES method developed earlier is an example of an on-policy method.

On-policy Monte Carlo Control

In on-policy control methods the policy is generally *soft*, meaning that $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}(s)$, but gradually shifted closer and closer to a deterministic optimal policy. Many of the methods discussed in Chapter 2 provide mechanisms for this. The on-policy method we present in this section uses ε-*greedy* policies, meaning that most of the time they choose an action that has maximal estimated action value, but with probability ε they instead select an action at random. That is, all nongreedy actions are given the minimal probability of selection, $\frac{\varepsilon}{|\mathcal{A}(s)|}$, and the remaining bulk of the probability, $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$, is given to the greedy action. The ε-greedy policies are examples of ε-*soft* policies, defined as policies for which $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ for all states and actions, for some $\varepsilon > 0$. Among ε-soft policies, ε-greedy policies are in some sense those that are closest to greedy.

From Module 02 : Optimistic Initial values, Upper confidence Bound, Gradient Bandit

On-policy Monte Carlo Control

The overall idea of on-policy Monte Carlo control is still that of GPI. As in Monte Carlo ES, we use first-visit MC methods to estimate the action-value function for the current policy. Without the assumption of exploring starts, however, we cannot simply improve the policy by making it greedy with respect to the current value function, because that would prevent further exploration of nongreedy actions. Fortunately, GPI does not require that the policy be taken all the way to a greedy policy, only that it be moved *toward* a greedy policy. In our on-policy method we will move it only to an ε-greedy policy. For any ε-soft policy, π, any ε-greedy policy with respect to q_π is guaranteed to be better than or equal to π. The complete algorithm is given in the box below.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \text{argmax}_a \, Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Need of Off-policy Methods

All learning control methods face a dilemma: They seek to learn action values conditional on subsequent *optimal* behavior, but they need to behave non-optimally in order to explore all actions (to *find* the optimal actions). How can they learn about the optimal policy while behaving according to an exploratory policy? The on-policy approach in the preceding section is actually a compromise—it learns action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior. The policy being learned about is called the *target policy*, and the policy used to generate behavior is called the *behavior policy*. In this case we say that learning is from data "off" the target policy, and the overall process is termed *off-policy learning*.

Comparing On-policy and Off-policy methods

- On-policy methods are generally simpler and are considered first.
- Off-policy methods require additional concepts and notation, and because the data is due to a different policy, off-policy methods are often of greater variance and are slower to converge.
- On the other hand, off-policy methods are more powerful and general. They include on-policy methods as the special case in which the target and behavior policies are the same.
- Off-policy methods also have a variety of additional uses in applications.
  - For example, they can often be applied to learn from data generated by a conventional non-learning controller, or from a human expert.

Off-policy Prediction via Importance Sampling

- To be added

TD Prediction

To be added

TD control using Q-Learning

To be added

End