1 Introduction

Humans learn by interacting with our environment.

RL is focused on goal-directed learning from interaction.

1.1 Reinforcement Learning

RL is learning what to do to maximise a numerical reward signal. The learner discovers which action yields the most reward by trial-and-error. Actions may affect immediate reward and all subsequent rewards (delayed rewards).

The RL problem is the optimal control of incompletely-known Markov Decisions Processes (MDPs).

A challenge in RL is the trade-off between exploration and exploitation.

1.3 Elements of Reinforcement Learning

Policy: defines learning agent's way of behaving at a given time.

Reward Signal: defines goal of a RL problem; objective of agent is to maximise total reward it receives over the long run.

Value function: specifies what is good in the long run.

Value of a state: total amount of reward an agent can expect to accumulate over the features, starting from that state.

(**Model:** mimics behaviour of the environment and allows inferences to be made about how the environment will behave; used for planning.)

1.4 Limitations and Scope

Policy and value function take state as input and output another state.

1.5 An Extended Example: Tic-Tac-Toe

Let S_t denote the state before the greedy move, and S_{t+1} the state after that move. The update to the estimated value of S_t, $V(S_t)$, can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha \left[V(S_{t+1}) - V(S_t)\right]$$

where alpha, the *step-size parameter* is a small positive fraction which influences the rate of learning.

Temporal-difference learning: changes are based on a difference between estimates at two successive times.

2 Multi-armed Bandits

2.1 A k-armed bandit Problem

k-armed bandit problem: k levers on a slot machine; each of k actions has an expected reward given that that action is selected.

Value of an action: expected reward given that that action is selected.

A_t: action selected on time step t.

R_t: corresponding reward of A_t.

$q_*(a)$: value of an action a.

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

Greedy action(s): action(s) whose estimated value is greatest.

Exploiting: selecting one of greedy action(s).

Exploring: selecting non-greedy action.

2.2 Action-value Methods

Action-value methods: methods for estimating the values of actions and for using the estimates to make action selection decisions.

True value of an action: mean reward when the action is selected.

Sample-average method $Q_t(a)$: average of rewards received.

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

Greedy method: $A_t \doteq argmax_a Q_t(a)$.

Epsilon-greedy method: with probability ϵ select randomly from all actions with equal probability, otherwise behave greedily.

- as number of steps increases, every action will be sampled an infinite number of times, ensuring all $Q_t(a)$ converge to their respective $q_*(a)$.

Exercise 2.2: Bandit example

Consider a k-armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4.

Consider applying to this problem a bandit algorithm using ϵ-greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a.

Suppose the initial sequence of actions and rewards is $A_1 = 1$, $R_1 = -1$, $A_2 = 2$, $R_2 = 1$, $A_3 = 2$, $R_3 = -2$, $A_4 = 2$, $R_4 = 2$, $A_5 = 3$, $R_5 = 0$.

On some of these time steps the ϵ case may have occurred, causing an action to be selected at random.

1) On which time steps did this definitely occur?
2) On which time steps could this possibly have occurred?

Solution:

Build a table for $Q_t(a)$ for each time step t:

	a=1	a=2	a=3	a=4
t=1	0	0	0	0
t=2	-1	0	0	0
t=3	-1	1	0	0
t=4	-1	-0.5	0	0
t=5	-1	0.33	0	0

- $A_1 = 1$: random or greedy
- $A_2 = 2$: random or greedy
- $A_3 = 2$: random or greedy
- $A_4 = 2$: definitely ϵ
- $A_5 = 3$: definitely ϵ

2.4 Incremental Implementation

Let R_i denote the reward received after the ith selection of an action, and let Q_n denote the estimate of its action value after it has been selected $n-1$ times, where

$$Q_n \doteq \frac{R_1 + \ldots + R_{n-1}}{n-1}.$$

If we record all rewards then computate whenever the estimated value was needed, memory and computational requirements would grow over time.

Instead, do

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \ldots = Q_n + \frac{1}{n}[R_n - Q_n]$$

Requires memory only for Q_n and n and the small computation for each new reward.

General form:

$$NewEstimate \leftarrow OldEstimate + Stepsize[Target - OldEstimate]$$

$[Target - OldEstimate]$ is the *error* in the estimate. Denote step-size parameter by α or $\alpha_t(a)$

2.5 Tracking a Nonstationary Problem

Averaging methods so far are appropriate for stationary bandit problems — reward probabilities do not change over time. For non stationary problems, makes more sense to give more weight to recent rewards than to long-past rewards. Can do this by using a constant step-size parameter, e.g. $\alpha \in (0, 1]$

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n] \ldots = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.$$

Results in Q_{n+1} being a weighted average of past rewards and initial estimate Q_1 since sum of the weights is $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^(n-i) = 1$. Weight given to R_i decays exponentially.

2.6 Optimistic Initial Values

Methods so far are dependent on initial action-value estimates, $Q_1(a)$ — *biased* by their initial estimates. For sample-average methods, bias disappears once all actions have been selected at least once, but for methods with constant α, the bias not usually a problem and can be helpful.

Initial action values can be used to encourage exploration by setting the initial estimate far from the true mean. Can be effective on stationary problems but not well-suited to non stationary problems. Any methods that focuses on initial conditions is unlikely to help with nonstationary problems.

2.7 Upper-Confidence-Bound Action Selection

In ϵ-greedy action selection, would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates. Can select actions according to

$$A_t \doteq argmax_a \left[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}} \right]$$

- $N_t(a)$ denotes the number of times that action a has been selected prior to time t

- $c > 0$ controls the degree of exploration
- If $N_t(a) = 0$ then a is considered to be a maximising action

This *upper confidence bound* (UCB) action selection is that the square-root term is a measure of the uncertainty or variance in the estimate of a's value. c determines the confidence level on the possible true value of action a. Each time a is elected, uncertainty is presumably reduced.

UCB often performs well on bandits, but is more difficult than ϵ-greedy to extend to the more general reinforcement learning settings.

2.8 Gradient Bandit Algorithms

Consider a numerical *preference* for each action a, which we denote $H_t(a)$ in R. The larger the preference, the more often that action is taken, but the preference has no interpretation in terms of reward. Only the relative preference of one action over another is important.

Let $\pi_t(a)$ be the probability of taking action a at time t.

$$Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

Initially all action preferences are the same so that all actions have equal probability of being selected

Without reward baseline term and gradient bandit algorithm, performance degraded.

2.9 Associative Search (Contextual bandits)

So far considered only non associative tasks. In these tasks the learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is non-stationary. In a general reinforcement learning task there is more than one situation, and the goal is to learn a policy: a mapping from situations to the actions that are best in those situations.

An *associative search* task (contextual bandits) involves both trial-and-error learning to *search* for the best actions, and *association* of these actions with the situations in which they are best. Associative search tasks are intermediate between k-armed bandit problem and full reinforcement learning problem. Full reinforcement learning problem when each action affects immediate reward and *next situation*.

2.10 Summary

UCB methods choose deterministically but achieve exploration by subtly favouring at each step the actions that have so far received fewer samples.

Gradient bandit algorithms estimate not action values, but action preferences, and favour them ore preferred actions in a graded, probabilistic manner using a soft-max distribution.

Chapter 3: Finite Markov Decision Processes

The problem of finite MDPs involves evaluative feedback, as in bandits, but also an associative aspect — choosing different actions in different situations. They are a classical formalisation of seqeunential decisions making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards.

$$F = ma$$

3.1 The Agent-Environment Interface

MDPs are meant to be a straightforward framing of the problem of learning from interaction to acheive a goal.

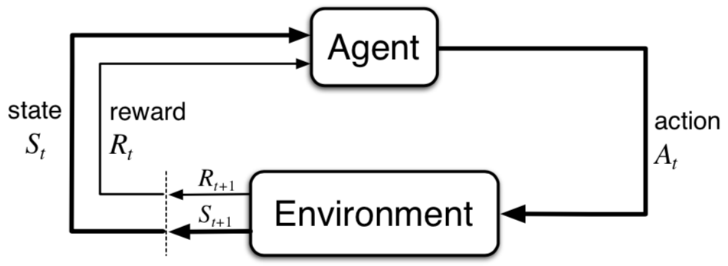


Figure 3.1: The agent–environment interaction in a Markov decision process.

The MDP and agent together give rise to a sequence or *trajectory* like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots.$$

In a *finite* MDP, the sets of states, actions, and rewards ($\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$) all have a finite number of elements.

Given $s' \in \mathcal{S}$ and $r \in \mathcal{R}$:

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\},$$

for all $s' \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(\int)$.

The function p defines the *dynamics* of the MDP.

State-transition probabilities, $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$\begin{aligned} p(s'|s,a) &\doteq Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} \\ &= \sum_{r \in \mathcal{R}} p(s', r|s, a). \end{aligned}$$

Expected rewards for state-action pairs, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$\begin{aligned} r(s,a) &\doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] \\ &= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a). \end{aligned}$$

Expected rewards for state-action-next-state triples, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$\begin{aligned} r(s,a,s') &\doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] \\ &= \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)} \end{aligned}$$

Sensory receptors of an agent should be considered part of the environment rather than part of the agent. Rewards, too, are computed inside the artificial learning system but are considered external to the agent.

Anything that cannot be changed arbitrarily by the agent is considered to be outside of it and this part of its environment. The agent-environment boundary represents the limit of the agent's *absolute control*, not of its knowledge

3.5 Policies and Value Functions

State-value function for policy π (value function of a state s under a policy π):

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_T|S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \text{ for all } s \in \mathcal{S}, \end{aligned} \tag{3.12}$$

is the expected return when starting in s and following π thereafter. The value of the terminal state, if any, is always zero.

$\mathbb{E}_\pi[\cdot]$: the expected value of a random variable given that the agent follows policy π

Action-value function for policy π (value of taking action a in state s under a policy π):

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi[G_t|S_t=s, A_t=a] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t=s, A_t=a\right], \end{aligned} \tag{3.13}$$

is the expected return starting from s, taking the action a, and thereafter following policy π.

Bellman equation for v_π:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_T|S_t=s\right] \\ &= \sum_a \pi\left(a|s\right)\sum_{s',r} p\left(s',r|s,a\right)\left[r+\gamma v_\pi(s')\right], \text{ for all } s\in\mathcal{S}, \end{aligned}$$

the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

Example 3.6: Golf

- Reward: -1 for each stroke until we hit the ball into the hole.
- State: location of the ball.
- Value of a state: negative of the number of strokes to the hole from that location.
- Actions: how we aim and swing at the ball and which club we select.

Exercise 3.17

Bellman equation for action values, q_π:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi\left[G_t|S_t=s, A_t=a\right] \\ &= \sum_{s',r} p(s',r|s,a)\left[r+\gamma\sum_{a'}\pi(a'|s')q_\pi(s',a')\right] \end{aligned}$$

Exercise 3.18

Value of a state:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[q_\pi(s,a)] \\ &= \sum_a \pi(a|s)q_\pi(s,a) \end{aligned}$$

depends on the values of the actions possible in that state and how likely each action is to be taken under the current policy.

Exercise 3.19

Value of an action (state-action pair):

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \end{aligned}$$

depends on the expected next reward and the expected sum of the remaining rewards.

3.6 Optimal Policies and Optimal Value Functions

$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \; \forall \; s \in \mathcal{S}$

Optimal policy, π_*: the one or more policy that is better than or equal to all other policies.

Optimal state-value function:

$$v_*(s) \doteq \max_\pi v_\pi(s), \quad \forall \; s \in \mathcal{S}, \tag{3.15}$$

is the shared state-value function of the optimal policies.

Optimal action-value function:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \quad \forall \; s \in \mathcal{S} \text{ and } a \in \mathcal{A}, \tag{3.16}$$

is also shared by the optimal policies.

For the state-action pair (s,a), q_* gives the expected return for taking action a in state s and thereafter following an optimal policy. Hence,

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]. \tag{3.17}$$

Bellman optimality equation (the Bellman equation for v_*):

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = t, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s,a)[r + \gamma v_*(s')]. \end{aligned}$$

Because v_* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (3.14). The equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state.

Bellman optimality equation for q_*:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \middle| S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r|s,a) \left[r + \gamma \max_{a'} q_*(s', a')\right] \end{aligned}$$

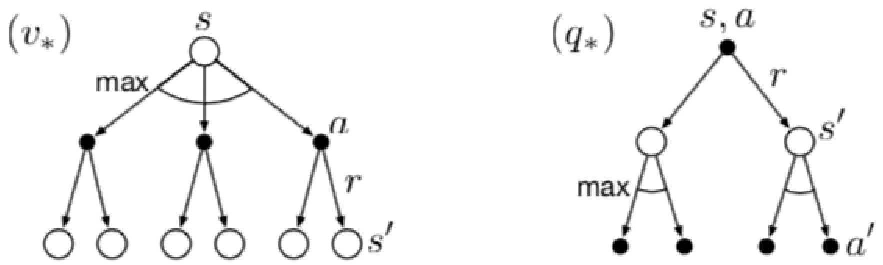


Figure 3.4: Backup diagrams for v_* and q_*

For finite MDPs, the Bellman optimality equation for v_* (3.19) has a unique solution. The bellman optimality equation is actually a system of equations, one for each state, so if there are n states, then there are n equations in n unknowns. If the dynamics p of the environment are known, then in principle one can solve this system of equations for v_*.

Once one has v_*, it is relatively easy to determine an optimal policy. Any policy that is *greedy* with respect to the optimal evlauaton function v_* is an optimal policy. The agent chooses optimal actions by, for any state s, finding any action that maximises $q_*(s,a)$.

Explictly solving the Bellman optimality equations provides one route to finding an optimal policy, and thus to solving the reiforcement learning problem. It is akin to an exhaustive search. This solution relies on at least three assumptions that are rarely true in practice:

1. the dynamics of the environment are accurately known
2. computational resources are sufficient to complete the calculation
3. the states have the Markov property

For backgammon, the first and third assumptions present no problems but the third is an impediment. The game has about 10^{20} states.

Many different decision-making methods can be viewed as ways of approximately solving the Bellman optimality equation

3.7 Optimality and Approximation

In practice, an agent rarely learns an optimal policy. Computational costs are too extreme.

Memory is an important constraint. In small tasks, it is possible to approximate with tables with one entry for each state, *tabular* case, with corresponding tabular methods.

The online nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states

4 Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).

We usually assume that the environment is a finite MDP. A common way of obtaining approximate solutions for tasks with continuous state and actions is to quantise the state and action spaces adn then apply finite-state DP methods.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organise and structure the search for good policies.

4.1 Policy Evaluation (Prediction)

Policy evaluation/The prediction problem: how to compute the state-value function v_π for an arbitrary policy π.

If the environment's dynamics are completely known, then (3.14) is a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns (the $v_\pi(s), s \in \mathcal{S}$). Iterative solution methods are most suitable.

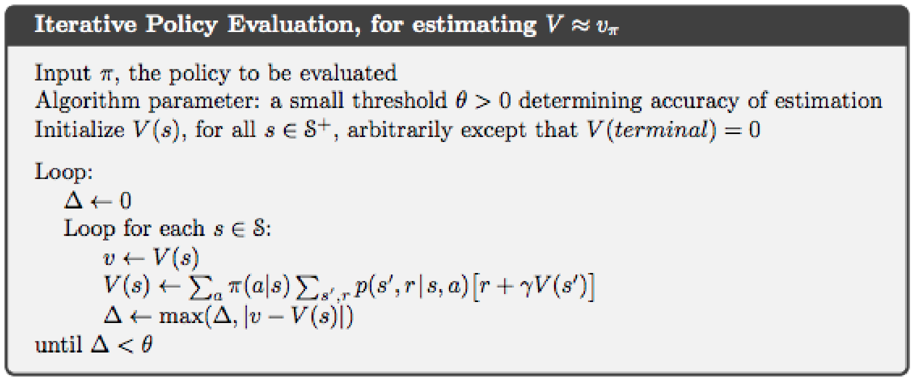
Iterative policy evaluation:

1. Initial approximation, v_0, is chosen arbitrarily.
2. Each successive approximation is obtained by using the Bellman equation for v_π as an update rule.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \text{ for all } s \in \mathcal{S}. \end{aligned}$$

Expected update: To produce each successive approximation, v_{k+1} from v_k, iterative policy evaluation applies the same operation to each state s: it replaces the old value of s with a new value obtained from the old values of the successor states of s, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated.

To write a sequential computer program to implement iterative policy evaluation as given by (4.5) you would have to use two arrays, one for the old values, $v_k(s)$, and one for the new values, $v_{k+1}(s)$.



Iterative Policy Evaluation, for estimating $V \approx v_\pi$

```
Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
    $\Delta \leftarrow 0$
    Loop for each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$
```

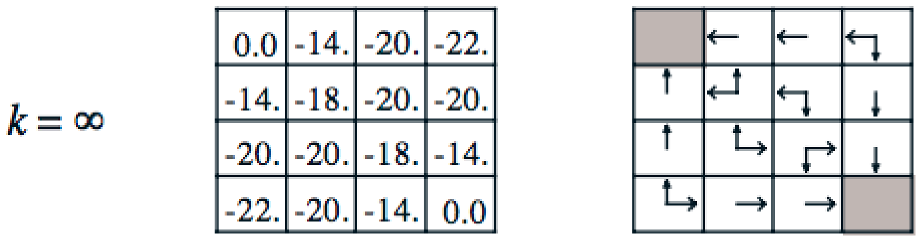
4.2 Policy Improvement

Suppose we have determined the value function v_π for an arbitrary deterministic policy π. We know how good it is to follow the current policy from s ($v_\pi(s)$), but would it be better or worse to change to a new policy?

If $q_\pi(s, a)$ is greater than $v_\pi(s)$—that is, if it is better to select a once in s and thereafter follow π than it would be to follow π all the time—then one would expect it to be better still to select a every time s is encountered, and that the new policy would in fact be a better one overall.

Policy improvement theorem: see page 78

Policy improvement: the process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy



0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

The original policy, π, is the equiprobable random policy, and the new policy, π', is greedy with respect to v_π. The value function v_π is shown in the bottom-left diagram and the set of possible π' is shown in the bottom-right diagram. Any apportionment of probability among actions in states with multiple arrows in the π' diagram is permitted. By inspection, the state values of the new policy π' can be seen to be either -1, -2, or -3.

4.3 Policy Iteration

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

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

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

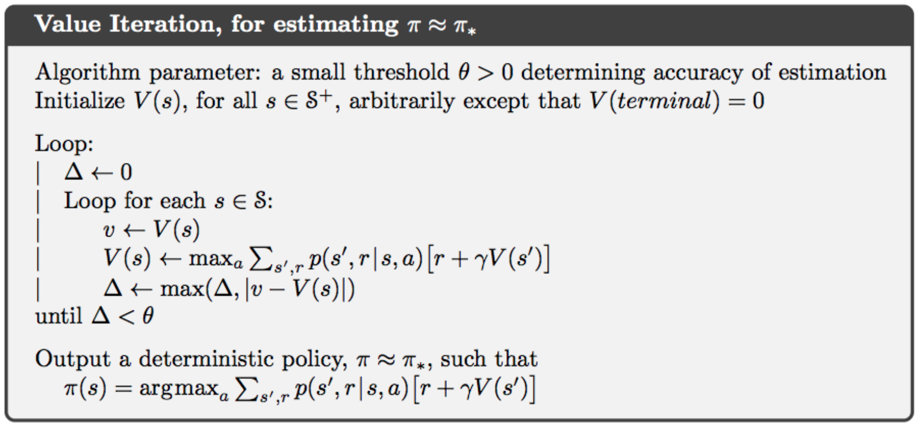
```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ true
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
      If old-action $\neq \pi(s)$, then policy-stable $\leftarrow$ false
   If policy-stable, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

4.4 Value Iteration

$$\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_k(s')\right], \qquad \text{for all } s \in \mathcal{S}
\end{aligned}$$



Value Iteration, for estimating $\pi \approx \pi_*$

```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
|  $\Delta \leftarrow 0$
|  Loop for each $s \in \mathcal{S}$:
|       $v \leftarrow V(s)$
|       $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
|       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
```

4.5 Asynchronous Dynamic Programming

DP methods involve operations over the entire state set of the MDP. If the state set is very large, like in Backgammon, then each set is expensive.

Asynchronous DP algorithms update the value of states in any order whatsoever, using whatever values of other states happen to be available.

4.6 Generalised Policy Iteration

Policy iteration consists of making the value function consistent with the current policy (policy evaluation), and the other making the policy greedy with respect to the current value function (policy improvement). The result is convergence to the optimal value function and an optimal policy.

Generalised policy iteration (GPI) is letting policy-evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes. The policy always improves with respect to the value function and the value function is always driven toward the value function for the policy. Both processes stabilise only when a policy has been found that is greedy with respect to its own evaluation function. This implies that the Bellman optimality equation (4.1) holds, and that the policy and value function are optimal

5 Monte Carlo Methods

No assumption of complete knowledge of the environment. Monte Carlo methods require only *experience*.

Ways of solving the reinforcement learning problem based on averaging sample returns. Assume experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. Only on the completion of an episode are value estimates and policies changed.

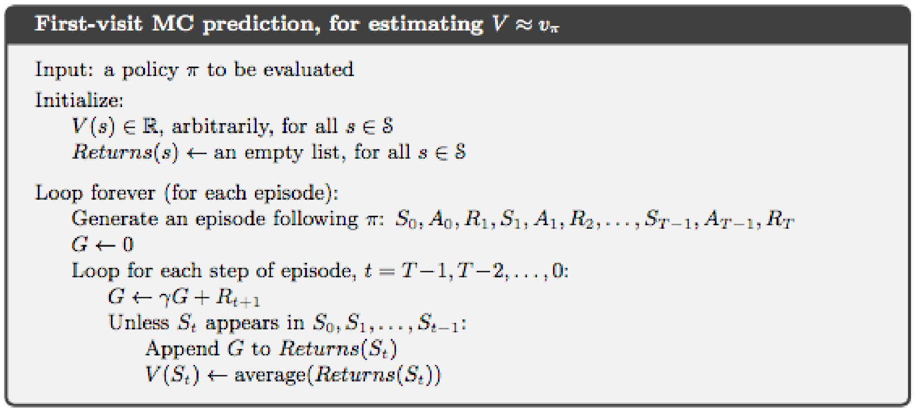
Because all the action selections are undergoing learning, the problem becomes nonstationary from the point of view of the earlier state.

5.1 Monte Carlo Prediction

Consider Monte Carlo methods for learning the state-value function for a given policy. To estimate the value of a state from experience is to average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value.

Visit to s: each occurrence of state s in an episode.

First-visit MC method: estimates $v_\pi(s)$ as the average of the returns following first visits to s.



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

```
Input: a policy $\pi$ 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 $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 MC method: averages the returns following all visits to s.

Both methods converge to $v_\pi(s)$ as the number of visits (or first visits) to s goes to infinity.

For MC estimation, the root of a backup diagram is a state node, and below it is the entire trajectory of transitions along a particular single episode, ending at the terminal state. Whereas the DP diagram shows all possible transitions, the Monte Carlo diagram shows only those sampled on the one episode. Whereas the DP diagram includes only one-step transitions,the Monte Carlo diagram goes all the way to the end of the episode.

Estimates for each state are independent.

The three advantages of Monte Carlo methods over DP methods:

- The computational expense of estimating the value of a single state is independent of the number of states.
- The ability to learn from actual experience
- The ability to learn from simulated experience

5.2 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 of our primary goals for Monte Carlo methods is to estimate q_*. TO achieve this, we first consider the policy evaluation problem for action values.

The policy evaluation problem for action values is to estimate $q_\pi(s, a)$. The MC methods for this are essentially the same as just presented for state values, replacing state with state-action pair.

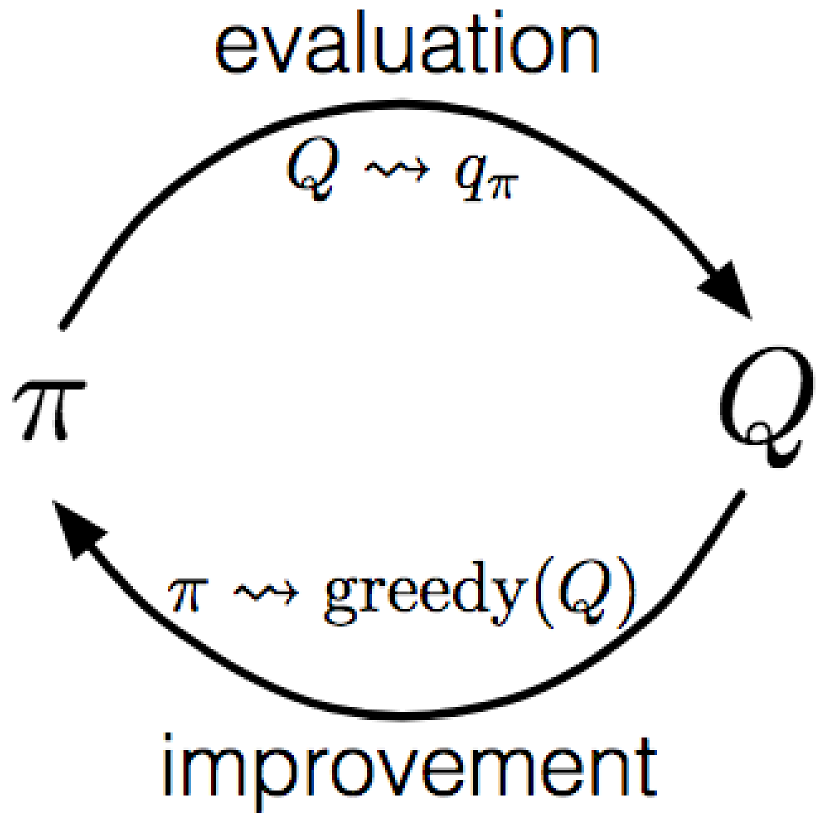
The complication is that many state-action pairs may never be visited. To compare alternatives we need to estimate the value of *all* the actions from each state.

This is the general problem of *maintaining 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 at the start. We call this the assumption of *exploring starts.*

5.3 Monte Carlo Control

Control: to approximate optimal policies

Procees according to the idea of generalised policy iteration (GPI)


evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Monte Carlo version of classical policy iteration: perform alternating 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.

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

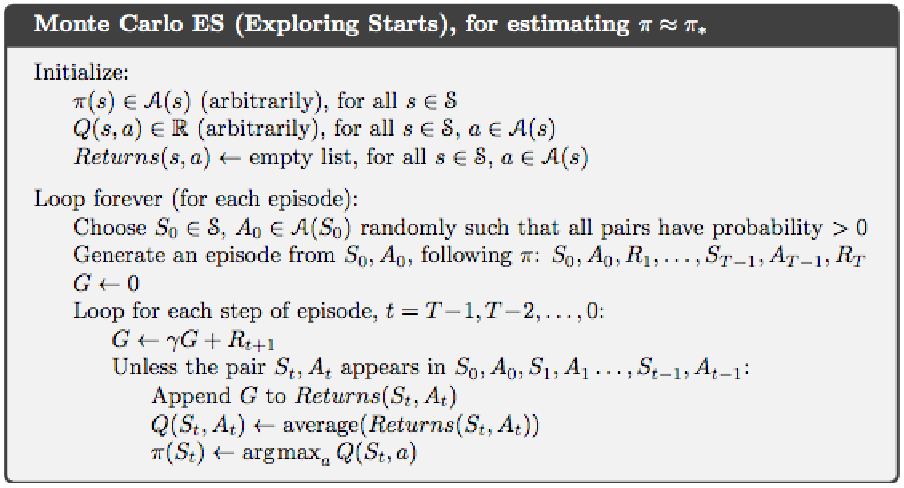
Assuming that we observe an infinite number of episodes and the episodes are generated with exploring starts, the MC methods will compute each q_{π_k} exactly, for arbitrary π_k.

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:

$$\pi(s) \doteq argmax_a q(s, a)$$

Policy improvement can be done by constructing each π_{k+1} as the greedy policy with respect to q_{π_k}. The policy improvement theorem applies to π_k and π_{k+1}.

Monte Carlo methods can be used to find optimal policiesgiven only sample episodes and no other knowledge of the environment's dynamics.



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 $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$
```

See Exercise 5.4 for improved version.

5.4 Monte Carlo Control without Exploring Starts

Assumption of exploring starts is unlikely. The only general way to ensure that all actions are selected infinitely often is for the agent ot continue to select them. There are two approaches:

On-policy methods: evaluate or improve the policy that is used to make decisions

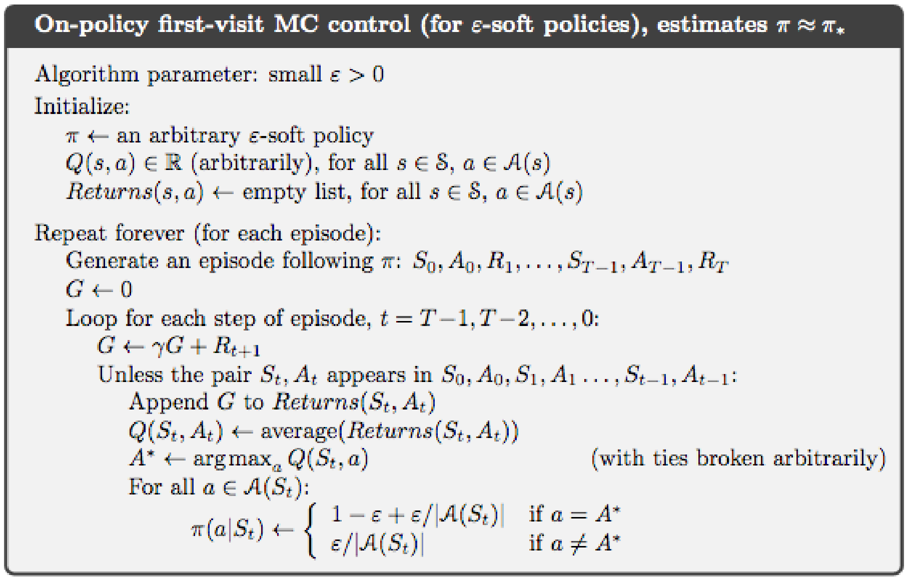
Off-policy methods evaluate or improve a policy different from that used to generate the data.

The Monte Carlo ES method is an example of an on-policy method.

Soft policy: $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}(s)$

ϵ-soft policy: $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$ for all states and actions, for some $\epsilon > 0$

The idea of on-policy Monte Carlo control is still that of GPI. In our on-policy methods we will move it only to an ϵ-greedy policy, since GPI does not require that the policy be taken all the way to a greedy policy.



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

```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-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 $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

Improvement is assured by the policy improvement theorem.

Policy iteration works for ϵ-soft policies.

5.5 Off-policy Prediction via Importance Sampling

All learning control methods face a dilemma: they seek to learn action values conditional on subsequent *optimal* behaviour, but they need to behave non-optimally in order to explore all actions (to *find* the optimal actions). The on-policy approach is a compromise $mdash; 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 behaviour.

Target policy: the policy being learned

Behaviour policy: the policy used to generate behaviour

Learning is from data "off" the target policy, and the overall process is termed *off-policy* learning.

Suppose we wish to estimate v_π or q_π, but all we have are episodes following another policy b, where $b \neq \pi$. In this case, π is the target policy, b is the behaviour policy, and both policies are considered fixed adn given.

In order to use episodes from b to estimate values for π, we require that every action taken under π is also taken, at least occasionally, under b. That is, we require and assume coverage.

Coverage: $\pi(a|s) > 0$ implies $b(a|s) > 0$

Off-policy methods utilise *importance sampling*

Importance sampling: technique for estimating expected values under one distribution given samples from another.

Apply importance sampling to off-policy learning by weighting returns according to the relative probability of their trajectories occurring under the target and behaviour policies, called the *importance-sampling ratio.* Given a starting state S_t, the probabiltiy of the subsequent state-action trajectory, $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$, occurring under any policy π is

$$\begin{aligned} Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} &= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \ldots p(S_T|S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k), \end{aligned}$$

where p is the state-transition probability function defined by (3.4)

The importance-sampling ratio is

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

The importance-sampling ratio depends only on the two policies and the sequence, not on the MDP.

The ratio $\rho_{t:T-1}$ transforms the returns to have the right expected value:

$$\mathbb{E}[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s).$$

Monte Carl algorithm that averages returns from a batch of observed episodes following policy b to estimate $v_\pi(s)$

For an every-visit method, let $\mathcal{T}(s)$ denote the set of all time steps in which state s is visited.

For a first-visit method, let $\mathcal{T}(s)$ denote the set of all time steps that were first visits to s within their episodes.

Let $T(t)$ denote the first time of termination following time t.

Let G_t denote the return after t up through $T(t)$.

Then $\{G_t\}_{t\in\mathcal{T}(s)}$ are the returns that pertain to state s, and $\{\rho_{t:T(t)-1}\}_{t\in\mathcal{T}(s)}$ are the corresponding importance-sampling ratios.

Ordinary importance sampling: to estimate $v_\pi(s)$, scale the returns by the ratios and average the results:

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|} \tag{5.5}$$

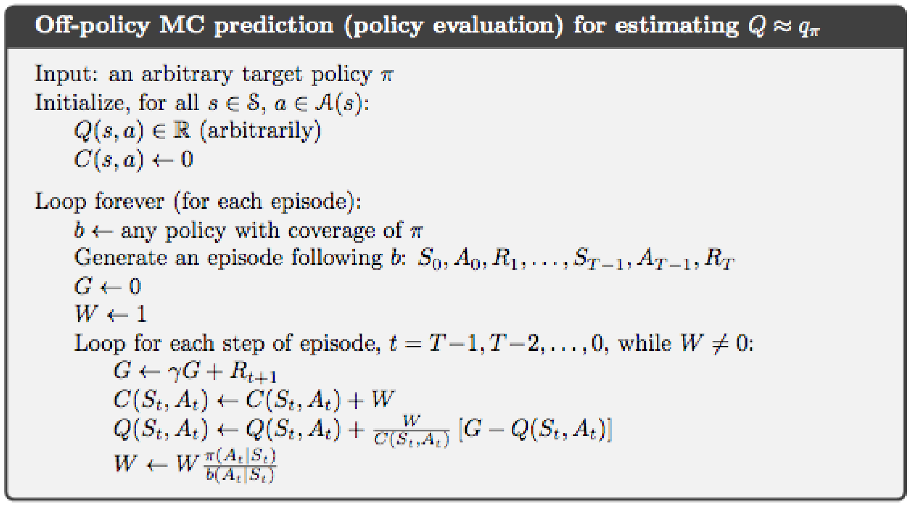
Weighted importance sampling: uses a *weighted* average

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}} \tag{5.6}$$

Ordinary importance sampling is unbiased whereas weighted importance sampling is biased (though the bias converges asymptotically to zero).

5.6 Incremental Implementation

Monte Carlo prediction methods can be implemented incrementally, on an episode-by-episode basis. In Monte Carlo methods we average *returns*.



Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

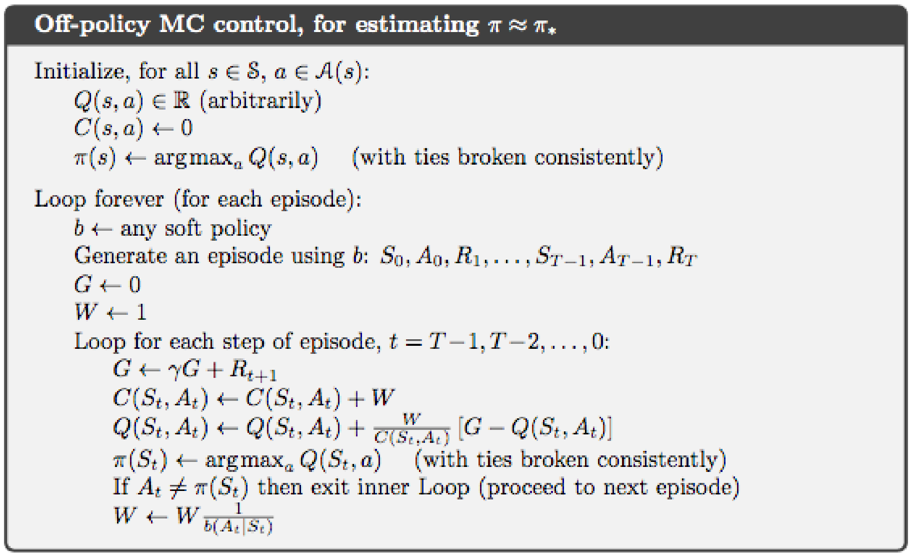
```
Input: an arbitrary target policy $\pi$
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of $\pi$
    Generate an episode following $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
```

5.7 Off-policy Monte Carlo Control

An advantage is that the target policy may be greedy, while the behaviour policy can continue to sample all possible actions.

These methods follow the behaviour policy while learning about and improving the target policy.



Off-policy MC control, for estimating $\pi \approx \pi_*$

```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \in \mathbb{R}$ (arbitrarily)
    $C(s, a) \leftarrow 0$
    $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$    (with ties broken consistently)
Loop forever (for each episode):
    $b \leftarrow$ any soft policy
    Generate an episode using $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$    (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
```

6 Temporal-Difference Learning

Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome.

For the *control* problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of generalised policy iteration (GPI).

6.1 TD Prediction

constant-α MC: a every-visit Monte Carlo method suitable for nonstationary environments is

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right], \tag{6.1}$$

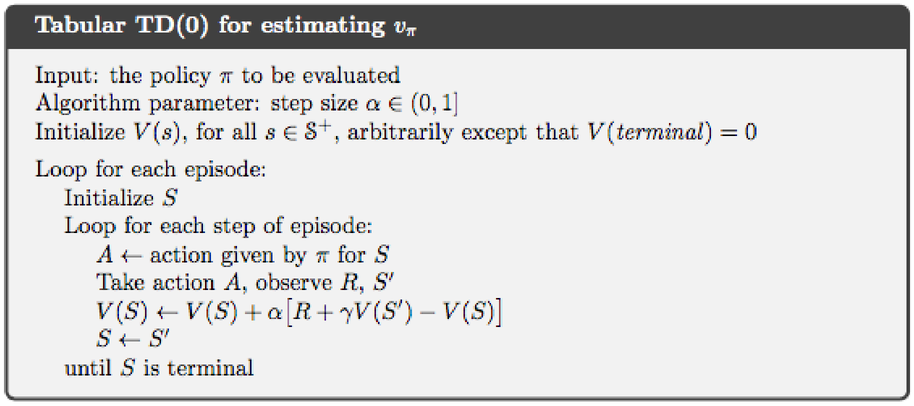
where G_t is the actual return following time t.

TD(0)/one-step TD: TD methods need to wait only until the next time step to determine the increment to $V(S_t)$. At time $t+1$ they form a target ad make a update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. THe simplest TD method makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{6.2}$$

on transition to S_{t+1} and receiving R_{t+1}.

The target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$.



Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    $A \leftarrow$ action given by $\pi$ for $S$
    Take action $A$, observe $R$, $S'$
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until $S$ is terminal
```

The TD target is an estimate because it samples the expected values *and* it uses the current estimate V instead of the true v_π.

TD error $\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

TD error at each time is the error in the estimate *made at that time.* δ_t is the error in $V(S_t)$, available at time $t+1$. If the array V does not change during the episode (as it does not in Monte Carlo methods), then the Monte Carlo error can be written as a sum of TD errors.

8.9 Heuristic Search

For each state encountered, a large tree of possible continuations is considered. The approximate value function is applied to the leaf nodes and then backed up toward the current state at the root. Then, the best is chosen as the current action.

In conventional heuristic search, no effort is made to save the backed-up values by changing the approximate value function. Heuristic search can be viewed as an extension of the idea of a greedy policy beyond a single step.

The point of searching deeper than one step is to obtain better action selections. The deeper the search, the more computation is required, usually resulting in a slower response time.

> "[Tesauro's TD-Gammon] used TD learning to learn an afterstate value function through many games of self-play, using a form of heuristic search to make its moves. As a model, TD-Gammon used a priori knowledge of the probabilities of dice rolls and the assumption that the opponent always selected the actions that TD-Gammon rated as best for it. Tesauro found that the deeper the heuristic search, the better the moves made by TD-Gammon, but the longer it took to make each move.