Temporal Difference Methods, Off-Policy Methods

Milan Straka

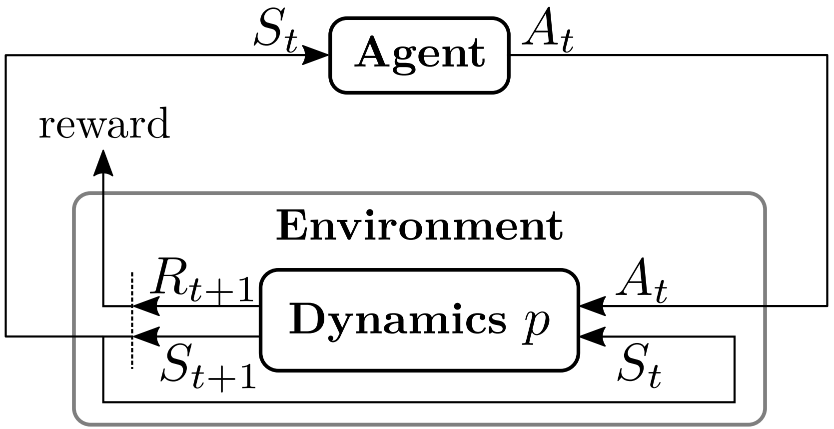
October 17, 2022

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research, Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

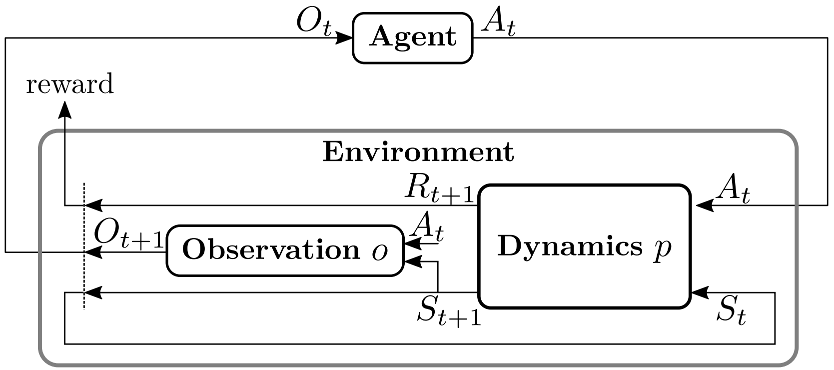
Recall that a **Markov decision process** (MDP) is a quadruple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a **reward** $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a **discount factor**.



Partially observable Markov decision process extends the Markov decision process to a sextuple $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{O}, o)$, where in addition to an MDP,

- $\mathcal{O}$ is a set of observations,
- $o(O_{t+1} | S_{t+1}, A_t)$ is an observation model, where observation O_t is used as agent input instead of the state S_t.



Refresh – Policies and Value Functions

A **policy** π computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action a in state s.

To evaluate a quality of a policy, we define **value function** $v_\pi(s)$, or **state-value function**, as

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t \middle| S_t = s\right] = \mathbb{E}_\pi\left[\sum\nolimits_{k=0}^\infty \gamma^k R_{t+k+1} \middle| S_t = s\right].$$

An **action-value function** for a policy π is defined analogously as

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t \middle| S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum\nolimits_{k=0}^\infty \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right].$$

Optimal state-value function is defined as $v_*(s) \stackrel{\text{def}}{=} \max_\pi v_\pi(s)$, analogously optimal action-value function is defined as $q_*(s, a) \stackrel{\text{def}}{=} \max_\pi q_\pi(s, a)$.

Any policy π_* with $v_{\pi_*} = v_*$ is called an **optimal policy**.

Refresh – Value Iteration

Optimal value function can be computed by repetitive application of Bellman optimality equation:

$$
\begin{aligned}
v_0(s) &\leftarrow 0 \\
v_{k+1}(s) &\leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \middle| S_t = s, A_t = a\right] = B v_k.
\end{aligned}
$$

Converges for finite-horizon tasks or when discount factor $\gamma < 1$.

Refresh – Policy Iteration Algorithm

Policy iteration consists of repeatedly performing policy evaluation and policy improvement:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}.$$

The result is a sequence of monotonically improving policies π_i. Note that when $\pi' = \pi$, also $v_{\pi'} = v_\pi$, which means Bellman optimality equation is fulfilled and both v_π and π are optimal.

Considering that there is only a finite number of policies, the optimal policy and optimal value function can be computed in finite time (contrary to value iteration, where the convergence is only asymptotic).

Note that when evaluating policy π_{k+1}, we usually start with v_{π_k}, which is assumed to be a good approximation to $v_{\pi_{k+1}}$.

Refresh – Generalized Policy Iteration

Generalized Policy Evaluation is a general idea of interleaving policy evaluation and policy improvement at various granularity.

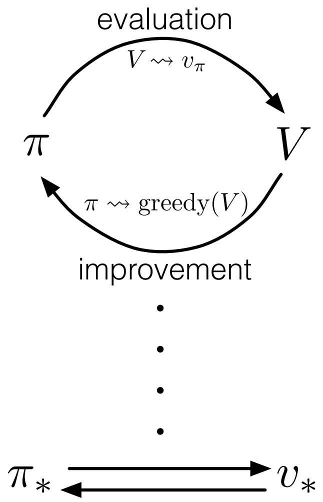


Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

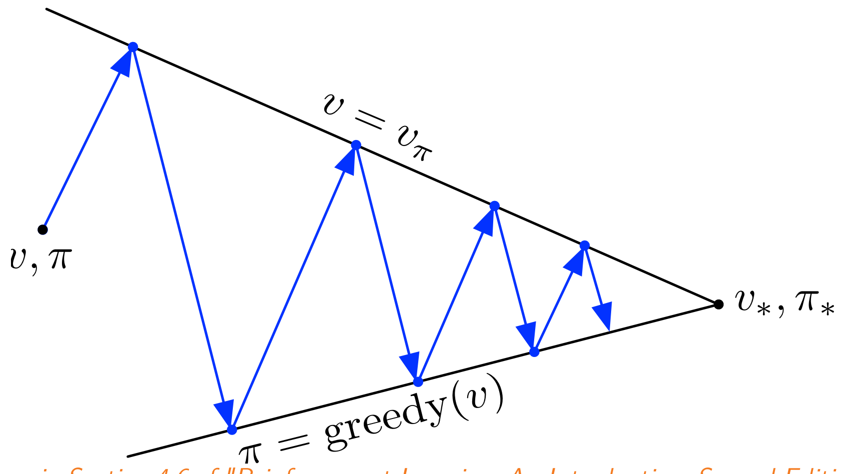


Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

If both processes stabilize, we know we have obtained optimal policy.

Refresh – Monte Carlo Methods

Monte Carlo methods are based on estimating returns from complete episodes. Furthermore, if the model (of the environment) is not known, we need to estimate returns for the action-value function q instead of v.

We can formulate Monte Carlo methods in the generalized policy improvement framework. Keeping estimated returns for the action-value function, we perform policy evaluation by sampling one episode according to current policy. We then update the action-value function by averaging over the observed returns, including the currently sampled episode.

We considered two variants of exploration:

- exploring starts
- ε-soft policies

On-policy every-visit Monte Carlo for ε-soft Policies

Algorithm parameter: small $\varepsilon > 0$

Initialize $Q(s, a) \in \mathbb{R}$ arbitrarily (usually to 0), for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $C(s, a) \in \mathbb{Z}$ to 0, for all $s \in \mathcal{S}, a \in \mathcal{A}$

Repeat forever (for each episode):

- Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, by generating actions as follows:
  - With probability ε, generate a random uniform action
  - Otherwise, set $A_t \stackrel{\text{def}}{=} \arg\max_a Q(S_t, a)$
- $G \leftarrow 0$
- For each $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{T+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}(G - Q(S_t, A_t))$

The reason we estimate *action-value* function q is that the policy is defined as

$$\begin{aligned}\pi(s) &\stackrel{\text{def}}{=} \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \sum\nolimits_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]\end{aligned}$$

and the latter form might be impossible to evaluate if we do not have the model of the environment.

However, if the environment is known, it is often better to estimate returns only for states, because there can be substantially less states than state-action pairs.

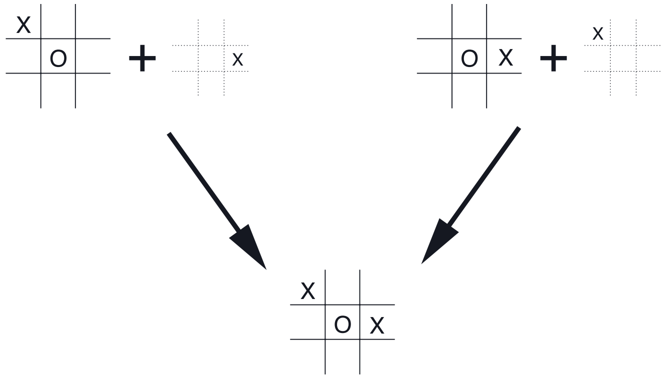


Figure from section 6.8 of "Reinforcement Learning: An Introduction, Second Edition".

Temporal-difference methods estimate action-value returns using one iteration of Bellman equation instead of complete episode return.

Compared to Monte Carlo method with constant learning rate α, which performs

$$v(S_t) \leftarrow v(S_t) + \alpha\big(G_t - v(S_t)\big),$$

the simplest temporal-difference method computes the following:

$$v(S_t) \leftarrow v(S_t) + \alpha\big(R_{t+1} + [\neg\mathrm{done}] \cdot \gamma v(S_{t+1}) - v(S_t)\big),$$

where $[\neg\mathrm{done}]$ has a value of 1 if the episode continues in the state S_{t+1}, and 0 otherwise.

We say TD methods are **bootstraping**, because they base their update on an existing (action-)value function estimate.

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Example 6.1 of "Reinforcement Learning: An Introduction, Second Edition".

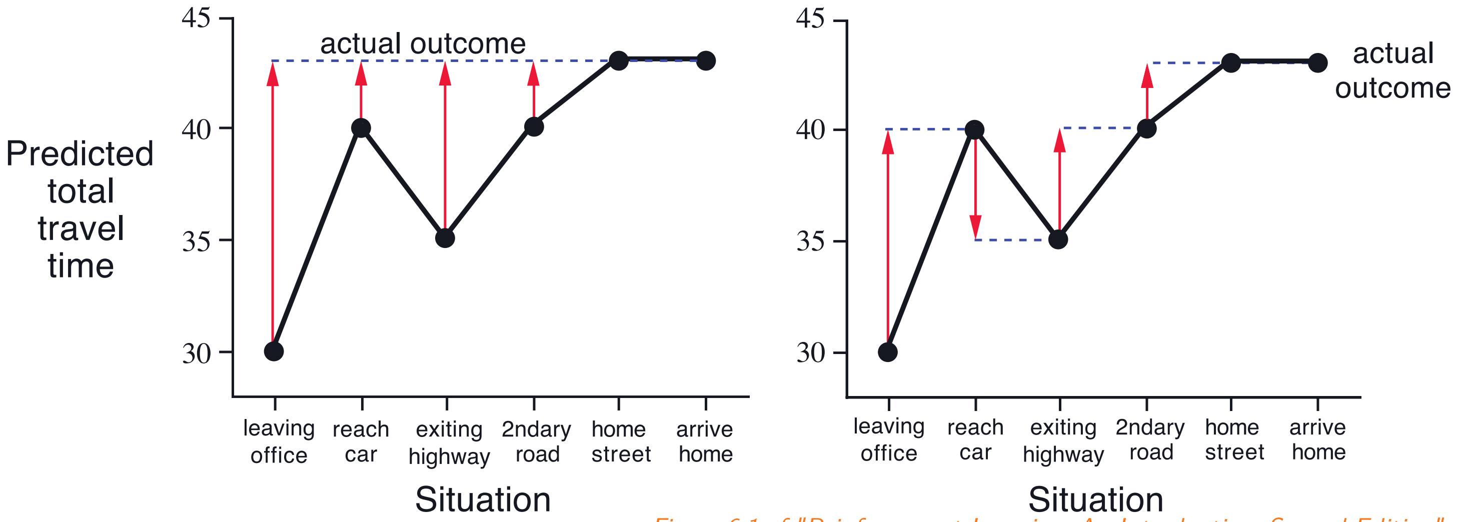


Figure 6.1 of "Reinforcement Learning: An Introduction, Second Edition".

TD Methods

An obvious advantage of TD methods compared to Monte Carlo is that they are naturally implemented in *online*, *fully incremental* fashion, while the Monte Carlo methods must wait until an episode ends, because only then the return is known.

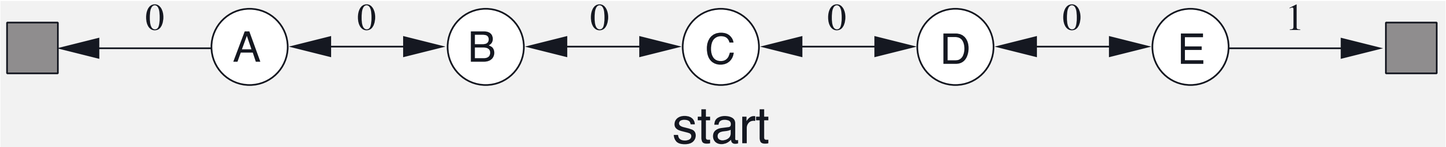
The possibility of immediate learning is useful for:

- continuous environments,
- environments with extremely large episodes,
- environments ending after some nontrivial goal is reached, requiring some coordinated strategy from the agent (i.e., it is improbable that random actions will reach it).

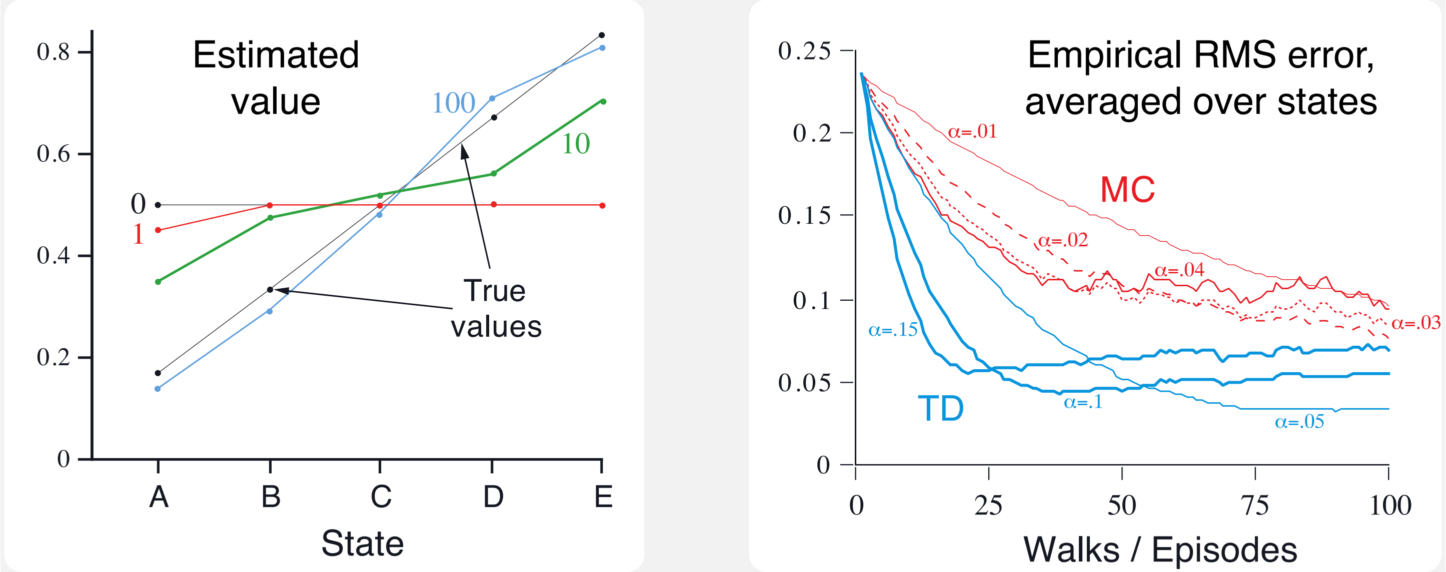
TD and MC Comparison

As with Monte Carlo methods, for a fixed policy π (i.e., the policy evaluation part of the algorithms), TD methods converge to v_π.

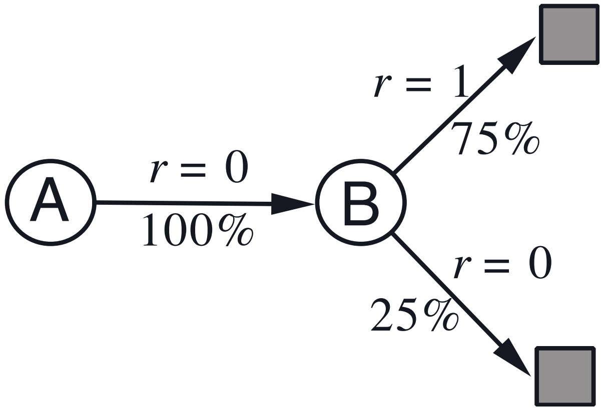
On stochastic tasks, TD methods usually converge to v_π faster than constant-α MC methods.



Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".



Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".



Example 6.4 of "Reinforcement Learning: An Introduction, Second Edition".

$\mathrm{A}, 0, \mathrm{B}, 0$	$\mathrm{B}, 1$
$\mathrm{B}, 1$	$\mathrm{B}, 1$
$\mathrm{B}, 1$	$\mathrm{B}, 1$
$\mathrm{B}, 1$	$\mathrm{B}, 0$

Example 6.4 of "Reinforcement Learning: An Introduction, Second Edition".

For state B, 6 out of 8 times return from B was 1 and 0 otherwise. Therefore, $v(B) = 3/4$.

- [TD] For state A, in all cases it transferred to B. Therefore, $v(A)$ could be $3/4$.
- [MC] For state A, in all cases it generated return 0. Therefore, $v(A)$ could be 0.

MC minimizes mean squared error on the returns from the training data, while TD finds the estimates that would be exactly correct for a maximum-likelihood estimate of the Markov process model (the estimated transition probability from s to t is the faction of observed transitions from s that went to t, and the corresponding reward is the average of the rewards observed on those transitions).

A straightforward application to the temporal-difference policy evaluation is Sarsa algorithm, which after generating $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ computes

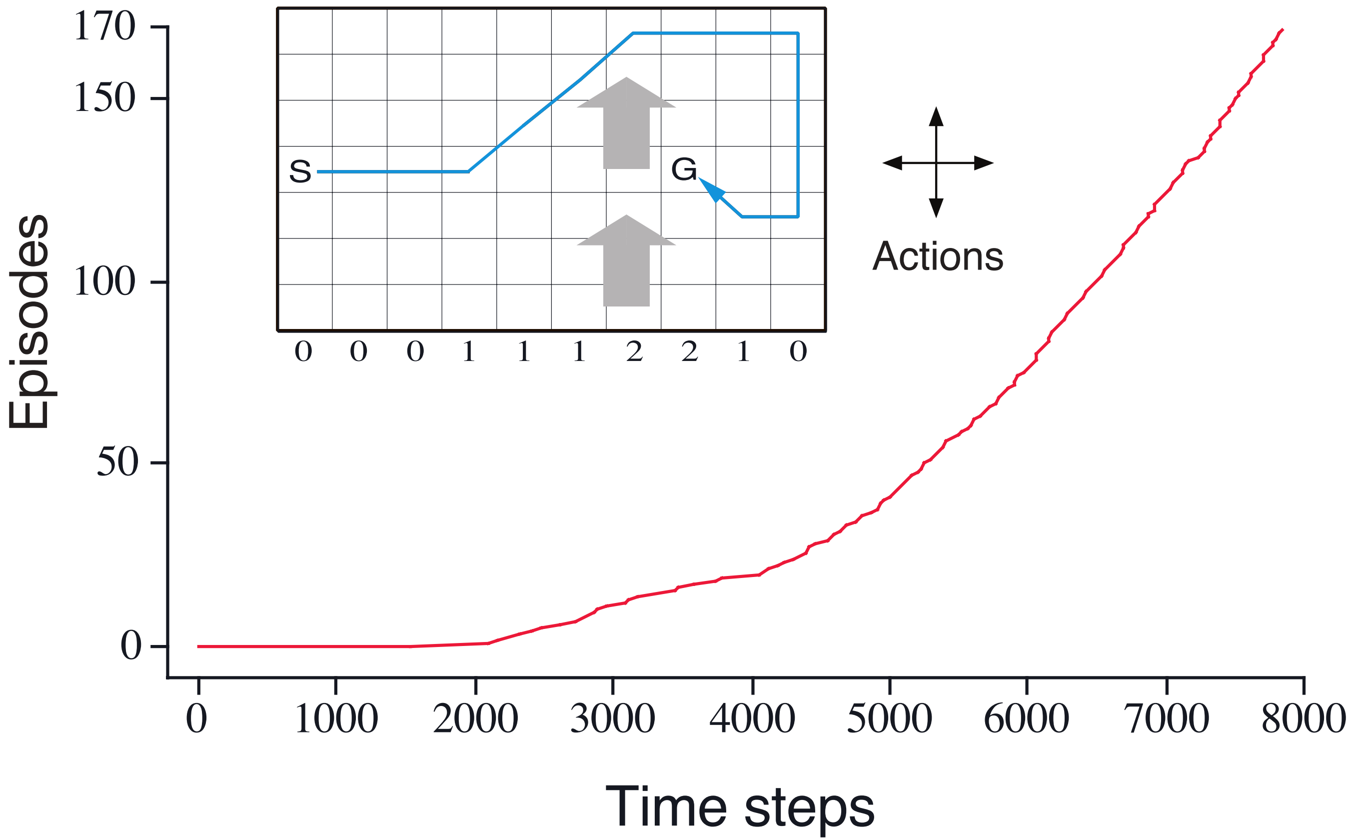
$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + [\neg\text{done}] \cdot \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\Big).$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Modification of Algorithm 6.4 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).



Example 6.5 of "Reinforcement Learning: An Introduction, Second Edition".

MC methods cannot be easily used, because an episode might not terminate if the current policy causes the agent to stay in the same state.

Q-learning

Q-learning was an important early breakthrough in reinforcement learning (Watkins, 1989).

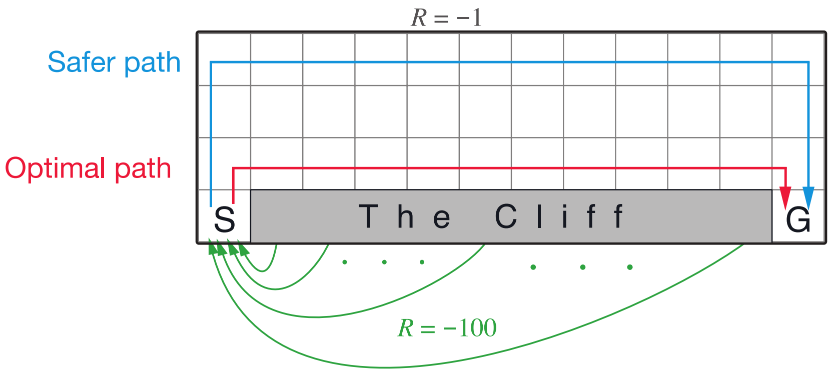
$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + [\neg\text{done}] \cdot \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)\Big).$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

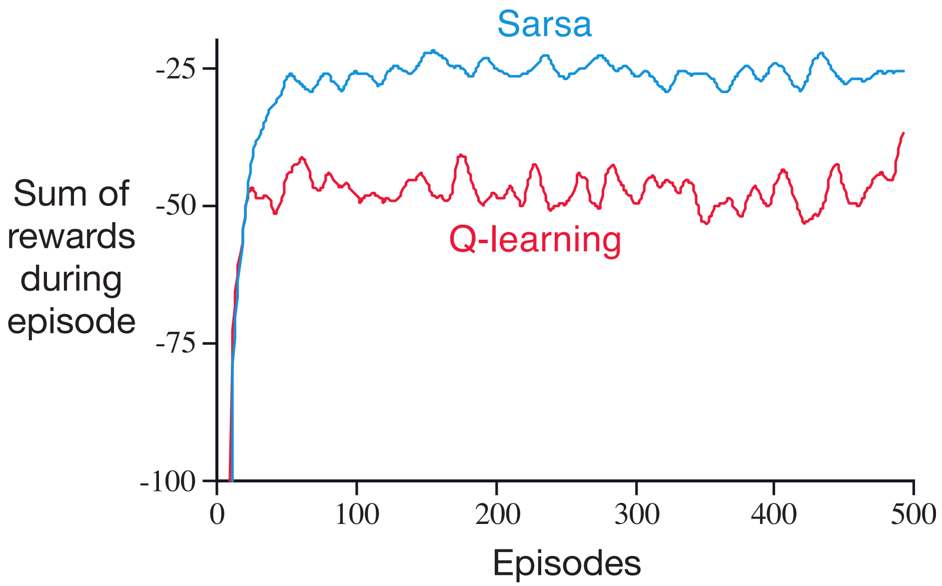
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Modification of Algorithm 6.5 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).



Example 6.6 of "Reinforcement Learning: An Introduction, Second Edition".



Example 6.6 of "Reinforcement Learning: An Introduction, Second Edition".

On-policy and Off-policy Methods

So far, most methods were **on-policy**. The same policy was used both for generating episodes and as a target of value function.

However, while the policy for generating episodes needs to be more exploratory, the target policy should capture optimal behaviour.

Generally, we can consider two policies:

- **behaviour** policy, usually b, is used to generate behaviour and can be more exploratory;
- **target** policy, usually π, is the policy being learned (ideally the optimal one).

When the behaviour and target policies differ, we talk about **off-policy** learning.

On-policy and Off-policy Methods

The off-policy methods are usually more complicated and slower to converge, but are able to process data generated by different policy than the target one.

The advantages are:

- can compute optimal non-stochastic (non-exploratory) policies;
- more exploratory behaviour;
- ability to process *expert trajectories*.

Off-policy Prediction

Consider prediction problem for off-policy case.

In order to use episodes from b to estimate values for π, we require that every action taken by π is also taken by b, i.e.,

$$\pi(a|s) > 0 \Rightarrow b(a|s) > 0.$$

Many off-policy methods utilize **importance sampling**, a general technique for estimating expected values of one distribution given samples from another distribution.

Assume that p and q are two distributions and let x_i be N samples of p. We can then estimate $\mathbb{E}_{\mathrm{x}\sim p}\big[f(x)\big]$ as $\frac{1}{N}\sum_i f(x_i)$.

In order to estimate $\mathbb{E}_{\mathrm{x}\sim q}\big[f(x)\big]$ using the samples x_i, we need to account for different probabilities of x_i under the two distributions. It is straightforward to verify that

$$\mathbb{E}_{\mathrm{x}\sim q}\big[f(x)\big] = \mathbb{E}_{\mathrm{x}\sim p}\left[\frac{q(x)}{p(x)}f(x)\right].$$

Therefore, we can estimate

$$\mathbb{E}_{\mathrm{x}\sim q}\big[f(x)\big] \approx \frac{1}{N}\sum_i \frac{q(x_i)}{p(x_i)}f(x_i),$$

with $q(x)/p(x)$ being the **relative probability** of x under the two distributions.

Both estimates mentioned on this slide are *unbiased*.

Given an initial state S_t and an episode $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$, the probability of this episode under a policy π is

$$\prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k).$$

Therefore, the relative probability of a trajectory under the target and behaviour policies is

$$\rho_t \stackrel{\text{def}}{=} \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k) p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

The ρ_t is usually called the **importance sampling ratio** or *relative probability*.

Therefore, if G_t is a return of episode generated according to b, we can estimate

$$v_\pi(S_t) = \mathbb{E}_b[\rho_t G_t].$$

Let $\mathcal{T}(s)$ be a set of times when we visited state s. Given episodes sampled according to b, we can estimate

$$v_\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t G_t}{|\mathcal{T}(s)|}.$$

Such simple average is called **ordinary importance sampling**. It is unbiased, but can have very high variance.

An alternative is **weighted importance sampling**, where we compute weighted average as

$$v_\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t}.$$

Weighted importance sampling is biased (with bias asymptotically converging to zero, i.e., a *consistent* estimate), but has smaller variance.

Off-policy Multi-armed Bandits

As a simple example, consider the 10-armed bandits from the first lecture, with single-step episodes.

Let the *behaviour policy* be a uniform policy, so the return is a reward of a randomly selected arm.

Let the *target policy* be a greedy policy always using action 3.

Assume that the first sample from the behaviour policy produced action 3 with reward R. Then

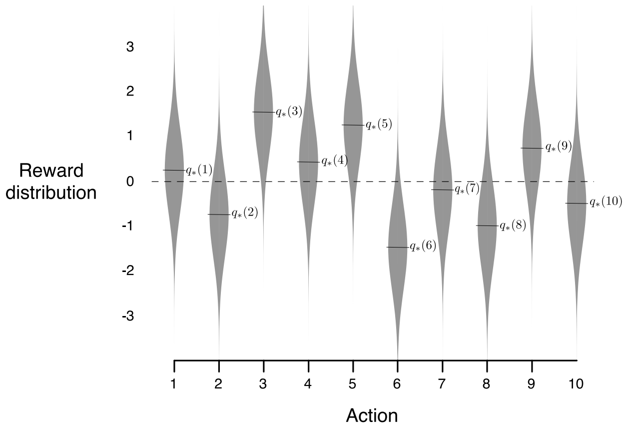


Figure 2.1 of "Reinforcement Learning: An Introduction, Second Edition".

- Ordinary importance sampling estimate the return for the target policy as

$$\frac{\pi(a)}{b(a)}R = \frac{1}{1/10}R = 10 \cdot R.$$

  The factor 10 is present, because the behaviour policy returns action 3 in 10% cases.

- Weighted importance sampling estimate the return for target policy as average of rewards for action 3.

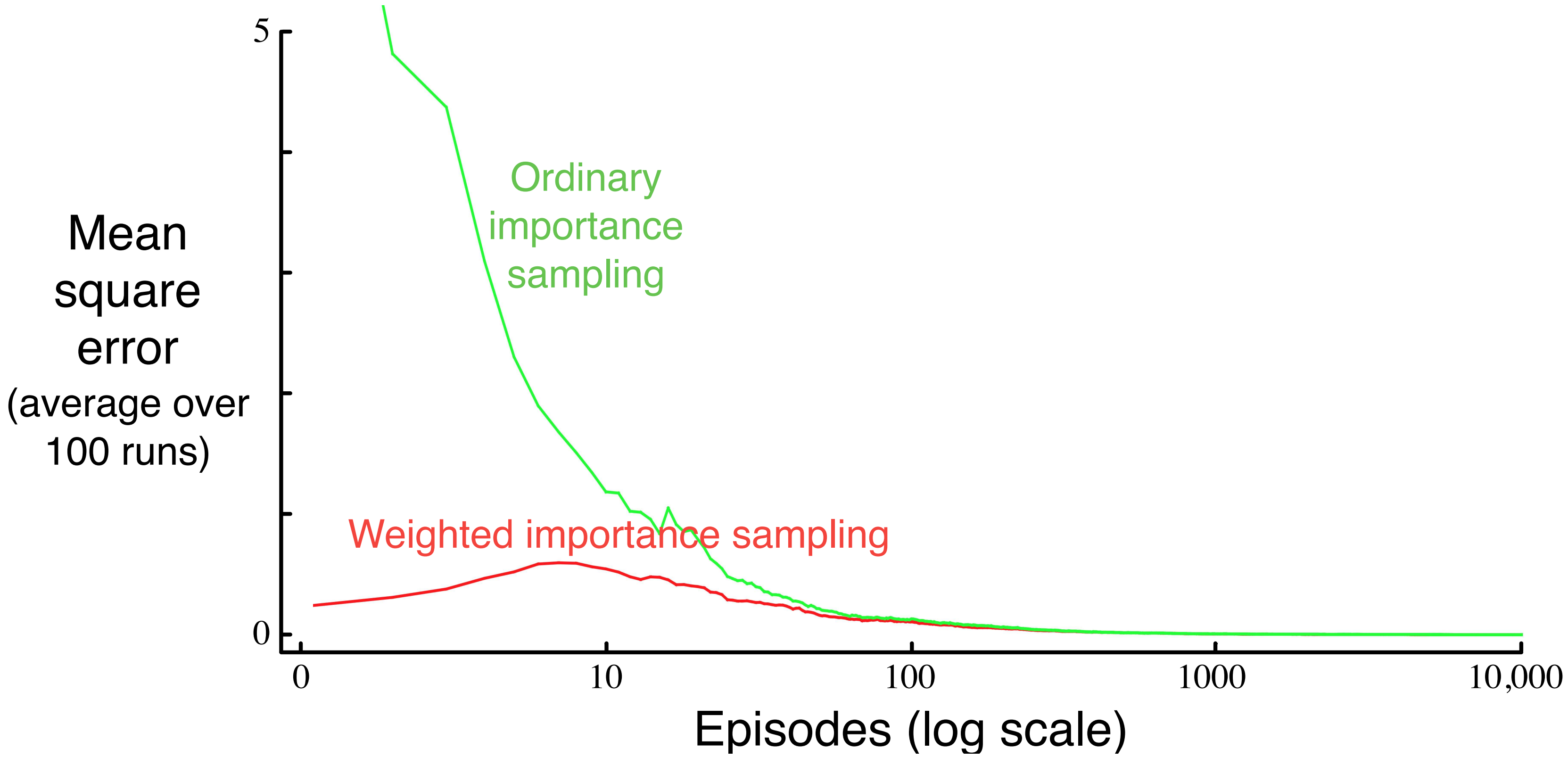


Figure 5.3 of "Reinforcement Learning: An Introduction, Second Edition".

Comparison of ordinary and weighted importance sampling on Blackjack. Given a state with sum of player's cards 13 and a usable ace, we estimate target policy of sticking only with a sum of 20 and 21, using uniform behaviour policy.

Off-policy Monte Carlo Policy Evaluation

We can compute weighted importance sampling similarly to the incremental implementation of Monte Carlo averaging.

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

Input: an arbitrary target policy π
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 π
    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 $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Algorithm 5.6 of "Reinforcement Learning: An Introduction, Second Edition".

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)
  $C(s, a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ (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, $t = T{-}1, T{-}2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Algorithm 5.7 of "Reinforcement Learning: An Introduction, Second Edition".

Expected Sarsa

The action A_{t+1} is a source of variance, providing correct estimate only *in expectation*.

We could improve the algorithm by considering all actions proportionally to their policy probability, obtaining Expected Sarsa algorithm:

$$\begin{aligned} q(S_t, A_t) &\leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + [\neg\textrm{done}]\cdot\gamma\mathbb{E}_\pi q(S_{t+1}, a) - q(S_t, A_t)\Big) \\ &\leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + [\neg\textrm{done}]\cdot\gamma\sum\nolimits_a \pi(a|S_{t+1})q(S_{t+1}, a) - q(S_t, A_t)\Big). \end{aligned}$$

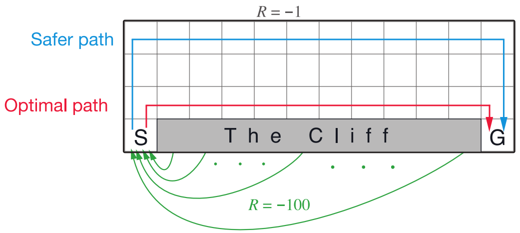
Compared to Sarsa, the expectation removes a source of variance and therefore usually performs better. However, the complexity of the algorithm increases and becomes dependent on number of actions $|\mathcal{A}|$.

Expected Sarsa as an Off-policy Algorithm

Note that Expected Sarsa is also an off-policy algorithm, allowing the behaviour policy b and target policy π to differ.

Especially, if π is a greedy policy with respect to current value function, Expected Sarsa simplifies to Q-learning.

Expected Sarsa Example



Example 6.6 of "Reinforcement Learning: An Introduction, Second Edition".

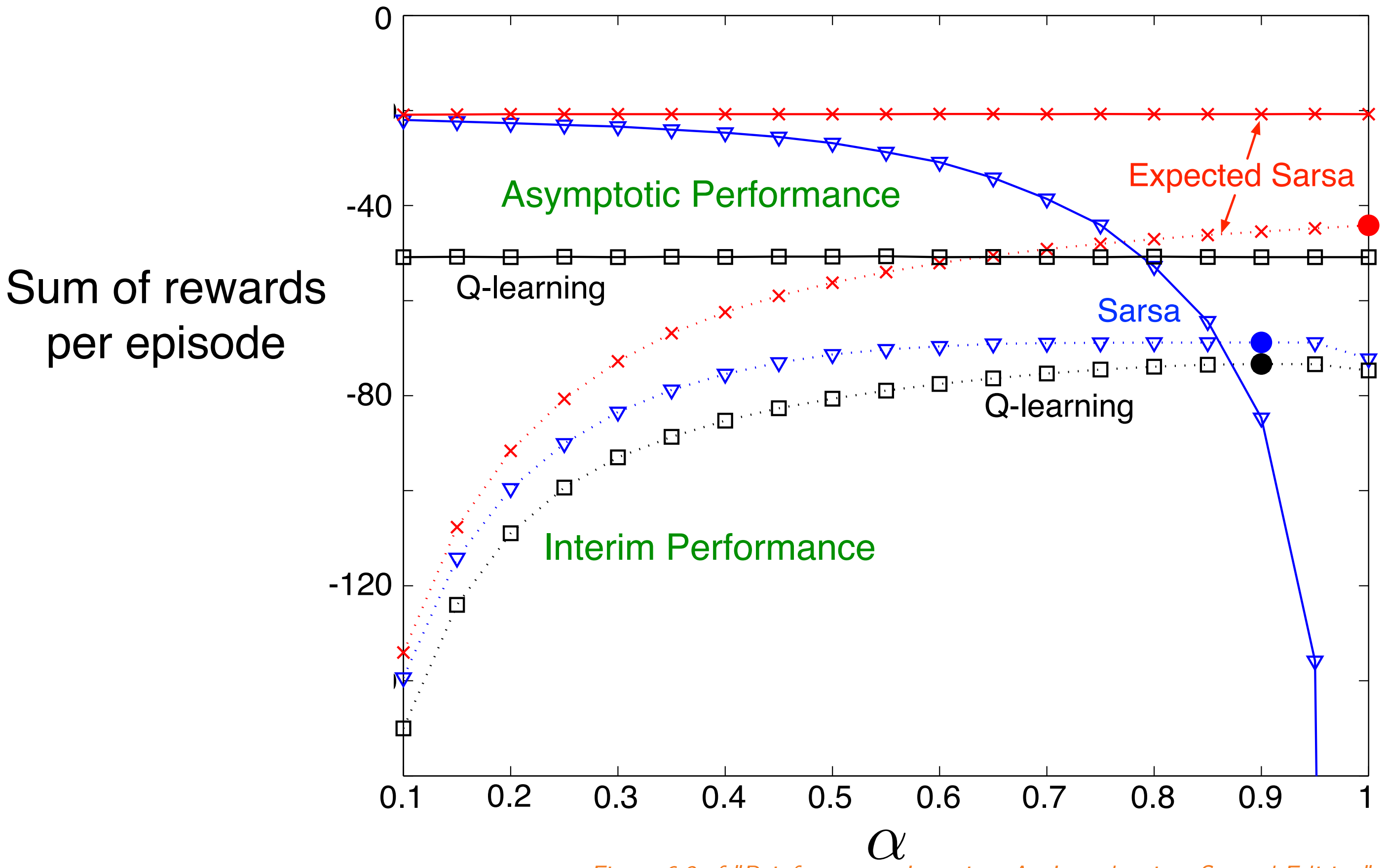


Figure 6.3 of "Reinforcement Learning: An Introduction, Second Edition".

Asymptotic performance is an average over 100k episodes (10 runs), interim performance over the first 100 episodes (50k runs); ε-greedy policy with $\varepsilon = 0.1$ is used.

Because behaviour policy in Q-learning is ε-greedy variant of the target policy, the same samples (up to ε-greedy) determine both the maximizing action and its value estimate.

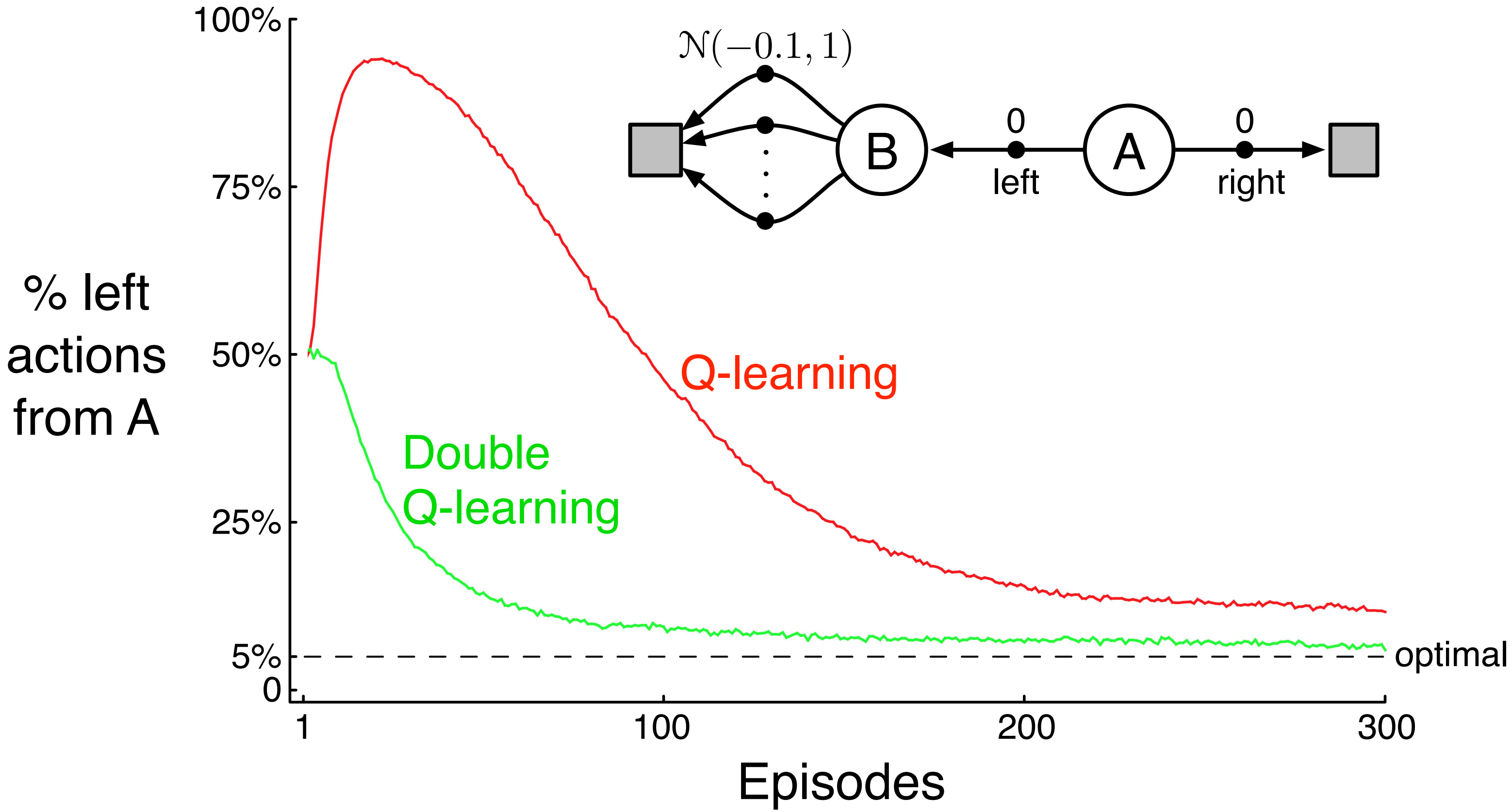


Figure 6.5 of "Reinforcement Learning: An Introduction, Second Edition".

Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$
    else:
$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$
    $S \leftarrow S'$
  until S is terminal

Modification of Algorithm 6.7 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).

Full return is

$$G_t = \sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1},$$

one-step return is

$$G_{t:t+1} = R_{t+1} + [\neg\mathrm{done}] \cdot \gamma V(S_{t+1}).$$

We can generalize both into n-step returns:

$$G_{t:t+n} \stackrel{\text{def}}{=} \left(\sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1}\right) + \gamma^n V(S_{t+n}).$$

with $G_{t:t+n} \stackrel{\text{def}}{=} G_t$ if $t+n \geq T$ (episode length).

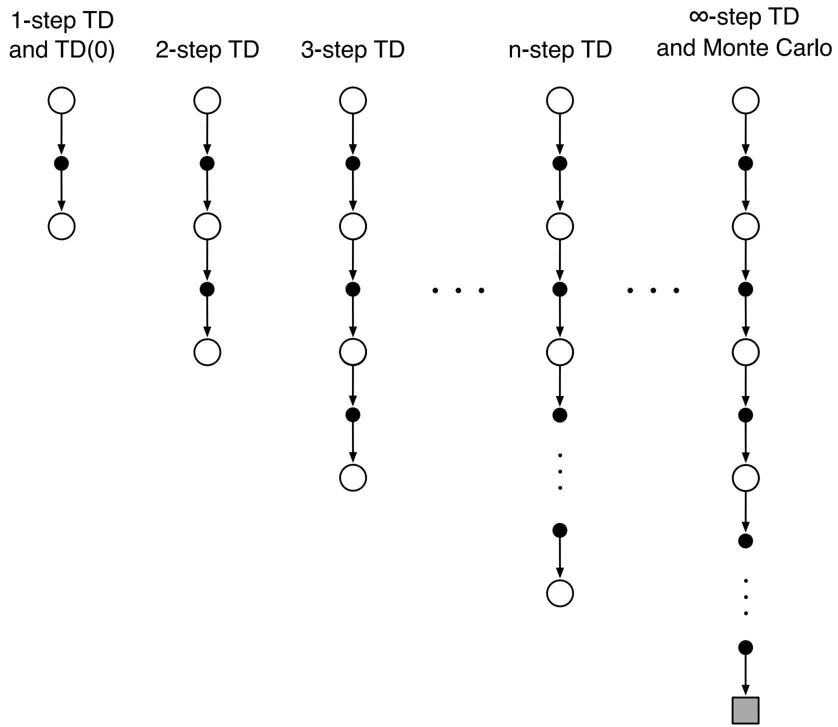


Figure 7.1 of "Reinforcement Learning: An Introduction, Second Edition".

A natural update rule is

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_{t:t+n} - V(S_t)\big).$$

n-step TD for estimating $V \approx v_\pi$

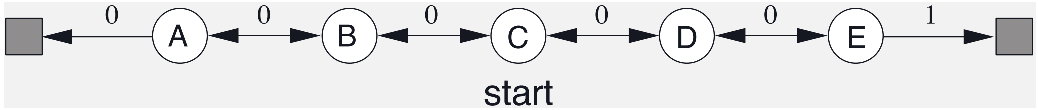
Input: a policy π
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$

Algorithm 7.1 of "Reinforcement Learning: An Introduction, Second Edition".

n-step Methods Example

Using the random walk example, but with 19 states instead of 5,



Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".

we obtain the following comparison of different values of n:

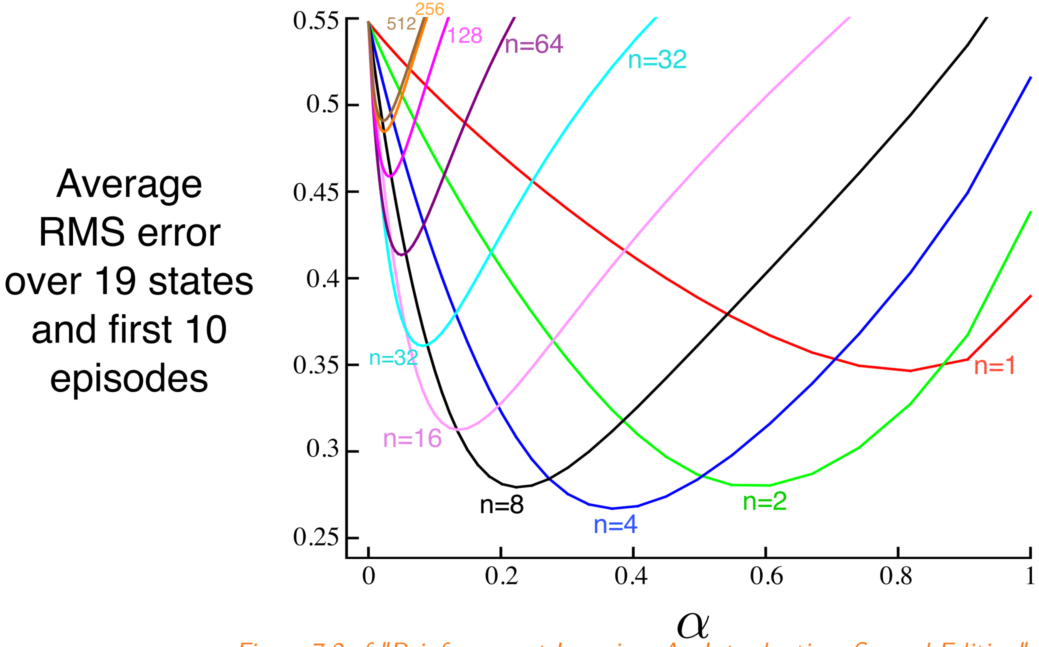


Figure 7.2 of "Reinforcement Learning: An Introduction, Second Edition".

Defining the n-step return to utilize action-value function as

$$G_{t:t+n} \stackrel{\text{def}}{=} \left(\sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1}\right) + \gamma^n Q(S_{t+n}, A_{t+n})$$

with $G_{t:t+n} \stackrel{\text{def}}{=} G_t$ if $t+n \geq T$, we get the following straightforward algorithm:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(G_{t:t+n} - Q(S_t, A_t)\big).$$

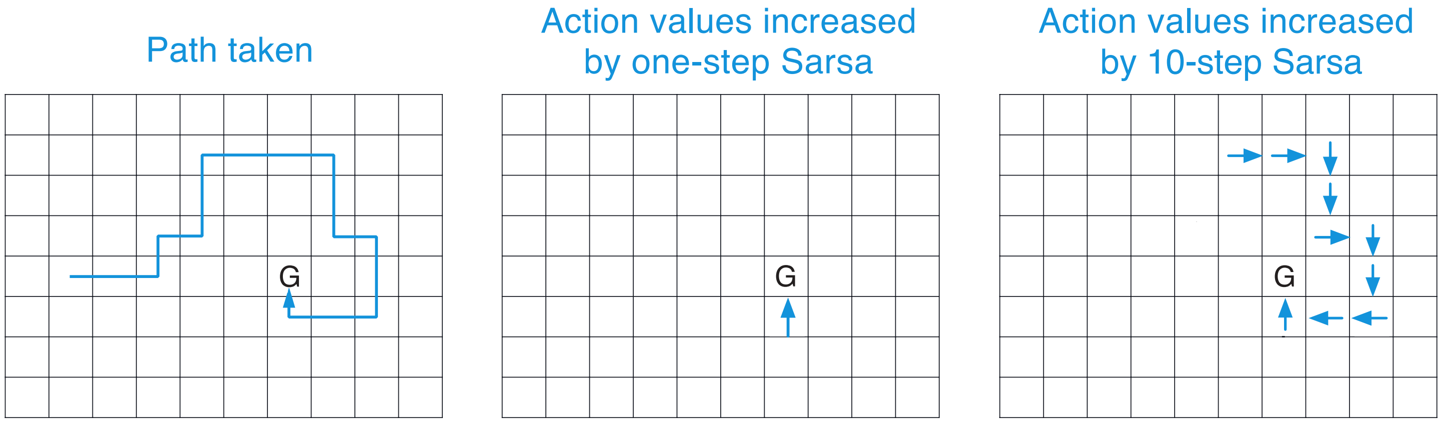


Figure 7.4 of "Reinforcement Learning: An Introduction, Second Edition".

n-step Sarsa Algorithm

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | Take action A_t
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then:
    | $T \leftarrow t+1$
    | else:
    | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
    | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
    | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
    Until $\tau = T - 1$

Algorithm 7.2 of "Reinforcement Learning: An Introduction, Second Edition".

Off-policy n-step Sarsa

Recall the relative probability of a trajectory under the target and behaviour policies, which we now generalize as

$$\rho_{t:t+n} \stackrel{\text{def}}{=} \prod_{k=t}^{\min(t+n,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

Then a simple off-policy n-step TD policy evaluation can be computed as

$$V(S_t) \leftarrow V(S_t) + \alpha\rho_{t:t+n-1}\big(G_{t:t+n} - V(S_t)\big).$$

Similarly, n-step Sarsa becomes

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\rho_{\boldsymbol{t+1:t+n}}\big(G_{t:t+n} - Q(S_t, A_t)\big).$$

Off-policy n-step Sarsa

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t + 1$
  | else:
  | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n\quad,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:t+n}\quad)$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

Modified from Algorithm 7.3 of "Reinforcement Learning: An Introduction, Second Edition" by changing ρ_{τ+1:τ+n-1} to ρ_{τ+1:τ+n}.

Off-policy n-step Without Importance Sampling

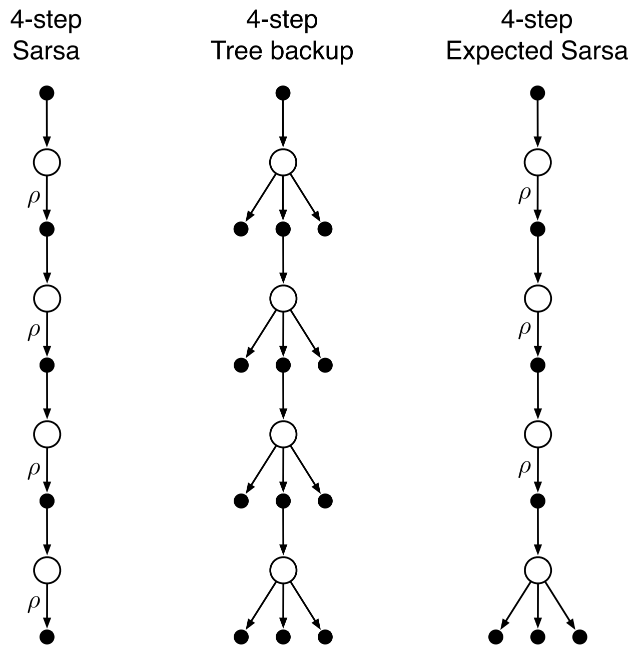


Figure 7.5 of "Reinforcement Learning: An Introduction, Second Edition".

Q-learning and Expected Sarsa can learn off-policy without importance sampling.

To generalize to n-step off-policy method, we must compute expectations over actions in each step of n-step update. However, we have not obtained a return for the non-sampled actions.

Luckily, we can estimate their values by using the current action-value function.

Off-policy n-step Without Importance Sampling

We now derive the n-step reward, starting from one-step:

$$G_{t:t+1} \stackrel{\text{def}}{=} R_{t+1} + [\neg\text{done}] \cdot \gamma \sum\nolimits_a \pi(a|S_{t+1}) Q(S_{t+1}, a).$$

For two-step, we get:

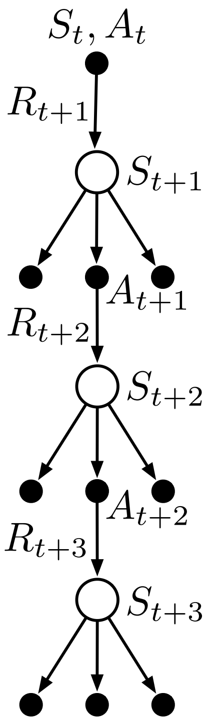
$$G_{t:t+2} \stackrel{\text{def}}{=} R_{t+1} + \gamma \sum\nolimits_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2}.$$

Therefore, we can generalize to:

$$G_{t:t+n} \stackrel{\text{def}}{=} R_{t+1} + \gamma \sum\nolimits_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n},$$

with $G_{t:t+n} \stackrel{\text{def}}{=} G_{t:T}$ if $t + n \geq T$ (episode length).

The resulting algorithm is n-step **Tree backup** and it is an off-policy n-step temporal difference method not requiring importance sampling.



the 3-step tree-backup update

Example in Section 7.5 of "Reinforcement Learning: An Introduction, Second Edition".

Off-policy n-step Without Importance Sampling

n-step Tree Backup for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$:
  | Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
  | If S_{t+1} is terminal:
  | $T \leftarrow t+1$
  | else:
  | Choose an action A_{t+1} arbitrarily as a function of S_{t+1}; Store A_{t+1}
  | $\tau \leftarrow t+1-n$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | If $t+1 \geq T$:
  | $G \leftarrow R_T$
  | else
  | $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$
  | Loop for $k = \min(t, T-1)$ down through $\tau+1$:
  | $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

Algorithm 7.5 of "Reinforcement Learning: An Introduction, Second Edition".