Temporal Difference Methods, Off-Policy Methods

Milan Straka

October 21, 2019

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

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*.

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.

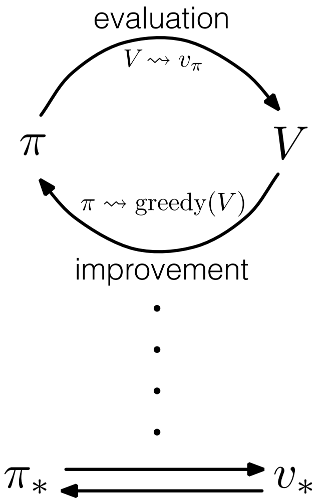


Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

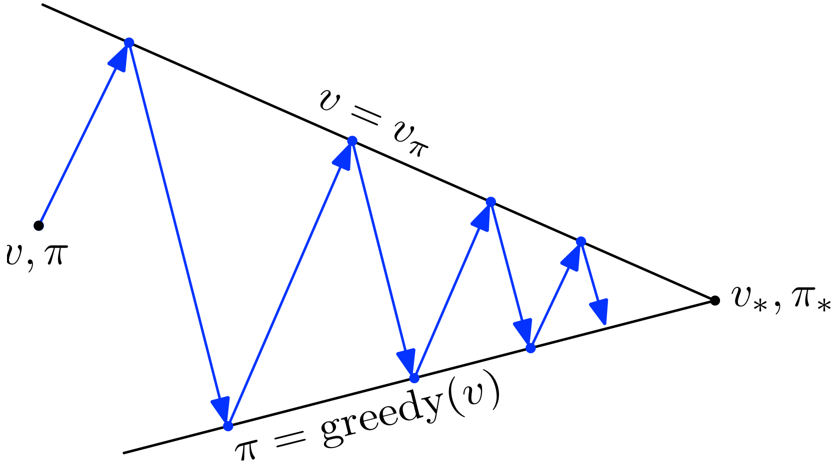


Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

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

Monte Carlo Methods

We now present the first algorithm for computing optimal policies without assuming a knowledge of the environment dynamics.

However, we still assume there are finitely many states $\mathcal{S}$ and we will store estimates for each of them.

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.

To guarantee convergence, we need to visit each state infinitely many times. One of the simplest way to achieve that is to assume *exploring starts*, where we randomly select the first state and first action, each pair with nonzero probability.

Furthermore, if a state-action pair appears multiple times in one episode, the sampled returns are not independent. The literature distinguishes two cases:

- *first visit*: only the first occurence of a state-action pair in an episode is considered
- *every visit*: all occurences of a state-action pair are considered.

Even though first-visit is easier to analyze, it can be proven that for both approaches, policy evaluation converges. Contrary to the Reinforcement Learning: An Introduction book, which presents first-visit algorithms, we use every-visit.

Monte Carlo with Exploring Starts

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, $t = T-1, T-2, \ldots, 0$:

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

Append G to $Returns(S_t, A_t)$

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

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Modification of algorithm 5.3 of "Reinforcement Learning: An Introduction, Second Edition" from first-visit to every-visit.

Monte Carlo and ε-soft Policies

A policy is called ε-soft, if

$$\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}.$$

For ε-soft policy, Monte Carlo policy evaluation also converges, without the need of exploring starts.

We call a policy ε-greedy, if one action has maximum probability of $1 - \varepsilon + \frac{\varepsilon}{|A(s)|}$.

The policy improvement theorem can be proved also for the class of ε-soft policies, and using ε-greedy policy in policy improvement step, policy iteration has the same convergence properties. (We can embed the ε-soft behaviour "inside" the environment and prove equivalence.)

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 $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{T+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}(G - Q(S_t, A_t))$

Action-values and Afterstates

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 might be better to estimate returns only for states, and there can be substantially less states than state-action pairs.

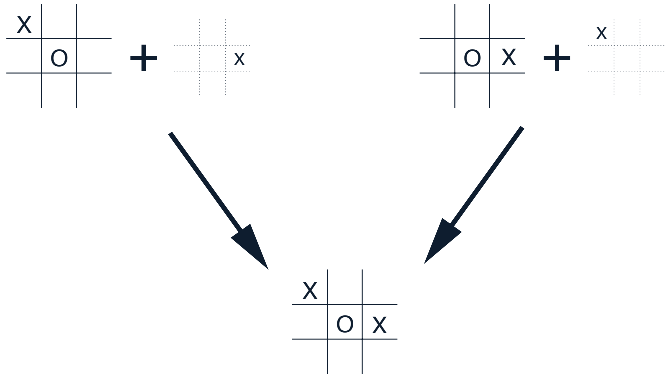


Figure from section 6.8 of "Reinforcement Learning: An Introduction, Second Edition".

Partially Observable MDPs

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,
- $o(O_t | S_t, A_{t-1})$ is an observation model.

Although planning in general POMDP is undecidable, several approaches are used to handle POMDPs in robotics (to model uncertainty, imprecise mechanisms and inaccurate sensors, …). In deep RL, partially observable MDPs are usually handled using recurrent networks, which model the latent states S_t.

TD Methods

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

$$v(S_t) \leftarrow v(S_t) + \alpha \left[G_t - v(S_t) \right],$$

the simplest temporal-difference method computes the following:

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

TD Methods

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".

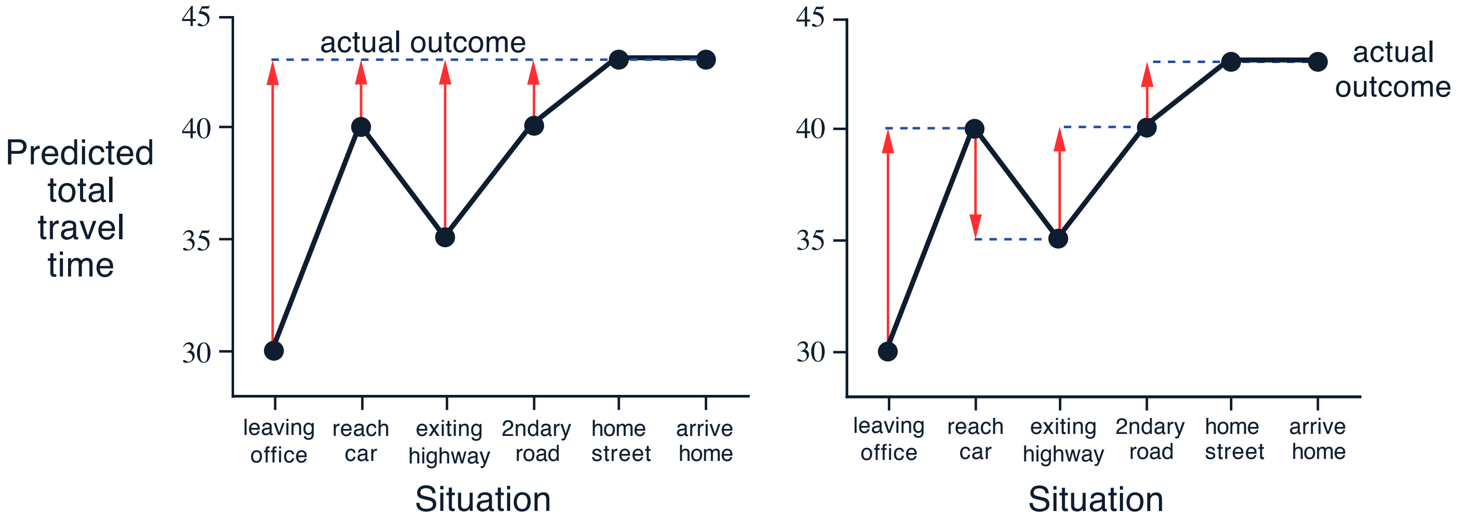
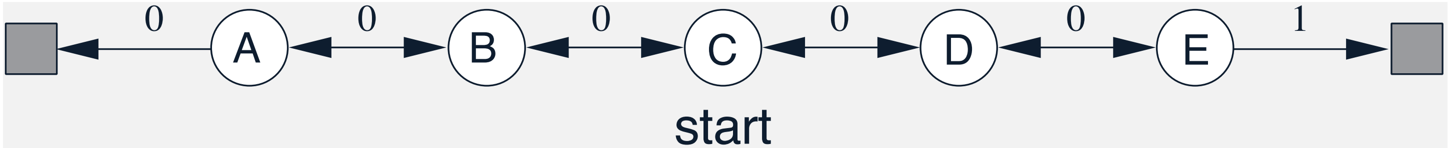


Figure 6.1 of "Reinforcement Learning: An Introduction, Second Edition".

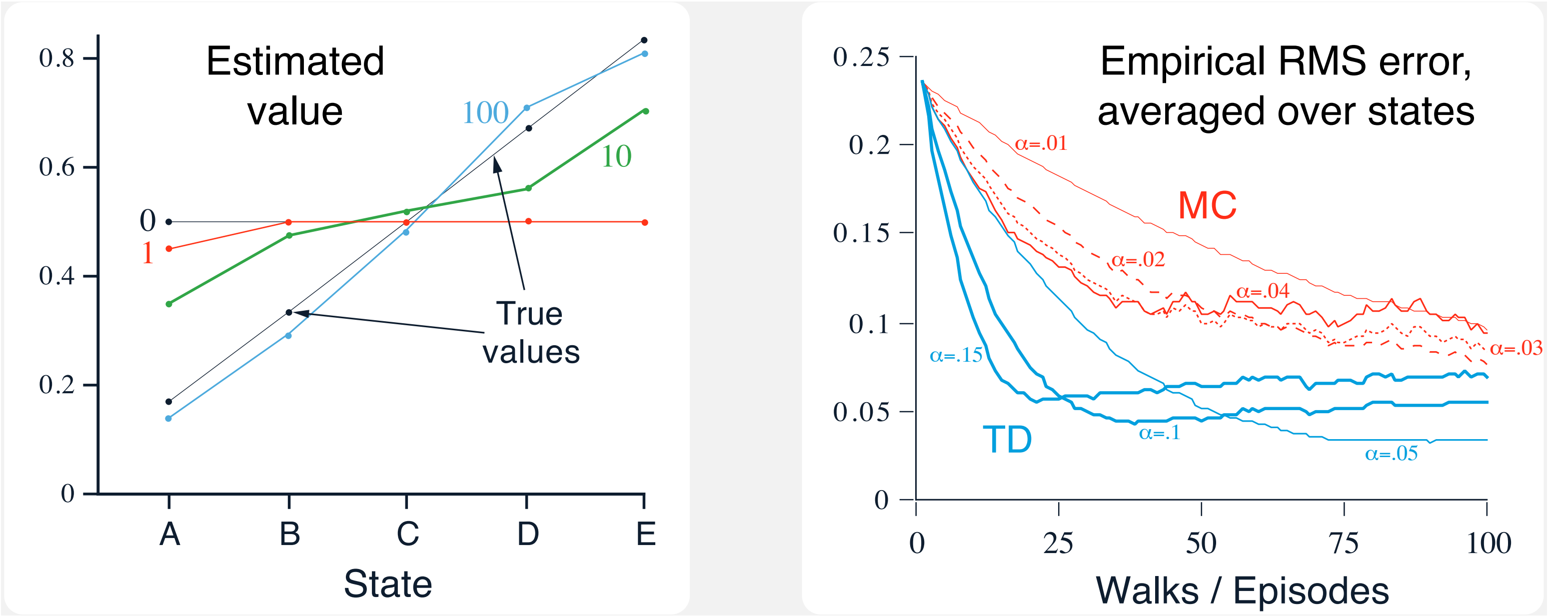
TD and MC Comparison

As with Monte Carlo methods, for a fixed policy π, TD methods converge to v_π.

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

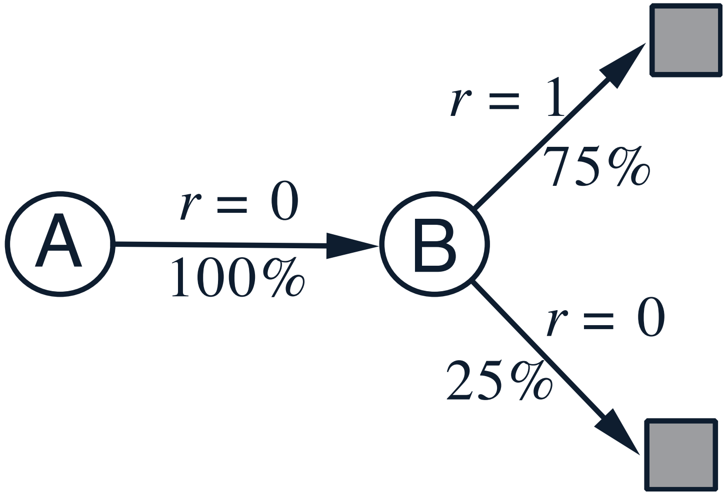


Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".



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

Optimality of MC and TD Methods



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

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	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 transfered 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 error on training data, TD minimizes MLE error for the Markov process.

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

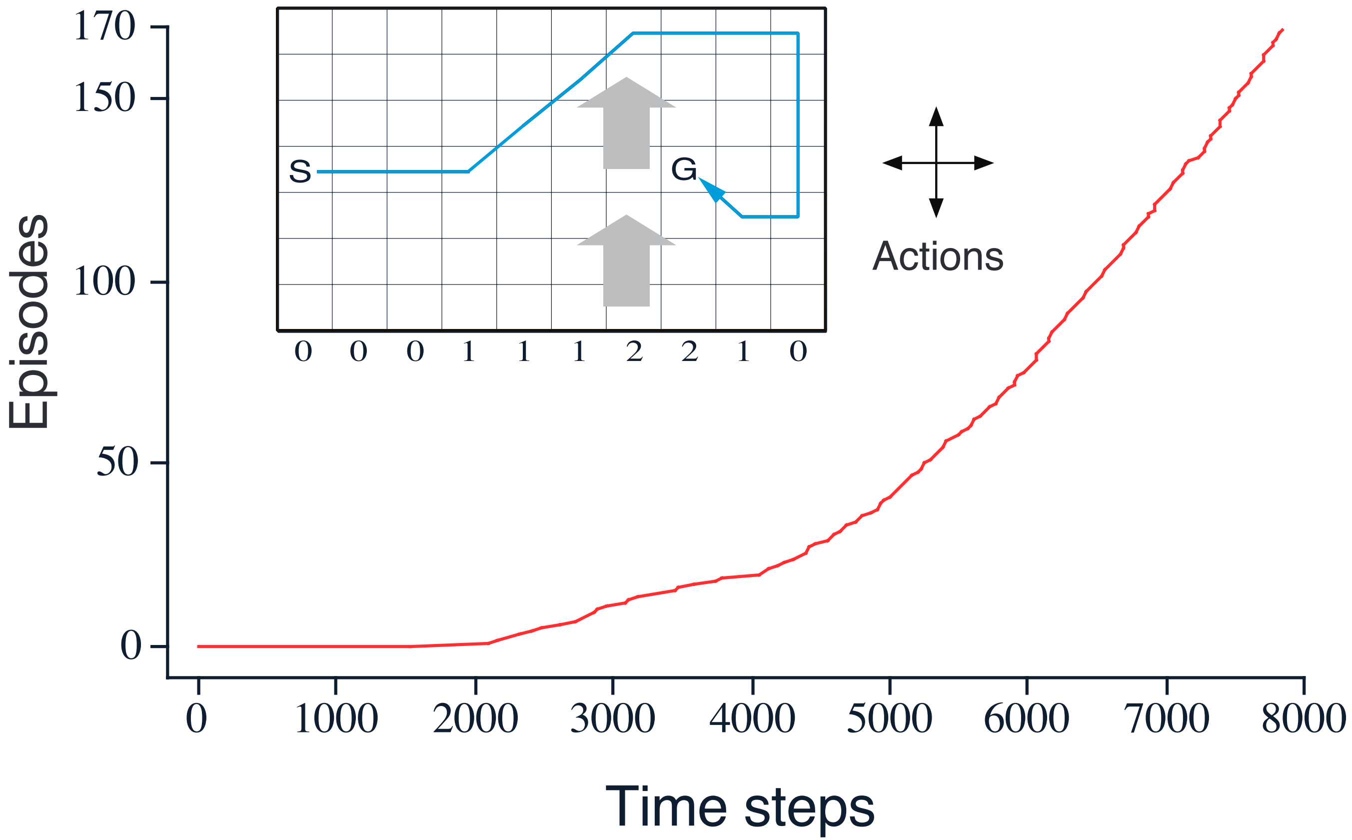
$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\right].$$

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)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  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 current policy caused the agent to stay in the same state.

Q-learning

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

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)\right].$$

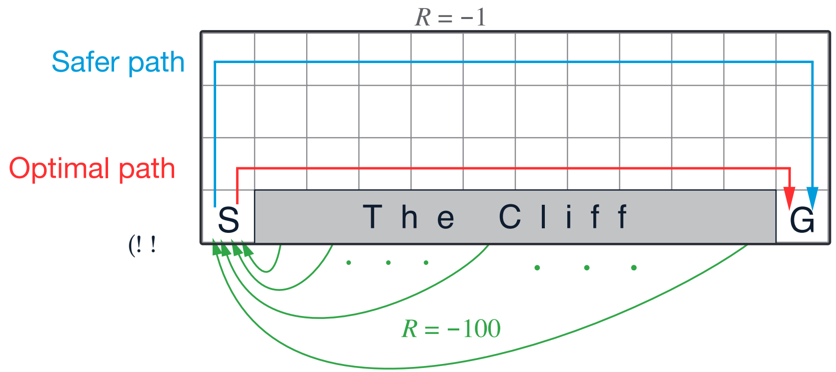
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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 $Q(terminal, \cdot) = 0$

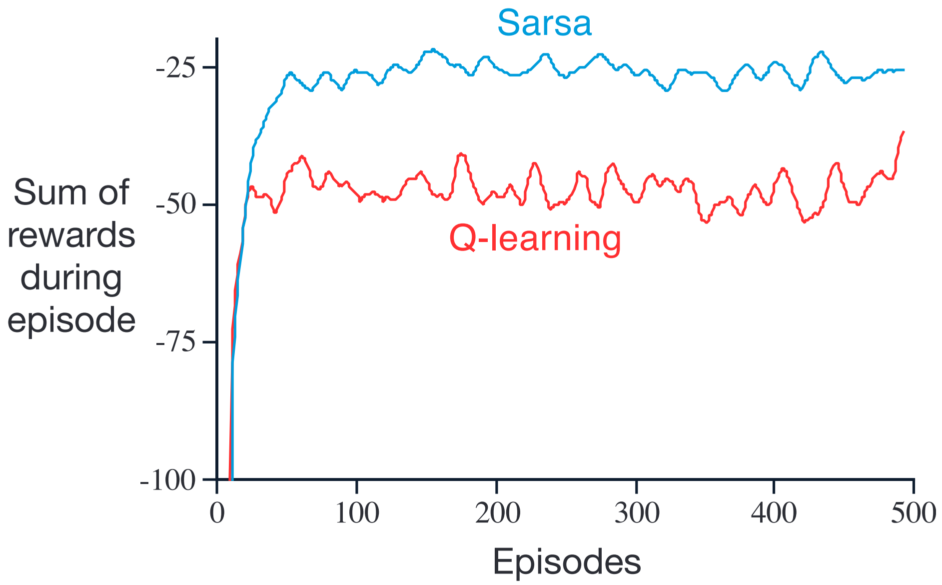
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

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

Q-learning versus Sarsa



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



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

Q-learning and Maximization Bias

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 estimate its value.

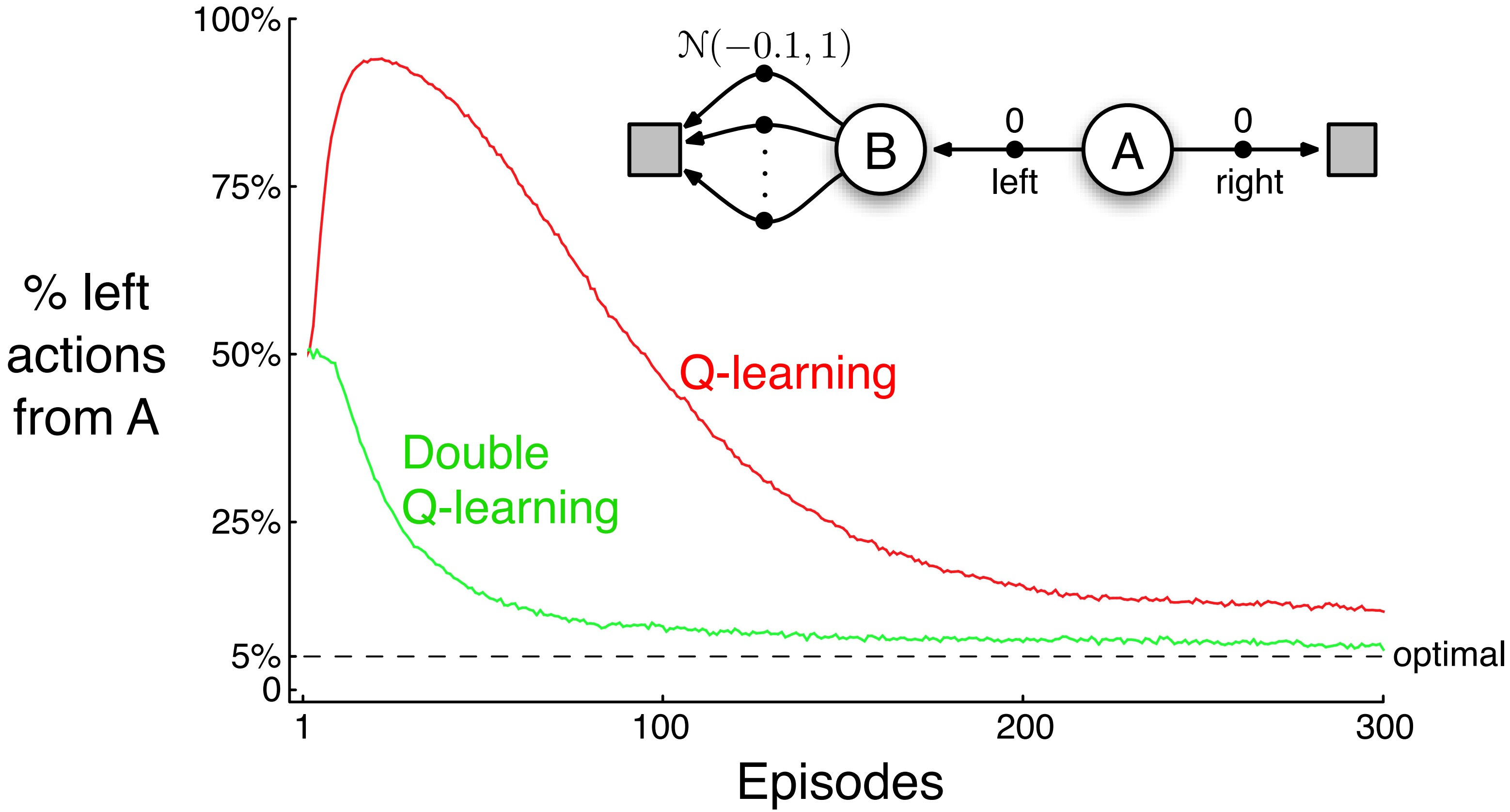


Figure 6.5 of "Reinforcement Learning: An Introduction, Second Edition".

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 $Q(\textit{terminal}, \cdot) = 0$

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).

On-policy and Off-policy Methods

So far, all 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.

Importance Sampling

Assume that b and π are two distributions and let x_i be the samples of b. We can then estimate $\mathbb{E}_{x\sim b}[f(x)]$ as

$$\mathbb{E}_{x\sim b}[f(x)] \sim \sum_i f(x_i).$$

In order to estimate $\mathbb{E}_{x\sim \pi}[f(x)]$ using the samples x_i, we need to account for different probabilities of x_i under the two distributions by

$$\mathbb{E}_{x\sim \pi}[f(x)] \sim \sum_i \frac{\pi(x_i)}{b(x_i)} f(x_i)$$

with $\pi(x)/b(x)$ being a *relative probability* of x under the two distributions.

Off-policy Prediction

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

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

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

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

Off-policy Monte Carlo Prediction

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), but has smaller variance.

Off-policy Monte Carlo Prediction

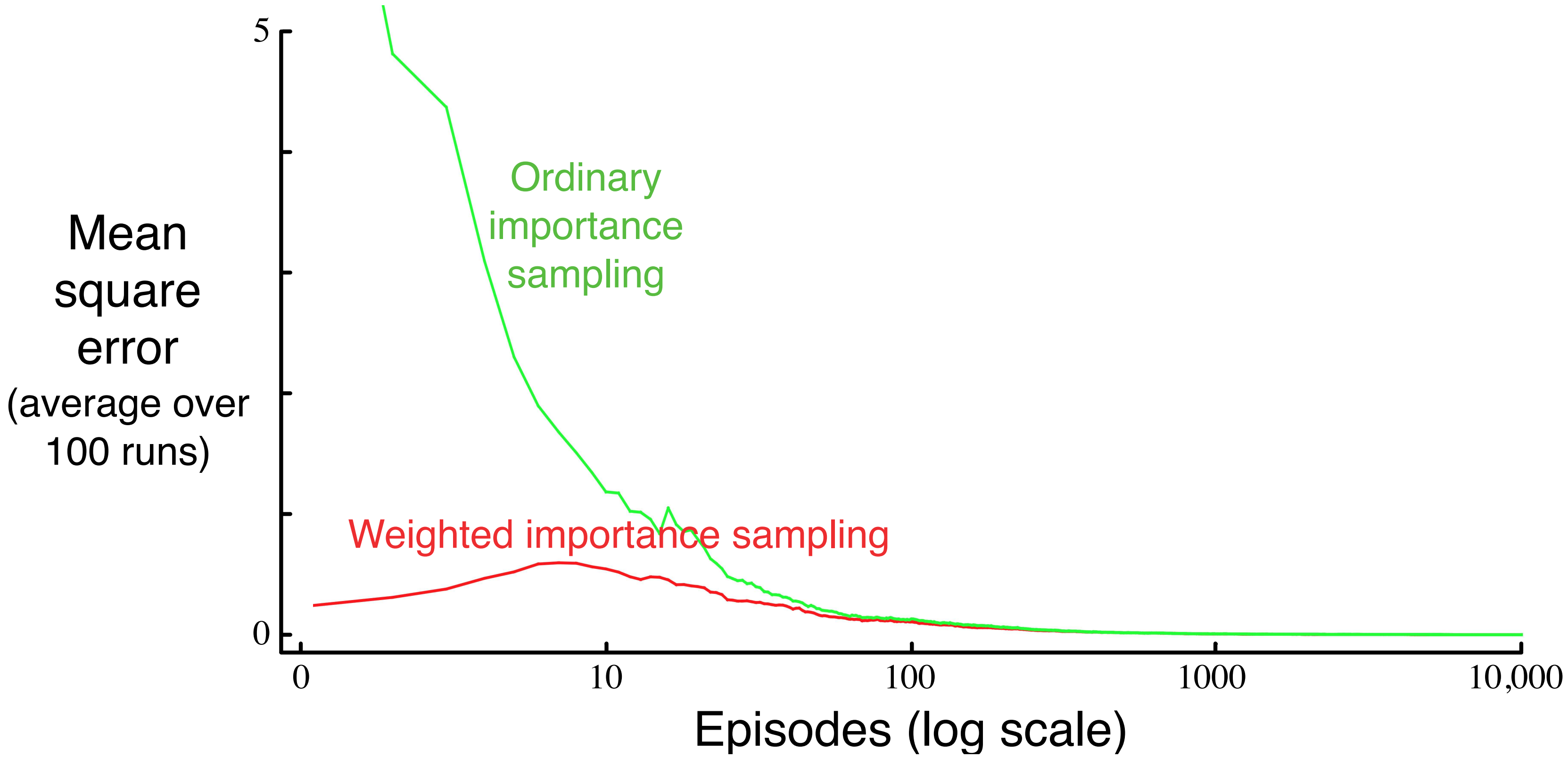


Figure 5.3 of "Reinforcement Learning: An Introduction, Second Edition".

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 Prediction

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, moving only *in expectation*.

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

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

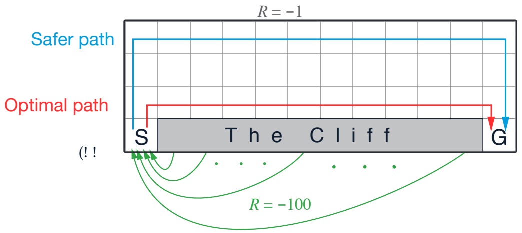
Compared to Sarsa, the expectation removes a source of variance and therefore usually performs better. However, the complexity of the algorithm increases and becomes dependent on number of actions $|\mathcal{A}|$.

Expected Sarsa as 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".

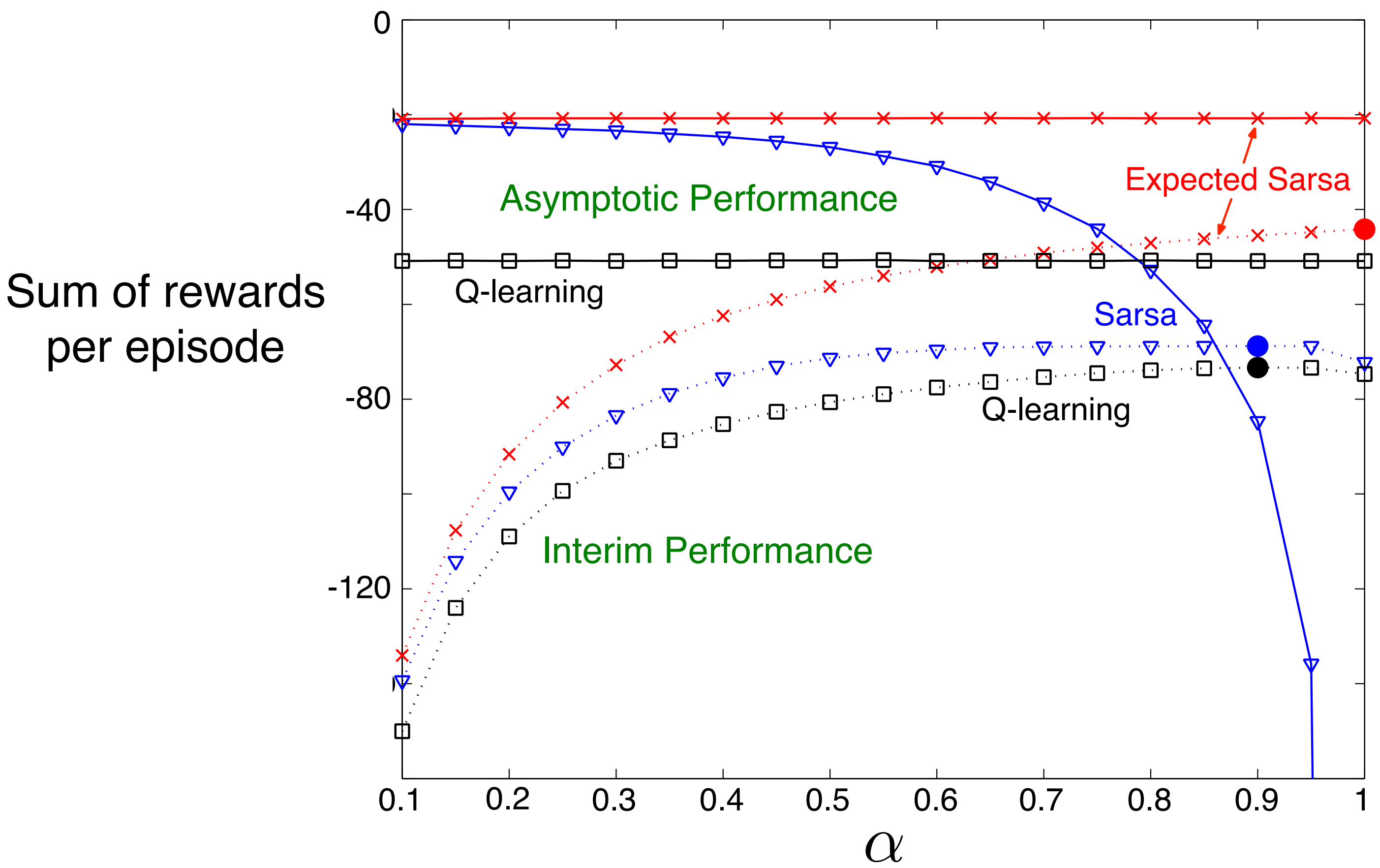


Figure 6.3 of "Reinforcement Learning: An Introduction, Second Edition".

Asymptotic performance is averaged over 100k episodes, interim performance over the first 100.