RL Ch7 n-step Bootstrapping

Le-Rong HSU

July 2023

1 Introduction

The disadvantage of one-step TD methods is that the "frequency" of update is fixed and not flexible. Sometime we wish to update values based on "more information" (rewards), not being restricted to only next reward. Thus, here comes to n-step bootstrapping.

In this chapter we unify the Monte Carlo (MC) methods and the one-step temporal-difference (TD) methods presented in the previous two chapters. In this chapter we present n-step TD methods that generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task.

The idea of n-step methods is usually used as an introduction to the algorithmic idea of eligibility traces (Chapter 12), which enable bootstrapping over multiple time intervals simultaneously.

2 n-step TD Prediction

- Monte Carlo methods $\xrightarrow{\text{based on}}$ **rewards** from that state **until the end of the episode**
- one-step TD methods $\xrightarrow{\text{based on}}$ **one next reward** which bootstraps from the value of the next state as a proxy for the remaining rewards.
- intermediate methods $\rightarrow$ an update based on a number of rewards

The methods that use n-step updates are still TD methods because they still change an earlier estimate based on how it differs from a later estimate. **Now the later estimate is not one step later, but n steps later**. Methods in which the temporal difference extends over n steps are called *n-step TD methods*.

What's the difference between Monte Carlo methods and n-step TD methods?

To be more precisely, consider the update of the estimated value of state S_t as a result of the state–reward sequence, $S_t, R_{t+1}, S_{t+1}, R_{t+2}, ..., R_T, S_T$ (omitting

the actions). In Monte Carlo methods, we wish to estimate $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$ in the direction of the "complete" return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t_3} + \cdots + \gamma^{T-t-1} R_T$$

where T is the last time step of the episode and G_t is the cumulative discounted future rewards from t to T (at time t, we choose an action A_t according to S_t). G_t is called the *target* of the update. In Monte Carlo methods, the target G_t the return.

In *one-step* updates, the target is the first reward plus the discounted estimated value of the next state:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}),$$

where V_t here is the estimate at time t of v_π. The difference between the target of Monte Carlo updates and *one-step* updates comes out: we replace $\gamma R_{t+2} + \gamma^2 R_{t_3} + \cdots + \gamma^{T-t-1} R_T$ with $\gamma V_t(S_{t+1})$ in *one-step* updates. The target for a two-step update is the *two-step return*:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

Similarly, the target for a n-step update is

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{7.1}$$

for all n, t such that $n \geq 1$ and $0 \leq t < T - n$. If $t + n \geq T$, then all the missing term are taken as zero and $G_{t+n} \doteq G_t$.

Note that n-step returns for $n > 1$ involve future rewards and states that are not available at the time of transition from t to $t + 1$. We have to see R_{t+n} and compute V_{t+n-1} for calculating the n-step return $G_{t:t+n}$. The first time these are available is $t + n$. The natural state-value learning algorithm for using n-step returns is thus

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)], 0 \leq t < T \tag{7.2}$$

while the values of all other states remain the same: $V_{t+n}(s) = V_{t+n-1}(s), \forall s \neq S_t$.

Algorithm 1 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 $V(s)$ for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod n + 1
for each episode **do**
  Initialize and store $S_0 \neq$terminal
  $T \leftarrow \infty$
  for $t = 0, 1, 2, ...$ **do**
    if $t < T$ **then**
      Take an action according to $\pi(\cdot|S_t)$
      Observe and store the next reward R_{t+1} and the next state S_{t+1}
      If S_{t+1} is terminal, then $T \leftarrow t + 1$
    end if
    $\tau \leftarrow t - n + 1$ (the time whose state's estimate is being updated)
    if $tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
      $V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$
    end if
  end for
  Until $\tau = T - 1$
end for

The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}. **An important property of n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense.** That is, for $n \geq 1$

$$\max_s |\mathbb{E}_\pi[G_{t:t+n}|S_t = s] - v_\pi(s)| \leq \gamma^n \max_s |V_{t+n-1}(s) - v_\pi(s)| \tag{7.3}$$

This is called the *error reduction property.* Because of error reduction property, one can show formally that all n-step TD methods converge to the correct predictions under appropriate technical conditions.

3 n-step Sarsa

We can use n-step methods to solve the control problem. The n-step version of Sarsa we call n-step Sarsa, and the original version presented in the previous chapter we henceforth call *one-step Sarsa, or Sarsa(0).*

The main idea is to simply switch states for actions (state–action pairs) and then use an ε-greedy policy. We redefine n-step returns (update targets) in

terms of estimated action values:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), n \geq 1, 0 \leq t < T-n, \tag{7.4}$$

with $G_{t:t+n} \doteq G_t$ if $t + n \geq T$. The natural algorithm is then

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], 0 \leq t < T \tag{7.5}$$

while the values of all other states remain the same: $Q_{t+n}(s, a) = Q_{t+n-1}(s, a), \forall s \neq S_t, a \neq A_t$. This is the algorithm we call *n-step Sarsa*.

Algorithm 2 n-step Sarsa for estimating $Q \approx q_\pi$ or q_*

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $Q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
All store and access operations (for S_t and R_t) can take their index mod n + 1
for each episode **do**
  Initialize and store $S_0 \neq$terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$'
  for $t = 0, 1, 2, ...$ **do**
    if $t < T$ **then**
      Take an action according to $\pi(\cdot|S_t)$
      Observe and store the next reward R_{t+1} and the next state S_{t+1}
      If S_{t+1} is terminal, then $T \leftarrow t + 1$
      else: Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    end if
    $\tau \leftarrow t - n + 1$ (the time whose state's estimate is being updated)
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$
    end if
  end for
  Until $\tau = T - 1$
end for

Example:

- rewards were all set 0, except for a positive reward at **G**
- first figure: the path taken by an agent (in an episode)
- second figure: action values were strengthened as a result of this path by one-step Sarsa

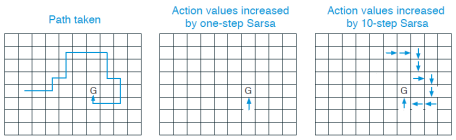


Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods.

- third figure: action values were strengthened as a result of this path by n-step Sarsa

Conclusion: The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

The n-step Sarsa is similarly defined. Let

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}), t+n < T \tag{7.7}$$

and for $t+n \geq T$, $G_{t:t+n} = G_t$, where

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \forall s \in \mathcal{S} \tag{7.8}$$

4 n-step Off-policy Learning

π is the greedy policy for the current action-value- function estimate, and b is a more exploratory policy, perhaps ε-greedy. In off-policy learning, we have to consider the relative probability of π and b. Recall the importance sampling ratio in Chapter 5:

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k) p(S_{k+1}|S_k, A_k)} = \prod_{T-1}^{k=t} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \tag{5.3}$$

The off-policy update of n-step TD for time t (which is actually made at time $t+n$) is:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} [G_{t:t+n} - V_{t+n-1}(S_t)], 0 \leq t < T \tag{7.9}$$

where

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \tag{7.10}$$

Note that

- if $\pi(A_k|S_k) = 0$, then the ratio would be zero

- if $\pi(A_k|S_k) >> b(A_k|S_k)$, then the ratio would increase. This is reasonable because that action is characteristic of π (and therefore we want to learn about it) but is selected only rarely by b and thus rarely appears in the data. To make up for this we have to over-weight it when it does occur.

The update rule for n-step Sarsa is

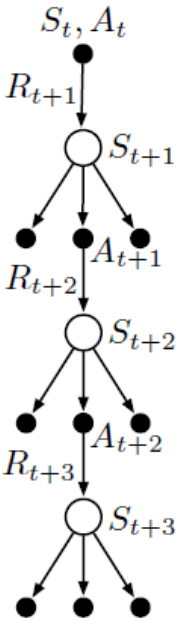
$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], 0 \leq t < T \tag{7.11}$$

Algorithm 3 Off-policy n-step Sarsa for estimating $Q \approx q_\pi$ or q_*

Input: a arbitrary behavior policy b such that $b(a|s) > 0$ for all s, a
Initialize $Q(s, a)$ for all s, 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 $n + 1$
for each episode **do**
  Initialize and store $S_0 \neq$terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  for $t = 0, 1, 2, ...$ **do**
    if $t < T$ **then**
      Take an action according to $\pi(\cdot|S_t)$
      Observe and store the next reward R_{t+1} and the next state S_{t+1}
      If S_{t+1} is terminal, then $T \leftarrow t + 1$
      else: Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    end if
    $\tau \leftarrow t - n + 1$ (the time whose state's estimate is being updated)
    if $\tau \geq 0$ **then**
      $\rho \leftarrow \Pi_{i=\tau+1}^{\min(\tau+n,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      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 w.r.t. Q
    end if
  end for
  Until $\tau = T - 1$
end for

5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

This chapter we talk about a tree-backup algorithm. Consider a tree backup diagram:



the 3-step tree-backup update

How to understand this diagram? The hollow circle means the state, and the solid circle means the action. The arrow from a hollow circle means taking action, and the arrow from a solid circle means the reward.

Initially we are at S_t and we take an action A_t, which leads to a reward R_{t+1} and state S_{t+1}. In the state S_{t+1}, **we sample three actions and we choose one of them**, the middle one, to be A_{t+1} and so on. For the last state S_{t+3}, none of actions is chosen.

In the tree-backup update, the target includes the chosen actions **and the actions that are sampled but not selected**.

More precisely, the update is from the estimated action values of the **leaf nodes** of the tree. Each leaf node contributes to the target with a weight proportional to its probability of occurring under the target policy π, e.g., $\pi(A_{t+1}|S_{t+1}), \pi(A_{t+2}|S_{t+2})$. Thus, each non-selected second-level action a' contributes with weight $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$, and third-level action contributes with weight $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a''|S_{t+s})$, and so on.

Let's develop the detailed equations for the n-step tree-backup algorithm. The one-step return (target) is the same as that of Expected Sarsa,

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a) \qquad (7.15)$$

for $t < T-1$, and the two-step tree-backup return is

$$\begin{aligned} G_{t:t+2} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a)) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2} \end{aligned}$$

for $t < T-2$. The general definition is

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \qquad (7.16)$$

for $t < T-1, n \geq 2$, with the $n = 1$ case handled by (7.15) except for $G_{T-1:t+n} \doteq R_T$.

This target is then used with the usual action-value update rule from n-step Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

for $0 \leq t < T$ while the values of all other state–action pairs remain unchanged.

Algorithm 4 On-step tree backup for estimating $Q \approx q_\pi$ or q_*

Initialize $Q(s, a)$ for all s, 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
for each episode **do**
  Initialize and store $S_0 \neq$terminal
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$'
  for $t = 0, 1, 2, ...$ **do**
    if $t < T$ **then**
      Take an action A_t and store R_{t+1}, S_{t+1}
      Observe and store the next reward R_{t+1} and the next state S_{t+1}
      If S_{t+1} is terminal, then $T \leftarrow t + 1$
      else: Choose an action A_{t+1} arbitrarily as a function of S_{t+1}; Store A_{t+1}
    end if
    $\tau \leftarrow t - n + 1$ (the time whose state's estimate is being updated)
    if $\tau \geq 0$ **then**
      If $t + 1 \geq T$: $G \leftarrow R_T$
      else: $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$
      for $k = \min(t, T-1)$ down to $\tau + 1$ **do**
        $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
      end for
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy w.r.t. Q
    end if
  end for
  Until $\tau = T - 1$
end for

6 A Unifying Algorithm: n-step $Q(\sigma)$

How can we unify our above algorithms? The backup diagram below suggests the idea:

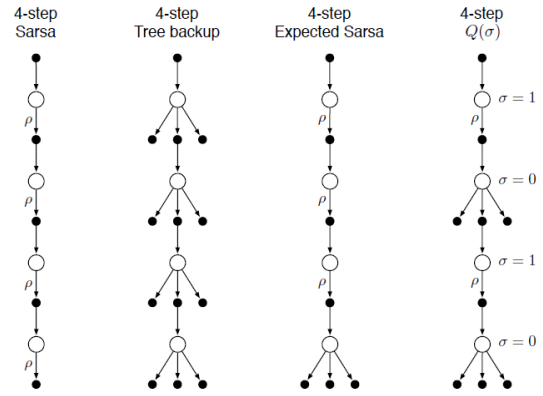


Figure 7.5

Remark. To identify these backup diagrams, it's necessary to look over how an update is performed for each learning.

- TD: $S_t, R_{t+1}, S_{t+1}, ..., R_T, S_T$
- Sarsa, Expected Sarsa: $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, ..., S_{T-1}, A_{T-1}, R_T, S_T$, and we use an quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ for one update.

The solid circle is an action from a specific state and thus the root is solid; it's an action from a specific state. Then we could understand why there are three actions connected to the last state. This is because the terminal state is in the quintuple and we cannot select an action.

Remark. Notations:

- ρ indicates the transitions on which which importance sampling is required in the off-policy case.
- $\sigma = 1 \rightarrow$ sample; $\sigma = 0 \rightarrow$ no sample (expectation)

To increase the possibilities even further we can consider a continuous variation between sampling and expectation. Let $\sigma_t \in [0, 1]$ denote the degree of sampling on step t, with $t = 1$ denoting full sampling and $t = 0$ denoting a pure expectation with no sampling. The random variable σ_t might be set as a function of the state, action, or state–action pair at time t.

First we write the tree-backup n-step return (7.16) in terms of the horizon $h = t + n$ and then in terms of the expected approximate value $\bar{V}$.

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \tag{7.16}$$

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \forall s \in \mathcal{S} \tag{7.8}$$

Then,

$$\begin{aligned} G_{t:h} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{h-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h} \\ &= R_{t+1} + \gamma \bar{V}_{h-1}(S_{t+1}) - \gamma \pi(A_{t+1}|S_{t+1}) Q_{h-1}(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h} \\ &= R_{t+1} + \gamma \pi(A_{t+1}|S_{t+1}) \left(G_{t+1:h} - Q_{h-1}\left(S_{t+1}, A_{t+1}\right) \right) + \gamma \bar{V}_{h-1}(S_{t+1}) \end{aligned}$$

For $Q(\sigma)$, we slide linearly between these two cases: for $t < h \leq T$

$$\begin{aligned} G_{t:h} \doteq R_{t+1} &+ \gamma(\sigma_{t+1}\rho_{t+1} + (1 - \sigma_{t+1})\pi(A_{t+1}|S_{t+1}))(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) \\ &+ \gamma \bar{V}_{h-1}(S_{t+1}) \end{aligned} \tag{7.17}$$

The recursion terminates as long as

- $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$ if $h < T$
- $G_{T-1:T} \doteq R_T$ if $h = T$.

The algorithm is given in the next page.

I'm tired of typing latex.

Off-policy n-step $Q(\sigma)$ 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 else it is 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 and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  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 and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | Select and store σ_{t+1}
  | Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as ρ_{t+1}
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | If $t+1 < T$:
  | $G \leftarrow Q(S_{t+1}, A_{t+1})$
  | Loop for $k = \min(t+1, T)$ down through $\tau + 1$:
  | if $k = T$:
  | $G \leftarrow R_T$
  | else:
  | $\bar{V} \leftarrow \sum_a \pi(a|S_k) Q(S_k, a)$
  | $G \leftarrow R_k + \gamma\big(\sigma_k \rho_k + (1 - \sigma_k)\pi(A_k|S_k)\big)\big(G - Q(S_k, A_k)\big) + \gamma \bar{V}$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$