N-step Temporal Difference Methods

Self-Study

November 4, 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

n-step Methods

Full return is

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

one-step return is

$$G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1}).$$

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).

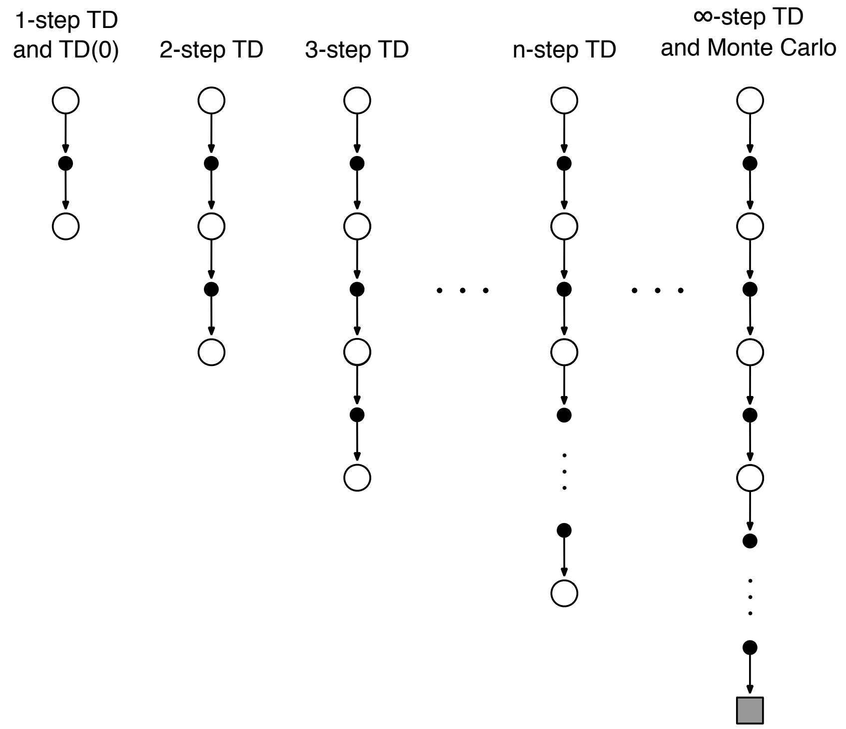


Figure 7.1 of "Reinforcement Learning: An Introduction, Second Edition".

n-step Methods

A natural update rule is

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_{t:t+n} - V(S_t)\right].$$

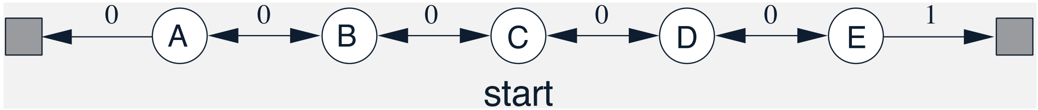
n-step TD for estimating $V \approx v_\pi$

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

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

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:

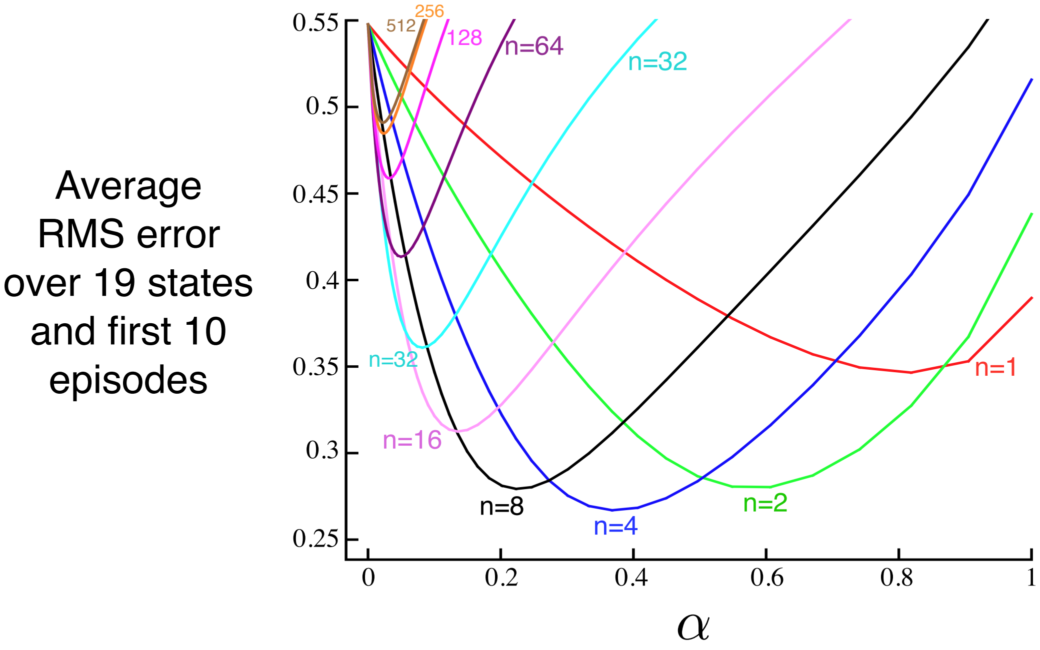


Figure 7.2 of "Reinforcement Learning: An Introduction, Second Edition".

n-step Sarsa

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:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_{t:t+n} - Q(S_t, A_t)\right].$$

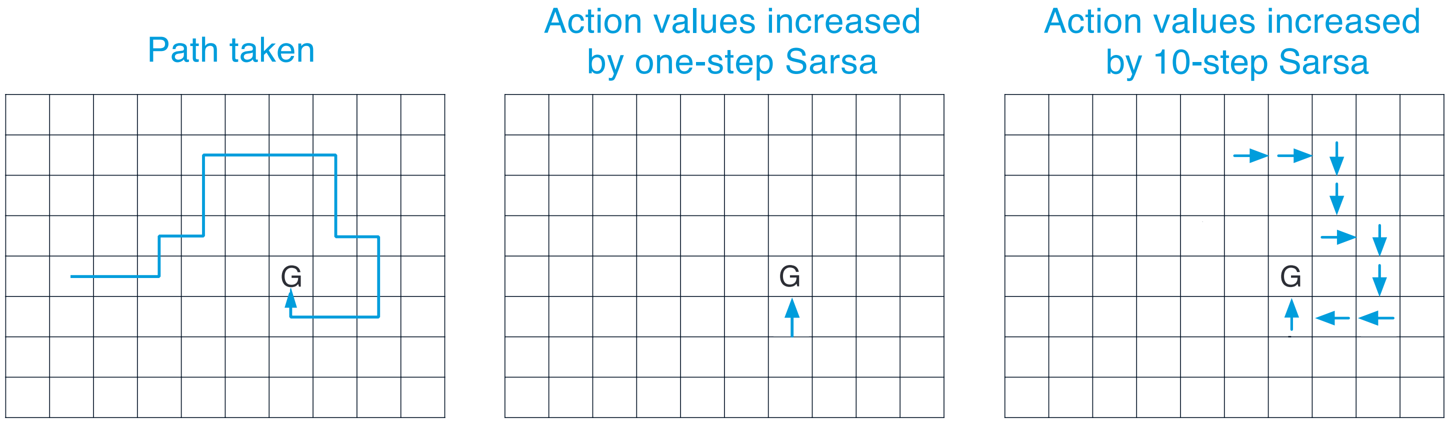


Figure 7.4 of "Reinforcement Learning: An Introduction, Second Edition".

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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
    | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
    | 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 can be computed as

$$V(S_t) \leftarrow V(S_t) + \alpha\rho_{t:t+n-1}\left[G_{t:t+n} - V(S_t)\right].$$

Similarly, n-step Sarsa becomes

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

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 $\tau \geq 0$:
    | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n\ \ \ \ ,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:t+n}$)
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)
    | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
    | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
    Until $\tau = T - 1$

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

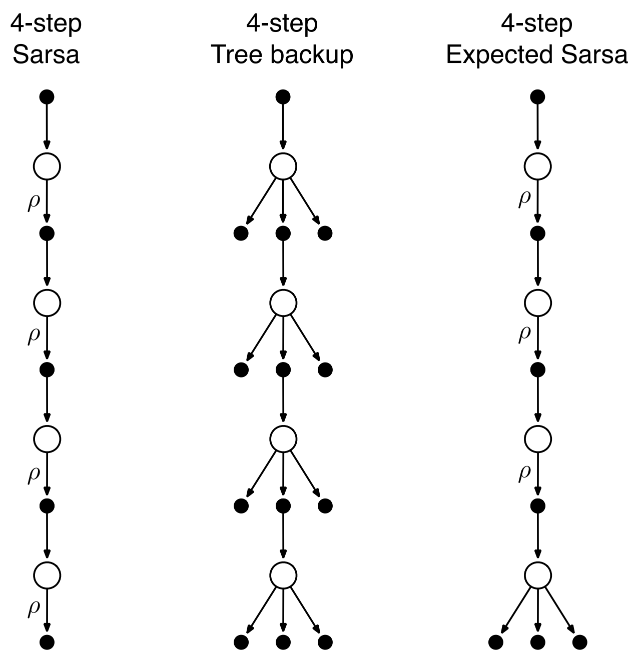


Figure 7.5 of "Reinforcement Learning: An Introduction, Second Edition".

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.

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

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

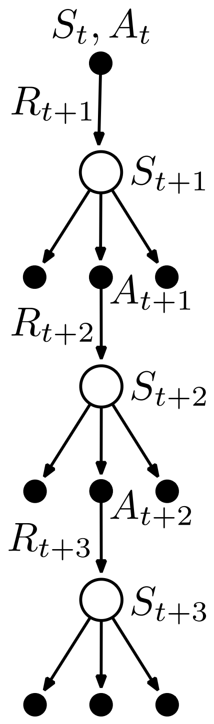
For two-step, we get:

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

Therefore, we can generalize to:

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

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 ≠ terminal
   Choose an action A_0 arbitrarily as a function of S_0; Store A_0
   T ← ∞
   Loop for t = 0, 1, 2, ... :
   |  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 ← t + 1
   |     else:
   |        Choose an action A_{t+1} arbitrarily as a function of S_{t+1}; Store A_{t+1}
   |  τ ← t + 1 − n    (τ is the time whose estimate is being updated)
   |  If τ ≥ 0:
   |     If t + 1 ≥ T:
   |        G ← R_T
   |     else
   |        G ← R_{t+1} + γ Σ_a π(a|S_{t+1}) Q(S_{t+1}, a)
   |     Loop for k = min(t, T − 1) down through τ + 1:
   |        G ← R_k + γ Σ_{a≠A_k} π(a|S_k) Q(S_k, a) + γ π(A_k|S_k) G
   |     Q(S_τ, A_τ) ← Q(S_τ, A_τ) + α [G − Q(S_τ, A_τ)]
   |     If π is being learned, then ensure that π(·|S_τ) is greedy wrt Q
   Until τ = T − 1
```

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