Chapter 7: n-step Bootstrapping

Seungjae Ryan Lee

endtoend.ai

Recap: MC vs TD

- Monte Carlo: wait until end of episode

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[\overbrace{\underbrace{G_t}_{\text{Return}} - V(S_t)}^{\text{MC error}} \Big],$$

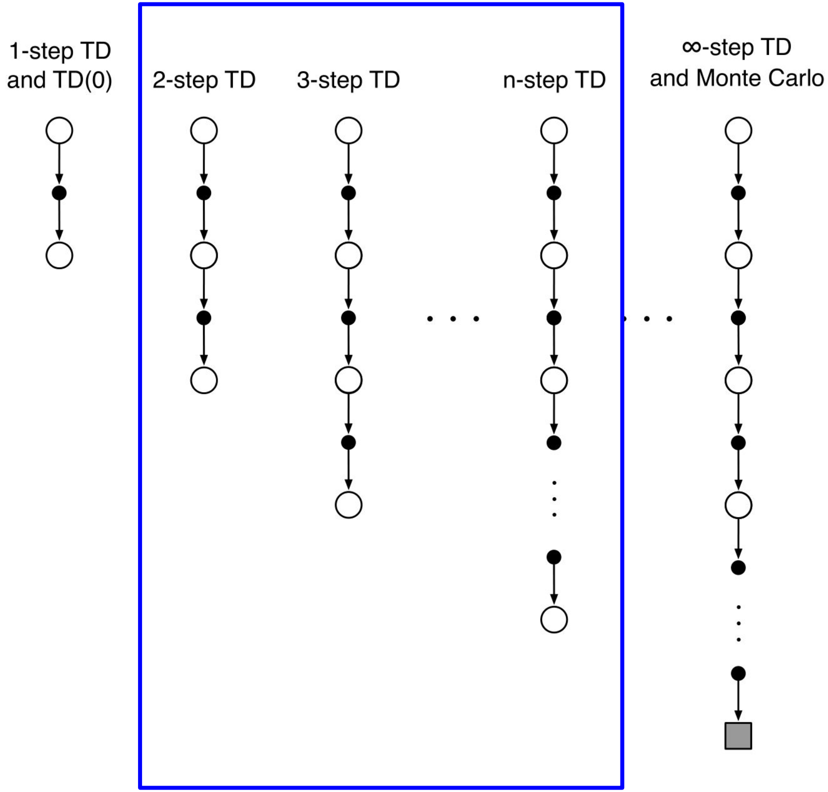
- 1-step TD / TD(0): wait until next time step

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[\overbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Bootstrapping target}} - V(S_t)}^{\text{TD error}} \Big]$$

n-step Bootstrapping

- Perform update based on intermediate number of rewards
- Freed from the “tyranny of the time step” of TD
  - Different time step for *action selection* (1) and *bootstrapping* interval (n)
- Called n-step TD since they still *bootstrap*

n-step Bootstrapping



endtoend.ai

n-step TD Prediction

- Use truncated *n-step return* as target
  - Use *n* rewards and bootstrap

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

- Needs future rewards not available at timestep t
- S_t cannot be updated until timestep $t+n$

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \big[G_{t:t+n} - V_{t+n-1}(S_t) \big], \qquad 0 \le t < T,$$

n-step TD Prediction: Pseudocode

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

endtoend.ai

n-step TD Prediction: Convergence

- The n-step return has the *error reduction property*
  - Expectation of n-step return is a better estimate of v_π than V_{t+n-1} in the worst-state sense

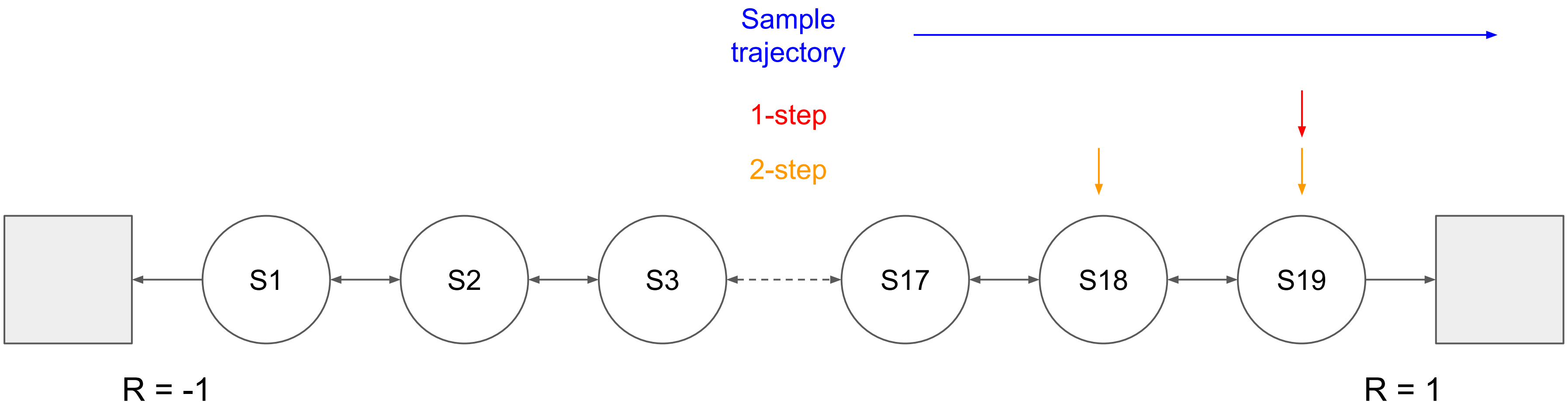
$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \leq \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|,$$

- Converges to true value under appropriate technical conditions

endtoend.ai

Random Walk Example

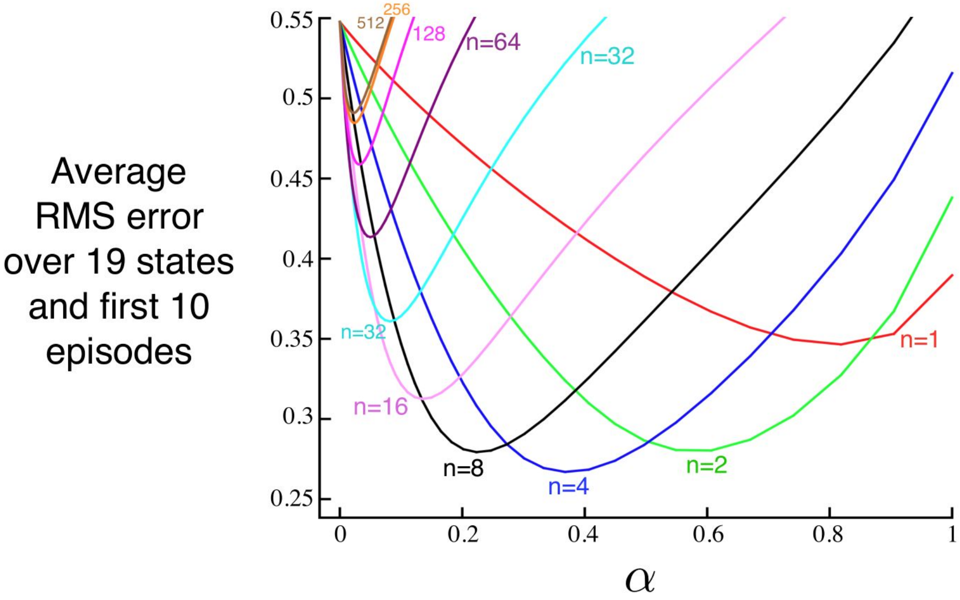
- Rewards only on exit (-1 on left exit, 1 on right exit)
- n-step return: propagate reward up to n latest states



endtoend.ai

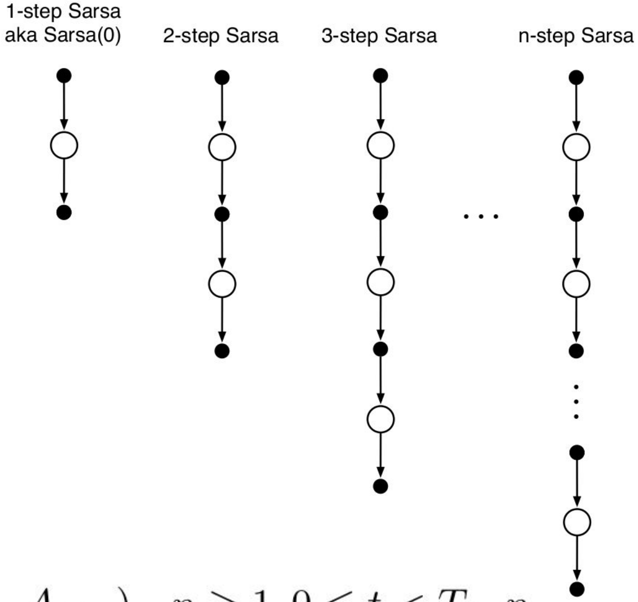
Random Walk Example: n-step TD Prediction

- Intermediate n does best



endtoend.ai

n-step Sarsa



- Extend n-step TD Prediction to Control (Sarsa)
  - Need to use Q instead of V
  - Use ε-greedy policy
- Redefine n-step return with Q

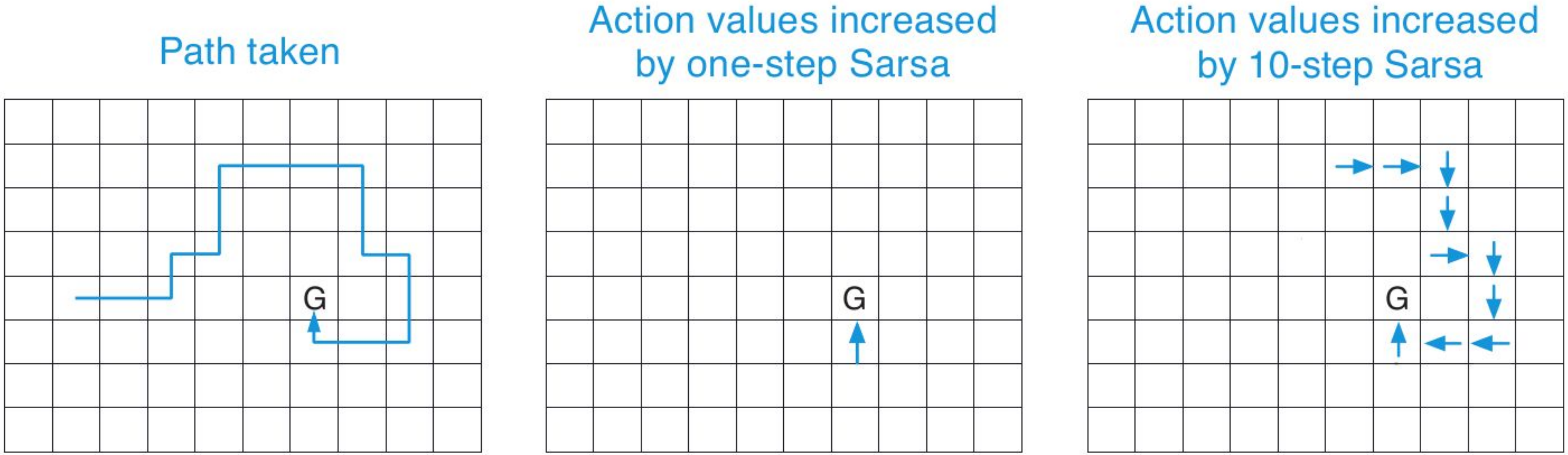
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n,$$

- Naturally extend to Sarsa

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

n-step Sarsa vs. Sarsa(0)

- Gridworld with nonzero reward only at the end
- n-step can learn much more from one episode



n-step Sarsa: Pseudocode

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 ≠ terminal
    Select and store an action A_0 ~ π(·|S_0)
    T ← ∞
    Loop for t = 0, 1, 2, ... :
    |   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 ← t + 1
    |       else:
    |           Select and store an action A_{t+1} ~ π(·|S_{t+1})
    |   τ ← t − n + 1    (τ is the time whose estimate is being updated)
    |   If τ ≥ 0:
    |       G ← Σ_{i=τ+1}^{min(τ+n,T)} γ^{i−τ−1} R_i
    |       If τ + n < T, then G ← G + γ^n Q(S_{τ+n}, A_{τ+n})                (G_{τ:τ+n})
    |       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 ε-greedy wrt Q
    Until τ = T − 1
```

endtoend.ai

n-step Expected Sarsa

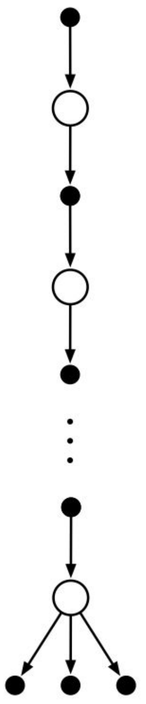
- Same update as Sarsa except the last element
  - Consider all possible actions in the last step
- Same n-step return as Sarsa except the last step

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

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a),$$

- Same update as Sarsa

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right], \qquad 0 \le t < T,$$



Off-policy n-step Learning

- Need *importance sampling*

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

- Update target policy's values with behavior policy's returns

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

- Generalizes the on-policy case
  - If $\pi = b$, then $\rho = 1$

Off-policy n-step Sarsa

- Update Q instead of V
- Importance sampling ratio starts one step later for Q values
  - A_t is already chosen

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

Off-policy n-step Sarsa: Pseudocode

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 $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:t+n-1})$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | 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 $\tau = T - 1$

endtoend.ai

Off-policy n-step Expected Sarsa

- Importance sampling ratio ends one step earlier for Expected Sarsa

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

- Use expected n-step return

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

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s, a),$$

Per-decision Off-policy Methods: Intuition*

- More efficient off-policy n-step method
- Write returns recursively:

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h},$$

$$G_{h:h} \doteq V_{h-1}(S_h).$$

- Naive importance sampling
  - If $\rho_t = 0$, $G_{t:h} = 0$
  - Estimate shrinks, higher variance

$$G_{t:h} \doteq \rho_t \left(R_{t+1} + \gamma G_{t+1:h}\right)$$

$$G_{t:h} \doteq \rho_t \left(R_{t+1} + \gamma G_{t+1:h}\right) + (1 - \rho_t) V_{h-1}(S_t),$$

Per-decision Off-policy Methods*

- Better: If $\rho_t = 0$, leave the estimate unchanged

$$G_{t:h} \doteq \rho_t \left(R_{t+1} + \gamma G_{t+1:h}\right) + \underbrace{(1-\rho_t)V_{h-1}(S_t),}_{\text{Control Variate}}$$

- Expected update is unchanged since $\mathbb{E}[\rho_t] = 1$

$$\mathbb{E}\left[\frac{\pi(A_k|S_k)}{b(A_k|S_k)}\right] \doteq \sum_a b(a|S_k)\frac{\pi(a|S_k)}{b(a|S_k)} = \sum_a \pi(a|S_k) = 1.$$

- Used with TD update *without* importance sampling

Per-decision Off-policy Methods: Q*

- Use Expected Sarsa's n-step return

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

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a),$$

- Off-policy form with control variate:

$$\begin{aligned} G_{t:h} &\doteq R_{t+1} + \gamma\Big(\rho_{t+1} G_{t+1:h} + \bar{V}_{h-1}(S_{t+1}) - \rho_{t+1} Q_{h-1}(S_{t+1}, A_{t+1})\Big), \\ &= R_{t+1} + \gamma\rho_{t+1}\Big(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\Big) + \gamma \bar{V}_{h-1}(S_{t+1}), \quad t < h \leq T. \end{aligned}$$

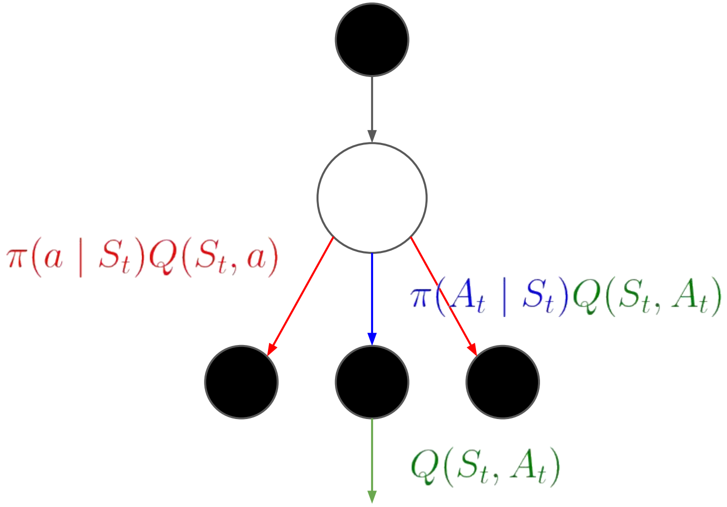
- Analogous to Expected Sarsa after combining with TD update algorithm

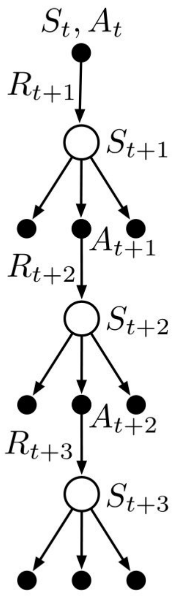
http://auai.org/uai2018/proceedings/papers/282.pdf

endtoend.ai

n-step Tree Backup Algorithm

- Off-policy *without* importance sampling
- Update from entire tree of estimated action values
  - Leaf action nodes (not selected) contribute to the target
  - Selected action nodes does not contribute but weighs all next-level action values





the 3-step tree-backup update

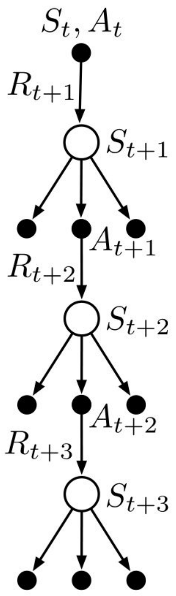
n-step Tree Backup Algorithm: n-step Return

- 1-step return

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

- 2-step return

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



the 3-step tree-backup update

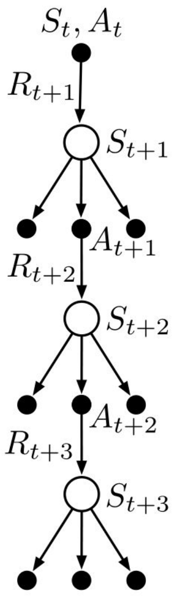
n-step Tree Backup Algorithm: n-step Return

- 2-step return

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

- n-step return

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



the 3-step tree-backup update

endtoend.ai

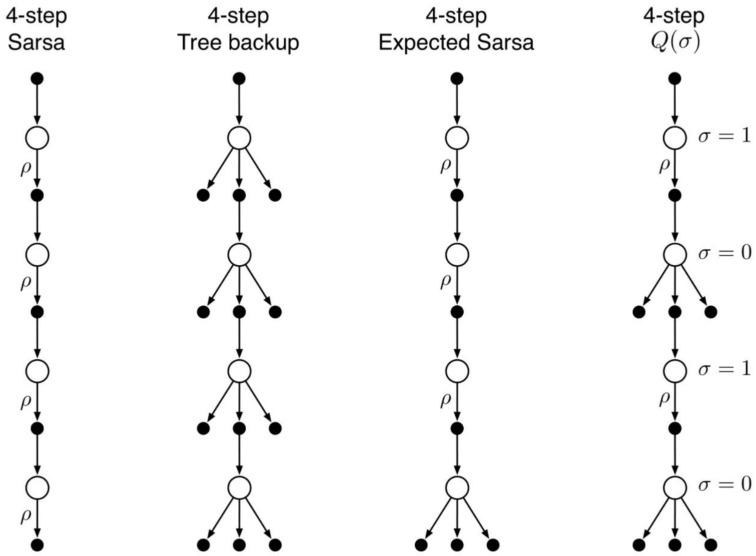
n-step Tree Backup Algorithm: Pseudocode

```
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
```

endtoend.ai

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

- Unify Sarsa, Tree Backup and Expected Sarsa
  - Decide on each step to use sample action (Sarsa) or expectation of all actions (Tree Backup)



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

- $\sigma_t \in [0,1]$: degree of sampling on timestep t

$$G_{t:h} \doteq R_{t+1} + \gamma\Big(\underline{\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1})}\Big)\Big(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\Big)$$
$$+ \gamma \bar{V}_{h-1}(S_{t+1}), \qquad (7.17)$$

- Slide linearly between two weights:
  - Sarsa: Importance sampling ratio ρ_{t+1}
  - Tree Backup: Policy probability $\pi(A_{t+1} \mid S_{t+1})$

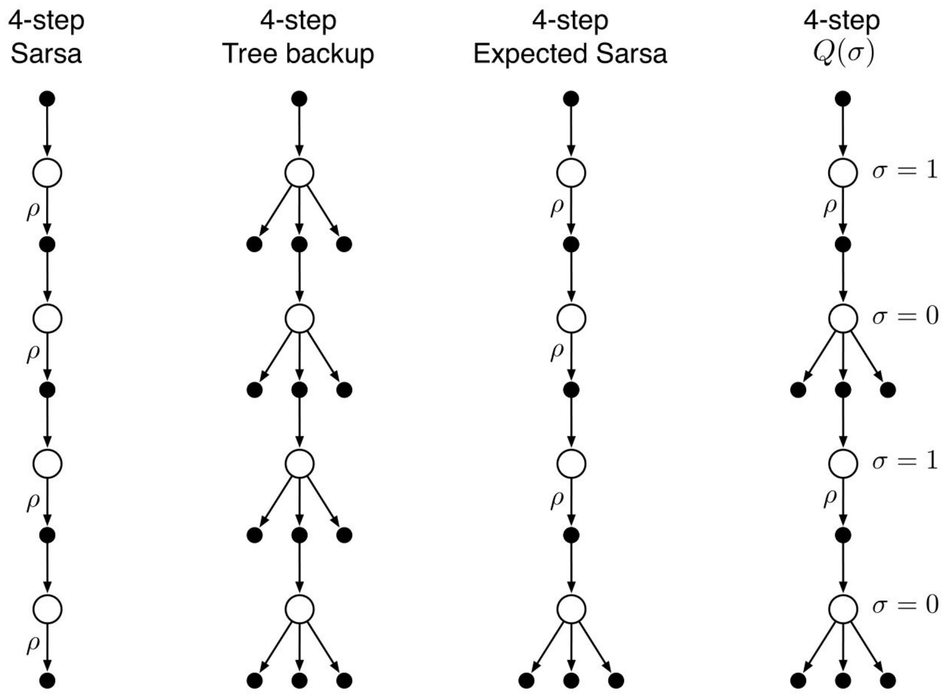
A Unifying Algorithm: n-step $Q(\sigma)$: Pseudocode*

```
Loop for each episode:
   Initialize and store S_0 ≠ terminal
   Choose and store an action A_0 ~ b(·|S_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 and store an action A_{t+1} ~ b(·|S_{t+1})
   |        Select and store σ_{t+1}
   |        Store π(A_{t+1}|S_{t+1}) / b(A_{t+1}|S_{t+1}) as ρ_{t+1}
   |  τ ← t − n + 1   (τ is the time whose estimate is being updated)
   |  If τ ≥ 0:
   |     G ← 0:
   |     Loop for k = min(t + 1, T) down through τ + 1:
   |        if k = T:
   |           G ← R_T
   |        else:
   |           V̄ ← Σ_a π(a|S_k)Q(S_k, a)
   |           G ← R_k + γ(σ_k ρ_k + (1 − σ_k)π(A_k|S_k))(G − Q(S_k, A_k)) + γV̄
   |     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
```

Summary

- n-step: Look ahead to the next *n* rewards, states, and actions
  - \+ Perform better than either MC or TD
  - \+ Escapes the *tyranny of the single time step*
  - \- Delay of *n* steps before learning
  - \- More memory and computation per timestep
- Extended to Eligibility Traces (Ch. 12)
  - \+ Minimize additional memory and computation
  - \- More complex
- Two approaches to off-policy n-step learning
  - Importance sampling: high variance
  - Tree backup: limited to few-step bootstrapping if policies are very different (even if n is large)

Summary



endtoend.ai

Thank you!

Original content from

- [Reinforcement Learning: An Introduction by Sutton and Barto]

You can find more content in

- github.com/seungjaeryanlee
- www.endtoend.ai