Theory and Methods for Reinforcement Learning

Prof. Volkan Cevher
volkan.cevher@epfl.ch

Lecture 5: n-step Bootstrapping

Laboratory for Information and Inference Systems (LIONS)
École Polytechnique Fédérale de Lausanne (EPFL)

EE-618 (Spring 2020)

License Information for Reinforcement Learning Slides

- This class:
  1. n-step TD Prediction
  2. n-step TD Control
  3. Eligibility Traces
- Next class:
  1. Value-based Methods for Deep RL

Recommended reading

- Chapters 7 & 12 in S. Sutton, and G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.

n-step bootstrapping

- Neither Monte-Carlo (MC) methods nor 1-step temporal-difference (TD) methods are always the best.

- n-step TD methods span a spectrum with MC methods at one end (∞-step) and one-step TD methods at the other (1-step).

- The best method is often an intermediate between the two extremes.

- n-step TD prediction and control.

n-step TD prediction

- Lets the TD target look n steps into the future.

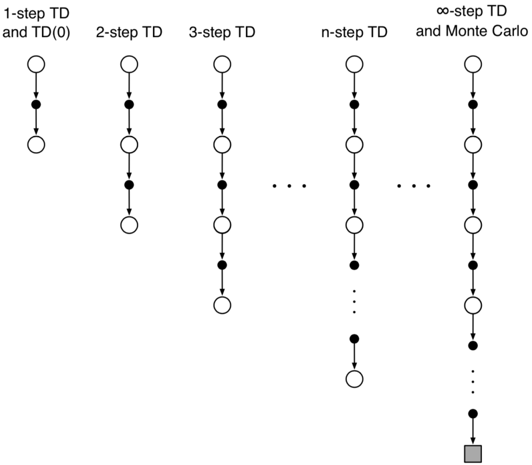


Figure: Backup Diagram for n-step TD methods, the two extreme ends are respectively 1-step TD and Monte-Carlo.

n-step return

Definition (n-step return)

Let T be the termination time step in a given episode, $\gamma \in [0, 1]$.

$$
\begin{aligned}
G_t^{(1)} &= R_{t+1} + \gamma V(S_{t+1}) && \textit{(one-step return)} \\
G_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) && \textit{(two-step return)} \\
G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) && \textit{(n-step return)} \\
G_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T && \textit{(complete return)}
\end{aligned}
$$

Note that $G_t^{(n)} = G_t^{(\infty)}$ if $t + n \geq T$.

- The n-step return computes discounted rewards for n steps, and uses the discounted $V(S_{t+n})$ as a proxy for the remaining terms.

- No real algorithm can use the n-step return until after it has seen R_{t+n}, as this would mean looking into the future.

n-step TD update

- Incremental update rule for the state-value prediction:

$$V(S_t) \; \leftarrow \; V(S_t) + \alpha \left[\text{target} - V(S_t) \right],$$

while the value of all the other states remains unchanged: $V(s) \leftarrow V(s)$, $\forall s \neq S_t$.

- Different targets can be used:
  - For one-step TD or TD(0): $\text{target} = G_t^{(1)}$
  - For two-step TD: $\text{target} = G_t^{(2)}$
  - For n-step TD: $\text{target} = G_t^{(n)}$
  - For MC: $\text{target} = G_t^{(\infty)}$

n-step TD Prediction Algorithm

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 $\tau = T - 1$

Error reduction property of n-step returns

Theorem (Error reduction property)

For all $n \geq 1$,

$$\max_{s \in \mathcal{S}} \left| \mathbb{E}\left[G_t^{(n)} \middle| S_t = s \right] - v_\pi(s) \right| \leq \gamma^n \max_{s \in \mathcal{S}} \left| V(s) - v_\pi(s) \right|.$$

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

- The error reduction property means that the worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V.

Example: Random walk

- Recall the 5-state Random walk example from Lecture 4.
- The outcome for ending up on the left is -1 and there are 19 states.

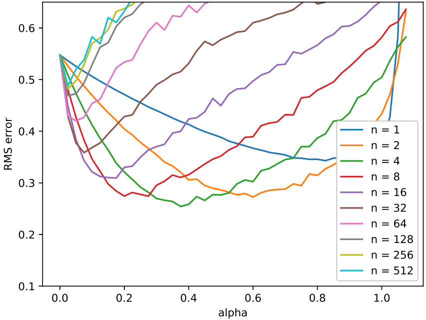


Figure: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task.

n-step TD control

- On-policy learning via n-step SARSA
- Off-policy learning with Importance Sampling
- Off-policy learning with n-step Tree Backup

Recall: SARSA Algorithm

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

Algorithm Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize: $Q(s, a)$, for all $s \in \mathcal{S}^+$ and $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

  Initialize state S

  Choose A from S using policy based on Q (e.g., ϵ-greedy)

  Loop for each step of episode:

    Take action A, observe reward R and next state S'

    Choose A' from S' using the policy based on Q (e.g., ϵ-greedy)

    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

    $S \leftarrow S'$; $A \leftarrow A'$

  until S is terminal

n-step SARSA

- We require the target policy π to be ϵ-greedy with respect to Q.
- Redefine the n-step return in terms of estimated action-values:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}),$$

with $G_t^{(n)} = G_t^{(\infty)}$ if $t + n \geq T$.

- n-step SARSA update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_t^{(n)} - Q(S_t, A_t) \right], \qquad 0 \leq t < T,$$

while the values of all other states remain unchanged: $Q(s,a) \leftarrow Q(s,a)$, for all $s \neq S_t, a \neq A_t$.

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 $\tau = T - 1$

n-step expected SARSA

- Redefine the n-step return as

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a \mid S_{t+n}) Q(S_{t+n}, a),$$

with $G_t^{(n)} = G_t^{(\infty)}$, if $t + n \geq T$.

- n-step expected SARSA update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_t^{(n)} - Q(S_t, A_t) \right],$$

while the values of all other states remain unchanged: $Q(s, a) \leftarrow Q(s, a)$, for all $s \neq S_t, a \neq A_t$.

Backup Diagrams for n-step Methods

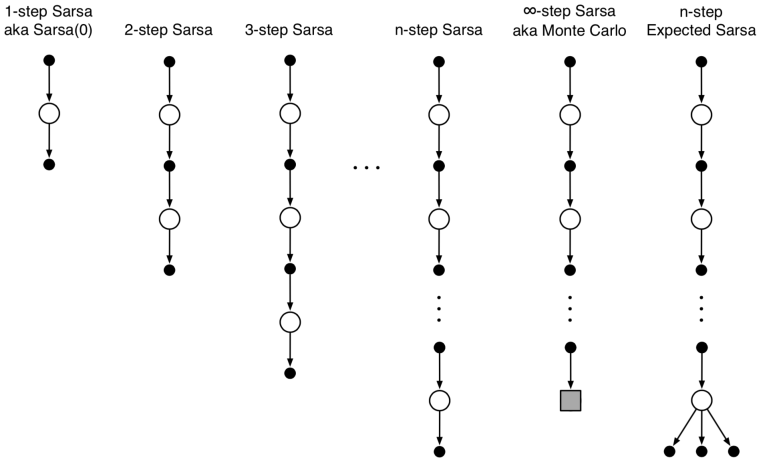


Figure: The backup diagrams for the spectrum of n-step methods for state-action values. They range from the one-step update of Sarsa(0) to the up-until-termination update of the MC method.

Example: Gridworld

- All action-state values and rewards are initialized to 0.
- Only the reward for G is set to 1.
- One-step SARSA focus on the last value.
- n-step SARSA “strengthens” last n actions.

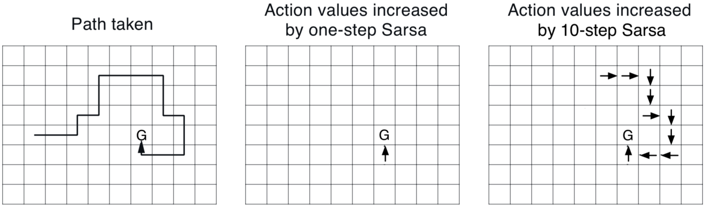


Figure: One-step SARSA strengthens the last action leading to the destination. n-step SARSA increases the action value for the last n actions. For a single episode, we could clearly see that multiple step approach learns more than its single step counterpart.

n-step off-policy learning

- *Off-policy learning:* Learn the value function for a policy π, while following another behaviour policy b.
  - π is the greedy policy w.r.t. current action-value function estimate.
  - b is a more exploratory policy (e.g., ϵ-greedy).

Definition (Importance sampling ratio)

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

- Off-policy n-step TD:

$$V(S_t) \leftarrow V(S_t) + \alpha \rho_t^{(n-1)} \left[G_t^{(n)} - V(S_t) \right], \qquad 0 \leq t < T.$$

n-step off-policy control

- Off-policy n-step SARSA:

$$Q(S_t, A_t) \;\leftarrow\; Q(S_t, A_t) + \alpha \rho_{t+1}^{(n-1)} \left[G_t^{(n)} - Q(S_t, A_t) \right].$$

- Note that the importance sampling ratio here starts and ends one step later than for n-step TD (for state value prediction).

- This is because we are selecting a *state-action* pair instead of only a state.

Off-policy n-step SARSA Algorithm

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 $\tau = T - 1$

n-step tree-backup algorithm: motivation

Motivation

Is off-policy learning possible without importance sampling?

Q-learning and Expected Sarsa do this for the one-step case, but is there a corresponding multi-step algorithm?

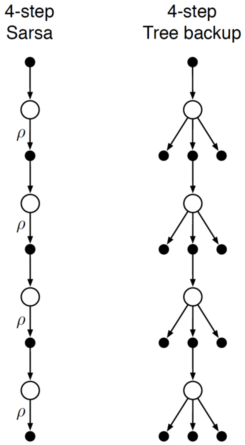
Answer: Yes! Use n-step tree backup.

n-step tree-backup algorithm

SARSA vs Tree-backup

Consider the backup diagram on the right. The estimated value for the top node can be updated in at least two ways:

- **So far** (SARSA): (discounted) rewards along the way + estimate for bottom nodes.
- **Tree-backup**: (discounted) rewards along the way + estimate for bottom nodes and *dangling* action nodes along the way.



Tree-backup return

- One-step return:

$$G_t^{(1)} = R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a).$$

- Two-step return:

$$\begin{aligned} G_t^{(2)} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) \\ &\quad + \gamma \pi(A_{t+1} \mid S_{t+1}) \left\{ R_{t+2} + \gamma \sum_a \pi(a \mid S_{t+2}) Q(S_{t+2}, a) \right\} \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1}^{(1)}. \end{aligned}$$

- n-step return:

$$G_t^{(n)} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1}^{(n-1)}.$$

n-step tree-backup algorithm

n-step Tree Backup for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given 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 as a fixed given policy
Parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  Store $Q(S_0, A_0)$ as Q_0
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$:
  | Take action A_t
  | Observe the next reward R; observe and store the next state as S_{t+1}
  | If S_{t+1} is terminal:
  | $T \leftarrow t + 1$
  | Store $R - Q_t$ as δ_t
  | else:
  | Store $R + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q_t$ as δ_t
  | Select arbitrarily and store an action as A_{t+1}
  | Store $Q(S_{t+1}, A_{t+1})$ as Q_{t+1}
  | Store $\pi(A_{t+1}|S_{t+1})$ as π_{t+1}
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $Z \leftarrow 1$
  | $G \leftarrow Q_\tau$
  | For $k = \tau, \ldots, \min(\tau + n - 1, T - 1)$:
  | $G \leftarrow G + Z\delta_k$
  | $Z \leftarrow \gamma Z \pi_{k+1}$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(a|S_\tau)$ is ε-greedy wrt $Q(S_\tau, \cdot)$
  Until $\tau = T - 1$

Eligibility traces: motivation

- n-step methods need to wait $n-1$ steps after the beginning of an episode before starting updates, and keeps running after the end of the episode.

- n-step methods do not make the best use use of a state as soon as it becomes available.

Motivation

How can we efficiently combine information from all time-steps?

Answer: Use eligibility traces.

Eligibility traces

- Eligibility traces unify and generalize TD and MC methods.
  - n-step TD methods also unify TD and MC.
  - But eligibility traces offer in addition:
    (i) an elegant algorithmic mechanism
    (ii) significant computational advantages.

- Eligibility traces produce a family of methods spanning a spectrum that has MC methods at one end ($\lambda = 1$) and one-step TD methods at the other ($\lambda = 0$).
  - In between $\lambda = 0$ and $\lambda = 1$ are intermediate methods that often perform better than either extreme method.

- Eligibility traces also provide a way of implementing Monte Carlo methods online and on continuing problems without episodes.

Averaging n-step returns

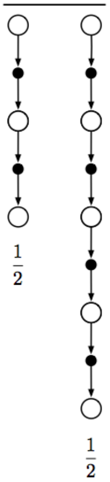
- We can average n-step returns over different n.

- e.g., average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

- Combines information from two different time-steps.

- How can we efficiently combine information from all time-steps?



λ-return

- The λ-return G_t^λ combines all n-step returns $G_t^{(n)}$:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

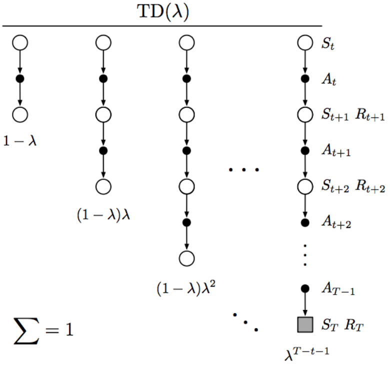
- Recall that $\sum_{n=0}^{\infty}\lambda^n = \frac{1}{1-\lambda}$ for all $\lambda \in [0,1]$.

- If T is the termination time step:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{T-t-1}G_t$$

- Forward-view of TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\left[G_t^\lambda - V(S_t)\right]$$



λ-return weighting function

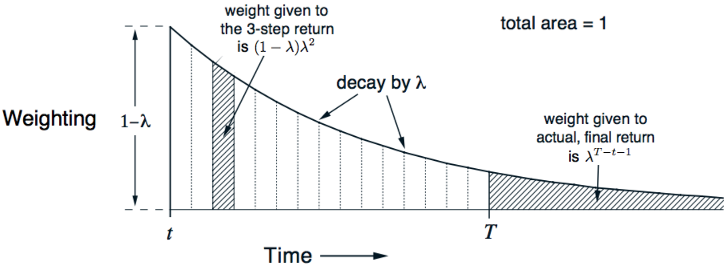


Figure: Weighting given in the λ-return to each of the n-step returns.

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

Forward-view TD(λ)

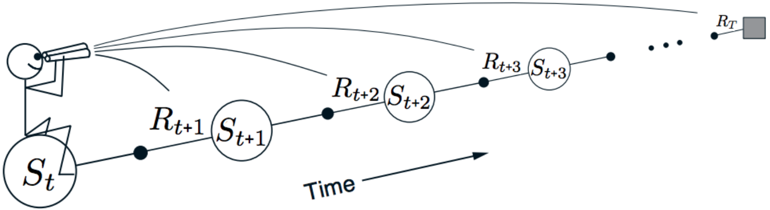


Figure: We decide how to update each state by looking forward to future rewards and states.

- Recall the forward view of TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t^\lambda - V(S_t) \right]$$

  - Updates the value function towards the λ-return
  - The forward view looks into the future to compute G_t^λ
  - Like MC, it can only be computed from complete episodes

Example: Random Walk

- The offline λ-return algorithm makes no changes to the weight vector during the episode. Then, at the end of the episode, a whole sequence of offline updates are made.

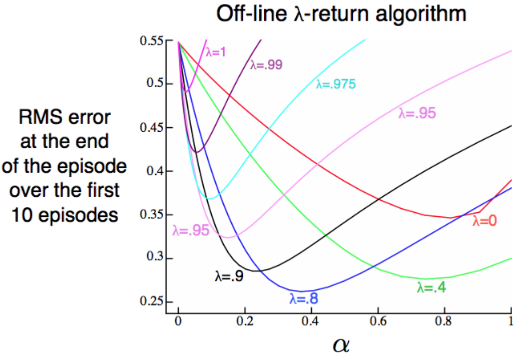


Figure: Performance of the offline λ-return algorithm in the 19-state random walk task.

Backward view of TD(λ)

- The forward view provides theory.

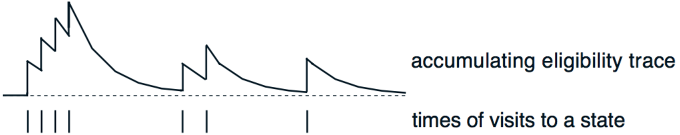
- The backward view provides a computationally efficient method through eligibility traces.

- Updates are performed online, at every step and from incomplete sequences.

Eligibility traces

- *Credit assignment problem:* did the bell or the light cause the shock?
- *Frequency heuristic:* assign credit to the most frequent states
- *Recency heuristic:* assign credit to most recent states
- Eligibility traces combine both heuristics

$$
\begin{aligned}
E_0(s) &= 0 \\
E_t(s) &= \gamma\lambda E_{t-1}(s) + 1_{\{S_t = s\}}
\end{aligned}
$$



Backward-view TD(λ)

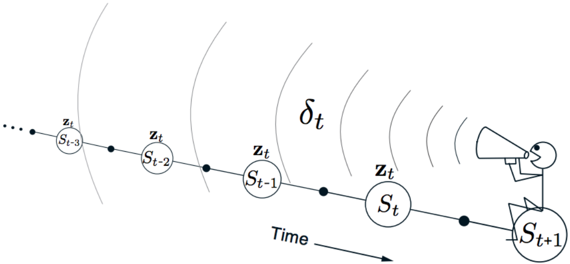


Figure: Each update depends on the current TD error combined with the current eligibility traces of past events.

- Keep an eligibility trace $E_t(s)$ for every state s.
- Compute the TD-error δ_t

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

- Update the value $V(s)$ of every state s

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

TD(λ) and TD(0)

- When $\lambda = 0$, only the current state is updated

$$\begin{aligned} E_t(s) &= 1_{\{S_t = s\}} \\ V(s) &\leftarrow V(s) + \alpha \delta_t E_t(s) \end{aligned}$$

- This is exactly equivalent to the TD(0) update

$$V(s) \leftarrow V(s) + \alpha \delta_t \text{ if } s = S_t$$

TD(λ) and MC

- When $\lambda = 1$, the credit is deferred until the end of the episode.
- Consider episodic environments with offline updates.
- Over the course of an episode, the total update for TD(1) is the same as the total update for MC

Theorem

The sum of offline updates is identical for forward-view and backward-view TD(λ)

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) \; = \; \sum_{t=1}^{T} \alpha \left[G_t^{\lambda} - V(S_t) \right] 1_{\{S_t = s\}}.$$

TD(1) and MC

- Consider an episode where s is visited only once at time-step k
- The eligibility trace of TD(1) discounts the time since the visit

$$\begin{aligned} E_t(s) &= \gamma E_{t-1}(s) + 1_{\{S_t = s\}} \\ &= \begin{cases} 0 & \text{if } t < k \\ \gamma^{t-k} & \text{if } t \geq k \end{cases}. \end{aligned}$$

- The TD(1) updates accumulate the error online

$$\sum_{t=1}^{T-1} \alpha \delta_t E_t(s) = \alpha \sum_{t=k}^{T-1} \delta_t \gamma^{t-k} = \alpha \left[G_k - V(S_k) \right].$$

- By the end of the episode, they accumulate the total error

$$\delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \cdots + \gamma^{T-1-k} \delta_{T-1}.$$

Telescoping in TD(1)

- When $\lambda = 1$, the sum of TD errors telescopes into the MC error,

$$
\begin{aligned}
& \delta_k + \gamma\delta_{k+1} + \gamma^2\delta_{k+2} + \cdots + \gamma^{T-1-k}\delta_{T-1} \\
= \; & R_{k+1} + \gamma V(S_{k+1}) - V(S_k) \\
+ \; & \gamma R_{k+2} + \gamma^2 V(S_{k+2}) - \gamma V(S_{k+1}) \\
+ \; & \gamma^2 R_{k+3} + \gamma^3 V(S_{k+3}) - \gamma^2 V(S_{k+2}) \\
+ \; & \ldots \\
+ \; & \gamma^{T-1-k} R_T + \gamma^{T-k} V(S_T) - \gamma^{T-1-k} V(S_{T-1}) \\
= \; & R_{k+1} + \cancel{\gamma V(S_{k+1})} - V(S_k) \\
+ \; & \gamma R_{k+2} + \cancel{\gamma^2 V(S_{k+2})} - \cancel{\gamma V(S_{k+1})} \\
+ \; & \gamma^2 R_{k+3} + \cancel{\gamma^3 V(S_{k+3})} - \cancel{\gamma^2 V(S_{k+2})} \\
+ \; & \ldots \\
+ \; & \gamma^{T-1-k} R_T + \cancelto{0}{\gamma^{T-k} V(S_T)} - \cancel{\gamma^{T-1-k} V(S_{T-1})} \\
= \; & R_{k+1} + \gamma R_{k+2} + \gamma^2 R_{k+3} + \cdots + \gamma^{T-1-k} R_T - V(S_k) \\
= \; & G_k - V(S_k)
\end{aligned}
$$

TD(λ) and TD(1)

- TD(1) is roughly equivalent to every-visit Monte-Carlo.
- Error is accumulated online, step-by-step
- If the value function is only updated offline at end of episode, then the total update is exactly the same as MC.

Telescoping in TD(λ)

- For general λ, TD errors also telescope to the λ-error, $G_t^\lambda - V(S_t)$

$$
\begin{aligned}
G_t^\lambda - V(S_t) &= -V(S_t) + (1-\lambda)\lambda^0 \left[R_{t+1} + \gamma V(S_{t+1})\right] \\
&+ (1-\lambda)\lambda^1 \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})\right] \\
&+ (1-\lambda)\lambda^2 \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})\right] \\
&+ \ldots \\
&= -V(S_t) + (\gamma\lambda)^0 \left[R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})\right] \\
&+ (\gamma\lambda)^1 \left[R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})\right] \\
&+ (\gamma\lambda)^2 \left[R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})\right] \\
&+ \ldots \\
&= (\gamma\lambda)^0 \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \\
&+ (\gamma\lambda)^1 \left[R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1})\right] \\
&+ (\gamma\lambda)^2 \left[R_{t+3} + \gamma V(S_{t+3}) - V(S_{t+2})\right] \\
&+ \ldots \\
&= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots
\end{aligned}
$$

Forward- and backward-TD(λ)

- Consider an episode where s is visited only once at time-step k
- The eligibility trace of TD(λ) discounts the time since the visit

$$
\begin{aligned}
E_t(s) &= \gamma\lambda E_{t-1}(s) + 1_{\{S_t=s\}} \\
&= \begin{cases} 0 & \text{if } t < k \\ (\gamma\lambda)^{t-k} & \text{if } t \geq k \end{cases}
\end{aligned}
$$

- Backward-TD(λ) updates accumulate the error online

$$
\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \alpha \sum_{t=k}^{T} \delta_t (\gamma\lambda)^{t-k} = \alpha \left[G_k^{\lambda} - V(S_k) \right]
$$

- By the end of the episode, they accumulate the total error for the λ-return.

Offline equivalence of forward- and backward-TD

- Offline updates:
  - updates are accumulated within an episode
  - but applied in batch at the end of the episode

Recall: n-step SARSA

Definition (n-step return)

Let T be the termination time step in a given episode.

$$\begin{aligned} G_t^{(1)} &= R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) && \textit{(one-step return)} \\ G_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}) && \textit{(two-step return)} \\ G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}) && \textit{(n-step return)} \\ G_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T && \textit{(complete return)} \end{aligned}$$

Note that $G_t^{(n)} = G_t^{(\infty)}$, if $t + n \geq T$.

- n-step SARSA update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_t^{(n)} - Q(S_t, A_t) \right]$$

Forward view Sarsa(λ)

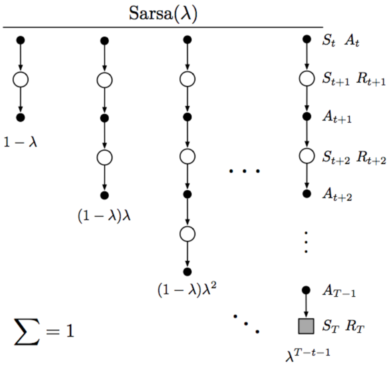
- The return G_t^λ combines the n-step returns $G_t^{(n)}$ for all n.

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- Recall that $\sum_{n=0}^{\infty}\lambda^n = \frac{1}{1-\lambda}$ for all $\lambda \in [0,1]$.

- Forward view Sarsa(λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[G_t^\lambda - Q(S_t, A_t)\right]$$



Backward view Sarsa(λ)

- Just like TD(λ), we use eligibility traces in an online algorithm.
- Sarsa(λ) has one eligibility trace for each *state-action pair*L

$$
\begin{aligned}
E_0(s,a) &= 0 \\
E_t(s,a) &= \gamma\lambda E_{t-1}(s,a) + 1_{\{S_t=s, A_t=a\}}
\end{aligned}
$$

- Compute the TD-error δ_t for every state-action pair (s, a)

$$
\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)
$$

- Update $Q(s,a)$ for all (s, a)

$$
Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)
$$

Sarsa(λ) Algorithm

Sarsa(λ)

Algorithm Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize: $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop for each episode:

  $E(s, a) = 0$, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

  Initialize S, A

  Loop for each step of episode:

    Take action A, observe reward R and next state S'

    Choose A' from S' using the policy based on Q (e.g., ϵ-greedy)

    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

    $E(S, A) \leftarrow E(S, A) + 1$

    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta E(s, a)$

      $E(S, A) \leftarrow \lambda\gamma E(S, A)$

    $S \leftarrow S'$; $A \leftarrow A'$

  until S is terminal

References