Lecture 06: Multi-Step Bootstrapping

Davit Ghazaryan

March 6, 2025

American University of Armenia

Lets unify MC and TD learning

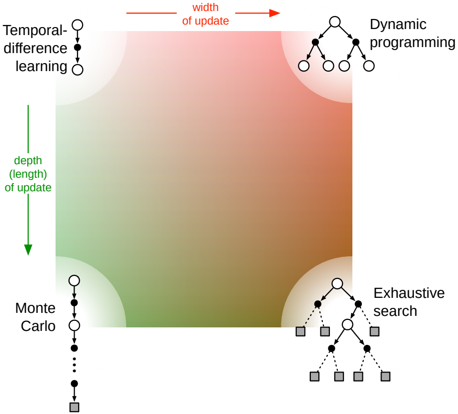

Fig. 6.1: MC and TD are the 'extreme options' in terms of the update's depth: what about intermediate solutions? (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Table of contents

n-step bootstrapping idea

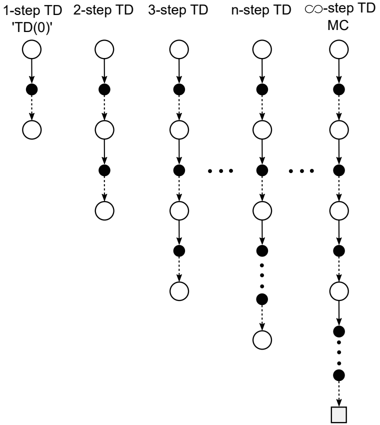

Fig. 6.2: Different backup diagrams of n-step state-value prediction methods

- n-step update: consider n rewards plus estimated value n-steps later (bootstrapping).

n-step bootstrapping idea

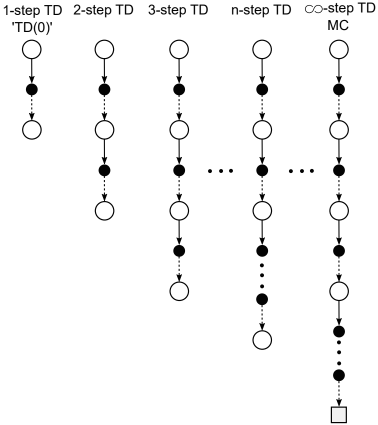

Fig. 6.2: Different backup diagrams of n-step state-value prediction methods

- n-step update: consider n rewards plus estimated value n-steps later (bootstrapping).
- Consequence: Estimate update is available only after an n-step delay.

n-step bootstrapping idea

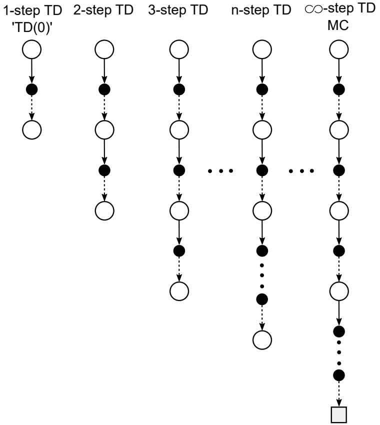

Fig. 6.2: Different backup diagrams of n-step state-value prediction methods

- n-step update: consider n rewards plus estimated value n-steps later (bootstrapping).
- Consequence: Estimate update is available only after an n-step delay.
- TD(0) and MC are special cases included in n-step prediction.

Formal notation (1)

Recap the update targets for the incremental prediction methods

- Monte Carlo: builds on the complete sampled return series

$$G_{t:T} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T. \tag{6.1}$$

  - $G_{t:T}$ denotes that all steps until termination at T are considered to derive an estimate target adressing step t.

Formal notation (1)

Recap the update targets for the incremental prediction methods

- Monte Carlo: builds on the complete sampled return series

$$G_{t:T} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T. \tag{6.1}$$

  - $G_{t:T}$ denotes that all steps until termination at T are considered to derive an estimate target adressing step t.

- TD(0): utilizes a one-step bootstrapped return

$$G_{t:t+1} = R_{t+1} + \gamma \hat{v}_t(s_{t+1}). \tag{6.2}$$

  - For TD(0), $G_{t:t+1}$ highlights that only one future sampled reward step is considered before bootstrapping.
  - $\hat{v}_t$ is an estimate of v_π at time step t.

Formal notation (2)

n-step state-value prediction target

Now, the target is generalized to an arbitrary n-step target:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}_{t+n-1}(s_{t+n}). \tag{6.3}$$

Formal notation (2)

n-step state-value prediction target

Now, the target is generalized to an arbitrary n-step target:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}_{t+n-1}(s_{t+n}). \tag{6.3}$$

- Approximation of full return series truncated after n-steps.
- If $t + n \geq T$ (i.e., n-step prediction exceeds termination lookahead), then all missing terms are considered zero.

Formal notation (2)

n-step state-value prediction target

Now, the target is generalized to an arbitrary n-step target:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}_{t+n-1}(s_{t+n}). \tag{6.3}$$

- Approximation of full return series truncated after n-steps.
- If $t + n \geq T$ (i.e., n-step prediction exceeds termination lookahead), then all missing terms are considered zero.

n-step TD

The state-value estimate using the n-step return approximation is

$$\hat{v}_{t+n}(s_t) = \hat{v}_{t+n-1}(s_t) + \alpha \left[G_{t:t+n} - \hat{v}_{t+n-1}(s_t) \right], \quad 0 \leq t < T. \tag{6.4}$$

Formal notation (2)

n-step state-value prediction target

Now, the target is generalized to an arbitrary n-step target:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}_{t+n-1}(s_{t+n}). \tag{6.3}$$

- Approximation of full return series truncated after n-steps.
- If $t + n \geq T$ (i.e., n-step prediction exceeds termination lookahead), then all missing terms are considered zero.

n-step TD

The state-value estimate using the n-step return approximation is

$$\hat{v}_{t+n}(s_t) = \hat{v}_{t+n-1}(s_t) + \alpha \left[G_{t:t+n} - \hat{v}_{t+n-1}(s_t) \right], \quad 0 \leq t < T. \tag{6.4}$$

- Delay of n-steps before $\hat{v}(s)$ is updated.
- Additional auxiliary update steps required at the end of each episode.

Convergence

Theorem 6.1: Error reduction property

The worst error of the expected n-step return is always less than or equal to γ^n times the worst error under the estimate $\hat{v}_{t+n-1}$:

$$\max_s \left| \mathbb{E}_\pi \left[G_{t:t+n} | S_t = s \right] - v_\pi(s) \right| \leq \gamma^n \max_s \left| \hat{v}_{t+n-1}(s) - v_\pi(s) \right|. \tag{6.5}$$

Convergence

Theorem 6.1: Error reduction property

The worst error of the expected n-step return is always less than or equal to γ^n times the worst error under the estimate $\hat{v}_{t+n-1}$:

$$\max_s \left|\mathbb{E}_\pi\left[G_{t:t+n} | S_t = s\right] - v_\pi(s)\right| \leq \gamma^n \max_s \left|\hat{v}_{t+n-1}(s) - v_\pi(s)\right|. \tag{6.5}$$

- Assuming an infinite number of steps/episodes and an appropriate step-size control n-step TD prediction converges to the true value.

Convergence

Theorem 6.1: Error reduction property

The worst error of the expected n-step return is always less than or equal to γ^n times the worst error under the estimate $\hat{v}_{t+n-1}$:

$$\max_s \left|\mathbb{E}_\pi\left[G_{t:t+n}|S_t = s\right] - v_\pi(s)\right| \leq \gamma^n \max_s \left|\hat{v}_{t+n-1}(s) - v_\pi(s)\right|. \tag{6.5}$$

- Assuming an infinite number of steps/episodes and an appropriate step-size control n-step TD prediction converges to the true value.
- In a more practical framework with limited number of steps/episodes:
  - Choosing the best n-step lookahead horizon is an engineering degree of freedom.
  - This is highly application-dependent (i.e., no predefined optimum).
  - Prediction/estimation errors can remain due to limited data.

Algorithmic implementation: n-step TD prediction

input: a policy π to be evaluated, **parameter:** step size $\alpha \in (0, 1]$, prediction steps $n \in \mathbb{Z}^+$
init: $\hat{v}(s)\,\forall\, s \in \mathcal{S}$ arbitrary except $v_0(s) = 0$ if s is terminal
for $j = 1, \ldots, J$ *episodes* **do**
    initialize and store s_0;
    $T \leftarrow \infty$;
    repeat $t = 0, 1, 2, \ldots$
        if $t < T$ **then**
            take action from $\pi(s_t)$, observe and store s_{t+1} and R_{t+1};
            if s_{t+1} is terminal: $T \leftarrow t + 1$;
        $\tau \leftarrow t - n + 1$ (τ time index for estimate update);
        if $\tau \geq 0$ **then**
            $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$;
            if $\tau + n < T$: $G \leftarrow G + \gamma^n \hat{v}(s_{\tau+n})$;
            $\hat{v}(s_\tau) \leftarrow \hat{v}(s_\tau) + \alpha \left[G - \hat{v}(s_\tau)\right]$;
    until $\tau = T - 1$;

Algo. 6.1: n-step TD prediction (output is an estimate $\hat{v}_\pi(s)$)

Example: 19 state random walk

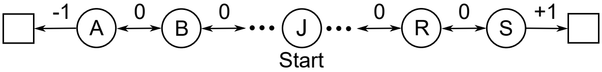

Fig. 6.3: Exemplary random walk Markov reward process (MRP)

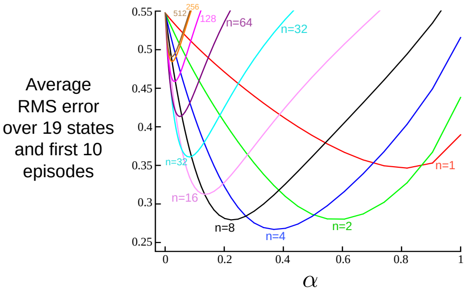

Fig. 6.4: n-step TD performance (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

- Early stage performance after only 10 episodes
- Averaged over 100 independent runs
- Best result here: $n = 4, \alpha \approx 0.4$
- Picture may change for longer episodes (no generalizable results)

Table of contents

Transfer the n-step approach to state-action values (1)

- For on-policy control by SARSA action-value estimates are required.
- Recap the one-step action-value update as required for 'SARSA(0)':

$$\hat{q}(s_t, a_t) \leftarrow \hat{q}(s_t, a_t) + \alpha \left[\underbrace{R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1})}_{\text{target } G} - \hat{q}(s_t, a_t) \right] . \tag{6.6}$$

Transfer the n-step approach to state-action values (1)

- For on-policy control by SARSA action-value estimates are required.
- Recap the one-step action-value update as required for 'SARSA(0)':

$$\hat{q}(s_t, a_t) \leftarrow \hat{q}(s_t, a_t) + \alpha \left[\underbrace{R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1})}_{\text{target } G} - \hat{q}(s_t, a_t) \right]. \tag{6.6}$$

n-step state-action value prediction target

Analog to n-step TD, the state-action value target is rewritten as:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}_{t+n-1}(s_{t+n}, a_{t+n}). \tag{6.7}$$

Transfer the n-step approach to state-action values (1)

- For on-policy control by SARSA action-value estimates are required.
- Recap the one-step action-value update as required for 'SARSA(0)':

$$\hat{q}(s_t, a_t) \leftarrow \hat{q}(s_t, a_t) + \alpha \left[\underbrace{R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1})}_{\text{target } G} - \hat{q}(s_t, a_t) \right] . \tag{6.6}$$

n-step state-action value prediction target

Analog to n-step TD, the state-action value target is rewritten as:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}_{t+n-1}(s_{t+n}, a_{t+n}). \tag{6.7}$$

- Again, if an episode terminates within the lookahead horizon $(t + n \geq T)$ the target is equal to the Monte Carlo update:

$$G_{t:t+n} = G_t. \tag{6.8}$$

Transfer the n-step approach to state-action values (2)

- For n-step expected SARSA, the update is similar but the state-action value estimate at step $t+n$ becomes the expected approximate value of s under the target policy valid at time step k:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|s)\hat{q}_t(s,a). \tag{6.9}$$

Transfer the n-step approach to state-action values (2)

- For n-step expected SARSA, the update is similar but the state-action value estimate at step $t+n$ becomes the expected approximate value of s under the target policy valid at time step k:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|s)\hat{q}_t(s,a). \tag{6.9}$$

- Finally, the modified n-step targets can be directly integrated to the state-action value estimate update rule of SARSA:

n-step SARSA

$$\hat{q}_{t+n}(s_t,a_t) = \hat{q}_{t+n-1}(s_t,a_t) + \alpha\left[G_{t:t+n} - \hat{q}_{t+n-1}(s_t,a_t)\right], \quad 0 \le t < T. \tag{6.10}$$

n-step bootstrapping for state-action values

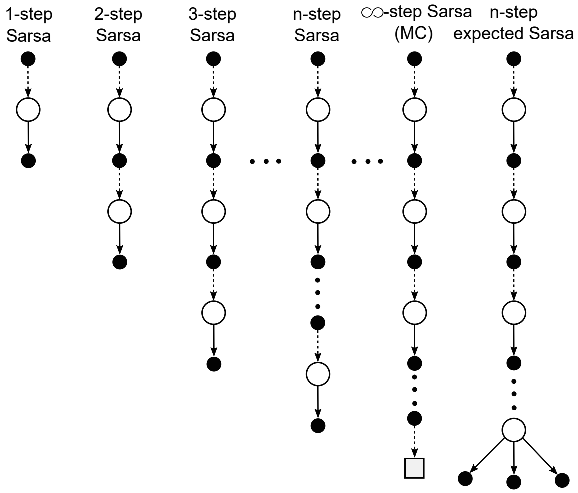

Fig. 6.5: Different backup diagrams of n-step state-action value update targets

Algorithmic implementation: n-step SARSA

parameter: $\alpha \in (0,1]$, $n \in \mathbb{Z}^+$, $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$
init: $\hat{q}(s,a)$ arbitrarily (except terminal states) $\forall\ \{s \in \mathcal{S}, a \in \mathcal{A}\}$
init: π to be ε-greedy with respect to $\hat{q}$ or to a given, fixed policy
for $j = 1, \dots, J$ *episodes* **do**
  initialize s_0 and action $a_0 \sim \pi(\cdot|s_0)$ and store them;
  $T \leftarrow \infty$;
  repeat $t = 0, 1, 2, \dots$
    if $t < T$ **then**
      take action a_t, observe and store s_{t+1} and R_{t+1};
      if s_{t+1} *is terminal* **then** $T \leftarrow t+1$ **else** store $a_{t+1} \sim \pi(\cdot|s_{t+1})$;
    $\tau \leftarrow t - n + 1$ (τ time index for estimate update);
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$;
      if $\tau + n < T$: $G \leftarrow G + \gamma^n \hat{q}(s_{\tau+n}, a_{\tau+n})$;
      $\hat{q}(s_\tau, a_\tau) \leftarrow \hat{q}(s_\tau, a_\tau) + \alpha\left[G - \hat{q}(s_\tau, a_\tau)\right]$;
      if $\pi \approx \pi^*$ is being learned, ensure $\pi(\cdot|s_\tau)$ is ε-greedy w.r.t $\hat{q}$;
  until $\tau = T - 1$;

Algo. 6.2: n-step SARSA (output is an estimate $\hat{q}_\pi$ or $\hat{q}^*$)

Illustration with grid-world example

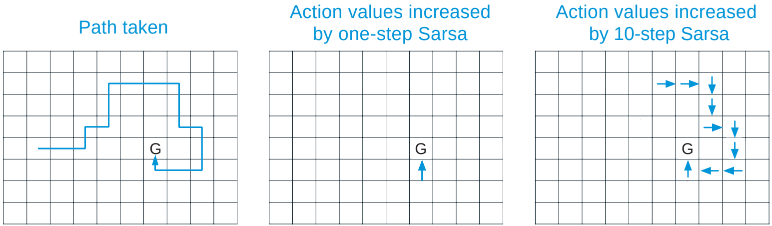

Fig. 6.6: Executed updates (highlighted by arrows) for different n-step SARSA implementations during an episode (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

- For one-step SARSA, one state-action value is updated.

Illustration with grid-world example

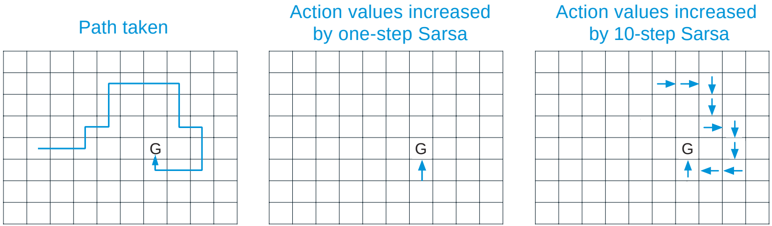

Fig. 6.6: Executed updates (highlighted by arrows) for different n-step SARSA implementations during an episode (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

- For one-step SARSA, one state-action value is updated.
- For ten-step SARSA, ten state-action values are updated.

Illustration with grid-world example

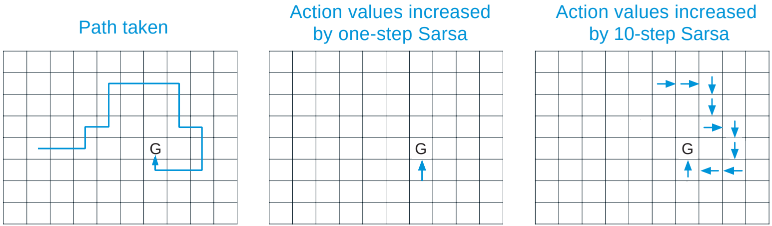

Fig. 6.6: Executed updates (highlighted by arrows) for different n-step SARSA implementations during an episode (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

- For one-step SARSA, one state-action value is updated.
- For ten-step SARSA, ten state-action values are updated.
- Consequence: a trade-off between the resulting learning delay and the number of updated state-action values results.

Table of contents

Recap on off-policy learning with importance sampling

Consider two separate policies in order to break the on-policy optimality trade-off:

- Behavior policy $b(a|s)$: Explores in order to generate experience.
- Target policy $\pi(a|s)$: Learns from that experience to become the optimal policy.

Recap on off-policy learning with importance sampling

Consider two separate policies in order to break the on-policy optimality trade-off:

- Behavior policy $b(a|s)$: Explores in order to generate experience.
- Target policy $\pi(a|s)$: Learns from that experience to become the optimal policy.
- Important requirement is coverage: Every action taken under π must be (at least occasionally) taken under b, too. Hence, it follows:

$$\pi(a|s) > 0 \Rightarrow b(a|s) > 0 \quad \forall \{s \in \mathcal{S}, a \in \mathcal{A}\}\,. \tag{6.11}$$

Recap on off-policy learning with importance sampling

Consider two separate policies in order to break the on-policy optimality trade-off:

- Behavior policy $b(a|s)$: Explores in order to generate experience.
- Target policy $\pi(a|s)$: Learns from that experience to become the optimal policy.
- Important requirement is coverage: Every action taken under π must be (at least occasionally) taken under b, too. Hence, it follows:

$$\pi(a|s) > 0 \Rightarrow b(a|s) > 0 \quad \forall \{s \in \mathcal{S}, a \in \mathcal{A}\}\,. \tag{6.11}$$

Importance sampling ratio

The relative probability of a trajectory under the target and behavior policy, the importance sampling ratio, from sample step t to T is:

$$\rho_{t:T} = \frac{\prod_t^{T-1} \pi(a_t|s_t)p(s_{t+1}|s_t, a_t)}{\prod_t^{T-1} b(a_t|s_t)p(s_{t+1}|s_t, a_t)} = \frac{\prod_t^{T-1} \pi(a_t|s_t)}{\prod_t^{T-1} b(a_t|s_t)}. \tag{6.12}$$

Transfer importance sampling to n-step updates

For a straightforward n-step off-policy TD-style update, just weight the update by the importance sampling ratio:

$$\hat{v}_{t+n}(s_t) = \hat{v}_{t+n-1}(s_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - \hat{v}_{t+n-1}(s_t) \right], \quad 0 \leq t < T,$$

$$\rho_{t:h} = \prod_{t}^{\min(h,T-1)} \frac{\pi(a_t|s_t)}{b(a_t|s_t)}. \tag{6.13}$$

- $\rho_{t:t+n-1}$ is the relative probability under the two polices taking n actions from a_t to a_{t+n}.

Transfer importance sampling to n-step updates

For a straightforward n-step off-policy TD-style update, just weight the update by the importance sampling ratio:

$$
\begin{aligned}
\hat{v}_{t+n}(s_t) &= \hat{v}_{t+n-1}(s_t) + \alpha \rho_{t:t+n-1}\left[G_{t:t+n} - \hat{v}_{t+n-1}(s_t)\right], \quad 0 \leq t < T, \\
\rho_{t:h} &= \prod_{t}^{\min(h,T-1)} \frac{\pi(a_t|s_t)}{b(a_t|s_t)}.
\end{aligned} \tag{6.13}
$$

- $\rho_{t:t+n-1}$ is the relative probability under the two polices taking n actions from a_t to a_{t+n}.

Analog, an n-step off-policy SARSA-style update exists:

$$
\begin{aligned}
\hat{q}_{t+n}(s_t, a_t) = \hat{q}_{t+n-1}&(s_t, a_t) \\
&+ \alpha \rho_{t+1:t+n}\left[G_{t:t+n} - \hat{q}_{t+n-1}(s_t, a_t)\right], \quad 0 \leq t < T.
\end{aligned} \tag{6.14}
$$

- Here, ρ starts and ends one step later compared to the TD case since state-action pairs are updated.

Algorithmic implementation: off-policy n-step TD-based prediction

input: a target policy π and a behavior policy b with coverage of π
parameter: step size $\alpha \in (0, 1]$, prediction steps $n \in \mathbb{Z}^+$
init: $\hat{v}(s)\,\forall\, s \in \mathcal{S}$ arbitrary except $v_0(s) = 0$ if s is terminal
for $j = 1, \ldots, J$ *episodes* **do**
    initialize and store s_0 and set $T \leftarrow \infty$;
    repeat $t = 0, 1, 2, \ldots$
        if $t < T$ **then**
            take action from $b(s_t)$, observe and store s_{t+1} and R_{t+1};
            if s_{t+1} is terminal: $T \leftarrow t + 1$;
        $\tau \leftarrow t - n + 1$ (τ time index for estimate update);
        if $\tau \geq 0$ **then**
            $\rho \leftarrow \prod_{i=\tau}^{\min(\tau+n-2, T-1)} \frac{\pi(a_i|s_t)}{b(a_i|s_i)}$;
            $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$;
            if $\tau + n < T$: $G \leftarrow G + \gamma^n \hat{v}(s_{\tau+n})$;
            $\hat{v}(s_\tau) \leftarrow \hat{v}(s_\tau) + \alpha\rho\,[G - \hat{v}(s_\tau)]$;
    until $\tau = T - 1$;

Algo. 6.3: Off-policy n-step TD prediction (output is an estimate $\hat{v}_\pi(x)$)

Algorithmic implementation: off-policy n-step SARSA

input: an arbitrary behavior policy b with $b(a|s) > 0 \,\forall\, \{s \in \mathcal{S}, a \in \mathcal{A}\}$
parameter: $\alpha \in (0, 1]$, $n \in \mathbb{Z}^+$, $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$
init: $\hat{q}(s, a) \,\forall\, \{s \in \mathcal{S}, a \in \mathcal{A}\}$ and a policy π to be greedy with respect to $\hat{q}$ or to a given, fixed policy
for $j = 1, \ldots, J$ *episodes* **do**
    initialize s_0 and action $a_0 \sim b(\cdot|s_0)$ and store them, set also $T \leftarrow \infty$;
    repeat $t = 0, 1, 2, \ldots$
        if $t < T$ **then**
            take action $a_t \sim b(\cdot|s_t)$, observe and store s_{t+1} and R_{t+1};
            if s_{t+1} *is terminal* **then** $T \leftarrow t + 1$ **else** store $a_{t+1} \sim b(\cdot|s_{t+1})$;
        $\tau \leftarrow t - n + 1$ (τ time index for estimate update);
        if $\tau \geq 0$ **then**
            $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(a_i|s_i)}{b(a_i|s_i)}$;
            $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$;
            if $\tau + n < T$: $G \leftarrow G + \gamma^n \hat{q}(s_{\tau+n}, a_{\tau+n})$;
            $\hat{q}(s_\tau, a_\tau) \leftarrow \hat{q}(s_\tau, a_\tau) + \alpha\rho \left[G - \hat{q}(s_\tau, a_\tau)\right]$;
            if $\pi \approx \pi^*$ is being learned, ensure $\pi(\cdot|s_\tau)$ is ε-greedy w.r.t to $\hat{q}$;
    until $\tau = T - 1$;

Algo. 6.4: Off-policy n-step SARSA (output is an estimate $\hat{q}_\pi$ or $\hat{q}^*$)

Table of contents

Averaging of n-step returns

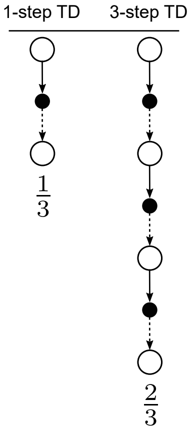

Fig. 6.7: Exemplary averaging of n-step returns

- Averaging different n-step returns is possible without introducing a bias (if sum of weights is one).
- Example on the left:

$$G = \frac{1}{3} G_{t:t+1} + \frac{2}{3} G_{t:t+3}$$

- Horizontal line in backup diagram indicates the averaging.

Averaging of n-step returns

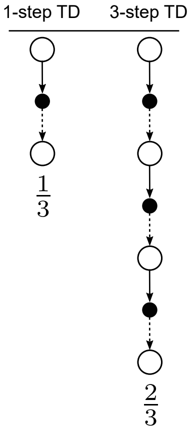

Fig. 6.7: Exemplary averaging of n-step returns

- Averaging different n-step returns is possible without introducing a bias (if sum of weights is one).
- Example on the left:

$$G = \frac{1}{3}G_{t:t+1} + \frac{2}{3}G_{t:t+3}$$

- Horizontal line in backup diagram indicates the averaging.
- Enables additional degree of freedom to reduce prediction error.
- Such updates are called compound updates.

λ-return (1)

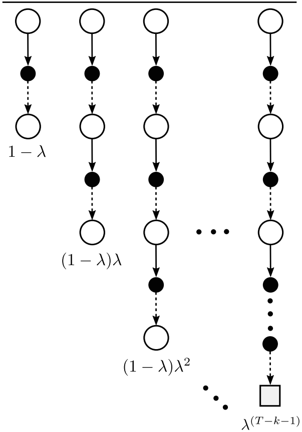

Fig. 6.8: Backup diagram for λ-returns

- λ-return: is a compound update with exponentially decaying weights:

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{(n-1)} G_{t:t+n} \,. \qquad (6.15)$$

- Parameter is $\lambda \in \{\mathbb{R} | 0 \leq \lambda \leq 1\}$.
- Geometric series of weights is one:

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

λ-return (2)

- Rewrite λ-return for episodic tasks with termination at $t = T$:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{(n-1)} G_{t:t+n} + \lambda^{T-t-1} G_t \,. \tag{6.16}$$

- Return G_t after termination is weighted with residual weight λ^{T-t-1}.

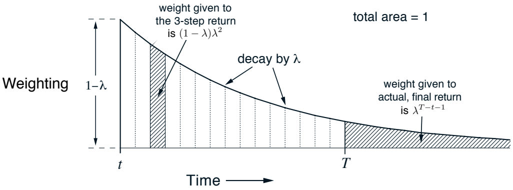

Fig. 6.9: Weighting overview in λ-return series (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

λ-return (2)

- Rewrite λ-return for episodic tasks with termination at $t = T$:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{(n-1)} G_{t:t+n} + \lambda^{T-t-1} G_t \,. \tag{6.16}$$

- Return G_t after termination is weighted with residual weight λ^{T-t-1}.
- Above, (6.16) includes two special cases:
  - If $\lambda = 0$: becomes TD(0) update.
  - If $\lambda = 1$: becomes MC update.

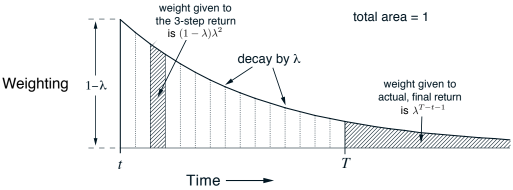

Fig. 6.9: Weighting overview in λ-return series (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Truncated λ-returns for continuing tasks

- Using λ-returns as in (6.15) is not feasible for continuing tasks.
- One would have to wait infinitely long to receive the trajectory.

Truncated λ-returns for continuing tasks

- Using λ-returns as in (6.15) is not feasible for continuing tasks.
- One would have to wait infinitely long to receive the trajectory.
- Intuitive approximation: truncate λ-return after h steps

$$G_{t:h}^{\lambda} = (1-\lambda) \sum_{n=1}^{h-t-1} \lambda^{(n-1)} G_{t:t+n} + \lambda^{h-t-1} G_{t:h} \,. \tag{6.17}$$

- Horizon h divides continuing tasks in rolling episodes.

Forward view

- Both, n-step and λ-return updates, are based on a forward view.
- We have to wait for future states and rewards to arrive before we are able to perform an update.

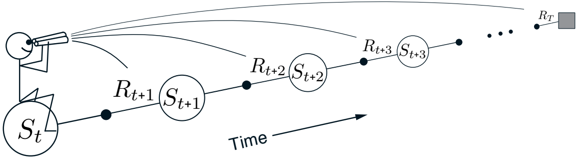

Fig. 6.10: The forward view: an update of the current state value is evaluated by future transitions (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Forward view

- Both, n-step and λ-return updates, are based on a forward view.
- We have to wait for future states and rewards to arrive before we are able to perform an update.
- Currently, λ-returns are only an alternative to n-step updates with different weighting options.

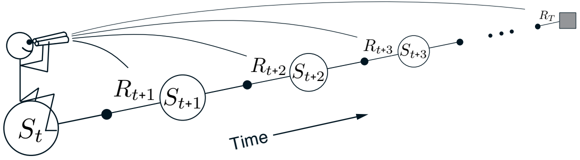

Fig. 6.10: The forward view: an update of the current state value is evaluated by future transitions (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Backward view of TD(λ)

General idea:

- Use λ-weighted returns looking into the past.
- Implement this in a recursive fashion to save memory.

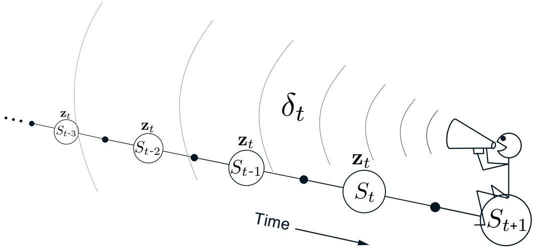

Fig. 6.11: The backward view: an update of the current state value is evaluated based on a trace of past transitions (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Backward view of TD(λ)

General idea:

- Use λ-weighted returns looking into the past.
- Implement this in a recursive fashion to save memory.
- Therefore, an eligibility trace z_t denoting the importance of past events to the current state update is introduced.

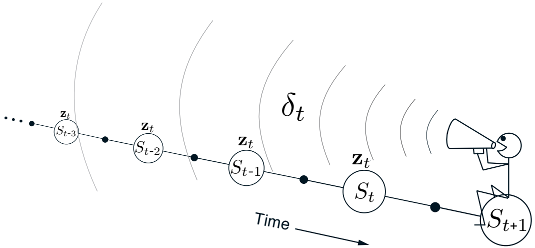

Fig. 6.11: The backward view: an update of the current state value is evaluated based on a trace of past transitions (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Eligibility trace

The eligibility trace $z_t(s) \in \mathbb{R}$ is defined and tracked for each state s separately:

$$
\begin{aligned}
z_0(s) &= 0, \\
z_t(s) &= \gamma\lambda z_{t-1}(s) + \begin{cases} 0, & \text{if } s_t \neq s, \\ 1, & \text{if } s_t = s. \end{cases}
\end{aligned}
\tag{6.18}
$$

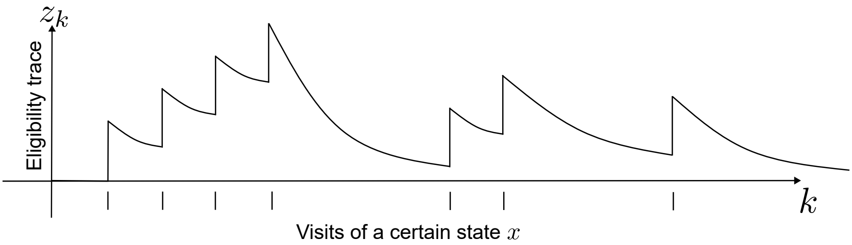

Fig. 6.12: Simplified representation of updating an eligibility trace of an arbitrary state in a finite MDP

TD(λ) updates using eligibility traces

Based on the eligibility trace definition from (6.18) we can modify our value estimates:

TD(λ) state-value update

The TD(λ) state-value update is:

$$\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha \left[R_{t+1} + \gamma \hat{v}(s_{t+1}) - \hat{v}(s_t)\right] z_t(s_t). \tag{6.19}$$

TD(λ) updates using eligibility traces

Based on the eligibility trace definition from (6.18) we can modify our value estimates:

TD(λ) state-value update

The TD(λ) state-value update is:

$$\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha \left[R_{t+1} + \gamma \hat{v}(s_{t+1}) - \hat{v}(s_t)\right] z_t(s_t). \tag{6.19}$$

SARSA(λ) action-value update

The SARSA(λ) action-value update is:

$$\hat{q}(s_t, a_t) \leftarrow \hat{q}(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}) - \hat{q}(s_t, a_t)\right] z_t(s_t, a_t). \tag{6.20}$$

TD(λ) updates using eligibility traces

Based on the eligibility trace definition from (6.18) we can modify our value estimates:

TD(λ) state-value update

The TD(λ) state-value update is:

$$\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha \left[R_{t+1} + \gamma \hat{v}(s_{t+1}) - \hat{v}(s_t)\right] z_t(s_t). \tag{6.19}$$

SARSA(λ) action-value update

The SARSA(λ) action-value update is:

$$\hat{q}(s_t, a_t) \leftarrow \hat{q}(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}) - \hat{q}(s_t, a_t)\right] z_t(s_t, a_t). \tag{6.20}$$

Already known prediction and control methods can be modified accordingly. In contrast to n-step forward updates, one can conclude:

- Advantage: recursive updates based on past updates (no additional waiting time),
- Disadvantage: effort for storing an eligibility trace for each state (scaling problem).

Algorithmic implementation: SARSA(λ)

parameter: $\alpha \in (0, 1]$, $\lambda \in (0, 1]$, $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$
init: $\hat{q}(s, a)$ arbitrarily (except terminal states) $\forall \ \{s \in \mathcal{S}, a \in \mathcal{A}\}$
init: π to be ε-greedy with respect to $\hat{q}$ or to a given, fixed policy
for $j = 1, \ldots, J$ *episodes* **do**
  initialize s_0 and action $a_0 \sim \pi(\cdot|s_0)$;
  initialize $z_0(s, a) = 0 \ \forall \ \{s \in \mathcal{S}, a \in \mathcal{A}\}$
  repeat
    take action a_t, observe s_{t+1} and R_{t+1};
    choose $a_{t+1} \sim \pi(\cdot|s_{t+1})$

$$z_t(s, a) \leftarrow \gamma\lambda z_{t-1}(s, a) + \begin{cases} 0, & \text{if } s_t \neq s \text{ or } a_t \neq a, \\ 1, & \text{if } s_t = s \text{ and } a_t = a. \end{cases} \quad \forall \ \{s \in \mathcal{S}, a \in \mathcal{A}\}$$

    $\delta \leftarrow R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}) - \hat{q}(s_t, a_t)$
    $\hat{q}(s, a) \leftarrow \hat{q}(s, a) + \alpha\delta z_t(s, a) \ \forall \ \{s \in \mathcal{S}, a \in \mathcal{A}\}$
    $t \leftarrow t + 1$;
  until s_t *is terminal*;

Algo. 6.5: SARSA(λ) (output is an estimate $\hat{q}_\pi$ or $\hat{q}^*$)

SARSA learning comparison in gridworld example

- λ can be interpreted as the discounting factor acting on the eligibility trace (see right-most panel below).
- Intuitive interpretation: more recent transitions are more certain/relevant for the current update step.

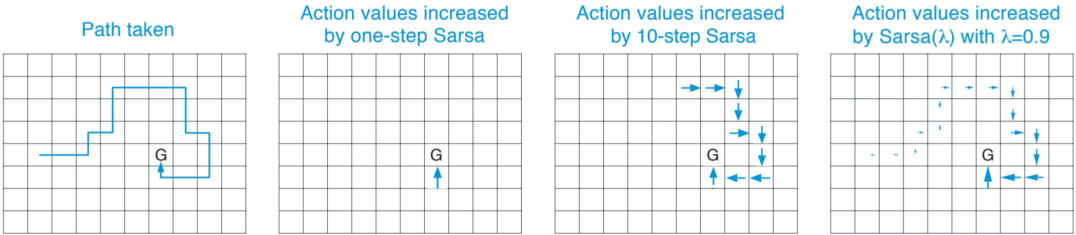

Fig. 6.13: SARSA variants after an arbitrary episode within a gridworld environment – arrows indicate action-value change starting from initially zero estimates (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018)

Summary

- n-step updates allow for an intermediate solution in between temporal difference and Monte Carlo:
  - $n = 1$: TD as special case,
  - $n = T$: MC as special case.

Summary

- n-step updates allow for an intermediate solution in between temporal difference and Monte Carlo:
  - $n = 1$: TD as special case,
  - $n = T$: MC as special case.
- The parameter n is a delicate degree of freedom:
  - It contains a trade-off between the learning delay and uncertainty reduction when considering more or less steps.
  - Choosing it is non-trivial and sometimes more art than science.
- λ-returns lead to compound updates which introduce an exponential weighting to visited states.
  - Rationale: states which have been already visited long ago are less important for the current learning step.
- TD(λ) transfers this idea into a recursive, backward oriented approach.
  - Eligibility traces store the long-term visiting history of each state in a recursive fashion.