Lecture 06: $n$-Step Bootstrapping

Oliver Wallscheid
Figure 8.11: A slice through the space of reinforcement learning methods, highlighting the two of the most important dimensions explored in Part I of this book: the depth and width of the updates.

Ranging from one-step TD updates to full-return Monte Carlo updates. Between these is a spectrum including methods based on \( n \)-step updates (and in Chapter 12 we will extend this to mixtures of \( n \)-step updates such as the \( \beta \)-updates implemented by eligibility traces).

Dynamic programming methods are shown in the extreme upper-right corner of the space because they involve one-step expected updates. The lower-right corner is the extreme case of expected updates so deep that they run all the way to terminal states (or, in a continuing task, until discounting has reduced the contribution of any further rewards to a negligible level). This is the case of exhaustive search. Intermediate methods along this dimension include heuristic search and related methods that search and update up to a limited depth, perhaps selectively. There are also methods that are intermediate along the horizontal dimension. These include methods that mix expected and sample updates, as well as the possibility of methods that mix samples and distributions within a single update. The interior of the square is filled in to represent the space of all such intermediate methods.

A third dimension that we have emphasized in this book is the binary distinction between on-policy and off-policy methods. In the former case, the agent learns the value function for the policy it is currently following, whereas in the latter case it learns the...
Table of Contents

1. $n$-Step TD Prediction
2. $n$-Step Sarsa
3. $n$-Step Off-Policy Learning with Importance Sampling
4. $n$-Step Off-Policy Learning with Tree Backup Approach
5. Bringing Everything Together: $n$-Step $Q(\sigma)$
$n$-Step Bootstrapping Idea

- $n$-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.

Fig. 6.2: Different backup diagrams of $n$-step state-value prediction methods
Recap the **update targets** for the incremental prediction methods (4.3):

- **Monte Carlo:** Builds on the complete sampled return series

\[ g_{k:T} = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots + \gamma^{T-k-1} r_T. \]  \hfill (6.1)

- \( g_{k:T} \) denotes that all steps until termination at \( T \) are considered to derive an estimate target addressing step \( k \).

- **TD(0):** Utilizes a one-step bootstrapped return

\[ g_{k:k+1} = r_{k+1} + \gamma \hat{v}_k(x_{k+1}). \]  \hfill (6.2)

- For TD(0), \( g_{k:k+1} \) highlights that only one future sampled reward step is considered before bootstrapping.
- \( \hat{v}_k \) is an estimate of \( v_\pi \) at time step \( k \).
Formal Notation (2)

\( n \)-step state-value prediction target

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

\[
g_{k:k+n} = r_{k+1} + \gamma r_{k+2} + \cdots + \gamma^{n-1} r_{k+n} + \gamma^n \hat{v}_{k+n-1}(x_{k+n}).
\] (6.3)

- Approximation of full return series truncated after \( n \)-steps.
- If \( k + n \geq T \) (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

\[
\hat{v}_{k+n}(x_k) = \hat{v}_{k+n-1}(x_k) + \alpha [g_{k:k+n} - \hat{v}_{k+n-1}(x_k)], \quad 0 \leq k < T.
\] (6.4)

- Delay of \( n \)-steps before \( \hat{v}(x) \) is updated.
- Additional auxiliary update steps required at the end of each episode.
Theorem 6.1: Error reduction property

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

$$\max_x |\mathbb{E}_\pi [G_{k:k+n} | X_k = x] - v_\pi(x)| \leq \gamma^n \max_x |\hat{v}_{k+n-1}(x) - v_\pi(x)|.$$  \hspace{1cm} (6.5)

- Assuming an infinite number of steps/episodes and an appropriate step-size control according to Theo. 5.1, $n$-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 \( \pi \) to be evaluated

**parameter:** step size \( \alpha \in (0, 1] \), prediction steps \( n \in \mathbb{Z}^+ \)

**init:** \( \hat{v}(x) \forall x \in X \) arbitrary except \( v_0(x) = 0 \) if \( x \) is terminal

**for** \( j = 1, \ldots, J \) **episodes** **do**

- initialize and store \( x_0 \);
- \( T \leftarrow \infty \);
- **repeat** \( k = 0, 1, 2, \ldots \)
  - **if** \( k < T \) **then**
    - take action from \( \pi(x_k) \), observe and store \( x_{k+1} \) and \( r_{k+1} \);
      - if \( x_{k+1} \) is terminal: \( T \leftarrow k + 1 \);
  - \( \tau \leftarrow k - n + 1 \) (\( \tau \) 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 \( \tau + n < T \): \( g \leftarrow g + \gamma^n \hat{v}(x_{\tau+n}) \);
    - \( \hat{v}(x_\tau) \leftarrow \hat{v}(x_\tau) + \alpha [g - \hat{v}(x_\tau)] \);
- **until** \( \tau = T - 1 \);

**Algo. 6.1:** \( n \)-step TD prediction (output is an estimate \( \hat{v}_\pi(x) \))
7.2. n-step Sarsa

How can n-step methods be used not just for prediction, but for control? In this section we show how n-step methods can be combined with Sarsa in a straightforward way to...

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, CC BY-NC-ND 2.0)
Table of Contents

1. $n$-Step TD Prediction

2. $n$-Step Sarsa

3. $n$-Step Off-Policy Learning with Importance Sampling

4. $n$-Step Off-Policy Learning with Tree Backup Approach

5. Bringing Everything Together: $n$-Step $Q(\sigma)$
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)'

$$
\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \alpha \left[ r_{k+1} + \gamma \hat{q}(x_{k+1}, u_{k+1}) - \hat{q}(x_k, u_k) \right].
$$

(6.6)

$n$-step state-action value prediction target

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

$$
g_{k:k+n} = r_{k+1} + \gamma r_{k+2} + \cdots + \gamma^{n-1} r_{k+n} + \gamma^n \hat{q}_{k+n-1}(x_{k+n}, u_{k+n}).
$$

(6.7)

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

$$
g_{k:k+n} = g_k.
$$

(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 $k + n$ becomes the expected approximate value of $x$ under the target policy valid at time step $k$:

$$g_{k:k+n} = r_{k+1} + \gamma r_{k+2} + \cdots + \gamma^{n-1} r_{k+n} + \gamma^n \sum_u \pi(u|x) \hat{q}_k(x, u). \quad (6.9)$$

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

$n$-step Sarsa

$$\hat{q}_{k+n}(x_k, u_k) = \hat{q}_{k+n-1}(x_k, u_k) + \alpha \left[ g_{k:k+n} - \hat{q}_{k+n-1}(x_k, u_k) \right], \quad 0 \leq k < T. \quad (6.10)$$
$n$-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:** $\hat{q}(x, u)$ arbitrarily (except terminal states) $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$

**init:** $\pi$ to be $\varepsilon$-greedy with respect to $\hat{q}$ or to a given, fixed policy

**for** $j = 1, \ldots, J$ **episodes** **do**

- initialize $x_0$ and action $u_0 \sim \pi(\cdot|x_0)$ and store them;
- $T \leftarrow \infty$;
- **repeat** $k = 0, 1, 2, \ldots$
  - **if** $k < T$ **then**
    - take action $u_k$, observe and store $x_{k+1}$ and $r_{k+1}$;
    - **if** $x_{k+1}$ **is terminal** **then** $T \leftarrow k + 1$ **else** store $u_{k+1} \sim \pi(\cdot|x_{k+1})$;
  - $\tau \leftarrow k - n + 1$ ($\tau$ 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** $\tau + n < T$: $g \leftarrow g + \gamma^n \hat{q}(x_{\tau+n}, u_{\tau+n})$;
    - $\hat{q}(x_{\tau}, u_{\tau}) \leftarrow \hat{q}(x_{\tau}, u_{\tau}) + \alpha [g - \hat{q}(x_{\tau}, u_{\tau})]$;
    - **if** $\pi \approx \pi^*$ is being learned, ensure $\pi(\cdot|x_{\tau})$ is $\varepsilon$-greedy w.r.t to $\hat{q}$;
- **until** $\tau = T - 1$;

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

n-step Sarsa for estimating $Q \mapsto q^\star$ or $q^\pi$

Initialize $Q(s, a)$ arbitrarily, for all $s \in S$, $a \in A$.

Initialize $\pi$ to be "-$\text{greedy with respect to}$ $Q$, or to a fixed given policy.

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\gamma > 0$, 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:

1. Initialize and store $S_0$, $\text{terminal}$.
2. Select and store an action $A_0 \sim \pi(\cdot|S_0)$.
3. $T_1$ 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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$.
     - $\tau \leftarrow t \mod n + 1$ ($\tau$ is the time whose estimate is being updated).
     - If $\tau = 0$:
       - $G_P \leftarrow \min(\tau + n, T)$
       - $i \leftarrow \tau + 1$
     - If $\tau + n < T$, then
       - $G_{\tau + n} \leftarrow Q(S_\tau, A_\tau) + \alpha [G_{\tau + n} - Q(S_\tau, A_\tau)]$.

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, CC BY-NC-ND 2.0)

- For one-step Sarsa, one state-action value is updated.
- For ten-step Sarsa, ten state-action values are updated.
- In the latter case, much more is learned during one episode.
- Nevertheless, a trade-off between the resulting learning delay and the number of updated state-action values remains.
Table of Contents

1. $n$-Step TD Prediction

2. $n$-Step Sarsa

3. $n$-Step Off-Policy Learning with Importance Sampling

4. $n$-Step Off-Policy Learning with Tree Backup Approach

5. Bringing Everything Together: $n$-Step $Q(\sigma)$
Recap on Off-Policy Learning with Importance Sampling

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

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

$$\pi(u|x) > 0 \Rightarrow b(u|x) > 0 \quad \forall \{x \in X, u \in U\}. \quad (6.11)$$

**Importance sampling ratio (revision from Def. 4.2)**

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

$$\rho_{k:T} = \frac{\prod_{k}^{T-1} \pi(u_k|x_k)p(x_{k+1}|x_k, u_k)}{\prod_{k}^{T-1} b(u_k|x_k)p(x_{k+1}|x_k, u_k)} = \frac{\prod_{k}^{T-1} \pi(u_k|x_k)}{\prod_{k}^{T-1} b(u_k|x_k)}. \quad (6.12)$$
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:

\[
\hat{v}_{k+n}(x_k) = \hat{v}_{k+n-1}(x_k) + \alpha \rho_{k:k+n-1} [g_{k:k+n} - \hat{v}_{k+n-1}(x_k)], \quad 0 \leq k < T,
\]

\[
\rho_{k:h} = \min(h,T-1) \prod_k \frac{\pi(u_k|x_k)}{b(u_k|x_k)}.
\]  

\[(6.13)\]

\( \rho_{k:k+n-1} \) is the relative probability under the two policies taking \( n \) actions from \( u_k \) to \( u_{k+n} \).

Analog, an \( n \)-step off-policy Sarsa-style update exists:

\[
\hat{q}_{k+n}(x_k, u_k) = \hat{q}_{k+n-1}(x_k, u_k)
\]

\[
+ \alpha \rho_{k+1:k+n} [g_{k:k+n} - \hat{q}_{k+n-1}(x_k, u_k)], \quad 0 \leq k < T.
\]

\[(6.14)\]

\( \rho \) starts and ends one step later compared to the TD case since state-action pairs are updated.
Implementation: Off-Policy \(n\)-step TD-Based Prediction

**input:** a target policy \(\pi\) and a behavior policy \(b\) with coverage of \(\pi\)  
**parameter:** step size \(\alpha \in (0, 1]\), prediction steps \(n \in \mathbb{Z}^+\) 
**init:** \(\hat{v}(x) \forall x \in \mathcal{X}\) arbitrary except \(v_0(x) = 0\) if \(x\) is terminal 

for \(j = 1, \ldots, J\) episodes do 
initialze and store \(x_0\);  
\(T \leftarrow \infty\);  
repeat \(k = 0, 1, 2, \ldots\) 
  if \(k < T\) then 
    take action from \(b(x_k)\), observe and store \(x_{k+1}\) and \(r_{k+1}\);  
    if \(x_{k+1}\) is terminal: \(T \leftarrow k + 1\);  
  \(\tau \leftarrow k - n + 1\) (\(\tau\) time index for estimate update);  
  if \(\tau \geq 0\) then 
    \(\rho \leftarrow \prod_{i=\tau}^{\min(\tau+n-2, T-1)} \frac{\pi(u_i|x_k)}{b(u_i|x_i)}\);  
    \(g \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} r_i\);  
    if \(\tau + n < T\): \(g \leftarrow g + \gamma^n \hat{v}(x_{\tau+n})\);  
    \(\hat{v}(x_\tau) \leftarrow \hat{v}(x_\tau) + \alpha \rho [g - \hat{v}(x_\tau)]\);  
  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 $b(u|x) > 0 \ \forall \ \{x \in X, u \in U\}$

**Parameter:** $\alpha \in (0, 1], \ n \in \mathbb{Z}^+, \ \varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$

**Init:** $\hat{q}(x,u)$ arbitrarily (except terminal states) $\forall \ \{x \in X, u \in U\}$

**Init:** $\pi$ to be greedy with respect to $\hat{q}$ or to a given, fixed policy

**For** $j = 1, \ldots, J$ episodes **do**

initialize $x_0$ and action $u_0 \sim b(\cdot|x_0)$ and store them;

$T \leftarrow \infty$;

**Repeat** $k = 0, 1, 2, \ldots$

if $k < T$ then

take action $u_k \sim b(\cdot|x_k)$, observe and store $x_{k+1}$ and $r_{k+1}$;

if $x_{k+1}$ is terminal then $T \leftarrow k + 1$ else store $u_{k+1} \sim b(\cdot|x_{k+1})$;

$\tau \leftarrow k - n + 1$ ($\tau$ time index for estimate update);

if $\tau \geq 0$ then

$\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(u_i|x_i)}{b(u_i|x_i)}$;

$g \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1}r_i$;

if $\tau + n < T$: $g \leftarrow g + \gamma^n \hat{q}(x_{\tau+n},u_{\tau+n})$;

$\hat{q}(x_\tau,u_\tau) \leftarrow \hat{q}(x_\tau,u_\tau) + \alpha \rho [g - \hat{q}(x_\tau,u_\tau)]$;

if $\pi \approx \pi^*$ is being learned, ensure $\pi(\cdot|x_\tau)$ is $\varepsilon$-greedy w.r.t to $\hat{q}$;

until $\tau = T - 1$;

**Algorithm 6.4:** Off-policy $n$-step Sarsa (output is an estimate $\hat{q}_\pi$ or $\hat{q}^*$)
Table of Contents

1. $n$-Step TD Prediction
2. $n$-Step Sarsa
3. $n$-Step Off-Policy Learning with Importance Sampling
4. $n$-Step Off-Policy Learning with Tree Backup Approach
5. Bringing Everything Together: $n$-Step $Q(\sigma)$
Importance sampling might be tedious and with high variance. Are there alternatives?

For one-step updates, $Q$-learning and expected Sarsa are available as off-policy learning alternatives (see prev. lecture).

Now, tree backups will do the same for $n$-step updates.

General idea:

- Mix $n$-step sampling (middle path on the left) plus bootstrap updates along all not selected actions (dangling nodes hanging off the sides).
Updates come from the estimated action values of the leaf nodes.

Actions actually taken deliver weights for subsequent nodes proportional to the probability of occurring under $\pi$.

First-level actions contribute to value estimate with weight $\pi(u|x_{k+1})$.

Second-level actions contribute to value estimate with weight $\pi(u_{k+1}|x_{k+1})\pi(u'|x_{k+2})$.

Third-level actions contribute to value estimate with weight $\pi(u_{k+2}|x_{k+2})\pi(u_{k+1}|x_{k+1})\pi(u''|x_{k+3})$.
Derive Formal Tree-Backup Equations (1)

The 1-step tree-backup target is equal to that of expected Sarsa:

\[ g_{k:k+1} = r_{k+1} + \gamma \sum_u \pi(u|x_{k+1}) \hat{q}_k(x_{k+1}, u), \quad k < T - 1. \quad (6.15) \]

In the 2-step case, the probability of being in \( x_{k+1} \) and applying the sampled action \( u_{k+1} \) weights the second node (if \( k < T - 2 \)):

\[
g_{k:k+1} = r_{k+1} + \gamma \sum_{u \neq u_{k+1}} \pi(u|x_{k+1}) \hat{q}_{k+1}(x_{k+1}, u) \\
+ \gamma \pi(u_{k+1}|x_{k+1}) \left( r_{k+2} + \gamma \sum_u \pi(u|x_{k+2}) \hat{q}_{k+1}(x_{k+2}, u) \right) \\
= r_{k+1} + \gamma \sum_{u \neq u_{k+1}} \pi(u|x_{k+1}) \hat{q}_{k+1}(x_{k+1}, u) + \gamma \pi(u_{k+1}|x_{k+1}) g_{k+1:k+2}. \quad (6.16)\]
Derive Formal Tree-Backup Equations (2)

Extending the previous scheme delivers a general definition of the tree-backup target in a recursive form:

**n-step tree-backup return**

For $k < T - n$ and $n \geq 2$ the $n$-step tree-backup return is:

$$g_{k:k+n} = r_{k+1} + \gamma \sum_{u \neq u_{k+1}} \pi(u|x_{k+1}) \hat{q}_{k+1}(x_{k+1}, u) + \gamma \pi(u_{k+1}|x_{k+1}) g_{k+1:k+n}.$$  \hspace{1cm} (6.17)

- Due to recursive formulation, calculation starts at the bottom node.
- State-action value update is realized again via the $n$-step Sarsa rule (for $0 \leq k < T$):

$$\hat{q}_{k+n}(x_k, u_k) = \hat{q}_{k+n-1}(x_k, u_k) + \alpha [g_{k:k+n} - \hat{q}_{k+n-1}(x_k, u_k)].$$
parameter: $\alpha \in (0, 1]$, $n \in \mathbb{Z}^+$
init: $\hat{q}(x, u)$ arbitrarily (except terminal states) $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$
init: $\pi$ to be greedy with respect to $\hat{q}$ or to a given, fixed policy

for $j = 1, \ldots, J$ episodes do
  initialize $x_0$ and action $u_0 \sim \pi(\cdot|x_0)$ and store them;
  $T \leftarrow \infty$;
  repeat $k = 0, 1, 2, \ldots$
    if $k < T$ then
      take action $u_k$, observe and store $x_{k+1}$ and $r_{k+1}$;
      if $x_{k+1}$ is terminal then $T \leftarrow k + 1$ else store arbitrary $u_{k+1} \sim f(x_{k+1})$ (arbitrary mapping from a behavior policy);
      $\tau \leftarrow k - n + 1$ ($\tau$ time index for estimate update);
    if $\tau \geq 0$ then
      if $k + 1 \geq T$ then $g \leftarrow r_T$ else
        $g \leftarrow r_{k+1} + \gamma \sum_u \pi(u|x_{k+1})\hat{q}(x_{k+1}, u)$;
      for $i = \min(k, T - 1) : -1 : \tau + 1$ do
        $g \leftarrow r_i + \gamma \sum_{u \neq u_i} \pi(u|x_i)\hat{q}(x_i, u) + \gamma \pi(u_i|x_i)g$
        $\hat{q}(x_\tau, u_\tau) \leftarrow \hat{q}(x_\tau, u_\tau) + \alpha [g - \hat{q}(x_\tau, u_\tau)]$;
      if $\pi \approx \pi^*$ is being learned, ensure $\pi(\cdot|x_\tau)$ is greedy w.r.t to $\hat{q}$;
    until $\tau = T - 1$;

Algo. 6.5: $n$-step tree-backup (output is an estimate $\hat{q}_\pi$ or $\hat{q}^*$)
Table of Contents

1. $n$-Step TD Prediction

2. $n$-Step Sarsa

3. $n$-Step Off-Policy Learning with Importance Sampling

4. $n$-Step Off-Policy Learning with Tree Backup Approach

5. Bringing Everything Together: $n$-Step $Q(\sigma)$
Fig. 6.8: The backup diagrams of the previous of $n$-step state-action value updates plus the so-called $n$-step $Q(\sigma)$ method that unifies them all (exemplary 4-step case is depicted). The latter allows for choosing to sample a transition (i.e., $\sigma = 1$) or not (i.e., $\sigma = 0$) on a state-to-state basis. A $\rho$ indicates transitions on which importance sampling is required in the off-policy case.
In each time step $k$, consider a continuous mixing of sampling and expectation by $\sigma_k \in [0, 1]$.

Special cases are included:

- $\sigma_k = 0$: pure expectation,
- $\sigma_k = 1$: full sampling.

$\sigma_k$ can be set individually for each state either manually or dynamically by superimposed optimization algorithms.

In general, $\sigma_k$ is an important (hyper-)parameter which increases flexibility but also complexity.
**$n$-Step $Q(\sigma)$ Target (1)**

Firstly, we rewrite the $n$-step tree backup (6.17)

\[
g_{k:k+n} = r_{k+1} + \gamma \sum_{u \neq u_{k+1}} \pi(u|x_{k+1})\hat{q}_{k+1}(x_{k+1}, u) + \gamma \pi(u_{k+1}|x_{k+1})g_{k+1:k+n}
\]

using $\overline{v}_k(x_k) = \sum_u \pi(u|x)\hat{q}_k(x_k, u)$ for the time horizon $h = k + n$:

**Rewritten $n$-step tree backup target**

\[
g_{k:h} = r_{k+1} + \gamma \overline{v}_{h-1}(x_{k+1}) - \gamma \pi(u_{k+1}|x_{k+1})\hat{q}_{h-1}(x_{k+1}, u_{k+1}) \\
+ \gamma \pi(u_{k+1}|x_{k+1})g_{k+1:h} \\
= r_{k+1} + \gamma \pi(u_{k+1}|x_{k+1}) (g_{k+1:h} - \hat{q}_{h-1}(x_{k+1}, u_{k+1})) + \gamma \overline{v}_{h-1}(x_{k+1})
\]

(6.18)
Similarly, a recursive formulation for the off-policy importance sampling target (6.13) can be found:

\[ g_{k:h} = r_{k+1} + \gamma \rho_{k+1} (g_{k+1:h} - \hat{q}_{h-1}(x_{k+1}, u_{k+1})) + \gamma \overline{\nu}_{h-1}(x_{k+1}). \]  

(6.19)

Inspecting (6.18) for the expectation-based target and (6.19) for the sample-based target, \( Q(\sigma) \) is a linear weighting in terms of the probability \( \pi(u_{k+1}|x_{k+1}) \) and the importance sampling ratio \( \rho_{k+1} \):

\[ g_{k:h} = r_{k+1} + \gamma \sigma_{k+1} \rho_{k+1} + (1 - \sigma_{k+1}) \pi(u_{k+1}|x_{k+1}) (g_{k+1:h} - \hat{q}_{h-1}(x_{k+1}, u_{k+1})) + \gamma \overline{\nu}_{h-1}(x_{k+1}). \]  

(6.20)
Algorithmic Implementation: \( n \)-step \( Q(\sigma) \)

\[ n \text{-step } Q(\sigma) \]

\[
\begin{align*}
\text{input:} & \quad \text{a behavior policy } b \text{ with } b(u|x) > 0 \forall \{x \in X, u \in U\} \\
\text{parameter:} & \quad \alpha \in (0, 1], n \in \mathbb{Z}^+, \varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}, \sigma_k \in \{\mathbb{R}|0 \leq \sigma_k \leq 1\} \\
\text{init:} & \quad \hat{q}(x, u) \text{ arbitrarily (except terminal states)} \forall \{x \in X, u \in U\} \\
\text{init:} & \quad \pi \text{ to be } \varepsilon\text{-greedy with respect to } \hat{q} \text{ or to a given, fixed policy}
\end{align*}
\]

**Algo. 6.6: \( n \)-step \( Q(\sigma) \)** (output is an estimate \( \hat{q}_\pi \) or \( \hat{q}^* \))

- Due to the algorithm’s length, the core code is on the next slide.
for $j = 1, \ldots, J$ episodes do
initialize $x_0$ and action $u_0 \sim b(\cdot|x_0)$ and store them;
$T \leftarrow \infty$;
repeat $k = 0, 1, 2, \ldots$
  if $k < T$ then
    take action $u_k$, observe and store $x_{k+1}$ and $r_{k+1}$;
    if $x_{k+1}$ is terminal then $T \leftarrow k + 1$;
  else
    Choose and store $u_{k+1} \sim b(\cdot|x_{k+1})$;
    Select $\sigma_{k+1}$ and store $\rho_{k+1} = \pi(u_{k+1}|x_{k+1})/b(u_{k+1}|x_{k+1})$;
    $\tau \leftarrow k - n + 1$ (τ time index for estimate update);
    if $\tau \geq 0$ then
      $g \leftarrow 0$;
      for $i = \min(k + 1, T) : -1 : \tau + 1$ do
        if $i=T$ then $g \leftarrow r_T$;
        else
          $\bar{v} \leftarrow \sum_u \pi(u|x_i)\hat{q}(x_i, u)$;
          $g \leftarrow r_i + \gamma(\sigma_i \rho_i + (1 - \sigma_i)\pi(u_i|x_i))(g - \hat{q}(x_i, u_i)) + \gamma \bar{v}$;
          $\hat{q}(x_{\tau}, u_{\tau}) \leftarrow \hat{q}(x_{\tau}, u_{\tau}) + \alpha [g - \hat{q}(x_{\tau}, u_{\tau})]$;
        if $\pi \approx \pi^*$ is being learned, ensure $\pi(\cdot|x_{\tau})$ is greedy w.r.t to $\hat{q}$;
      until $\tau = T - 1$;
Comparison on 19 State Random Walk Example

Fig. 6.8: 19 state random walk results for different $n$-step algorithms (source: De Asis et al., Multi-step Reinforcement Learning: A Unifying Algorithm, arXiv:1703.01327, 2018)

- Based on random walk example from Fig. 6.4.
- $n = 3, \alpha = 0.4$
- Left plot shows RMS error in the value function.
- The results are an average of 100 runs.
- Dynamic $Q(\sigma)$ starts like Sarsa (i.e., $\sigma = 1$) and shifts towards tree-backup (i.e., $\sigma \approx 0$) by $\sigma_{k+1} = 0.95\sigma_k$.
- Again: only exemplary application-dependent result.
Summary: What You’ve Learned Today

- $n$-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.

- Tree backups are an alternative for off-policy learning without importance sampling.

- $Q(\sigma)$ updates unify sample-based and expectation-based approaches:
  - A continuous intermixing of sampling and expectation is possible.
  - Here, $\sigma_k \in [0, 1]$ can be adjusted in every step.
  - Adds flexibility but obviously also complexity.
  - Again, optimal parameter choice is not straightforward.
Thanks for your attention and have a nice week!