Chapter 7: n-step Bootstrapping

Seungjae Ryan Lee
Recap: MC vs TD

- Monte Carlo: wait until end of episode
  \[ V(S_t) \leftarrow V(S_t) + \alpha \left[ G_t - V(S_t) \right], \]
  MC error

- 1-step TD / TD(0): wait until next time step
  \[ V(S_t) \leftarrow V(S_t) + \alpha \left[ R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \]
  TD error
  Bootstrapping target
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
n-step TD Prediction

- Use truncated \textit{n-step return} as target
  - Use \( n \) rewards and bootstrap

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

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

\[
V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \left[ G_{t:t+n} - V_{t+n-1}(S_t) \right], \quad 0 \leq t < T,
\]
n-step TD Prediction: Pseudocode

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

- **Input:** a policy $\pi$
- **Algorithm parameters:** step size $\alpha \in (0, 1]$, a positive integer $n$
- **Initialize $V(s)$ arbitrarily, for all $s \in S$**
- **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 $T \leftarrow t + 1$
  - $\tau \leftarrow t - n + 1$ ( $\tau$ 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$
    - $V(S_{\tau}) \leftarrow V(S_{\tau}) + \alpha [G - V(S_{\tau})]$
- **Compute n-step return**
- **Update $V$**
- **Until $\tau = T - 1$**
n-step TD Prediction: Convergence

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

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

- Converges to true value under appropriate technical conditions
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
Random Walk Example: n-step TD Prediction

- Intermediate $n$ does best

Average RMS error over 19 states and first 10 episodes
n-step Sarsa

- Extend n-step TD Prediction to Control (Sarsa)
  - Need to use Q instead of V
  - Use $\epsilon$-greedy policy

- Redefine n-step return with Q

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

- Naturally extend to Sarsa

\[ Q_{t+n}(S_t, A_t) \triangleq Q_{t+n-1}(S_t, A_t) + \alpha [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \quad 0 \leq t < T, \]
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 $q_\pi$

Initialize $Q(s,a)$ arbitrarily, for all $s \in S, a \in A$
Initialize $\pi$ to be $\varepsilon$-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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    $\tau \leftarrow t - n + 1$ ( $\tau$ 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:
        $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
        $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$
      If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$
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

\[ G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}), \quad t+n < T, \]

\[ \bar{V}_t(s) = \sum_a \pi(a|s)Q_t(s,a), \]

- Same update as Sarsa

\[ Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \quad 0 \leq t < T, \]
Off-policy n-step Learning

- Need *importance sampling*

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

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

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

- 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

\[ Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \]
Off-policy n-step Sarsa: Pseudocode

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 \( \pi \) 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 \( S_0 \neq \text{terminal} \)
  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 \( A_{t+1} \sim b(\cdot|S_{t+1}) \)
        \( \tau \leftarrow t - n + 1 \) \( (\tau \text{ is the time whose estimate is being updated}) \)
    If \( \tau > 0 \):
      \( \rho \leftarrow \Gamma_{\min(t+n-1,T-1)} \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \) \( (\rho_{\tau+t:n-1}) \)
      \( G \leftarrow \sum_{i=0}^{\min(\tau+n,T)} \gamma^{i-1} R_i \) \( (G_{\tau+t:n}) \)
      If \( \tau + n < T \), then:
        \( G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n}) \)
        \( Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \rho [G - Q(S_t, A_t)] \)
      If \( \pi \) is being learned, then ensure that \( \pi(\cdot|S_t) \) is greedy wrt \( Q \)
    Until \( \tau = T - 1 \)
Off-policy n-step Expected Sarsa

- Importance sampling ratio ends one step earlier for Expected Sarsa

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

- Use expected n-step return

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

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

\[ G_{t:h} \doteq \rho_t (R_{t+1} + \gamma G_{t+1:h}) \]
\[ G_{t:h} \doteq \rho_t (R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V_{h-1}(S_t), \]
Per-decision Off-policy Methods*

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

\[
G_{t:h} = \rho_t (R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V_{h-1}(S_t),
\]

Control Variate

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

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

- Used with TD update without importance sampling
Per-decision Off-policy Methods: $Q^*$

- Use Expected Sarsa’s n-step return
  \[ G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}), \quad t + n < T, \]
  \[ \bar{V}_t(s) = \sum_a \pi(a|s)Q_t(s,a), \]

- Off-policy form with control variate:
  \[ G_{t:h} = R_{t+1} + \gamma \left( \rho_{t+1}G_{t+1:h} + \bar{V}_{h-1}(S_{t+1}) - \rho_{t+1}Q_{h-1}(S_{t+1}, A_{t+1}) \right), \]
  \[ = R_{t+1} + \gamma \rho_{t+1} \left( G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}) \right) + \gamma \bar{V}_{h-1}(S_{t+1}), \quad t < h \leq T. \]

- Analogous to Expected Sarsa after combining with TD update algorithm

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
n-step Tree Backup Algorithm: n-step Return

- 1-step return

\[ G_{t:t+1} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q_t(S_{t+1}, a), \]

- 2-step return

\[ G_{t:t+2} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) \]
\[ + \gamma \pi(A_{t+1}|S_{t+1}) \left( R_{t+2} + \gamma \sum_a \pi(a|S_{t+2})Q_{t+1}(S_{t+2}, a) \right) \]
\[ = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+2}, \]
n-step Tree Backup Algorithm: n-step Return

- 2-step return

\[ G_{t:t+2} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) \]
\[ + \gamma \pi(A_{t+1}|S_{t+1}) \left( R_{t+2} + \gamma \sum_{a} \pi(a|S_{t+2})Q_{t+1}(S_{t+2}, a) \right) \]
\[ = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+2}, \]

- n-step return

\[ G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+n}, \]
n-step Tree Backup Algorithm: Pseudocode

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose an action $A_0$ arbitrarily as a function of $S_0$; Store $A_0$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$ :
    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:
        $T \leftarrow t + 1$
      else:
        Choose an action $A_{t+1}$ arbitrarily as a function of $S_{t+1}$; Store $A_{t+1}$
        $\tau \leftarrow t + 1 - n \quad (\tau$ is the time whose estimate is being updated$)$
    If $\tau \geq 0$:
      If $t + 1 \geq T$:
        $G \leftarrow R_T$
      else
        $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a)$
      Loop for $k = \min(t, T - 1)$ down through $\tau + 1$:
        $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k)Q(S_k, a) + \gamma \pi(A_k|S_k)G$
        $Q(S_{\tau}, A_{\tau}) \leftarrow Q(S_{\tau}, A_{\tau}) + \alpha [G - Q(S_{\tau}, A_{\tau})]$
    If $\pi$ is being learned, then ensure that $\pi(\cdot|S_{\tau})$ is greedy wrt $Q$
  Until $\tau = T - 1$
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 $t$

$$G_{t:h} = R_{t+1} + \gamma \left( \sigma_{t+1} \rho_{t+1} + (1 - \sigma_{t+1}) \pi(A_{t+1} | S_{t+1}) \right) \left( G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}) \right) + \gamma \hat{V}_{h-1}(S_{t+1})$$

(7.17)

- Slide linearly between two weights:
  - Sarsa: Importance sampling ratio $\rho_{t+1}$
  - Tree Backup: Policy probability $\pi(A_{t+1} | S_{t+1})$
A Unifying Algorithm: n-step $Q(\sigma)$: Pseudocode*

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$
    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:
        $T \leftarrow t + 1$
      else:
        Choose and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
        Select and store $\sigma_{t+1}$
        Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as $\rho_{t+1}$
        $\tau \leftarrow t - n + 1$ (\(\tau\) is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $G \leftarrow 0$
      Loop for $k = \min(t+1, T)$ down through $\tau + 1$:
        if $k = T$:
          $G \leftarrow R_T$
        else:
          $V \leftarrow \sum_a \pi(a|S_k)Q(S_k, a)$
          $G \leftarrow R_k + \gamma (\sigma_k \rho_k + (1 - \sigma_k)\pi(A_k|S_k)) (G - Q(S_k, A_k)) + \gamma V$
          $Q(S_{\tau}, A_{\tau}) \leftarrow Q(S_{\tau}, A_{\tau}) + \alpha [G - Q(S_{\tau}, A_{\tau})]$
    If $\pi$ is being learned, then ensure that $\pi(\cdot|S_{\tau})$ is greedy wrt $Q$
  Until $\tau = T - 1$
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
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