N-step Temporal Difference Methods

Self-Study

November 4, 2019
Full return is

\[ G_t = \sum_{k=t}^{\infty} R_{k+1}, \]

one-step return is

\[ G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1}). \]

We can generalize both into \( n \)-step returns:

\[ G_{t:t+n} \overset{\text{def}}{=} \left( \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} \right) + \gamma^n V(S_{t+n}). \]

with \( G_{t:t+n} \overset{\text{def}}{=} G_t \) if \( t + n \geq T \).
A natural update rule is

\[ V(S_t) \leftarrow V(S_t) + \alpha [G_{t:t+n} - V(S_t)] . \]

---

**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 \( S_0 \neq \text{terminal} \)
  - \( T \leftarrow \infty \)
  - 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: \( G \leftarrow G + \gamma^n V(S_{\tau+n}) \) \((G_{\tau+n})\)
      - \( V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)] \)
  - Until \( \tau = T - 1 \)

*Algorithm 7.1 of “Reinforcement Learning: An Introduction, Second Edition”.*
Using the random walk example, but with 19 states instead of 5,

we obtain the following comparison of different values of $n$:

Average RMS error over 19 states and first 10 episodes

Figure 7.2 of "Reinforcement Learning: An Introduction, Second Edition".
Defining the $n$-step return to utilize action-value function as

$$G_{t:t+n} \overset{\text{def}}{=} \left( \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} \right) + \gamma^n Q(S_{t+n}, A_{t+n})$$

with $G_{t:t+n} \overset{\text{def}}{=} G_t$ if $t + n \geq T$, we get the following straightforward algorithm:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [G_{t:t+n} - Q(S_t, A_t)].$$
**n-step Sarsa Algorithm**

### 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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$
- $\tau \leftarrow t - n + 1$  \hspace{1cm} ($\tau$ is the time whose estimate is being updated)
- If $\tau \geq 0$:
  - $G \leftarrow \sum_{i=\tau+1}^{\tau+n} \gamma^{i-\tau-1} R_i$
  - If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
  - $Q(S_{\tau}, A_{\tau}) \leftarrow Q(S_{\tau}, A_{\tau}) + \alpha [G - Q(S_{\tau}, A_{\tau})]$  \hspace{1cm} ($G_{\tau, \tau+n}$)
- If $\pi$ is being learned, then ensure that $\pi(\cdot|S_{\tau})$ is $\varepsilon$-greedy wrt $Q$

Until $\tau = T - 1$

*Algorithm 7.2 of "Reinforcement Learning: An Introduction, Second Edition".*
Off-policy $n$-step Sarsa

Recall the relative probability of a trajectory under the target and behaviour policies, which we now generalize as

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

Then a simple off-policy $n$-step TD can be computed as

$$V(S_t) \leftarrow V(S_t) + \alpha \rho_{t:t+n-1} \left[ G_{t:t+n} - V(S_t) \right].$$

Similarly, $n$-step Sarsa becomes

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \rho_{t+1:t+n} \left[ G_{t:t+n} - Q(S_t,A_t) \right].$$
**Off-policy n-step Sarsa**

For estimating $Q \approx q_*$ or $q_\pi$

**Input:** an arbitrary behavior policy $b$ such that $b(a|s) > 0$, for all $s \in S, a \in A$

- Initialize $Q(s,a)$ arbitrarily, for all $s \in S, a \in A$
- 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 $S_0 \neq$ 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 is the time whose estimate is being updated)

**If $\tau \geq 0$:**

- $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \pi(A_{i+1}|S_{i+1}) b(A_i|S_i)$
- $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau} R_i$

- 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 $\tau = T - 1$**

---

*Algorithm 7.3 of "Reinforcement Learning: An Introduction, Second Edition".*
Q-learning and Expected Sarsa can learn off-policy without importance sampling.

To generalize to $n$-step off-policy method, we must compute expectations over actions in each step of $n$-step update. However, we have not obtained a return for the non-sampled actions.

Luckily, we can estimate their values by using the current action-value function.
Off-policy $n$-step Without Importance Sampling

We now derive the $n$-step reward, starting from one-step:

$$G_{t:t+1} \overset{\text{def}}{=} R_{t+1} + \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a).$$

For two-step, we get:

$$G_{t:t+2} \overset{\text{def}}{=} R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+2}.$$

Therefore, we can generalize to:

$$G_{t:t+n} \overset{\text{def}}{=} R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+n}.$$

The resulting algorithm is $n$-step Tree backup and it is an off-policy $n$-step temporal difference method not requiring importance sampling.
**n-step Tree Backup for estimating** $Q \approx q_\pi$ or $q_\pi$

Initialize $Q(s, a)$ arbitrarily, for all $s \in S, a \in A$
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 can take their index mod $n + 1$

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 $A_{t+1}$
      - $\tau \leftarrow t + 1 - n$ (\(\tau\) is the time whose estimate is being updated)
      - If $\tau \geq 0$:
        - $G \leftarrow R_T$
      - 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 $\tau = T - 1$

*Algorithm 7.5 of "Reinforcement Learning: An Introduction, Second Edition".*