1 n-step Bootstrapping

n-step methods allow us to observe multiple time-steps of returns before updating a state with the observed data and a bootstrapped estimate of the value of the nth succeeding state.

1.1 n-step TD Prediction

Define the n-step return

$$G_{t:t+n} \doteq \sum_{i=t}^{t+n-1} \gamma^{i-t} R_{i+1} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{1}$$

where $n \geq 1$, $0 \leq t < T - n$ and V_i is the estimated state-value function as of time i. If $t + n > T$ then $G_{t+n} \equiv G_t$, the standard return. The n-step return is the target for *n-step TD* methods, note that $n - 1$ rewards are observed and the succeeding value is bootstrapped with the latest estimate of the value function. The corresponding update for state-values is

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)] \qquad 0 \leq t < T. \tag{2}$$

Note that Monte-Carlo can be thought of as TD(∞)Pseudocode for n-step TD is given in the box below.

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$

The n-step return obeys the *error-reduction property*, and because of this n-step TD can be shown to converge to correct predictions (given a policy) under appropriate technical conditions. This property states that the n-step return is a better estimate than V_{t+n-1} in the sense that the error on the worst prediction is always smaller

$$\max_s |\mathbb{E}_\pi[G_{t:t+n}|S_t = s] - v_\pi(s)| \leq \gamma^n \max_s |V_{t+n-1}(s) - v_\pi(s)| \tag{3}$$

1.2 n-step Sarsa

Sarsa

We develop n-step methods for control. We generalise Sarsa to n-step Sarsa, or Sarsa(n). This is done in much the same way as above, but with action-values as opposed to state-values. The n-step return in this case is defined as

$$G_{t:t+n} \doteq \sum_{i=t}^{t+n-1} \gamma^{i-t} R_{i+1} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{4}$$

where $n \geq 1$, $0 \leq t < T - n$ and Q_i is the estimated action-value function as of time i. If $t + n > T$ then $G_{t+n} \equiv G_t$, the standard return. The corresponding update is

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \qquad 0 \leq t < T. \tag{5}$$

Expected Sarsa

We define n-step expected Sarsa similarly

$$G_{t:t+n} \doteq \sum_{i=t}^{t+n-1} \gamma^{i-t} R_{i+1} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}) \tag{6}$$

where $n \geq 1$, $0 \leq t < T - n$ and $\bar{V}_i$ is the *expected approximate value* of state s

$$\bar{V}_i(s) \doteq \sum_a \pi(a|s) Q_i(s, a). \tag{7}$$

As always, if $t + n > T$ then $G_{t+n} \equiv G_t$, the standard return. The corresponding update is formally the same as above.

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 $\tau = T - 1$

1.3 n-step Off-policy Learning

We can learn with n-step methods off-policy using the importance sampling ratio (target policy π and behaviour policy b)

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

For state-values we have

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)]$$

and for action-values we have

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

note that for action values the importance sampling ratio starts one time-step later, because we are attempting to discriminate between actions at time t.

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: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

1.4 *Per-decision Methods with Control Variates

We have the standard recursion relation for the n-step return

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h}.$$

For an off-policy algorithm, one would be tempted to simply weight this target by the importance sampling ratio. This method, however, shrinks the estimated value functions when the importance sampling ratio is 0, hence increasing variance. We thus introduce the *control-variate* $(1-\rho_t)V_{h-1}(S_t)$, giving an off-policy update of

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

where $G_{h:h} = V_{h-1}(S_h)$. Note that the control-variate has expected value 0, since the factors are uncorrelated and the expected value of the importance sampling ratio is 1.

We can do a similar thing for action-values

$$G_{t:h} \doteq R_{t+1} + \gamma\rho_{t+1:h}\left(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\right) - \gamma\bar{V}_{h-1}(S_{t+1}),$$

where once again the importance sampling ratio starts one time-step later.

Control Variates in General

Suppose we want to estimate μ and assume we have an unbiased estimator for μ in m. Suppose we calculate another statistic t such that $\mathbb{E}[t] = \tau$ is a known value. Then

$$m^\star = m + c\,(t - \tau)$$

is also an unbiased estimator for μ for any c, with variance

$$\mathrm{Var}\left(m^\star\right) = \mathrm{Var}\left(m\right) + c^2\,\mathrm{Var}\left(t\right) + 2c\,\mathrm{Cov}\left(m,t\right).$$

It is easy to see that taking

$$c = -\frac{\mathrm{Cov}\left(m,t\right)}{\mathrm{Var}\left(t\right)}$$

minimizes the variance of $m^\star$. With this choice

$$\mathrm{Var}(m^\star) = \mathrm{Var}(m) - \frac{\left[\mathrm{Cov}(m,t)\right]^2}{\mathrm{Var}(t)} \tag{8}$$

$$= (1-\rho_{m,t}^2)\mathrm{Var}(m) \tag{9}$$

where $\rho_{m,t} = \mathrm{Corr}\left(m,t\right)$ is the Pearson correlation coefficient of m and t. The greater the value of $|\rho_{m,t}|$, the greater the variance reduction achieved.

1.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

We introduce the n-step *tree-backup algorithm* algorithm using the return

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1},a) + \gamma\pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \tag{10}$$

for $t < T-1$, $n > 1$ and with $G_{i:i} = 0$ and $G_{T-1:t+n} = R_T$. This algorithm updates S_t with bootstrapped, probability weighted action-values of *all* actions that were not taken all along the trajectory and recursively includes the rewards realised, weighted by the probability of their preceding actions under the policy. Pseudocode given below.

n-step Tree Backup 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 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}
    $\tau \leftarrow t+1-n$ (τ 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 π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

1.6 *A Unifying Algorithm: n-step $Q(\sigma)$

We introduce an algorithm which, at each time step, can choose to either take an action as a sample as in Sarsa or to take an expectation over all possible actions as in tree-backup.

Define a sequence $\sigma_t \in [0,1]$ that at each time step chooses a proportion of sampling vs. expectation. This generalises Sarsa and tree-backup by allowing each update to be a linear combination of the two ideas. The corresponding return (off-policy) is

$$G_{t:h} \doteq R_{t+1} + \gamma \left(\sigma_{t+1}\rho_{t+1}(1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1})\right)\left(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\right) \tag{11}$$

$$+ \gamma \bar{V}_{h-1}(S_{t+1}), \tag{12}$$

for $t < h < T$, with $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$ if $h < T$ and $G_{T-1:T} \doteq R_t$ if $h = T$. Pseudocode given below.

Off-policy n-step $Q(\sigma)$ 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 ε-greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations can take their index mod $n+1$

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 σ_{t+1}
        Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as ρ_{t+1}
    $\tau \leftarrow t - n + 1$ (τ 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:
          $\bar{V} \leftarrow \sum_a \pi(a|S_k) Q(S_k, a)$
          $G \leftarrow R_k + \gamma\big(\sigma_k \rho_k + (1 - \sigma_k)\pi(A_k|S_k)\big)\big(G - Q(S_k, A_k)\big) + \gamma \bar{V}$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$