Chapter 7 n-step Bootstrapping

Created	@December 4, 2023 9:08 AM
Tags	

written by Yee

7-1. n-step TD Prediction

n-step TD 大概為 MC 跟 TD 的綜合體，MC更新的時候是在整個episode跑完之後才去做更新，而TD只會基於上一個值的變化做更新。我們若是可以基於多個step，但又不想要跑完整個episode在做更新，比如2-step、3-step等等多-step的更新。這些n-step更新仍然適用TD因為她仍然基於先前幾步的差值去做更新。有別於 $TD(0)$ 的每步更新，現在更新得要延遲 n-step。或是我們可以稱之為 **n-step TD method**。

假設有一個episode 有state-reward sequence, $S_t, R_{t+1}, S_{t+1}, R_{t+2}, S_{t+2}, \ldots, R_T, S_T$，我們知道MC是需要到episode結束才能更新資料。

$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

有別於MC目標是return，one-step return 是的目標是reward及下個state的期望值。

$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$

這個 $t : t+1$的寫法表示了 擷取 time t 到 $t+1$ 的 reward總值的 return，之後再以 $\gamma V_t(S_{t+1})$ 取代 $\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$，我們以此類推到下一個step，以下為**two-step return**

$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$

接下來再繼續推到 **n-step return**

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \qquad (7.1)$$

for all n, t such that $n \geq 1$ and $0 \leq t < T - n$

如果 $t + n \geq T$，那所有缺少的元素都補 0，這時候 n-step return 定義回原來的 G_t Full return

$(G_{t:t+n} = G_t \text{ if } \ t + n \geq T)$

n-step如果 $n > 1$ 會需要未來的Reward，需要直到 R_{t+n} 之後並計算 V_{t+n-1} ，再 $t+n$的時間才能得到下列這兩個的值。

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)] \qquad (7.2)$$

我們稱之為 **n-step TD**。在一開始的 $n-1$ step 我們不會改動任何東西，再最後到terminal state後在一次更新之後還沒更新過的地方。

---

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 update)

If $\tau \geq 0$:

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

If $r + n < T,$ then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$

$V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$

Until $\tau = T - 1$

7.2 n-step Sarsa (on-policy)

這個 method 僅僅是把 prediction 改成 control method，然後使用的是 ϵ-greedy policy， Sarsa 是在 action 下結束而非 state。我們重新定義 n-step Sarsa 的 預估action values：

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{7.4}$$
$$n \geq 1, 0 \leq t < T - n$$
$$\text{with } G_{t:t+n} = G_t \text{ if } t + n \geq T$$

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \tag{7.5}$$
$$0 \leq t < T$$

這個就稱作 n-step Sarsa

---

n-step Sarsa for estimating $Q \approx q_\pi$ or q_*

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$

Initialize π to be ϵ-greedy w.r.t $Q,$ or to a fixed given policy

Algorithm parameter: step size $\alpha \in (0, 1],$ small $\epsilon > 0$, a positive integer n

All store and access operation (for S_t, A_t, 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 state's estimate is being update)

If $\tau \geq 0$:

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

If $r+n<T,$ 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 π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ϵ-greedy w.r.t Q

Until $\tau = T-1$

那 Expected Sarsa呢?

在前面behavior的過程仍舊跟Sarsa一樣是線性的，除了最後一個元素需要分支計算下一個state的所有期望值總和。 $\bar{V}_t$

$$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}) \\ t+n<T \tag{7.9}$$

$(G_{t:t+n} = G_t \text{ if } \ t+n \geq T)$ $\bar{V}_t$ 是 state s 的期望預估值 (expected approximate value)

7.3 n-step Off-policy Sarsa

回顧 off-policy 是讓 policy π 從 behavior policy b 上學習。 π 通常是以目前 Action-value function所組成的 greedy policy， b 則通常是比較具探索性(exploratory) 的 policy，通常是 ϵ-greedy policy。為了可以使用 b 產出的資料，我們需要計算走這步的相對機率(Section 5.5)。回顧 importance sampling ratio

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)] \\ \text{, for } 0 \leq t < T$$

$\rho_{t:t+n-1}$就是 importance sampling ratio，是兩個policy使用這 n 個 action 的相對機率

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

舉個例子，如果有任一個 action 不會被 policy π 採用 (i.e. $\pi(A_k|S_k)=0$) 那這個 n-step return 會被忽略掉。在另一方面如果 這個如果被 π 挑中的機率大於 b ，那他的比重(weight)會大幅增加。若為on-policy 這兩個得值的比重永遠為1，所以我們先前的n-step Sarsa公式可以直接替換成：

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \\ \text{, for } 0 \leq t < T$$

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 w.r.t $Q,$ or as a fixed given policy

Algorithm parameter: step size $\alpha \in (0,1],$ small $\epsilon > 0$, a positive integer n

All store and access operation (for S_t, A_t, 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 state's estimate is being update)

    If $\tau \geq 0$:

      $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n,T-1)} \frac{\pi(A_i,S_i)}{b(A_i,S_i)}$ $\qquad (\rho_{\tau+1:\tau+n})$

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

      If $r + n < T,$ then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $\qquad (G_{\tau:\tau+n})$

      $\mathrm{Q}(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$

      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ϵ-greedy w.r.t Q

  Until $\tau = T - 1$

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

這個點子是來自於右圖 3-step tree-backup backup。 在主幹上有三個被標籤的 states 和 rewards 還有兩個 actions， S_t, A_t 是任意的初始 state-action pair。在旁邊的枝葉 state 是沒有被選擇到的action，因為我們沒有未選擇action的資料，我們用bootstrap的方式使用殘枝的期望值。我們目前都會更新主幹的值，**而在tree-backup我們會額外去找主幹連接殘枝的值回來更新**，這就是所謂的 tree-backup update。簡單來說我們會整合整棵樹的預測值再去做更新。

每個殘枝的 weight 會是在 policy π 的機率採取這個 action，因此在第一層會貢獻 $\pi(a|S_{t+1})$除了實際上走的那條路 A_{t+1} 不會被計算在內，而他的機率

$\pi(A_{t+1}|S_{t+1})$

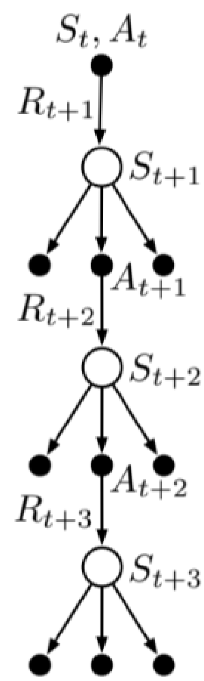
，會是所有第二層樹 action value 的比重。例如 a'貢獻了

$\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$

，每個第三層樹則是

$\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a'', S_{t+3})$

以此類推。



the 3-step tree-backup update

我們把 3-step tree-backup update 拆解成6個half-step，一半為 action 到接下來的 state，另一半是考慮那個state的所有有可能的action 以及其在該 policy 發生的機率。

我們開始來推導 n-step tree-backup update出這個演算法的細節。one-step return跟Expected Sarsa一模一樣

$$G_{t:t+1} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q_t(S_{t+1}, a) \tag{7.15}$$

若 $t < T-1$ 那 2-step return 為

$$\begin{aligned} G_{t:t+2} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) + \gamma\pi(A_{t+1}|S_{t+1}) \left(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2})Q_{t+1}(S_{t+2}, a) \right) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) + \gamma\pi(A_{t+1}, S_{t+1})G_{t+1:t+2} \end{aligned}$$

對於 $t < T-2$ 後面一般化的公式如下：

$$G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+n-1}(S_t+1, a) + \gamma\pi(A_{t+1}|S_{t+1})G_{t+1:t+n} \tag{7.16}$$

對於在 $t < T-1, n \geq 2$， $n = 1$ 已經包含在 (7.15) 了，這個是用於n-step Sarsa對於一般 action value 的更新

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

對於 $0 \le t < T$ 當所有的 state-action pair 並未改變， $Q_{t+n}(s,a) = Q_{t+n-1}(s,a)$對於所有的 s, a 使得 $s \neq S_t$ or $a \neq A_t$。

---

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 ϵ-greedy w.r.t $Q,$ or to a fixed given policy

Algorithm parameter: step size $\alpha \in (0,1],$ small $\epsilon > 0$, a positive integer n

All store and access operation (for S_t, A_t, R_t) can take their index mod $n+1$

Loop for each episode:

  Initialize and store $S_0 \neq$ terminal

  Choose an action arbitrarily as a function of $S_0;$ store A_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 and state as R_{t+1}, S_{t+1}

      If S_{t+1} is terminal

        then $T \leftarrow t+1$

      else:

        Choose an action arbitrarily A_{t+1} as a function of $S_{t+1};$ Store A_{t+1}

    $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being update)

    If $\tau \ge 0:$

      If $t + 1 \ge T$

        $G \leftarrow R_T$

      else:

        $G = R_{t+1} + \gamma \sum_{a \neq A_k} \pi(a|S_k)Q(S_k, a) + \gamma\pi(A_k|S_k)G$

      Loop for $k = \min(t, T-1)$ down through $\tau + 1:$

        $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k)Q(S_k, a) + \gamma\pi(A_k|S_k)G$

      $\mathrm{Q}(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$

      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ϵ-greedy w.r.t Q

  Until $\tau = T - 1$
