Chapter 6 Temporal-Difference Learning

Created	@November 16, 2023 2:07 PM
Tags	

Written by Yee

TD 是合併 Monte Carlo 以及 DP 的點子。他像 MC 可以從第一手經驗(raw experience)去學習，他也有 DP 的優勢去參考其他人所學的 (learned) 去預測，不需要等到 episode 結束(他們可以bootstrap即時更新)。

6-1. TD Prediction

TD 以及 MC 都是以經驗去預測問題解。如果我們用 policy π 來，兩個 method 都是用來預測在 non-Terminal state S_t 中 v_π 的 V 的。大致方向是 MC 需要等到得到該個 visit 的 Return 才能去使用這個 return 想辦法更新 $V(S_t)$。 最簡單的 MC method 在變動的環境下式子如下

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{6.1}$$

G_t 是在 time t 下的 return， α 是一個 step-size parameter。我們先使用 constant-α MC。不過 MC 需要等到episode 的結束才能去做 $V(S_t)$ 的更新，TD 則可以在下一步馬上做更新。在 $t+1$ 時他就直接使用 R_{t+1} 的值做即時更新。最簡單的 TD 如下

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{6.2}$$

MC 用 G_t 做更新， 這裡 TD 是用 $R_{t+1} + \gamma V(S_{t+1})$。這個 TD 也可以被稱作為 $TD(0)$ 或是稱為 **one-step TD**。因為這是第12章 $TD(\lambda)$或是第七章 n-step TD的特殊變體。下面為 pseudocode

---

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

$\text{Algorithm Parameter: step size } \alpha \in (0,1]$

$\text{Initialize } V(s)$, for all $s \in \mathcal{S}$,arbitrarily except that $V(terminal) = 0$

Loop for each episode:

  Initialize S

  Loop for each step of episode:

    $A \leftarrow$ action given by π for S

    Take action A, observe R, S'

    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

    $S \leftarrow S'$

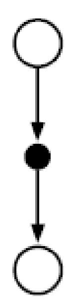
  until S is terminal

因為 $TD(0)$ 是基於既有已知的預測去做 update 的。也就是我們說的 bootstrapping。從第三章我們知道：

$$\begin{aligned} v_\pi(s) =& \mathbb{E}_\pi[G_t|S_t = s] && (6.3) \\ =& \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] && (\text{from}(3.9)) \\ =& \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] && (6.4) \end{aligned}$$

MC是以 (6.3) 作為目標，而DP是以 (6.4) 作為目標

TD的目標也是一種估測值，這有兩個理由：(1)它從6.4得到樣本期望值，(2)它使用當前的估測值。而因為 $v_\pi(S_{t+1})$為未知，因此我們使用 $V(S_{t+1})$。這個TD target是使用 (6.4) 而且是使用 V 而不是 v_π。因此我們可以說它結合了MC的sampling以及DP的bootstrapping。



TD(0)

上圖可以看的到，backup diagram的最上面的那個state節點的估測值是依據它後面那個state一次樣本轉移之後來做更新的。我們可以將TD跟MC更新視為 **sample updates**(採樣更新) 因為他們需要向前看下一代的state(或是state-action)中的 state value $V(S_{t+1})$還有reward R_{t+1} 然後去更新原來 $V(S_t)$的值。

我們可以把 $TD(0)$ 的更新想成錯誤的修正，或是稱之為TD error

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{6.5}$$

每一次TD error 都是當時預測的error因為 TD error 需要下個 State 以及下一個 reward，這些在去到下一步前並無法得知。所以說 $V(S_t)$ 的更新會延遲一步。也就是說， δ_t 需要等到 $t+1$ 時才能得知。若是 V 無法在當下的episode改變(MC method並不會在episode中改變 V 值)。那麼我們可以把 TD error的總和表示成 MC error 。

$$\begin{aligned}
G_t - V(S_t) =& R_{t+1} + \gamma(G_{t+1}) - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) && (\text{from}(3.9)) \\
=& \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\
=& \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\
=& \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) && (6.6) \\
=& \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) \\
=& \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k
\end{aligned}$$

6.2 Advantage of TD Prediction Method

1. Bootstrapping:
2. 不需要model
3. 只需要one-step的時間就能做即時更新
4. MC必須忽略或是減少exploration的權重，可能會造成學習進程緩慢，TD並沒有這個問題。
5. 適合用於極長或無限大小的episode。
6. 仍舊可以收束到 v_π

7. 絕大部分情況下 TD的收束速度會大於MC。

6.3 Optimality of TD(0)

假設我們只有有限的experience，10個episode或是100個episode for example。在這個情況下我們只能重複使用這些資料去讓預測收束至true value。給定一個value function V ，並使用 $(6.1),(6.2)$的更新方法。但是我在計算error後先將同一個State的(action-state) 的 error 做累積而不是直接更新 V，最後再將累積的值直接去做 V 的修正。這個方法我們稱作 **batch updating**因為update只會在跑完一個batch之後發生。

TD(0) 在 batch-updating 版本在 α 足夠小的情況下會收束。constant-α MC 亦同。不過兩種 method 會收束到不同的答案。

EX 6.4 You are the Predictor

假設我們的八個 episode 為下列值。

$A, 0, B, 0$

$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 0$

這八個episode為一個batch，那我們求 $V(A)$ 跟 $V(B)$ 的值為多少?

每個人應該都同意 $V(B)=\frac{3}{4}$

但是 $V(A)$ 是多少呢?

根據 batch MC method $G_t - V(S_t)$

$V(A) = 0$

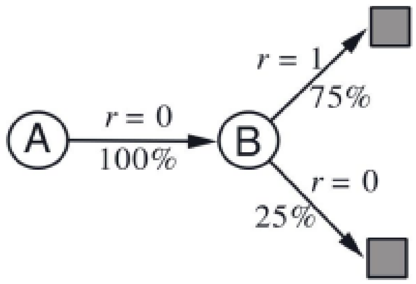
但是根據 batch TD $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

$V(A) = 0 + \frac{3}{4} - 0 = \frac{3}{4}$

根據 MDP 則是

A有100%機率進入state B，而state B會有 75% 機率 reward為1，25%為0。

所以



$V(A) = 0 * 0.25 + 1 * 0.75 = \frac{3}{4}$

最後一個是純粹觀察 A 的回傳值得到

$V(A) = 0$

在EX 6.4 我們可以明顯的看出兩個batch method的顯著差異。

batch MC會試著找出最低的方均差，而batch TD(0)會找出最大概似(maximum-likelihood)Markov process模型產出的估計值。一般來說 maximum-likelihood estimate會挑出能誰有最高的機率製造出資料，在這個例子就是從先前觀察到的 episode 捏出來的 Markov process 的模型。這個預估從分布機率從 i 轉變成 j 會是從 i 到 j 所觀察到轉移的分數，而相關的預期的報酬(expected reward)就會是這些轉移中所觀察到的 rewards 的平均。給定這個模型，如果這個模型是完全正確的，那我們就可以精確的計算這個 value function 的估測值。這樣稱知會 **certainly-equivalence estimate** 這等價於它會預測淺在地過程(underlying process)而不是只是去預測他。一般來說 batch TD(0) 會收束在確定等價(certainly equivalence)預估。

http://ccckmit.wikidot.com/st:maximumlikelihood

TD的速度會比MC快就是因為他是運算certainly-equivalence的預估。

雖然non-batch TD(0) method並不會收束於certainly-equivalence的預估，但是他相較於constant-αMC還是會收束在較佳的預估上。

還有一些要提到的地方是，在部分的地方 certainly-equivalence的預估是最佳解，但是通常都很難直接去計算出來他的實際值。假設說我有 $n = |\mathcal{S}|$ 個 state，那我計算就他就需要 n^2 個memory，計算相對應的value function就需要 n^3 運算次數。所以，如果你的state space非常大，那TD應該是唯一能夠求近似確定等估計的作法。

6.4 Sarsa: On-policy TD Control

我們現在要用TD prediction 來解問題了。我們同樣會使用 GPI 的方，但是這次我利用TD methods來做evaluation 跟prediction。

Sarsa是從 action-value function 學習的。所以我們必須以先在的behavior policy π、state s 、action a 去預測 $q_\pi(s,a)$。

我們把 $q_\pi(s,a)$ 帶入 $TD(0)$

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \tag{6.7}$$

這個rule使用了所有 $(S_t,A_t,R_{t+1},S_{t+1},A_{t+1})$ 的元素故取名為 **Sarsa**。

跟其他on-policy method很像，我們會持續地去用policy π 去預測 q_π，我們再用 Sarsa 來更新policy。

Sarsa的收束條件取決於這個policy 對於 Q 的依賴程度，比如說上述Sarsa也可以使用 ϵ-greedy 或是 ϵ-soft的policy。Sarsa同樣也可以在policy逐漸轉為greedy的情況下($\epsilon = 1/t$) 每個state到訪無限次數的情況下收束。

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(treminal,\cdot) = 0$

Loop for each episode:

  Initialize S

  Choose A from S using policy derived from Q (e.g., ϵ-greedy)

  Loop for each step of episode:

    Take action A observe R, S'

    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)

    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$

    $S \leftarrow S'; A \leftarrow A'$

  until S is terminal

6.5 Q-learning: Off-policy TD Control

在早期，Off-policy TD Control，或是稱之為 Q-learning以下列為定義

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

在這個method，我們拿學到的 action-value function Q 直接預測 q_π ，這使得演算法的分析部分變得十分簡單並且可能可以使其更早收束。policy仍然需要決定哪個state-action需要被visit及更新，只要這些state-action有不斷的被更新，終究會使其收束到正確的值。在第五章我們有談到，上述個方法是獲得最佳解的最低要求。

off-policy 的原因是因為我們**不管是用哪種policy**，我們最後都是使用 $max_a Q(S', a)$**去更新 estimated action value**

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in$
$\mathcal{A}(s)$, arbitrarily except that $Q(treminal, \cdot) = 0$

Loop for each episode:

  Initialize S

  Loop for each step of episode:

    Choose A from S using policy derived from Q (e.g., ϵ-greedy)

    Take action A observe R, S'

    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

    $S \leftarrow S'$

  until S is terminal

6.6 Expected Sarsa

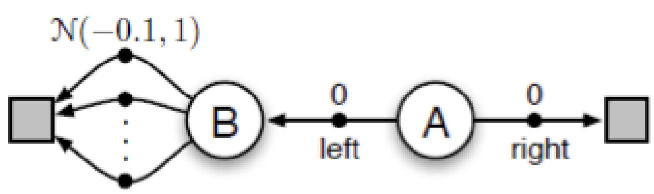
這個method類似Q-learning但是他用的是期望值而不是action-value最大值，他顧及到以現在的policy下各個未來的action的機率。

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}|S_{t+1})] - Q(S_t, A_t)] \\
&\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \qquad (6.9)
\end{aligned}
$$

其他地方則遵守 Q-Learning的規則。給定下個State S_{t+1} ，這演算法跟Sarsa預期的方向一致，因此稱為Expected Sarsa。

Expected Sarsa 比 Sarsa難計算，但是它去除掉了對 A_{t+1} 做出的隨機選擇所產生的方差。在相同的經驗下他會表現得比Sarsa更好。Expected Sarsa可以用於on-policy或是off-policy，off-policy 僅需要額外設計一個 behavior policy，**然而 off-policy 的 expected Sarsa 就是 Q-Learning如果 π 是 greedy-policy的話**。Expected Sarsa整合並一般化了Q-learning並改善了Sarsa，這點額外的運算可以讓Expected Sarsa優於其他兩個algorithm。

6.7 Maximization Bias and Double Q learning



假設 starting state 是 A，向左之後下個 所有 action 都會獲得一個平均為 -0.1 標準差為 1 的隨機reward，向右則會直接結束這個 episode，我們可以說向左本身就比向右差。它最大的真實值就是0，但是估測的最大值卻是正值，這就是一個正偏差(positive bias)。我們稱為 **maximization bias**。

在傳統的 Q-learning，很**容易就被當時的最大值牽著走**，殊不知他們都是平均值或是說真實期望值為 -0.1 。

Double Q-learning的特點就是會用到**兩個Q table**： $Q_1(a)$ 以及 $Q_2(a)$，兩個都是來預測真實值 $q(a)$ ，我們可以用一個預測值， $Q_1(a)$ 為例子我們需要預測可以獲得的最大值的最大action： $A^* = \arg\max_a Q_1(a)$ ， $Q_2(a)$ 提供預測值 $Q_2(A^*) = Q_2(\arg\max_a Q_1(a))$ 。這個預測值會是 unbiased因為 $\mathbb{E}[Q_2(A^*)] = q(A^*)$，我們同樣也可以反過來預測 $Q_1(\arg\max_a Q_2(a))$，這就是所謂的 double learning，不過要注意的是每一次step只有一邊的Q table有做更新。

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

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

Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s),$ arbitrarily except that $Q(treminal, \cdot) = 0$

Loop for each episode:

  Initialize S

  Loop for each step of episode:

    Choose A from S using policy ϵ-greedy derived from $Q_1 + Q_2$

    Take action A observe R, S'

    with 0.5 probability

      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha[R + \gamma Q_2(S', \arg\max_a Q_1(S', a)) - Q_1(S, A)]$

    else:

      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha[R + \gamma Q_1(S', \arg\max_a Q_2(S', a)) - Q_2(S, A)]$

    $S \leftarrow S'$

  until S is terminal
