Chapter 12 Eligibility Trace

Created	@December 27, 2023 8:43 AM
Tags	

12.1 The λ return

在第七章我們定義 n-step return是 n個reward的總和加上到達n-step後的預測值，每個都適當地做 discounted (7.1)。

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}) \\ 0 \leq t \leq T - n \tag{12.1}$$

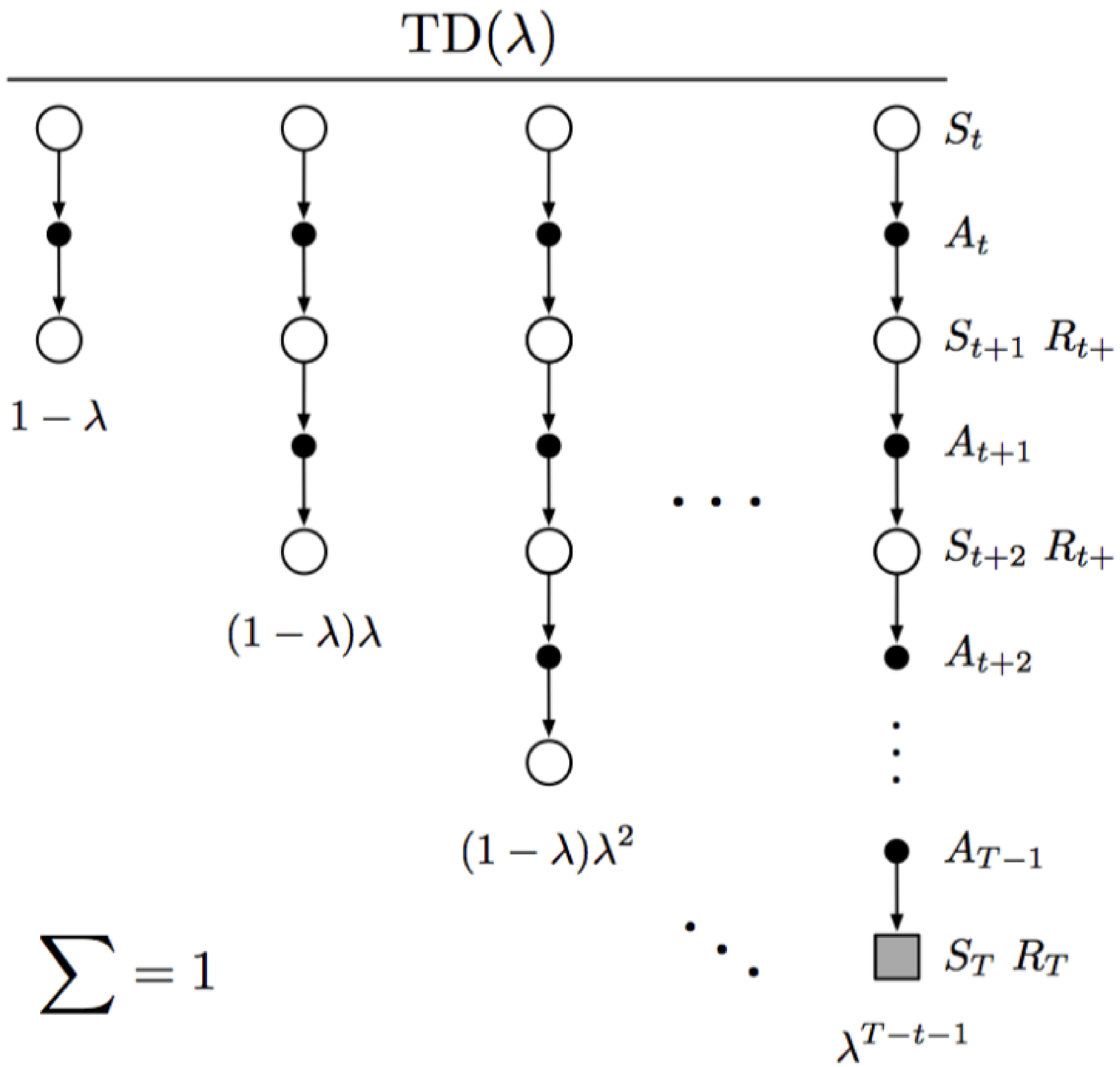
$\hat{v}(s, \mathbf{w})$是state s 在給定 $\mathbf{w}$ 權重的情況下預估值， T為episode終結的時間(如果有的話)。我們在第七章標示說n-step return $(n \geq 1)$ 在tabular learning update是一個有效的update target，在 (9.7) 的近似 SGD 亦同。

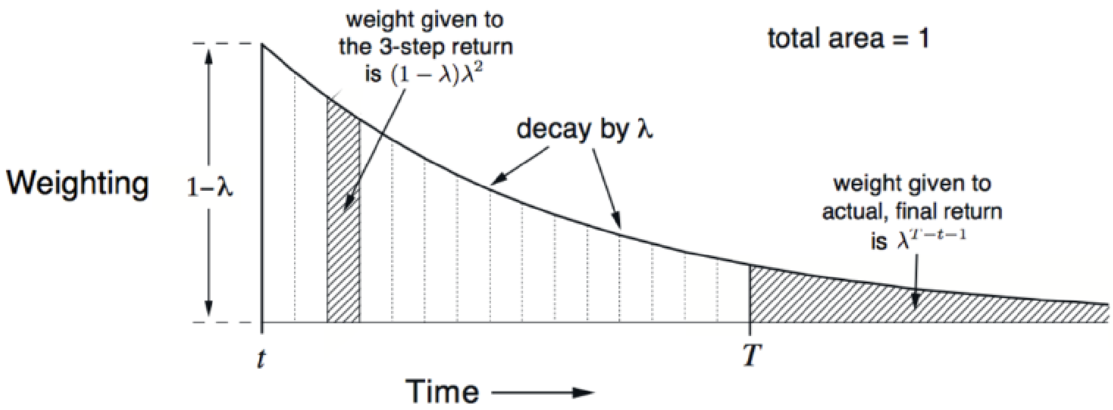
現在我們要告訴你的是，這些不只可以被應用在固定的n-step return，而可以被用在任意"平均"的n-step return 這個 n 可以是不同 n 的平均。像是一半的 2-step 跟一半的 4-step return $\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$，任意一個Set都可以這樣被平均，甚至是無限長的set，只要**這些組合return的weight是正的並且總合為1**，這個複合式的Return 也有一個可以減少 error 的特性如同個別的 n-step return一樣。我們可以平均 1-step跟 無限-step的return去使得MC 跟 TD取得相連。原則上我們可以使用以經驗為基礎的DP更新去獲得一個以經驗跟model為基礎的method。

這個用我們這組合的平均去做更新的方法叫做 compound update，這個compound update的圖以每個個別的component跟一個權重去組成的，那個在全部components上水平線，下面則是它的個別權重。

$\mathrm{TD}(\lambda)$可以被理解為一個方法n-step update的平均，這個包含所有的n-step，並每個權重為 λ^{n-1} $(\lambda \in [0, 1])$並被 $(1 - \lambda)$normalized。這個被稱為 λ return定義如下

$$G_t^\lambda = (1 - \lambda) \sum_{n-1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{12.2}$$





我們也可以將Termination之後區域分別出來

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t \tag{12.3}$$

$\lambda = 1$時主要左邊的合會變成0，右邊就是以前所學過的return，所以當 $\lambda = 1$時，這個就是**傳統的MC演算法**，如果 $\lambda = 0$，這裡就或變成 **one-step TD**因為 λ-return 會變成 one-step return，更新就會變成 one-step TD。

我們現在可以定義我們第一個基於 λ return 的學習演算法：off-line λ return algorithm，就像一個 off-line的演算法，它在一個episode中並沒有改動weight-vector，然後在episode要結束時，整個系列off-line更新會直接用semi-gradient rule去做更新，再以 λ-return 作為更新目標。

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla\hat{v}(S_t, \mathbf{w}_t), \quad t = 0, \ldots, T-1 \tag{12.4}$$

這個 λ return 是另一種介於 one-step TD跟 MC 之間並可以與 n-step bootstrapping來做比較，我們舉 範例7.1 19-state random walk 的例子。

目前我們講述這些理論，都是”向前看”的，也就是說每個拜訪我們都需要向未來去觀測未來的reward然後再想辦法去利用它們。就像我們坐在當下的state併用望遠鏡去窺視未來。

12.2 TD(λ)

TD(λ)是其中一個最古老也是一個最廣泛應用的強化學習演算法之一。這是第一個演算法運用eligibility trace的方式使演算以”向後看”的方式解決理論上”向前看”的問題，現在我們就要看我們如何熟練的預測off-line λ return的approximate表示出來。

$\mathrm{TD}(\lambda)$ 用了三種方法使其完善，**第一它會更新每一代weight vector而不是到episode結束才做更新；第二，其運算可以等量的分配而不是擠到最後的episode結束才開始做運算；最後一個則是它可以被應用在continuous problem而不是episodic problems**。在這個章節我們將會介紹 semi-gradient版本的 $\mathrm{TD}(\lambda)$ 。

藉由function approximation 這個 eligibility trace是一個向量 $\mathbf{z}_t \in \mathbb{R}^d$ 有著跟 weight vector $\mathbf{w}_t$ 相同數量的維度，不同於 weight vector是個長時間的記憶， eligibility trace是個短期的記憶，通常會小於episode的長度，Eligibility trace可以幫助學習，然後他僅會影響weight vector之後weight vector再去影響 estimated value。

在 $\mathrm{TD}(\lambda)$ ，eligibility trace table會在episode 的一開始被初始化至 0，並在每個timestep做梯度增加其值，然後再以 $\gamma\lambda$的速率漸漸消散。

$$
\begin{aligned}
\mathbf{z}_{-1} &= \mathbf{0} \\
\mathbf{z}_t &= \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}_t(S_t, \mathbf{w}_t),\ 0 \le t \le T
\end{aligned}
\tag{12.5}
$$

γ是discounted rate， λ 則是如先前介紹的那樣，所以我們把它稱呼為消散(trace-decay)參數，eligibility trace會持續監視貢獻給weight vector 的component，不管事正貢獻還是負貢獻，至最近的State的值，我們的**"最近"取決於這個** $\gamma\lambda$，(我們回想線性function approximation，也就是 $\nabla\hat{v}(S_t, \mathbf{w}_t))$ feature vector $\mathbf{x}_t$，這個eligibility vector就只是存放著過去的總和並逐漸消散的input vector，這個trace隱含著各個weight vector部件的合格性(eligibility)經歷強化學習所改變發生的過程。這個強化學習的事件會把時時刻刻的(moment by moment)的TD error考慮進去，這個TD error是

$$
\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{12.6}
$$

weight的更新在 $\mathrm{TD}(\lambda)$則是以一部分的TD error跟vector eligibility trace

$$
\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \tag{12.7}
$$

Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$, such that $\hat{v}(\text{terminal}, \cdot) = 0$

Algorithm parameters: step size $\alpha > 0$, trace decay $\lambda \in [0, 1]$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g.,$\mathbf{w} = \mathbf{0}$)

Loop for each episode:

  Initialize S

  $\mathbf{z} \leftarrow 0$ (A d-dimensional vector)

  Loop for each step of episode:

    Choose $A \sim \pi(\cdot|S)$

    Take action A, observe R, S'

    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S, \mathbf{w})$

    $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$$
$$S \leftarrow S'$$
$$\text{until } S' \text{ is terminal}$$

$\mathrm{TD}(\lambda)$會導向過去，我們看向現在TD error然後把它分派到各個先前的state端看他對state的貢獻到現在的eligibility trace。我們可以想像成騎在state上順流而下，計算TD errors然後用大聲公傳到先前經過的state，當trace跟TD error結合，我們就可以用(12.7)更新，當他未來再次發生的時候改變以前的state值。

去更進一步認識 $\mathrm{TD}(\lambda)$ 我們從 λ 的值下手，如果 $\lambda = 0$，那在 trace t 就會完全跟value gradient 對應 S_t 一樣，這樣的話 $\mathrm{TD}(\lambda)$ (12.7) 就會退化成 one-step semi-gradient method ，這也是為甚麼 one-step semi-gradient method叫做 $\mathrm{TD}(0)$。在較大的 λ ($\lambda < 1$)，先前的許多State都改變了，但是越遠的State改變的幅度就會越小，因為跟其相對應的eligibility trace也會比較少。我們就會說較早的State會從 TD error 中給予較少的"credit"。

如果 $\lambda = 1$那這個State每個step只會受這個 γ消散，這也是向MC靠攏的現象，在傳遞後k-step之後，其 reward 的 return 為 γ^k，就是 eligibility trace要做的(消散)。如果 $\lambda = 1$且 $\gamma = 1$，那這個演算法就會理論上跟MC的 undiscounting method episodic task的表現概念一樣，基於 $\mathrm{TD}(1)$ 的control method。

$\mathrm{TD}(1)$ 是一個**能更廣義的表示MC演算法的一個方式**，大幅增加其應用性與適應性。舉例來說，還記得以前MC有只能在episodic task運作的限制嗎？現在MC也可以被應用在discounted continuing task。再來就是 $\mathrm{TD}(1)$ 可以用於 on-line的學習，MC其中一個劣勢是他需要等到episode結束才能夠開始學習，舉個例子，MC control 選擇 action 然後產生出了不佳的結果然後尚未結束episode時，這個agent在這個episode作相同抉擇時還是不會考慮到上次的不佳結果。不過在 On-line $\mathrm{TD}(1)$，會在n-step TD 的模式學習非完整的episode，從n-step到現在的step，如果在episode中有發生甚麼好事或是壞事，我們可以馬上去學習並調整該個episode 的 behavior。

在19-state random walk範例，我們可以看到 $\mathrm{TD}(\lambda)$ 在off-line λ return 的表現，在最優的 α選擇下兩個表現一樣，在大於最佳的 α 時off-line λ return只有稍微比 $\mathrm{TD}(\lambda)$ 差一點，但是在較高的 α會遠差於 off-line λ return，在這個問題 $\mathrm{TD}(\lambda)$ 並非極度糟糕，因為我們本來就不常使用高 α 值，不過對於其他的問題可能就會暴露巨大的弱點。

線性的 on-policy $\mathrm{TD}(\lambda)$ 已經被證明說可以收束如果step size 參數 α 逐漸降低，就像我們先前在 (9.4) 所提到的，他並非收束在最小誤差的weight vector，而是附近跟 λ 有關的weight vector，解答的品質的界線在 (9.14) 之間並且被 λ generalized for discounted case。

$$\overline{\mathrm{VE}}(\mathbf{w}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} \min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) \tag{12.8}$$

所以說，這個漸進式錯誤不會多於 $\frac{1-\gamma\lambda}{1-\gamma}$ 倍最小錯誤，而當 λ 越發接近 1， 這個界線就會越接近最小錯誤，不過在實際上，讓 $\lambda = 1$ 會是較差的決定，後面會以圖 (12.14) 說明。

12.3 n-step method Truncated λ-return method

這個off-line λ-return 有重要的點子，不過它只有有限的應用因為 λ-return (12.2) 的關係，**因為其值只有在 episode 的最後才可以知曉**，所以這個 λ-return 理論上是未知的，因為其需要 n-step return 且 n 為任意大的數，所以那個需要知道的 reward 有任意大的 n 的距離。不過其依賴性會隨著 $\gamma\lambda$ 下降，自然的近似之後會在幾個step之後被去除掉，所以我們現在的 n-step 自然可以勝任這個讓這些缺失的reward以 estimated value 作替代。

一般來說，我們定義這個裁減過的 λ-return (truncated λ-return) 在時間 t 時，給的資料只到時間 h

$$G_{t:h}^{\lambda} = (1-\lambda)\sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} \ + \ \lambda^{h-t-1} G_{t:h} \tag{12.9}$$

如果你與 λ-return 做比較 (12.3)，我們可以很清楚的知道 horizon h 跟 terminal state T 扮演的是相同的角色，然而 λ-return 給予剩下的weight給 G_t 而這裡是給予剩下的 weight $G_{t:h}$。

這個裁減過的 λ-return 我們可以直接建立一個 λ 家族，與第七章n-step類似，在這些演算法中，更新需要延遲 n 步只會考慮第 n 步之前的reward，而現在我們所以 k-step return都需要考慮到 $1 \le k \le n$並其權重以幾何分布，在 state-value的情況下，這個演算法被稱為 Truncated $\mathrm{TD}(\lambda)$ 或是 $\mathrm{TTD}(\lambda)$。

$\mathrm{TTD}(\lambda)$可以被定義為 $(\mathrm{cf}\ (9.15))$

$$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n}^{\lambda} - \hat{v}(S_t, \mathbf{w}_{t+n-1})]\nabla\hat{v}(S_t, \mathbf{w}_{t+n-1}), \quad 0 \le t < T$$

這個演算法可以被有效的被應用所以per-step的計算不會跟 n 成正比(雖然記憶體需要)，跟n-step TD一樣，在一開始的 $n-1$ step是沒有更新的，還有額外的 $n-1$ 步會在到達時繼續更新，我們可以把公式改寫如下去使其更有效率：

$$G_{t:t+k}^{\lambda} = \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-t}\delta_i' \tag{12.10}$$

where

$$\delta'_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_{t-1})$$

12.4 Redoing updates: Online λ-return Algorithm

選擇裁減的參數 n 在 $\mathrm{TTD}(\lambda)$是有代價的，其 n 要夠大才能接近 off-line λ-return 演算法的近似值，但又要足夠小才能相對即時的更新，我們可以同時做到這兩個要求嗎？我們其實可以，代價就是犧牲一些算力。

我們的點子是，我們每一步獲得資料時，你就重頭開始更新。這個新的更新會比之前幾次的更新都還優秀因為他會考慮到這個time step新的資料。所以說，這個更新會朝向 n-step truncated λ-return，不過他會使用最新的horizon，一個episode中若是多次經過就可以使用較長的horizon去獲得較好的結果。我們回顧 (12.9)

$$G^\lambda_{t:h} = (1-\lambda)\sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} \;+\; \lambda^{h-t-1} G_{t:h}$$

我們來看這個可以怎麼去應用這個公式。這個episode的初始weight $\mathbf{w}_0$是從上一個episode的結束得來的。開始時horizon會擴展至 timestep 1，我們把timestep 1 的資料給那我們得到的就是 one-step return G^1_0，包含 R_1 跟 從預測 $\hat{v}(S_1, \mathbf{w}_0)$bootstrap的值，這完全就是 $G^\lambda_{0:1}$，第一個部分的總合為0，我們會用上述式子更新 weight 到 $\mathbf{w}_1$，那我們的data horizon到 step 2 時我們要做甚麼呢？我們現在有 R_2, S_2跟新的 $\mathbf{w}_1$，所以我們可以算出第一次update target $G^\lambda_{0:2}$ 跟更優良的第二次update target $G^\lambda_{1:2}$，有了這些改良的目標後，我們重新更新 S_1 跟 S_2 並從 $\mathbf{w}_0$ 開始以此類推，每次horizon往下步移動，所有的更新都要被重新計算，從 $\mathbf{w}_0$

這個概念透過多次經過，每次產出不同的 weight vector，我們必須要區分我們每次產出的weight vector，我們把它標記為 $\mathbf{w}^h_t$，第一個weight vector是 $\mathbf{w}^h_0$，到最後的weight vector $\mathbf{w}^h_h$，然後以此類推到最後的horizon就是 $\mathbf{w}^T_T$，他會被推到新個 episode當下個episode的初始 weight。有了上述描述，我們可以把前三個序列寫出來

$$h=1: \;\mathbf{w}^1_1 = \mathbf{w}^1_0 + \alpha[G^\lambda_{0:1} - \hat{v}(S_0, \mathbf{w}^1_0)]\nabla\hat{v}(S_0, \mathbf{w}^1_0)$$

$$\begin{aligned} h=2: \mathbf{w}^2_1 &= \mathbf{w}^2_0 + \alpha[G^\lambda_{0:2} - \hat{v}(S_0, \mathbf{w}^2_0)]\nabla\hat{v}(S_0, \mathbf{w}^2_0) \\ \mathbf{w}^2_2 &= \mathbf{w}^2_1 + \alpha[G^\lambda_{1:2} - \hat{v}(S_1, \mathbf{w}^2_1)]\nabla\hat{v}(S_1, \mathbf{w}^2_1) \end{aligned}$$

$$
\begin{aligned}
h = 3 : \mathbf{w}_1^3 &= \mathbf{w}_0^3 + \alpha[G_{0:3}^\lambda - \hat{v}(S_0, \mathbf{w}_0^3)]\nabla\hat{v}(S_0, \mathbf{w}_0^3) \\
\mathbf{w}_2^3 &= \mathbf{w}_1^3 + \alpha[G_{1:3}^\lambda - \hat{v}(S_1, \mathbf{w}_1^3)]\nabla\hat{v}(S_1, \mathbf{w}_1^3) \\
\mathbf{w}_3^3 &= \mathbf{w}_2^3 + \alpha[G_{2:3}^\lambda - \hat{v}(S_2, \mathbf{w}_2^3)]\nabla\hat{v}(S_2, \mathbf{w}_2^3)
\end{aligned}
$$

更新寫成通用的式子就是

$$
\mathbf{w}_{t+1}^h = \mathbf{w}_t^h + \alpha[G_{t:h}^\lambda - \hat{v}(S_t, \mathbf{w}_t^h)]\nabla\hat{v}(S_t, \mathbf{w}_t^h),\ 0 \le t < h \le T
$$

以這個為更新，並且 $\mathbf{w}_t = \mathbf{w}_t^t$ 可以定義為 online λ-return algorithm

這個演算法是可以完全online的，唯一的缺點就是計算十分複雜，每次進程都會傳過一部分該episode 的 experience。這比我們上次介紹的 off-line λ-return 的計算還要複雜，因為其要經過所有的step到episode結束才會更新。不過以此為代價，online λ-return algorithm 的表現會比off-line λ-return 再好一些，他在episode的最後使用bootstrapping(in $G_{t:h}^\lambda$)會有相對較大的資訊上的更新。

12.5 True Online $\mathrm{TD}(\lambda)$

剛剛的 online λ-return 演算法是目前我們提出過表現最好的 TD 演算法，不過過於複雜，所以我們想想可不可以反轉這個前瞻性(forward-review) 演算法使他變成有效的後顧型(backward-view)演算法？事實上線性approximation的計算確有與其相似之處。這個 “True” online $\mathrm{TD}(\lambda)$的確比 online λ-return 還要 “真實” 相較於 $\mathrm{TD}(\lambda)$。

$$
\begin{array}{cccccc}
\mathbf{w}_0^0 & & & & & \\
\mathbf{w}_0^1 & \mathbf{w}_1^1 & & & & \\
\mathbf{w}_0^2 & \mathbf{w}_1^2 & \mathbf{w}_2^2 & & & \\
\mathbf{w}_0^3 & \mathbf{w}_1^3 & \mathbf{w}_2^3 & \mathbf{w}_3^3 & & \\
\vdots & \vdots & \vdots & \vdots & \ddots & \\
\mathbf{w}_0^T & \mathbf{w}_1^T & \mathbf{w}_2^T & \mathbf{w}_3^T & \cdots & \mathbf{w}_T^T
\end{array}
$$

我們的策略很簡單，以前online λ-return的全重可以被排成一個三角形，三角形中的每一行都是一個time step之中產生出來的，不過我們知道這個對角這個 $\mathbf{w}_t^t$才是我們所要的。

首先這個 $\mathbf{w}_0^0$ 是整個episode初始權重，$\mathbf{w}_T^T$是最後的權重。 $\mathbf{w}_t^t$在前兩者之間扮演 n-step return的 bootstrapping的更新。在最後演算法這個對腳的 weight vector我們把上標給拿掉變成： $\mathbf{w}_t = \mathbf{w}_t^t$。再來就是找一個有效且簡短的方法去計算一個個的 $\mathbf{w}_t^t$。在這之後，如果是線性的 $\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$，我們就可以寫出這個 true online $\mathrm{TD}(\lambda)$演算法：

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t + \alpha(\mathbf{w}_t^\top \mathbf{x}_t - \mathbf{w}_{t-1}^\top \mathbf{x}_t)(\mathbf{z}_t - \mathbf{x}_t)$$

我們用了簡化的寫法 $\mathbf{x}_t = \mathbf{x}(S_t)$， δ_t已經在 $\mathrm{TD}(\lambda)$ 上定義了， $\mathbf{z}_t$則被定義為：

◗ (12.11)

其在2016年已經被證明說其 $\mathbf{w}_t, 0 \leq t \leq T$跟online λ-return所產生出來的權重完全吻合。現在這個演算法已經沒以前這麼吃資源了，記憶體的優勢不變而計算速度快了50%左右，整體上來說計算複雜度維持 $O(d)$，與 $\mathrm{TD}(\lambda)$一樣

True online TD (λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx v_\pi$

Input: the policy π to be evaluated

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$

Algorithm parameters: step size $\alpha > 0,$ trace decay rate $\lambda \in [0, 1]$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$(e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:

  Initialize state and obtain initial feature vector $\mathbf{x}$

  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)

  $V_{old} \leftarrow 0$ (a temporary scalar variable)

  Loop for each step of episode:

    Choose $A \sim \pi$

    Take action $A,$ observe $R, \mathbf{x}'$ (feature vector of the next state)

    $V \leftarrow \mathbf{w}^\top \mathbf{x}$

    $V' \leftarrow \mathbf{w}^\top \mathbf{x}'$

    $\delta \leftarrow R + \gamma V' - V$

    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + (1 - \alpha\gamma\lambda\mathbf{z}^\top \mathbf{x})\mathbf{x}$

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{old})\mathbf{z} - \alpha(V - V_{old})\mathbf{x}$

$V_{old} \leftarrow V$

$\mathbf{x} \leftarrow \mathbf{x}'$

Until $\mathbf{x'} = \mathbf{0}$ (signaling arrival at a terminal state)

---

true online $\mathrm{TD}(\lambda)$ 所使用的 eligibility trace (12.11)被稱作為 dutch trace，為了要跟 (12.5) $\mathrm{TD}(\lambda)$ 的 trace 做區分，他則是叫座 accumulating trace。在較早用過的第三種叫做 replacing trace，只有在 tabular case 或是只有在 二元(binary) feature vectors在 tile coding被製造出來的，這個replacing trace定義為各個部件是否為feature vector 為 1 或是 0。

$$z_{i,t} = \begin{cases} 1 & \text{if } x_{i,t} = 1 \\ \gamma\lambda z_{i,t-1} & \text{otherwise.} \end{cases} \tag{12.12}$$

現在我們可以把 replacing trace 視為 dutch trace的粗略估計，大量的取代(supersede) dutch trace，dutch trace通常表現的會比 replacing trace。Accumulating trace則是用於非線性 function 也就是當 dutch trace無法使用時。

12.7 $Sarsa$(λ)

我們僅需要些微的改變就可以將演算法應用在 action value method，去學習預測 action-value ，我們需要去預測 $\hat{q}(s, a, \mathbf{w})$ 而非 $\hat{v}(s, \mathbf{w})$。我們需要使用 n-step action-value 的回傳型態

$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}), \quad t+n < T$

with $G_{t:t+n} = G_t$ if $t+n \geq T$

我們可以藉由上述型態去組成 truncated λ-return，除此之外其他都跟 (12.9) state-value 一樣， action-value 型態的 off-line λ-return algorithm (12.4) 純粹只是用 $\hat{q}$ 而不是 $\hat{v}$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[G_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \\ t = 0, \ldots, T-1 \tag{12.15}$$

where $G_t^\lambda = G_{t:\infty}^\lambda$，我們可以注意到它跟 $\mathrm{TD}(\lambda)$的相似之處。

這個 method 也可以稱之為 $Sarsa(\lambda)$ 其更新的，有著跟 $\mathrm{TD}(\lambda)$ 一樣的更新方法

$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t$

我們用它的 action-value 計算 TD error

$$\tag{12.15}$$

action-value 型態的 eligible trace

$$
\begin{aligned}
&\mathbf{z}_{-1} = \mathbf{0} \\
&\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t),\ 0 \le t \le T
\end{aligned}
$$

Sarsa(λ) with binary features and linear function approximation for estimating $\mathbf{w}^\top\mathbf{x} \approx q_\pi$ or q_*

Input: a function $\mathcal{F}(s,a)$ returning the set of (indices of) active features for s, a
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize: $\mathbf{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1, \ldots, z_d)^\top \in \mathbb{R}^d$

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{z} \leftarrow \mathbf{0}$
  Loop for each step of episode:
    Take action A, observe R, S'
    $\delta \leftarrow R$
    Loop for i in $\mathcal{F}(S, A)$:
      $\delta \leftarrow \delta - w_i$
      $z_i \leftarrow z_i + 1$ (accumulating traces)
      or $z_i \leftarrow 1$ (replacing traces)
    If S' is terminal then:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
      Go to next episode
    Choose $A' \sim \pi(\cdot|S')$ or near greedily $\sim \hat{q}(S', \cdot, \mathbf{w})$
    Loop for i in $\mathcal{F}(S', A')$: $\delta \leftarrow \delta + \gamma w_i$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
    $S \leftarrow S'; A \leftarrow A'$

這裡也有action-value 版本的理想(ideal) TD method，就是 online λ-return 演算法還有其衍伸效率較高的演算法 True online TD (12.4)的部分幾乎不用做任何改動就可以套上 action value，其中改變只有 feature vector $\mathbf{x}_t = \mathbf{x}(S_t)$ 需要改成 $\mathbf{x}_t = \mathbf{x}(S_t, A_t)$，True online $Sarsa(\lambda)$ 為其名稱。

True Online Sarsa(λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or near greedily from S using $\mathbf{w}$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
  | Take action A, observe R, S'
  | Choose $A' \sim \pi(\cdot|S')$ or near greedily from S' using $\mathbf{w}$
  | $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
  | $Q \leftarrow \mathbf{w}^\top \mathbf{x}$
  | $Q' \leftarrow \mathbf{w}^\top \mathbf{x}'$
  | $\delta \leftarrow R + \gamma Q' - Q$
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
  | $Q_{old} \leftarrow Q'$
  | $\mathbf{x} \leftarrow \mathbf{x}'$
  | $A \leftarrow A'$
  until S' is terminal

最後剪裁版本的 $Sarsa(\lambda)$ 可以被稱為 forward $Sarsa(\lambda)$，其對於多層無model的類神經網絡有一定的效果

12.8 變數 λ 與變數 γ

我們逐漸到達基礎TD learning 發展史的結尾，為了展示的演算法的最終型態，我們必須要generalize bootstrapping以及discounting 從一個常數參數轉變為依附在state跟action上的，所以，每個不同的 timestep都有不同的 λ 以及 γ，表示為 λ_t 及 γ_t，我們現在將 λ 與獨立的 state 及 action做連結 $\lambda : \mathcal{S} \times \mathcal{A} \to [0, 1]$現在是一個函數使得 $\lambda_t = \lambda(S_t, A_t)$，然後嘎瑪也一樣 $\gamma : \mathcal{S} \to [0, 1]$是與 state 做連結使得 $\gamma_t = \gamma(S_t)$。

我們現在來介紹 γ 現在為我們的 termination function，他因為會改變 return 所以她頗為重要，是我們所想要預測的期望值。現在這個 return被可以更廣義的被定義為

$$\begin{aligned} G_t &= R_{t+1} + \gamma_{t+1} G_{t+1} \\ &= R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \gamma_{t+1}\gamma_{t+2}\gamma_{t+3} R_{t+4} + \dots \\ &= \sum_{k=t}^{\infty} \left(\prod_{i=t+1}^{k} \gamma_i \right) R_{k+1} \end{aligned} \tag{12.17}$$

假設這個總和是有限的，我們需要得到 $\prod_{k=t}^{\infty} \gamma_k = 0$在機率為1在所有 t 下，其中一個理念是他可以將一系列的經驗在沒有特殊terminal state，起始的或是終結的時間點的該episode的設定及演算法表示出來。在過往(erstwhile)的terminal state在 $\gamma(s) = 0$跟轉移(transition)至分布的起始，我們便可以將以前經典的 episodic setting給帶回來(在選擇 $\gamma(\cdot)$ 為常數在所有其他的 states 下)。獨立於各個State 的中止(termination)包含在prediction cases像是 pseudo termination，在我們不影響到Markov process下我們估算它的量。Discounted return 可以被解讀為 state-獨立的 termination 統一了 episodic 以及 discounted-continuing cases。

這個 **對變數的 bootstrapping 的 generalization並非在問題上做文章，而是對結果**，這個 generalization 影響 state, action 的 λ-return。這個新的 state-based λ-return 可以以遞迴的方法寫成

$$G_t^{\lambda s} = R_{t+1} + \gamma_{t+1}((1-\lambda_{t+1})\hat{v}(S_{t+1}, \mathbf{w}_t) + \lambda_{t+1} G_{t+1}^{\lambda s}) \tag{12.18}$$

💡 $G_t^{\lambda} = (1-\lambda)\sum_{n-1}^{\infty} \lambda^{n-1} G_{t:t+n}$
(12.1)

不過現在我們加上 “ s ” 在 λ 上表示他是從 state value 上去做 bootstrap的，藉此我們可以分辨它並非從 action value 去做 bootstrap 得來的，action value 去做 bootstrap 得來的我們會在下列表達式加入 “ a ”做區分。這個等式說明了他是 λ-return 的第一個reward 不會被 discounted 也不會被 bootstrapping 影響，加上有可能的第二個條件(term) 去延伸下個state不去做discounting(也就是說，我們回想以前的情況起下這個state如果是terminal則 γ_{t+1} 就會為 0) ，以至於我們下個 state並不會終結，我們會有第二個term自己分割成兩個 cases 表示端看於他在 state 做了多少程度的bootstrapping。當我們在進行 bootstrapping 的情況下，這是在預測 value of the state，而在沒有bootstrapping的情況下，我們用 λ-return去計算下個 time step。而這個 action-based可以是 Sarsa的形式

$$G_t^{\lambda a} = R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) + \lambda_{t+1}G_{t+1}^{\lambda a}\big) \tag{12.19}$$

或是 Expected Sarsa的形式

$$G_t^{\lambda a} = R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\bar{V}_t(S_{t+1}) + \lambda_{t+1}G_{t+1}^{\lambda a}\big) \tag{12.20}$$

where (7.8) function approximation 的公式被 generalization 為

$$\bar{V}_t(s) = \sum_a \pi(a|s)\hat{q}(s, a, \mathbf{w}_t) \tag{12.21}$$

12.9 Off-policy Trace with Control Variance

Sect 5.8 5.9 7.4傳送門

最後一步則是要合併 importance sampling，在非剪裁的 λ-returns，這裡沒有比較實用的方法可以將importance sampling 從 target return 分離出來(就像 (7.3) n-step method)，不過我們可以直接去 bootstrapping generalization of per-decision importance sampling with control variate (section (7.4), section (5.9) 補充教材)

在這個 cases ，我們最終 generalize λ-return (12.18) 的定義以公式 (7.13) 為模板

$$\begin{aligned} G_t^{\lambda s} = \rho_t\Big(R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\hat{v}(S_{t+1}, \mathbf{w}_t) + \lambda_{t+1}G_{t+1}^{\lambda s}\big)\Big) \\ +(1-\rho_t)\hat{v}(S_t, \mathbf{w}_t) \end{aligned} \tag{12.22}$$

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

where $\rho_t = \frac{\pi(A_t, S_t)}{b(A_t, S_t)}$ 是常見的一步importance sampling ratio，就像其他return一樣， 剪裁版本的 return可以被state-based TD error的總和預測。

$$\delta_t^s = R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{12.23}$$

而

$$G_t^{\lambda s} \approx \hat{v}(S_t, \mathbf{w}_t) + \rho_t \sum_{k=t}^{\infty} \delta_k^s \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \tag{12.24}$$

Exercise 12.9 Truncated version of $G_{t:h}^{\lambda s}$

$$G_{t:h}^{\lambda s} \approx \hat{v}(S_t, \mathbf{w}_t) + \rho_t \sum_{k=t}^{h} \delta_k^a \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i$$

以上的 λ-return形式 (12.24)是便於使用的 前瞻式(forward-review) 更新

$$\begin{aligned}
\mathbf{w}_{t+1} =& \mathbf{w}_t + \alpha\big(G_t^{\lambda s} - \hat{v}(S_t, \mathbf{w}_t)\big)\nabla\hat{v}(S_t, \mathbf{w}_t) \\
\approx& \mathbf{w}_t + \alpha\rho_t \left(\sum_{k=t}^{\infty} \delta_k^s \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i\right) \nabla\hat{v}(S_t, \mathbf{w}_t)
\end{aligned}$$

這很像我們所熟知的 eligibility-based TD update，這個乘積很像 eligibility trace 然猴他與 TD error相乘，不過這僅僅是單個 step的前瞻(forward view)。我們需要從 forward-review update的加總和相似的backward-view加總中尋找其關聯性(這只能求其近似因為我們需要忽略value function的改變)。forward-view的總合為

$$\begin{aligned}
\sum_{t=1}^{\infty}(\mathbf{w}_{t+1} - \mathbf{w}_t) &\approx \sum_{t=1}^{\infty}\sum_{k=t}^{\infty} \alpha\rho_t\delta_k^s \nabla\hat{v}(S_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \\
&= \sum_{k=1}^{\infty}\sum_{t=1}^{k} \alpha\rho_t \nabla\hat{v}(S_t, \mathbf{w}_t)\delta_k^s \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \\
&\text{(using summation rule)} \\
&= \sum_{k=1}^{\infty} \alpha\delta_k^s \sum_{t=1}^{\infty} \rho_t \nabla\hat{v}(S_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i
\end{aligned}$$

然後會以 backward-view TD update 去詮釋上面，我們現在可以證明這整個可以被改寫，甚至也可以 eligibility trace 做逐步更新。所以說我們假設這個trace是到時間 k ，我們可以在時間 $k-1$ 時更新

$$
\begin{aligned}
\mathbf{z}_k &= \sum_{t=1}^{k} \rho_t \nabla \hat{v}(S_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \\
&= \sum_{t=1}^{k-1} \rho_t \nabla \hat{v}(S_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i + \rho_k \nabla \hat{v}(S_k, \mathbf{w}_k) \\
&= \gamma_k \lambda_k \rho_k \underbrace{\sum_{t=1}^{k-1} \rho_t \nabla \hat{v}(S_t, \mathbf{w}_t) \prod_{i=t+1}^{k-1} \gamma_i \lambda_i \rho_i}_{\mathbf{z}_{k-1}} + \rho_k \nabla \hat{v}(S_k, \mathbf{w}_k) \\
&= \rho_k (\gamma_k \lambda_k \mathbf{z}_{k-1} + \nabla \hat{v}(S_k, \mathbf{w}_k))
\end{aligned}
$$

把 k 改成 t，我們就可以廣義化 accumulating trace update for state values

$$
\mathbf{z}_t = \rho_t (\gamma_t \lambda_t \mathbf{z}_{t-1} + \nabla \hat{v}(S_t, \mathbf{w}_t)) \tag{12.25}
$$

💡 $\mathbf{z}_t = \gamma \lambda \mathbf{z}_{t-1} + \nabla \hat{v}_t(S_t, \mathbf{w}_t)$

Eligibility trace 結合 semi-gradient parameter update rule 可以產生一個 TD 演算法可以被應用在 on-policy 以及 off-policy 上。如果應用在 on-policy，這個 ρ_t 就會是 1 ，那這個演算法就會跟 $\mathrm{TD}(\lambda)$完全一樣， (12.25) 就會變成一般的 accumulating trace (12.5)，不過在 off-policy 的版本，這個演算法偶爾表現得不錯，不過在 semi-gradient method 他並不保證會穩定。在後面的區域我們會考慮它會保證穩定的時候。

而在action value 我們也可以用相同的手法做針對 action value 的 off-policy eligibility traces，其相對應的演算法就是 $Sarsa(\lambda)$ 的演算法，我們可以以遞迴的型式的 (12.19) 或是 (12.20)， 但是 Expected Sarsa好像會比較單純一點。我們將 (12.20) 做延伸到 off-policy 的形式至 (7.14) 可以產生

$$
\begin{aligned}
G_t^{\lambda a} =& R_{t+1} + \gamma \Big((1 - \lambda_{t+1}) \bar{V}_t(S_{t+1}) + \\
& \lambda_t [\rho_{t+1} G_{t+1}^{\lambda a} + \bar{V}_t(S_{t+1}) - \rho_{t+1} \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)] \Big) \\
=& R_{t+1} + \gamma_{t+1} \Big(\bar{V}_t(S_{t+1}) + \lambda_{t+1} \rho_{t+1} [G_{t+1}^{\lambda a} - \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)] \Big)
\end{aligned} \tag{12.26}
$$

$\bar{V}_t(S_{t+1})$ 在 (12.21) 已經有提到過了，我們再一次看到 λ-return 可以被大略寫成 sum of TD error

$$G_t^{\lambda a} \approx \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) + \sum_{k=t}^{\infty} \delta_k^a \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \tag{12.27}$$

使用的是期望值形式的 action-based TD error

$$\delta_t^a = R_{t+1} + \gamma_{t+1} \hat{V}_t(S_{t+1}) - \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{12.28}$$

Exercise 12.11 Truncated version of $G_{t:h}^{\lambda a}$

$$G_{t:h}^{\lambda a} \approx \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) + \sum_{k=t}^{h} \delta_k^a \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i$$

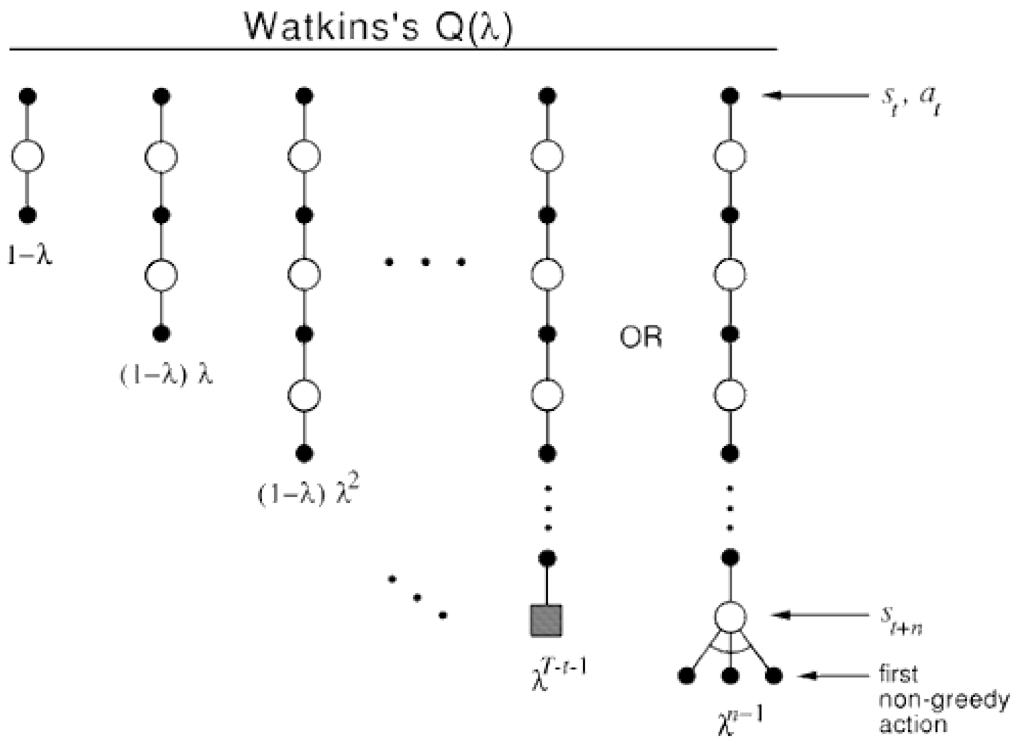
我們可以類比(analogous) state case去寫出一個 forward-view update基於 (12.27)，用 summation rule計算 the sum of the update，我們再根據其形態組成我們要的 action value 的 eligibility trace。

$$\mathbf{z}_t = \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)) \tag{12.29}$$

這個 eligibility trace 結合了 experience-base TD error (12.28)以及正規的 semi-gradient parameter update rule (12.7) 組成了一個有效且優雅的 $\text{Expected Sarsa}(\lambda)$ 可以被應用到 on-policy 以及 off-policy的演算法。這可能是目前表現最好的演算法

12.10 Watkin’s Q(λ) to Tree-Backup(λ)

歷史有一些 method被提出來讓 Q learning 帶入 eligibility trace，最原始的版本就是 Watkin的Q(λ)，他的 eligibility trace會逐漸消退前提是他需要選擇 greedy action，然後在走非 greedy 的時候將 trace清空。在章節6我們統一了 Q-learning 跟 Expected Sarsa 在 off-policy 版本，Q-learning 則作為特殊的 case，並且我們還 generalize了它使其可以是任意的 target policy，在上一個部份我們 generalize 它到 off-policy eligibility trace 的形式。在第七章，我麼卻把 n-step Expected Sarsa和 n-step Tree Backup 分開，只有後者不用 importance sampling。而現在我們這個 eligibility trace 版本的Tree-Backup 這裡稱為 Tree-Backup(λ) 或是 $\text{TB}(\lambda)$。這個是第一個成功提取 Q-learning 精隨並應用在 eligibility trace上的 method且也是不需要計算 importance sampling 並且還是可以應用在 off-policy data上。



$TB(\lambda)$ 很直覺

其每種長度的權重就是一樣依賴著 λ，而詳細的公式我們可以借鑑 (12.20) 的遞迴寫法然後再導入 λ, γ還有 (7.16) 的公式模型。

$$
\begin{aligned}
G_t^{\lambda a} =& R_{t+1} + \gamma_{t+1}\Big((1-\lambda_{t+1})\bar{V}_t(S_{t+1}) \\
&+ \lambda_{t+1}\Big[\sum_{a \neq A_{t+1}} \pi(a|S_{t+1})\hat{q}(S_{t+1}, a, \mathbf{w}) + \pi(A_{t+1}|S_{t+1})G_{t+1}^{\lambda a}\Big]\Big) \\
=& R_{t+1} + \gamma_{t+1}\Big(\bar{V}_t(S_{t+1} + \lambda_{t+1}\pi(A_{t+1}|S_{t+1}))\big(G_{t+1}^{\lambda a} - \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)\big)\Big)
\end{aligned}
$$

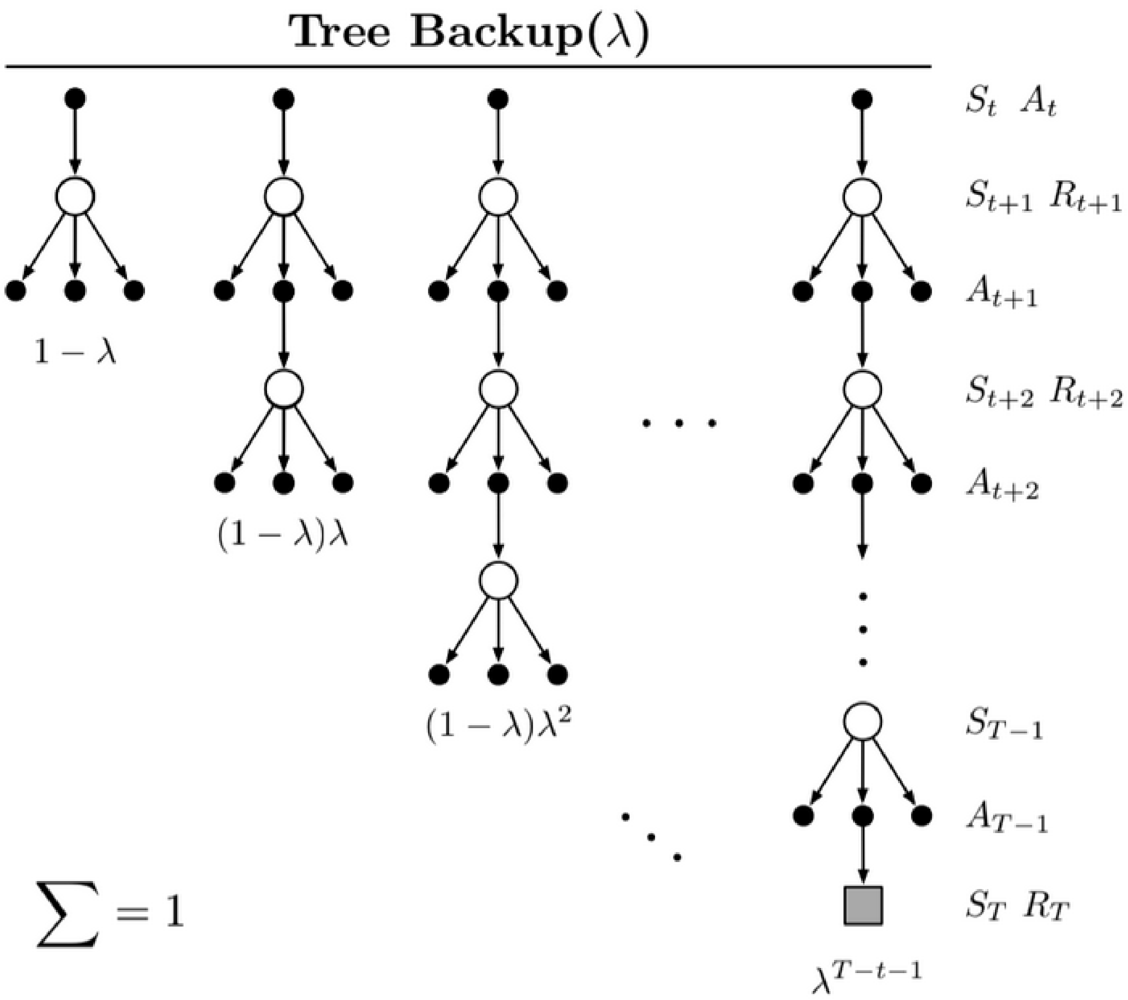
我們也可以依照同樣形式寫成近似 (忽略 approximation value function 的改變)是TD error的總和。

$$
G_t^{\lambda a} \approx \hat{q}(S_t, A_t, \mathbf{w}_t) + \sum_{k=t}^{\infty} \delta_k^a \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i)
$$

用的是期望值版本的 action-base TD error (12.28)

跟上一個 section 一樣，我們可以把它寫成一個特別的 eligibility trace update 包含 target-policy 的選擇機率的action

$$
\mathbf{z}_t = \gamma_t \lambda_t \pi(A_t|S_t)\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t)
$$



這個再加上 usual parameter-update rule (12.7)，定義了 $TB(\lambda)$ 演算法，就像所有的 semi-gradient algorithm，使用 off-policy data的 $TB(\lambda)$ 並不保證會穩定，要使其穩定需要結合下一個部分的其中一個method。

12.11 Stable Off-policy Methods with Traces