Chapter 8 Planning and Learning with Tabular Methods

Created	@December 7, 2023 8:43 AM
Tags	

written by Yee

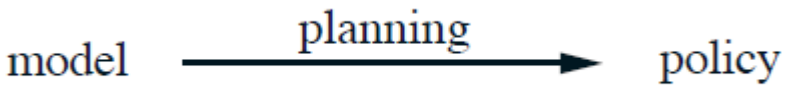
8.1 Model and Planning

我們在說的代表著環境environment的model是任何東西 the agent可以用來預測做出這個action後環境如何去做反饋。給定一個state 跟 action，model就會產預測之結果的next state以及next reward。如果model是隨機性的，那我們就會有多個next possible states 或是 rewards，每條路都有機率會發生。有些model 會給你所有可能及所有機率，我們稱之為 **distribution model**。而其他則是在其中一個有可能發生的去做採樣，我們則稱為 **sample models**。

比如說我們已眾多骰子的總合建立 model ，distribution model 是所有有可能的總合的所有機率，而sample model則會是單一個總和的機率。這種model是DP的，這種model以前已經應用在21點(BlackJack)上了。雖然distribution model 遠比 sample model強大，但是在絕大部分情況下獲得sample model的難度會低非常多。我們可以很容易的就把單一個sum的sample model寫出來，但是distribution model 就十分的困難。

經驗是可以用Model來模仿或是模擬的。如果有初始state 跟 action ， sample model 可以產出可能的下一代，而distribution model 可以產生所有可能的後代及其機率的比重。給定一個policy跟起始state，sampling model可以給你一個episode，而distribution model可以給你所有可能的episode 以及給你這些 episodes 的發生機率。這些model是用來**模擬環境並產出模擬資料(simulated experience)**。

“Planning” 這個字在很多個領域中有不同的理解，在RL它表示匯入的model並在這個modeled environment所做出的運算和改善policy的互動過程。



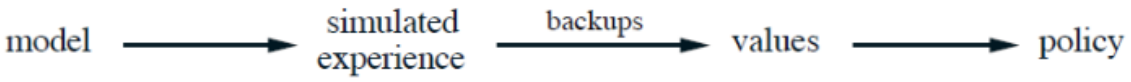
在AI，我們有兩種方法去做planning：

- State-space planning 包含在我們以前曾經使用過的方法，也就是以尋找 state space來判斷出最佳policy，action會讓一個state轉移到另一個state，然後value functions再依據states來計算。
- Plan-space planning ，會以planning去替代state space 的尋找。操作者可以從一個plan 轉變成另一個plan。如果有value function的話，那它就是根據plan space來定義。planning space包含了 evolutionary method以及 “partial-order planning”，一個AI中常見的planning方式，其steps的順序在planning的所有階段中是不完全確定的。

不過Plan-space planning比較難應用在隨機性的選擇，而這在是我們在RL理會重點討論的項目。

我們在這邊要提出的一個統一觀點就是，所有的state-space planning methods有一個通用的架構，這個架構也存在於本書所介紹的學習方法中。這一章剩餘的部份我們會來好好的說說這個觀點，不過有兩個基本的概念：

1. 所有的state-space planning methods都涉及計算value functions來做為改進policy的關鍵中間步驟
2. 它們會透過用於模擬經驗的updates或是backup的操作來計算value function



DP十分適合這個架構，他掃過所有的space of state 並產生每個state的機率分布以及可能的轉移(轉移下個state)，每個分布會被用來計算一個backup-value(更新目標)然後再去更新 state 的估計值。

用這樣的方法來看待planning methods就是為了強調planning methods跟我們提過的強化學習方法之間的關係。learning跟planning的核心利用backing-up update的操作來估測value function。不同的地方在於這是跑模擬得到的經驗還是實際上跟真實的環境互動所得到的經驗，這個差異會導致許多方面的不同，比如說效能的評估以及產生的經驗有多彈性。但是這個常見的架構表示了很多想法或是演算法都可以在planning跟learning之前互相轉換。在許多的情況下，演算法都可以被替換成planning method。學習method只需要透過經驗作為輸入，而這些經驗常常是透過模擬所得出的經驗而不是實際經驗。

Random-sample one-step tabular Q-planning

Loop forever:

1. Select a state $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain a sample next reward, R, and a sample text state, S'
3. Apply one-step tabular Q-learning to S, A, R, S' :
   $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

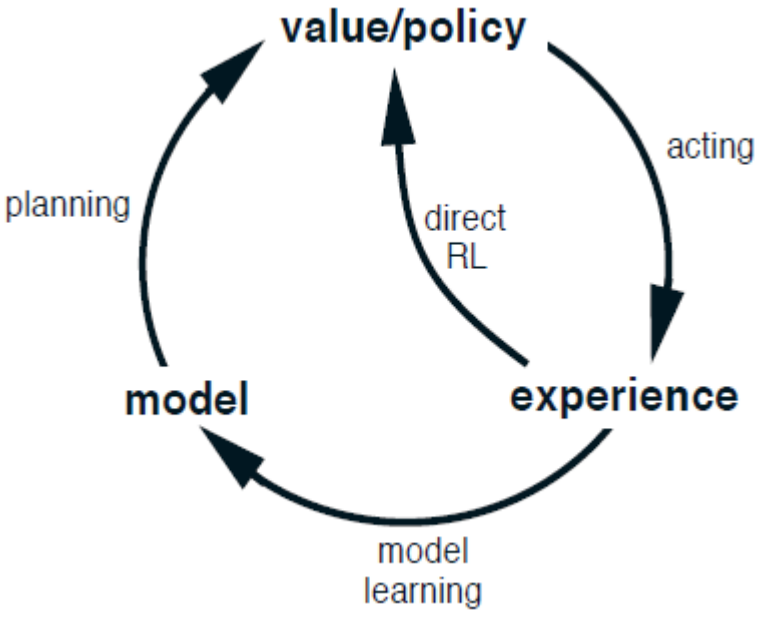
為了要統一planning跟learning methods，第二部分會講述小，逐步的steps。這讓planning可以隨時被中斷或是重新導向，僅僅只需花費極小的計算，而這計算的花費正是在整合planning及learning的必要關鍵。以極小的step去做planning可能是解決十分大的問題的最有效的方法既使是在純粹planning的問題上。

8.2 Dyna: Integrated Planning, Acting, and Learning

當planning是在線上，就在你跟環境作互動的時候，有一個有趣的問題會發生。新的資料有可能會改變model的樣貌而去跟planning作互動。也許我們應該用某種方法去改變正在考慮的state或是decision的planning過程。倘若決策跟model的建構都需要大量的運算，運算資源應該適當分配給他們兩個。這時候我們有 **Dyna-Q** 這個簡易的架構去整合主要會在planning時用到地的functions。每個Dyna-Q 的 function都是以最純粹簡易的形式羅列。現在我們只談論她的ideas。

在planning agent中，真實經驗(real experience)是有兩個主要的用途：

1. 改善model (讓 model 更貼近environment) ，我們稱之為**model learning**，或是**indirect reinforcement learning**。
2. 可以直接用我們前幾章節講的演算法去改善 value function 跟 policy，我們稱之為**direct reinforcement learning**。



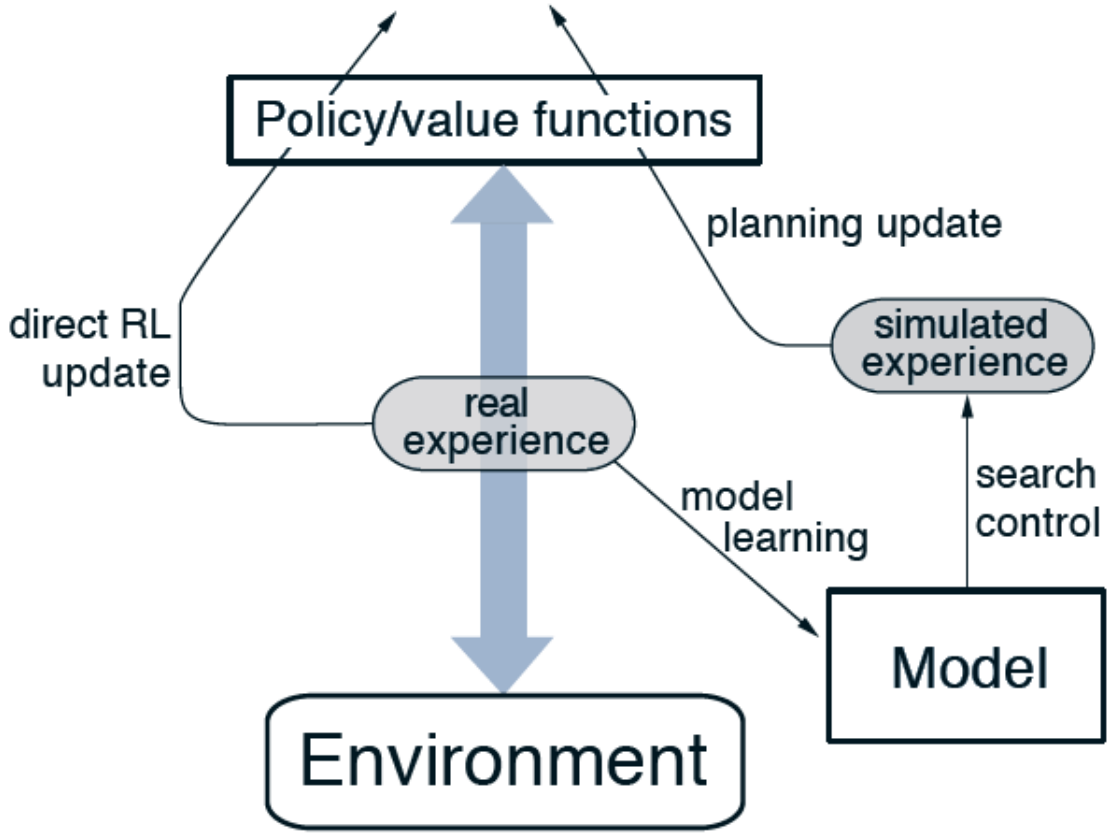
兩個方法都有其優劣

Indirect method 會充分應用有限的經驗，在最低限度的跟環境作互動下達成更新policy。

Direct method 比較簡單也不用牽涉到模型的設計及建構，也比較貼近人類及動物學習方式。

Dyna-Q 包含了以上圖的所有部分。planning method是 random sample tabular Q-learning。(前面的pseudocode)，model learning method是 table-based然後假設環境是deterministic的。在每個transition $S_t, A_t \rightarrow R_{t+1}, S_{t+1}$，model 會去紀錄檯面上的輸入直 S_t, A_t跟後面deterministic決策的 R_{t+1}, S_{t+1}，如果有遇到以前的state-action pair，他純粹會回傳上次觀測到的下一步的state跟下一個reward當作預測。在planning時，Q-planning algorithm只會從先前已經有過經驗的state-action pair來隨機採樣(Step 1)，因此model這輩子永遠不會被查詢到沒有信息的state-action pair。

Dyna agent的整體架構，Dyna-Q是其中一員，如下圖



中間那列代表agent跟environment做的互動，產出新的trajectory of real experience。右半邊為model-based 流程，model會從real experience學習並創立simulated experience，而所謂的 search control是在model生成的模擬經驗中選擇初始 state跟action，planning就是在最後再將模擬經驗應用在強化學習的 method上就像他以前有實際發生過。基本上 強化學習的方法像Dyna-Q， 是同時從真實的經驗跟模擬經驗中學習。Learning跟planning已經整合，其相異的地方僅僅是在產出經驗的地方的不同。

觀念上說planning, acting, model-learning跟direct RL可以同時發生並Dyna agent可以做平行處理(parallel)。我們將其在電腦中具體化(concreteness)並化為實作，但是我們需要給這些function每個一個先後被呼叫的順序。在Dyna-Q，acting, model learning跟direct RL需求較少的運算，我們可以忽略他們的運算時間，而最終運算都是要被用來做planning process的。

Tabular Dyna-Q

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

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \epsilon$-greedy(S,Q)

(c) Take action A; observe resultant reward, R, and state S'

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

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f)

Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

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

Direct reinforcement learning, model-learning, planning分別為 (d), (e), (f)

EX 8.1 Dyna Maze

4 方位迷宮, reward是0除了end state 1， γ是0.95，agent是決策性的

8.3 When the Model Is Wrong

在上一個example，model幾乎沒有改變，model一開始是空的，後來漸漸地塞滿正確資訊，一般來說，我們不能每次都這麼幸運，model可能會因為環境具有隨機性而且僅有有限的樣本被觀測到，或者因為model是用函數近似(function approximation)來學到的，而這個函數近似的一般化可能不是那麼樣的完美，或是環境已經改變但是model還沒有跟上。如果model是錯誤的，planning的過程可能就會執行非最佳的policy。

有時候這個非最佳的 policy 會在 planning 的時候發現自己並非最佳轉而改善成modeling error，這通常在model感覺到有更好的reward或是較優的state transition時往往會發生這個情況，所以這個planned policy 會嘗試著透過運算去利用這些機會但終究發現事實並非如此。

這個也是一個exploration跟exploitation的衝突，exploration本意是對action做嘗試，而exploitation是policy對現在model最佳的走法的實行。我們需要agent去尋找環境的變化，但又不要大到去影響到效能。

Dyna-Q+ agent 用了一個啟發式的解決方法解決了shortcut maze，這個 agent 持續觀察每個 state-action pair 然後計算上次更新過的時間是多久以前，如果時間經過得越多，探索走這步的機率就會提升，為了要鼓勵policy嘗試許久未經過的地方，我們要有

一個”特別獎勵”如果模擬有包含他，假設 model中state 的轉移有reward r 然後已經有 τ 的時間沒有到訪過了，那如果這一個state 轉移有產出reward r ， planning update 會用 reward $r + \kappa\sqrt{\tau}$, κ為值小常數。這鼓勵 agent去嘗試所有可行的state轉移，即使為了實現這樣的測試需要執行很多很多的actions，當然，這樣的測試是需要成本的，不過更多情況下，這種computational curiosity(好奇心)是值得的。

8.4 Prioritized Sweeping

在先前的 Dyna Agent，其模擬轉移是起始於從所有先前所經歷過的pairs中隨機均勻選擇的state-action pairs，但是這種均勻的選擇通常並非最佳的方式，如果我們能針對某幾個比較重要的 state-action pair去做較頻繁的更新。打個比方，如果第一個Maze task，在第二個episode的一開始，只有在終點旁邊的 state-action pair有值，其他都是0，這樣過多的模擬transition是沒有用的。所以均勻的模擬會有很多非必要的更新。隨著planning的進展，有效的區域會一直增加，但是planning的效能仍然是遠不如將重點放在在最能發揮的地方。做為我們真正目標的更大問題中，states的數量是很多，以致於這種非聚焦的搜尋十分沒有效率。

上面的範例告訴我們一件事，那就是透過從**目標狀態(goal state)反向作業**，我們可以讓搜尋(search)這個動作可以比較**聚焦**(focused)。當然，我們並不是真的想要使用任何一種特定於"目標狀態"這個概念的方法。我們想要的是一個通用的reward function。終點state只是一個特殊案例，方便去刺激直覺(stimulating intuition)。假設agent已經發現環境已經改變期望值，不管是變多還是變少。理論上來說，這可能暗示了其他state的期望值也被改動，然後在這更之前的state也許也會改變，以此類推，這之後的有受該state影響的所有值都會發生變化，我們可以從任意一個value有變化的的states反向作業(work backward)，要嘛做有效的更新或終止這個傳播(propagation)。這樣子的想法可以稱為backward focusing of planning computations。

隨著有效更新的邊界(frontier)一直迅速向後延伸，通常會產出大量的 state-action pair需要被更新，有些的值改變甚大，有些只改變一點點。而這些改變甚大的後代大概也有巨大的差異。在具有隨機性的環境下，預估的轉化(transition)的機率的差異也會影響著改變的大小以及最需要先變化的state-action pair 。我們自然會從她最需要變化的優先級先行去更新優先級高的目標，這個就是所謂的**priority sweeping**。會有一個queue(序列)去維持說每個State-action pair如果更新了會大幅度的更改到了哪些state-action pair的期望值。所以當該state-action pair值更新了，這些後代都會一併計算並更新。如果這個更新幅度大於某定值，這個pair會被插入到一個更新其優先度(較前面的優先度)，這樣子改變的效果可以被有效的傳播至後代到靜止(quiescence)

Priority Sweeping for a Deterministic Environment

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ and $PQueue$ to empty

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow policy(S,Q)$

(c) Take action A; observe resultant reward, R, and state S

(d) $Model(S,A) \leftarrow R, S'$

(e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$

(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P

(g)

Loop repeat n times, while $PQueue$ is not empty:

$S, A \leftarrow first(PQueue)$

$R, S' \leftarrow Model(S,A)$

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

Loop for all $\bar{S}, \bar{A}$ predicted to lead to S

$\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$

$P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S}, \bar{A})|$

if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

將prioritized sweeping擴展到隨機環境是很簡單的一件事。這模型是利用持續的計算每個state-action pair被看過的次數以及下一個會是什麼state來維護。而且不是像之前一樣的用一個樣本來更新，而是考慮所有可能的下一個state以及它們可能發生的機率來更新，也就是expected update。

Prioritized sweeping只是一種為了planning效率去分配計算頻率的方法，也許亦不是最佳的方法。其中一個限制是他是用期望值更新，在隨機性的環境可能會造成大量的計算浪費在低機率會發生的事情上。在下一節，樣本的更新可以在以較小的計算之下接近true value，即使sampling會導致較大的差異性。樣本更新會贏過傳統方法因為她把計算的部分拆解成小塊，根據相對應的轉換，轉而去針對變化較大的部分。這些是沿單一轉移的更新，如樣本更新(sample update)，但是是基於沒有採樣的轉移機率，像是expected update(期望更新?)。透過選擇完成那些小範圍更新的順序，我們可以大幅度的提升planning的效率，超過prioritized sweeping。

這個章節的 state-space planning可以被視為值的更新，指示是以預測或是樣本更新、大或小、更新順序去作出區別。這章節也強調哦 backward focusing，不過這也僅僅是其中一種策略。比如說舉個例子，也許有一種方法是關注在states上，根據什麼？根據當下的policy從那些經常遇到的states到達它們有多麼容易，這可以稱為forward focusing(前向關注?)。後續會介紹一下一種極端的形式。
