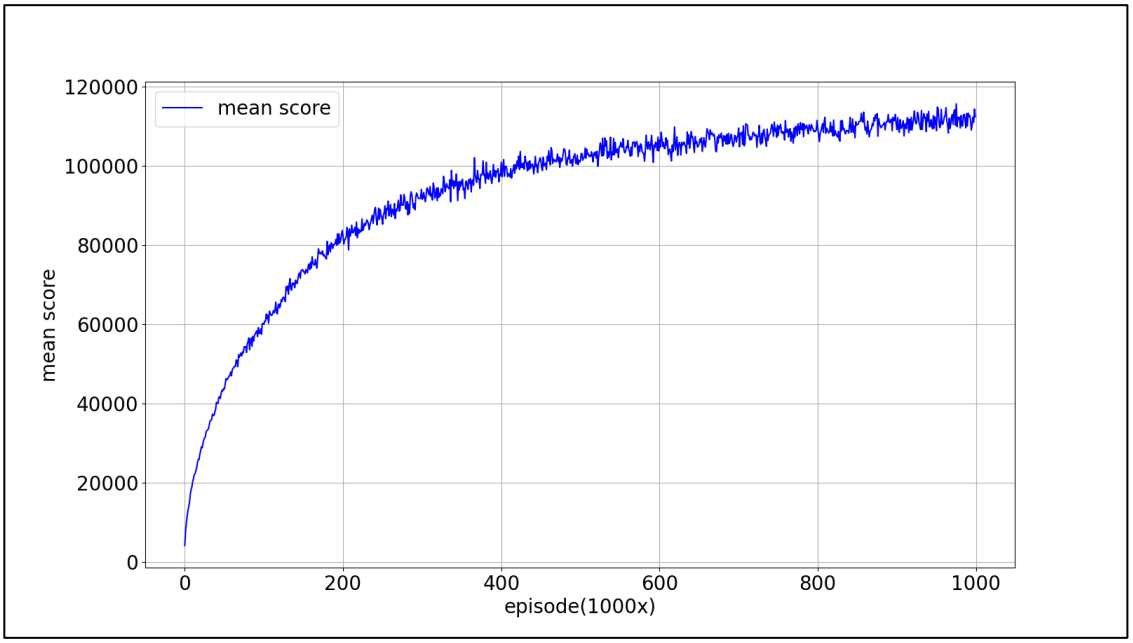
Temporal Difference Learning
Lab Report # 1

By
312581020
許瀚丰

Selected Topics in Reinforcement Learning
Fall 2023
Date Submitted: October 1, 2023

- A plot shows scores (mean) of at least 100k training episodes (20%)
  - 結果為運算 1000k 個 episodes 所獲得的平均分數

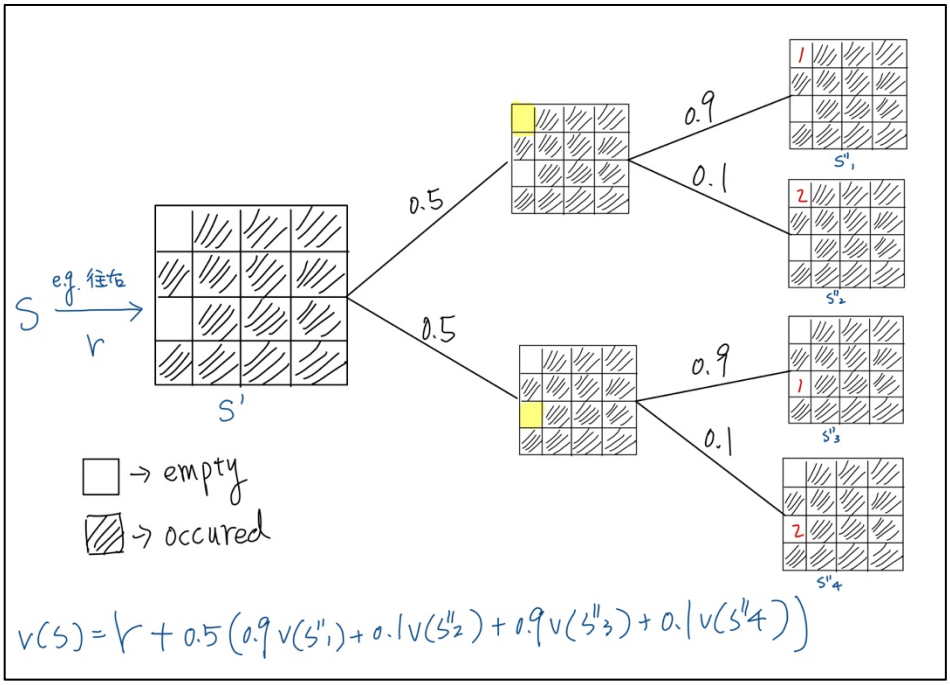


- Describe the implementation and the usage of *n*-tuple network.
  - 對於 2048 這個任務而言，若我們要儲存每一個盤面的資訊以用來估計可能的 value 值，顯然是不切實際的，這種做法不僅記憶體無法負荷，且要重複出現的盤面機率也非常低。因此才會設計 n-tuple 的方式，利用選取盤面上的特徵，也就是整個盤面中的多個連在一起的小塊位置(Patterns)，經過鏡射與翻轉後的所有產生的結果(每組 Pattern 有 8 個相對位置)的估計值的總和來衡量一個盤面的價值。而在實作上，假設有一組 Tuple 內剛好是(2^0(empty), 2^1, 2^3, 2^0(empty))，我們將其的次方作為特徵(0, 1, 3, 0)，再將其對應到 weights array 中 index 為 0130 的位置，作為此組特徵可能的答案，而在更新時也是以此組 index 來更新。而對於要估計整體版面的 value 值，我們就將每組 Pattern 所有可能位置(經過鏡射與翻轉每個 Pattern 八個)產生的 index 直接在 weights array 做 table lookup，再將結果總和就是當前盤面的 value 值。
- Explain the mechanism of TD(0).
  - TD Learning 與 Monte Carlo Learning 都是用以預估 Value Function 的方式，但兩者得差別在於 Monte Carlo 的更新是需要等到整個 episode 結束後才更新，而 TD Learning 則是每次每個連續的狀態都能夠更新，因此相比 Monte Carlo 的作法效率更好，但在估計並不如 Monte Carlo 準確。

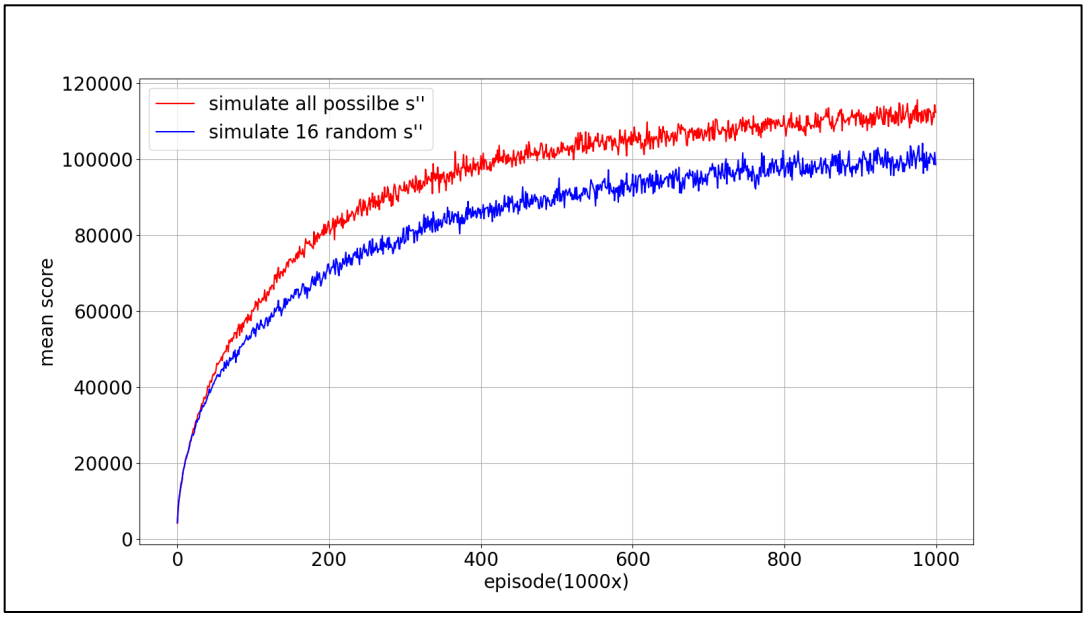
- 而 TD(0)中的 0 代表了只考慮了一步的(即當前這步與下一步)，在其中使用了 Bellman Equation 以更新 Value Function V，即 $V(S) = V(S_t) + \alpha * (R_{t+1} + \gamma * V(S_{t+1}) - V(S_t))$，其中 S_t 代表目前的狀態，而 S_{t+1} 則是下一個狀態，α 為學習率，而 γ 則是 discount factor。

- Describe your implementation in detail including action selection and TD-backup diagram.
  - Lab1 中共有 5 個 TODO 的地方，以下為 5 個需要撰寫的 function 的實作細節
  - estimate
    - 此為計算此盤面所預測的 value 值，我的作法是將所有 pattern 經過轉換(鏡射與旋轉後每個 pattern 共有八個)後利用 indexof 的 function 計算出每個特徵在 weights 的 index 值(也就 weights[index])，所有特徵的總和即是此盤面所預估出的 value 值。
  - update
    - 在 update 中，作法對於各個 pattern 所能產生出的特徵 index，將每個 weights[index]直接加上 td error * step_size(1 / 所有特徵的數量)，就可完成更新，最後再將更新完後新的 value 值回傳。
  - indexof
    - 此為將盤面上 pattern 的位置 mapping 到 weights 中的 index(如上面 n-tuple 的作法)，由於我們預期每個位置可能最多有 $2^0, 2^1, \ldots, 2^{14}, 2^{15}$ 共 16 個可能的值，因此最多需要使用 4 個 bit 來儲存某一格可能的次方數，因此我的作法是先宣告一個 index，對於從 pattern 最後一個到第一個位置，依序地將盤面上 pattern 出現的位置加到 index 中，除了最後一個外每次做完就將結果往左 shift 四格，就能夠依序且不重複地將特徵位置的次方數放上去，最後再回傳計算出的 index 值。
  - select_best_move
    - 由於此次作業需要撰寫的是 before-state 的版本，因此我們需要去模擬對於當前的 state s 中產生的 s"的所有情況，也就是 s 的 after-state s' (經過上下左右)再加上一次 popup(以 9:1 的比例產生一個 2^1 或 2^2 在一個空的位置)。因此我的作法找出所有 empty 的位置，嘗試放其放上 1 或 2(代表 2^1 或 2^2)，並依照其出現機率來計算期望值(0.9 與 0.1)，將所有 empty 經過模擬後結果的期望值就是 s"，

計算方式如圖所示。最終則以往哪個方向移動經過模擬的所預估的 value 最大作為結果回傳。



◆ 另外，我也嘗試比較以上述方法比較在 s' 模擬 16 次 popup()所產生的預估平均，由實驗可知前者結果不管是在運算時間與 mean score 與後者相比皆能獲得更佳的結果，如圖所示(only mean score)。



- ◆ 由於實驗結果在約 400 後 mean score 的提升就相對於前面趨緩，由於不確定是否是演算法可能落入區域最佳，導致無法更有效率的提升 mean score。因此我想嘗試使用ϵ-greedy 來增加 Exploration，但在“Temporal difference learning of n-tuple networks for the game 2048”這篇論文的結論中提到，由於此遊戲隨機產生 2 與 4 已讓此任務有足夠的隨機性，使用ϵ-greedy 上並不會有顯著的提升，因此最終並無實際測試。

- update_episode
  - ◆ 在 update_episode 的地方，由於最後一步之後是沒有任何 reward 的，且每一步的更新都與其後一步的結果有關，因此在實作上，可以透過從最後一個 state 往前到第一個 state 的方式更新，先將一開始的 TD target 設為 0(因為最後一步的 Value 值應該要是 0)，並依序計算 TD error 用來更新我們的 V，再以更新後的 V 來再往計算前一步的 TD target，以此類推直到將所有的結果(path)從後往前更新直到所有 state 皆更新完成。

Reference

[1] 從 2048 學習遊戲對局 AI 設計 https://ko19951231.github.io/2021/01/01/2048/

[2] “Temporal difference learning of n-tuple networks for the game 2048,” in 2014 IEEE Conference on Computational Intelligence and Games (CIG), August 2014, pp. 1–8.