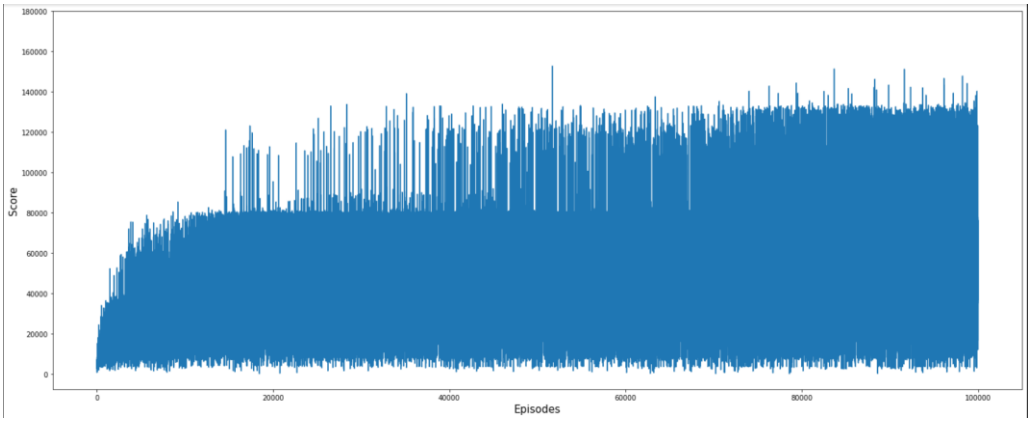
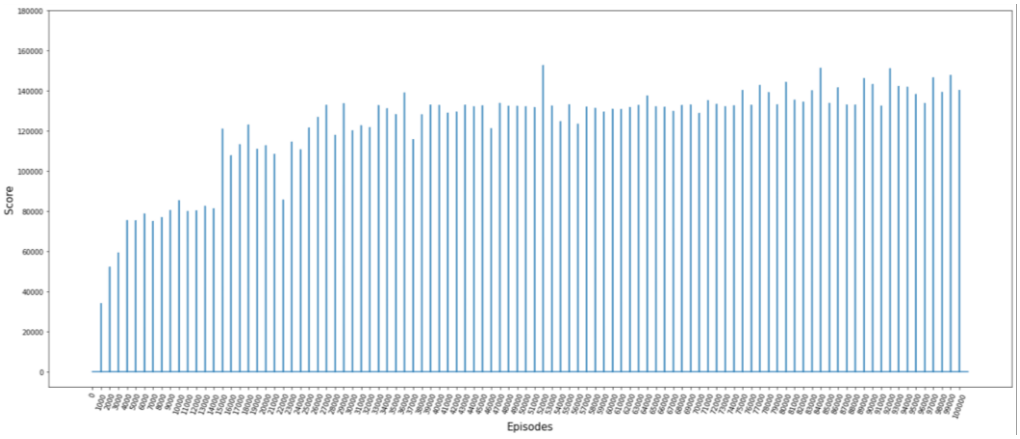
Lab 02 TDL

1. A plot of scores of 100000 training episodes.

   每一場:



   每 1000 場中最高:

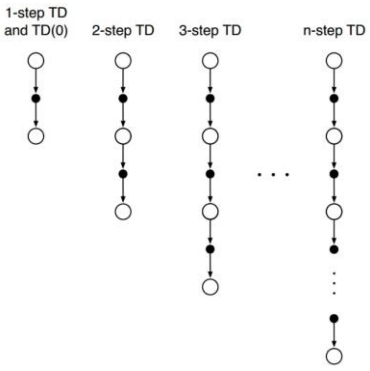


2. Implementation and the usage of n-tuple network.

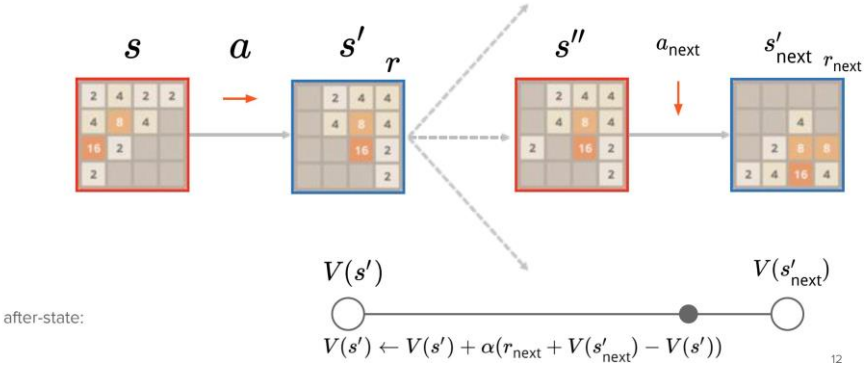
   用來估計盤面的工具，由於估計整個盤面的參數量過多，因此使用n-tuple 來做局部的估計，並由多個n-tuple 來近似整個盤面的評估。程式在執行單一個n-tuple 的估計時，也會旋轉該n-tuple 來做評估。

3. The mechanism of TD(0).

   0 代表只取下一個狀態，即是 1-step TD Learning：



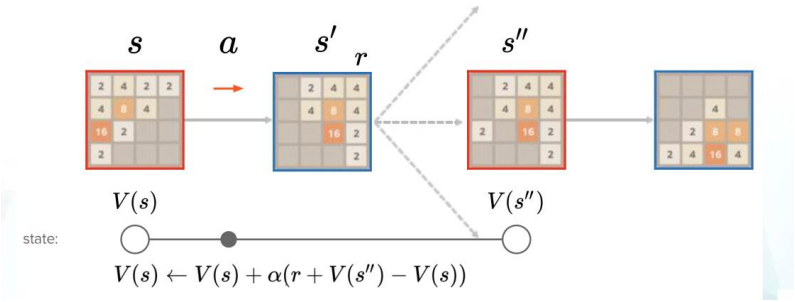
4. TD-backup diagram of V(after-state).



After state 在 2048 的遊戲中指的是在做出移動後，並還沒有跳出新的 2 或 4 的狀態，V(after-state) 就是 TDL 將該狀態做評估並學習。

5. Action selection of V(after-state) in a diagram.
程式在做動作決策時，會將盤面(S')分別上下左右都移動一次(變為 S'next)，並評估其盤面(V(S'next))，再加上移動後的 Reward 做為評估值，並取 4 個評估值中最大者之動作做為決策的結果。

6. TD-backup diagram of V(state).



State 在 2048 的遊戲中指的是在做出移動後並跳出新的 2 或 4 的狀態，V(state) 就是 TDL 將該狀態做評估並學習。

7. Action selection of V(state) in a diagram.
程式在做動作決策時，會將盤面(S)分別上下左右都移動一次並(變為 S')，再列出所有可能的(S'')，並評估其盤面，方式為:

$$r + \Sigma_{s'' \in S''} P(s, a, s'') V(s'')$$

將移動後所有可能出現的 V(S'')乘以發生的機率並相加，並加上(S')的 reward 做為該動作的評估值，並取 4 個評估值中最大者之動作做為決策的結果。

8. My implementation
    1. estimate() , update() , indexof 由於是計算 value 以及更新的方式，演算法的改變並不直接牽涉，因此皆使用 source code 的方法，未做改動。
    2. select_best_move() :

```
722     state select_best_move(const board& b) const {
723         state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
724         state* best = after;
725         float value_sss_zegma = 0;
726         for (state* move = after; move != after + 4; move++) {
727             value_sss_zegma = 0;
728             if (move->assign(b)) { // move up right left down , move is (s')
729                 board test = move->after_state();
730                 board temp = test;
731                 std::vector<int> infro = temp.popup(2);
732 
733                 for(int k= board temp_2 k +=1){
734                     board temp_2 = test;
735                     board temp_4 = test;
736                     temp_2.popup(2,1,k);
737                     temp_4.popup(4,1,k);
738                     value_sss_zegma = value_sss_zegma + 0.1*estimate(temp_4) + 0.9*estimate(temp_2);
739                 }
740                 float factor = (1.0/infro[0]);
741                 value_sss_zegma = factor*value_sss_zegma;
742                 move->set_value(move->reward() + value_sss_zegma);
743                 if (move->value() > best->value())
744                         best = move;
745             } else {
746                 move->set_value(-std::numeric_limits<float>::max());
747             }
748             debug << "test " << *move;
749 
750         }
751         return *best;
752     }
```

首先更改了 popup()，將其變為可控制新值跳出的位置以及是 2 或 4，並且會回傳可跳出的空格數以及該次跳出的位置(亦可選擇按原設計隨機跳出位置與值)。

```
std::vector<int> popup(int v=0, int map=0,int Loc=0)
    int space[16], num = 0;
    int count = 0;
    int loc = 0;
    std::vector<int> local;
    static int arr[2];
    for (int i = 0; i < 16; i++)
        if (at(i) == 0) {
            space[num++] = i;
            count += 1;
        }
    if (num)
        loc = rand() % num;
        if (map !=0){
            loc = Loc;
        }
        if (v == 2){
            set(space[loc], 1);
        }
        if (v == 4){
            set(space[loc], 2);
        }
        if (v == 0){
            set(space[loc], rand() % 10 ? 1 : 2);
        }
    local.push_back(count);
    local.push_back(loc);
    return local;
}
```

在 select_best_move 中，第 729~731 行，用一個 temp 暫存 S’，並做一次 popup 以取得空格數(infro[0])，接下來 733~739 用 for loop 計算所有可能 S’’的 value 並加起來(value_sss_zegma)，並在 738 行除以空格數。總得來

說就是，計算 S'的空格數，每個空格出現機率為(1/ infro[0])，並且每個空格按 9:1 的機率出現 2 和 4，計算所有可能結果。

3. update_episode() :

```
768     void update_episode(std::vector<state>& path_s, std::vector<state>& path_ss, std::vector<state>& path_sss, float alpha = 0.1) const {
769         float v_s = 0;
770         path_s.pop_back();
771         path_sss.pop_back();
772         for (path_ss.pop_back()/* terminal state */; path_ss.size(); path_ss.pop_back()) {
773             state& state_s = path_s.back();
774             state& state_sss = path_sss.back();
775             state& state_ss = path_ss.back();
776             float td_error = state_ss.reward() + (v_s - estimate(state_s.after_state()));
777             debug << "update error = " << td_error << " for state" << std::endl << state_s.after_state();
778             float td_target = state_ss.reward() + estimate(state_sss.after_state());
779             v_s = update(state_s.after_state(), alpha * td_error);
780             path_s.pop_back();
781             path_sss.pop_back();
782         }
783     }
```

此處改為需要三個 vector 輸入，path_s：儲存 S ，path_ss：儲存 S' ，path_sss：儲存 S'' ，用 loop 計算 TD error (776 行) :

$$(r + V(s'') - V(s))$$

以及 TD target (788 行) :

$$r + V(s'')$$

779 行:更新 V(s)。

(此處沿用 source code 的 state 變數型態，因此 state.after_state()並非指 after state，而是做為取出盤面之用，如 state_s.after_state() = S 盤面)

4. main() :

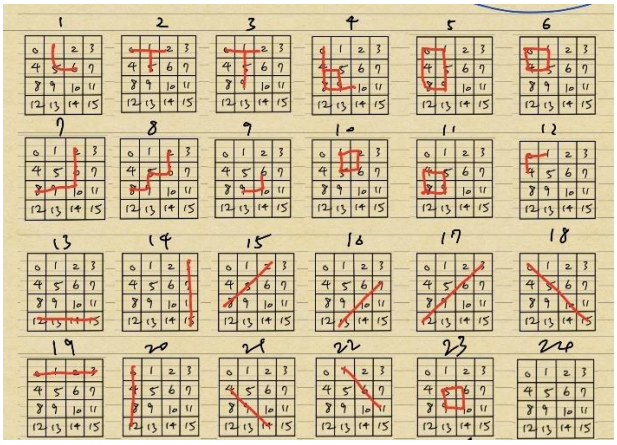
```
940         std::vector<state> path_s;
941         std::vector<state> path_ss;
942         std::vector<state> path_sss;
943         path_s.reserve(20000);
944         path_ss.reserve(20000);
945         path_sss.reserve(20000);
```

用三個 vector 儲存 S、S'、S''。

```
while (true) {
    debug << "state" << std::endl << b;
    state best = tdl.select_best_move(b);
    // sleep(1);
    if (best.is_valid()) {
        debug << "best " << best;
        path_s.push_back(b);
        score += best.reward();
        file.close();
        b = best.after_state();
        path_ss.push_back(best);
        b.popup(0);
        path_sss.push_back(b);
    } else {
        break;
    }
}
```

5. n-tuple:

   這裡按照 2048 遊戲的高分策略設計 feature，實驗方式是逐個增加 tuple(有時也會一次增加 2~3 個)，如勝率提升，便保留該 tuple，否則捨棄。設計的主要概念如下：



(tuple 設計圖，編號代表採用的順序)

1. 選擇一個角落，密集布置 feature (我選擇左上角 9 宮格)。
2. 由於將盤面的對角格湊成同樣的數有助於往後的合併(如對角線都是 8)，因此便設計了對角線分布的 feature。
3. 經逐次新增 tuple 並訓練後發現，增多 n = 6 或以上的 tuple 的數量帶來的勝率提升遠不如增加 n = 3 或 4 的 tuple，因此總共設計了 23 個 tuple :

   7 個 3-tuple、11 個 4-tuple，3 個 5-tuple，2 個 6-tuple。

9. Other discussions or improvements.

```
Playing...100000 run100000      mean = 64324.3  max = 148848
        128     100%    (0.1%)
        256     99.9%   (0.4%)
        512     99.5%   (1.8%)
        1024    97.7%   (4%)
        2048    93.7%   (17.7%)
        4096    76%     (67.2%)
        8192    8.8%    (8.8%)
```

```
Playing...1000 run1000  mean = 11077.7  max = 34880
        64      100%    (0.1%)
        128     99.9%   (1.8%)
        256     98.1%   (11.6%)
        512     86.5%   (41.5%)
        1024    45%     (40.6%)
        2048    4.4%    (4.4%)
```

訓練結果，2048 勝率可達 9 成以上，4096 可達近 8 成，且 2048 tile 在訓練的前 1000 場內便出現，但在約 50000 episode 左右便已收斂，並考慮程式會旋轉映射 feature，有些 feature 似乎是多餘的，可嘗試移除，增加訓練速度(目前訓練 100000 次需一天)。