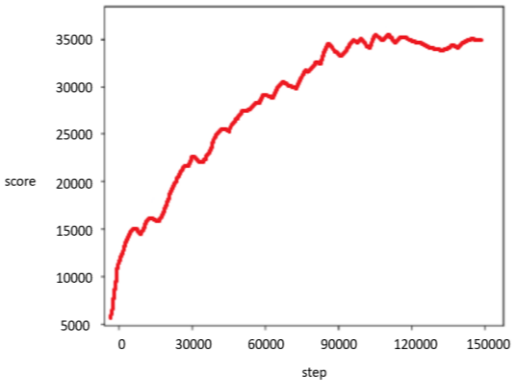
RL – Lab1

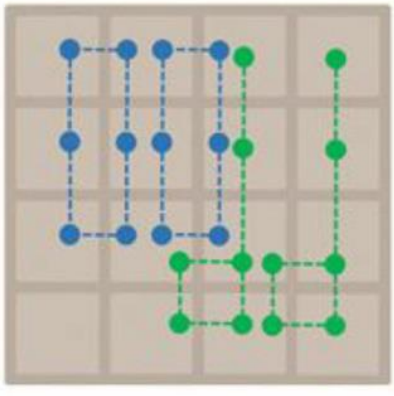
姓名: 蔡沅恆
學號: 311551136
日期: 2024/1/5

result:
在大約十萬步左右就達到收斂，並且約有 80%勝率，再練下去分數並沒有更高甚至有時候還會更低

```
TDL2048-Demo
alpha = 0.2
total = 150000
seed = 1785122674
6-tuple pattern 012345, size = 16777216 (64MB)
6-tuple pattern 456789, size = 16777216 (64MB)
6-tuple pattern 012456, size = 16777216 (64MB)
6-tuple pattern 45689a, size = 16777216 (64MB)
6-tuple pattern 012345 is loaded from weight.bin
6-tuple pattern 456789 is loaded from weight.bin
6-tuple pattern 012456 is loaded from weight.bin
6-tuple pattern 45689a is loaded from weight.bin
1000    mean = 29353    max = 50624
        128     100%    (0.1%)
        256     99.9%   (0.1%)
        512     99.8%   (1.7%)
        1024    98.1%   (15.7%)
        2048    82.4%   (80.8%)
        4096    1.6%    (1.6%)
```



1. Describe the implementation and the usage of n-tuple network
在 2048 中 state 非常大，如果用一個表去記錄每一個 state(棋盤狀態)的 value 非常不實際，因此使用 n-tuple 方是在保有部分 feature 同時也能夠有效縮減 state 大小。
而 n-tuple 紀錄方式是將棋盤每隔以 0~15 編號，假如其中一個 feature 是[0,2,5]則 0、2 和 5 對應的數字為 2、16、2 則此 feature 所得值是 141。同時由於 2048 棋盤分別可以找到 8 種 isomorphism(轉九十度四次、鏡像轉九十度四次)，因此還可以再額外透找到另外七組成 3*8*4bits 長的 state。此外每個 state 的估算值也由所有 feature 對應數字的值總和作為估計值。
而本次做也我採用的 feature 為:



2. Explain the mechanism of TD(0)

相比於 MC method 其必須完整收集軌跡，TD(0)每與環境互動一次就可以更新，如此可以讓 TD(0)用於沒有終結狀態的環境，並且其相比 MC 有較低的 variance(但同時會有 bias)。其更新方式是: TD target=a*r+V(S')，TD-error = TD target- V(S) ，V(S)<-V(S)+ TD-error

3. Describe your implementation in detail including action selection and TD

```
state select_best_move(const board& b) const {
td::cout<<"[select the action]:"<<std::endl;
    state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
    state* best = after;
    for (state* move = after; move != after + 4; move++) {
        if (move->assign(b)) {
            // code
            float total{0};
            size_t emptyNumber{0};
            for(size_t i{0};i<16;i++){
                if(move->after_state().at(i)==0){  //find the empty tile in the board
                    emptyNumber++;
                    //set to 2
                    board boardAdd2 = move->after_state();
                    boardAdd2.set(i,1); //set 2 in the position i
                    //set to 4
                    board boardAdd4 = move->after_state();
                    boardAdd4.set(i,2); //set 2 in the position i

                    total += estimate(boardAdd2)*0.9 + estimate(boardAdd4)*0.1;  //the probabil
                }
            }
td::cout<<"action-"<<move->action()<<"action reward: "<<move->reward()<<"  value:"<<move->reward
            move->set_value( move->reward() + total/emptyNumber );  //do not need to check if t
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
```

Action selection:

TD-learning 在選擇 action 時會檢查每個 action 所對應的 estimated value，並且選擇擁有最大預估值者。而在 2048 中執行一個 action 可以分為兩個步驟，第一步是 action 執行後棋盤往上/下/左/右靠；接著才是由環境依據 0.9 及 0.1 的機率在棋盤上隨機位置出現 2 或 4。而每個 action 的預估值是要預估在"第二步驟完成後 state 的期望值"。

由於我們只能夠知道第一重後棋盤的狀況，到第二步驟是由環境依據機率分布所決定。

1. 因此首先根據四個動作模擬出第一部完成後的棋盤狀況(s')。

```
        if (move->assign(b)) {
```

2. 接著檢查 s'上面是否有空格，如果有則分別計算如果是出現二或四的值

```
            for(size_t i{0};i<16;i++){
                if(move->after_state().at(i)==0){  //find the empty tile in the board
                    emptyNumber++;
                    //set to 2
                    board boardAdd2 = move->after_state();
                    boardAdd2.set(i,1); //set 2 in the position i
                    //set to 4
                    board boardAdd4 = move->after_state();
                    boardAdd4.set(i,2); //set 2 in the position i
```

3. 並依據 0.9 及 0.1 的機率相乘並再除以全部空格數算出此格的期望值貢獻度，並將所有空格算出期望值貢獻度相加作為 s''的期望值

```
                    total += estimate(boardAdd2)*0.9 + estimate(boardAdd4)*0.1;  //the probabil
                }
            }
td::cout<<"action-"<<move->action()<<"action reward: "<<move->reward()<<"  value:"<<move->reward
            move->set_value( move->reward() + total/emptyNumber );  //do not need to check if t
```

4. 選擇具有最高期望值的 action 作為此 state 的 action

```
            if (move->value() > best->value())
                best = move;
```

TD-backup:

TD-error = alpha*reward + ESTIMATE_NEXT - STIMATE_NOW

而 TD-error 是目標所要更新的量

```
 */
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    update( path[path.size()-1].before_state(), -40000);
    for( size_t i{path.size()-2} ; i<path.size() ; i-- ){
        float TDerr = alpha*( path[i].reward() + estimate(path[i+1].before_state()) - estimate(path[i].before_state()) );
        update( path[i].before_state(), TDerr );
    }
```

由於此次作業 estimate value 是尤其所有 value 的和加總而成，因此 TD-error 必須在平均分成 feats.size 份，而每個 feat 再將此值加到 weight(board)上面完成一次 TD-backup

```
 */
virtual float update(const board& b, float u){
    // TODO
    float est{0.0};
    float u_split = u/iso_last;
    for( int i{0} ; i < iso_last ; i++){   //each isomorphic
        size_t index{ indexof(isomorphic[i], b) };
        weight[index] +=  u_split;
        est = est + weight[index];
    }
    return est;
```