Lab2: Temporal Difference Learning

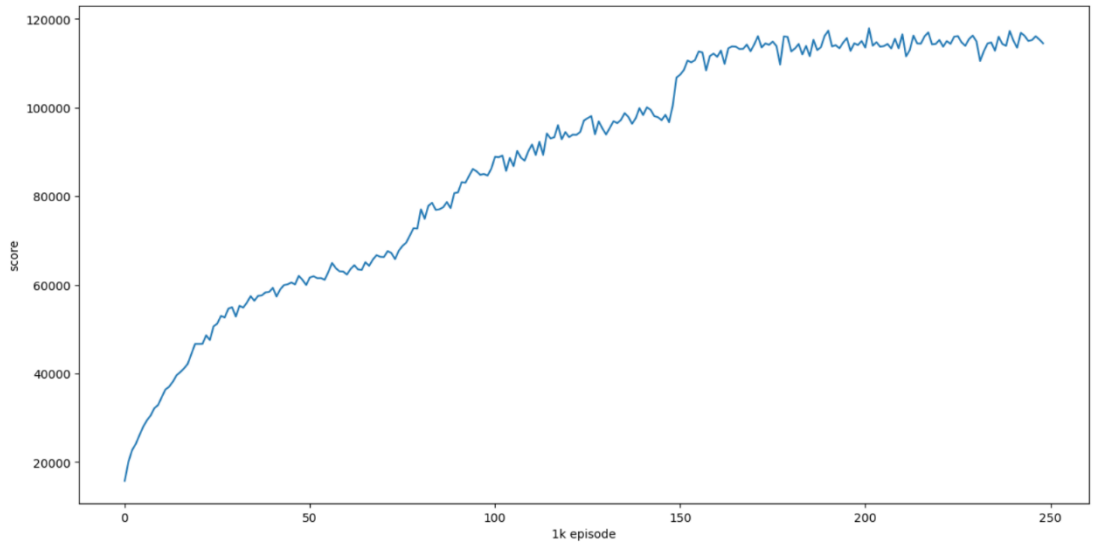
311553030 林亮丞

0. Introduction

在此次作業中，我們會學習到如何使用 TD(0)和 n-tuple network 去解 2048

1. A plot shows scores (mean) of at least 100k training episodes (10%)

- Alpha: 0.1, epochs: 1~150K
- Alpha: 0.01, epochs: 150k+1~250K
- 總共 250K 場遊戲



2. Describe the implementation and the usage of n-tuple network. (10%)

由於 2048 遊戲中總共只有 4*4 的格子，每個格子只有 [0, 2^1, 2^2, ..., 2^15] 等共 16 種數值，因此總共會有 16^16 種不同的盤面。記錄所有盤面的估計值將對記憶體構成巨大負擔，且在訓練過程中大多只會遇到沒有見過的盤面，這會使得模型難以訓練。

然而，在 2048 遊戲中，只需取出一小塊位置即可成為有用的特徵。因此，我們可以選擇一小塊位置作為盤面的「feature」，並且在計算估計值時僅針對該特定 feature 進行操作。此外由於可以透過水平、垂直和對角線的移動變換，將同樣的遊戲盤面轉換成不同的狀態，而這些狀態實際上具有相同的估計值。因此 N-tuple 在四個旋轉方向 x 兩種鏡像總共八種 isomorphism 中各取一次估計值，並全部相加來當作此盤面的總估計值，來避免等效狀態的估計值不同。

我在此作業中選擇了 6 個 6-tuple network 和 4 個 4-tuple network，Fig 2 為為此部分的程式截圖。

```
// initialize the features of the 4x6-tuple network
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 5 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 9 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));
tdl.add_feature(new pattern({ 4, 5, 6, 8, 9, 10 }));

tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 7 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 11 }));

tdl.add_feature(new pattern({ 0, 1, 2, 3 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7 }));
tdl.add_feature(new pattern({ 0, 1, 4, 5 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6 }));
```

Fig2: Design of 6-tuple and 4-tuple network

3. Explain the mechanism of TD(0). (10%)

在 2048 遊戲中，我們可以選擇上下左右四個方向來移動磚塊。在 TDL(0)算法中，我們可以通過比較當前狀態的值函數和下一個狀態的估計值函數的差異來計算誤差，並且利用誤差和尋找最大的移動獎勵去更新當前估計值。Fig3 為示意圖以及公式，S 是 action 的 before state，S'是 action 的 after state，S"是新 popup 好一個數字的盤面。

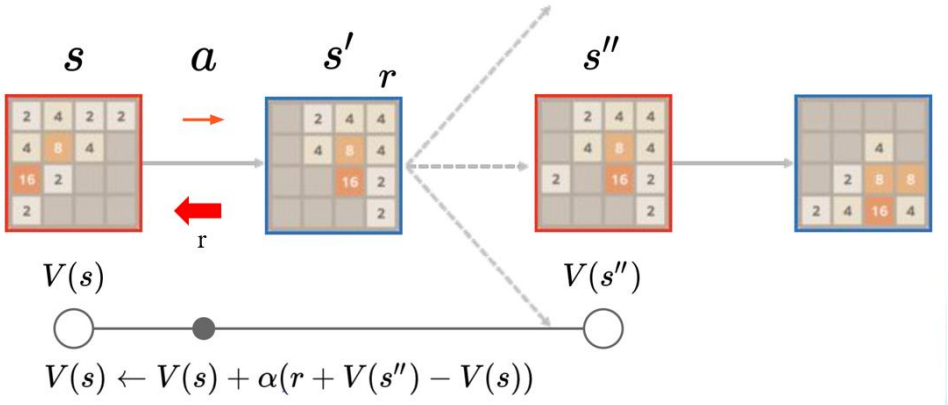


Fig3: mechanism of TD(0)

4. Describe your implementation in detail including action selection and TD-backup diagram. (20%)

4.1. Action selection:

在選擇從狀態 S 到狀態 S'的動作(上、下、左、右)時，我們需要考慮到每個動作所帶來的期望值，因此需要嘗試每一個可能的動作。這個期望值不僅需要考慮到當前動作可以帶來的即時獎勵，還需要考慮到在執行動作後，加入數字(90% 2,

10% 4)生成的新狀態 S"，因此我們需要納入所有可能的 S"的估計值來進行評估，最後期望值公式會如同 Fig4。

function EVALUATE(s, a)

$s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)

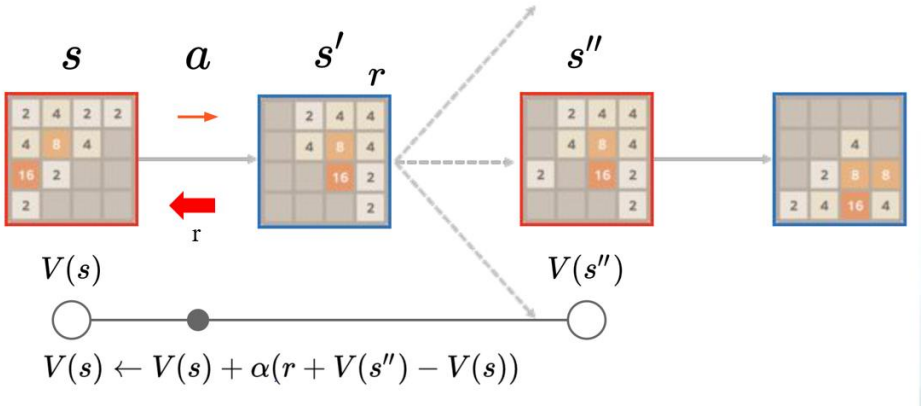
$S'' \leftarrow$ ALL POSSIBLE NEXT STATES(s')

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

Fig4: Expection Function

4.2. TD-backup diagram:

如同 Fig3，只是因為同時使用了 n-tuple network，所以這裡用來取得估計值的 V function 改成是對所有的 Feature 中 8 種 isomorphism 去取估計值，這些的總和才是此盤面的估計值。



5. All implement code

5.1. Value Function

```
virtual float estimate(const board& b) const {
    // TODO
    float value = 0;
    for(int i=0; i<iso_last; i++){
        size_t index = indexof(isomorphic[i], b);
        value += operator[](index);
    }
    return value;
}
```

總和 Feature 中 8 種 isomorphism 取得估計值並回傳。

5.2. Update Function

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

將要更新的值分散給各個 isomorphism 去更新 weight。

5.3. Index of pattern

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
    // TODO
    size_t index = 0;
    for (size_t i = 0; i < patt.size(); i++)
        index |= b.at(patt[i]) << (4 * i);
    return index;
}
```

棋盤中會使用 4 bit 去儲存每格的次方數，n-tuple 中會有 n 組 4 bit 的數字，將這些數字以 little edian 方式儲存並回傳。

5.4. Select best move

```
state select_best_move(const board& b) const {
    state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
    state* best = after;
    for (state* move = after; move != after + 4; move++) {
        // 找出最大價值的action
        if (move->assign(b)) {
            // TODO
            board after_move_state = move->after_state(); // S' is after_move_state
            float next_state_esti = 0;
            int num = 0;
            // get all possible estimate(S")
            for(int i = 0; i < 16; i++){
                if(after_move_state.at(i)==0){
                    // 90% popup 2
                    after_move_state.set(i,1);
                    next_state_esti += 0.9*estimate(after_move_state);
                    // 10% popup 4
                    after_move_state.set(i,2);
                    next_state_esti += 0.1*estimate(after_move_state);
                    after_move_state.set(i,0);
                    num++;
                }
            }
            next_state_esti /= num;
            move->set_value(move->reward() + next_state_esti);
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }

    return *best;
}
```

嘗試每一個可能的動作，計算這些動作的即時獎勵以及下一個 state S”的所有可能的估計值作為期望值，找到最大期望值的動作就是 best action。

5.5. Update epside

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    state& terminal_state = path.back(); // terminal_state.before_state() is S"
    float target_value = 0; // V(S")
    for (path.pop_back() ; path.size(); path.pop_back()) {
        state& curr_state = path.back();
        // error = r + V(S") - V(S)
        float error = curr_state.reward() + target_value - estimate(curr_state.before_state());
        // V(S) = V(S) + alpha * error
        target_value = update(curr_state.before_state(), alpha * error);
    }
}
```

通過比較當前狀態(S)和下一個狀態(S”)的估計值的差異來計算誤差，並且利用誤

差和尋找最大的移動獎勵去更新當前估計值。