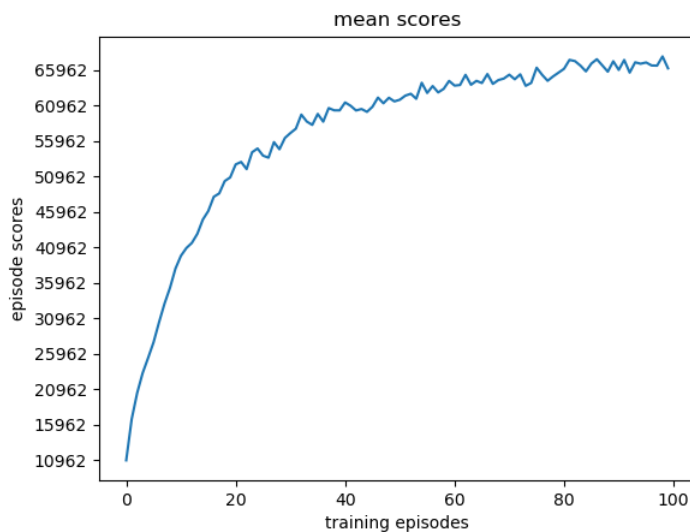
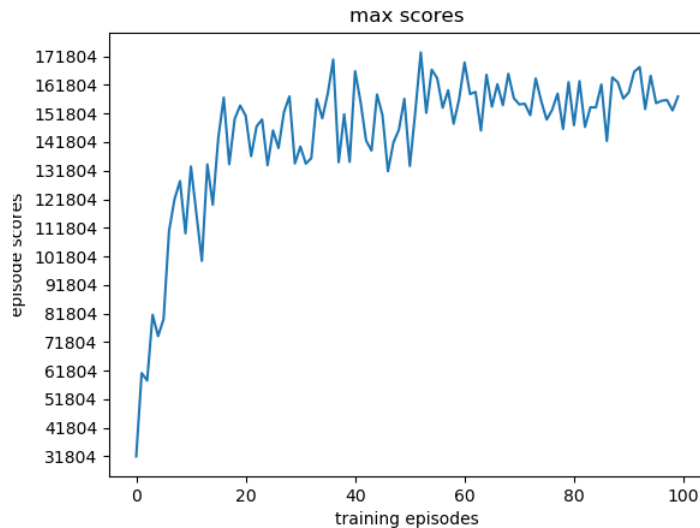


# Lab2 Temporal Difference Learning

311605014 鄧奕辰

1. A plot shows scores of at least 100k training episodes



上圖可以看到，在 max scores 中，震盪得比較大，但也是有持續上升，且 mean scores 中更明顯的看到平均下來是持續上升的。

2. Describe the implementation and the usage of  $n$ -tuple network

在 2048 遊戲中，所有可能的狀態數量很。每個方塊的值可以是  $2^1$  次方到  $2^{16}$  次方，總共有 16 個可能的值，因此總共可能有  $16^{16}$  次方個狀態。這個數量太大，無法在內存中保存。因此在這個實驗中，我們使用  $n$ -tuple network 來節省內存。在這個實驗的  $n$ -tuple network 中，它有一個稱為“權

重”的數組。一旦我們想要獲取一個 n-tuple 特徵的值，我們就會將該特徵轉換為整數索引，然後使用這個索引來查找它的權重值。

```
// initialize the features
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 }));
// 4-tuple
tdl.add_feature(new pattern({ 0, 1, 2, 3 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7 }));
tdl.add_feature(new pattern({ 8, 9, 10, 11 }));
tdl.add_feature(new pattern({ 12, 13, 14, 15 }));
tdl.add_feature(new pattern({ 0, 1, 4, 5 }));
tdl.add_feature(new pattern({ 2, 3, 6, 7 }));
tdl.add_feature(new pattern({ 8, 9, 12, 13 }));
tdl.add_feature(new pattern({ 10, 11, 14, 15 }));
tdl.add_feature(new pattern({ 5, 6, 9, 10 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6 }));
tdl.add_feature(new pattern({ 4, 5, 8, 9 }));
tdl.add_feature(new pattern({ 9, 10, 13, 14 }));
tdl.add_feature(new pattern({ 6, 7, 10, 11 }));
```

### 3. Explain the mechanism of TD(0)

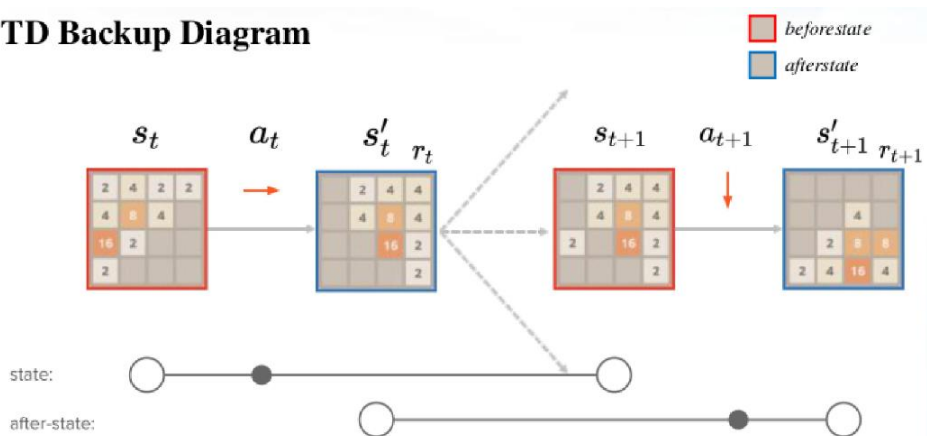
TD(0)是一種基於觀察到的獎勵和下一個狀態值來估計策略價值函數的強化學習算法，TD(0)估算當前狀態  $s$  的值函數  $V(s)$  通過使用 Bellman equation:  

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

其中  $\alpha$  是學習率， $\gamma$  是折扣因子， $r$  是即時獎勵， $V(s')$  是下一個狀態  $s'$  的估計值。TD(0)是一種無模型算法，不需要環境模型，並且是一種在每轉換之後更新價值函數估計的在線學習算法。它具有較低的計算成本，並且在某些條件下可以收斂到最優策略。

### 4. Describe your implementation in detail including action selection and TD-backup diagram

#### TD Backup Diagram



該圖表顯示，在訓練期間，它會更新後狀態的值。使用的 TD 目標等於下一個狀態的動作的獎勵加上下一個後狀態的估計值。

$V(\text{after-state})$ : after-state 是考慮藍色的 board，也就是執完 action 後產生的 board

$V(\text{state})$ : state 是考慮紅色的 board，也就是 popup 之後的 board

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

在 estimate() 函式中，由於 2048 的棋盤可以進行旋轉和反射，因此回傳值將是這些特徵值的總和。使用 indexof() 函式將該棋盤上的特徵轉換為整數索引，然後取得該索引的權重值，加總至最終回傳值中。

```
virtual float update(const board& b, float u) {
    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;
}
```

在 update() 函式中，由於估計值來自於不同的特徵，我們需要平均更新值到這些特徵上。使用 indexof() 函式將該棋盤上的特徵轉換為整數索引，然後將這些值更新到權重中。

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

```

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++) {
        if (move->assign(b)) {
            // TODO
            board next = move->after_state();
            int count = 0;
            double sigma_P_V = 0;
            for (int i = 0; i < 16; i++){
                if (next.at(i) == 0) {
                    count ++ ;
                    next.set(i,2);
                    float V1 = estimate(next);
                    next.set(i,0);
                    next.set(i,1);
                    float V9 = estimate(next);
                    next.set(i,0);

                    sigma_P_V += (V1*0.1 + V9*0.9);
                }
            }
            if(count!=0){
                move->set_value(move->reward() + sigma_P_V/count);
            }
            if(count==0){
                move->set_value(move->reward() + 0);
            }
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}

```

在 TD 的過程中，我們需要計算“ $r + \sum_{s'' \in S} P(s, a, s'') V(s'')$ ”，且  $V(s'')$  是 popup 之後的 state  $s'$ ，2 的 popup 機率為 90%，4 的 popup 機率為 10%，因此  $\sum_{s'' \in S} P(s, a, s'') V(s'')$  可被表示成  $0.9 * (V(s') + 2) + 0.1 * (V(s') + 4)$

```

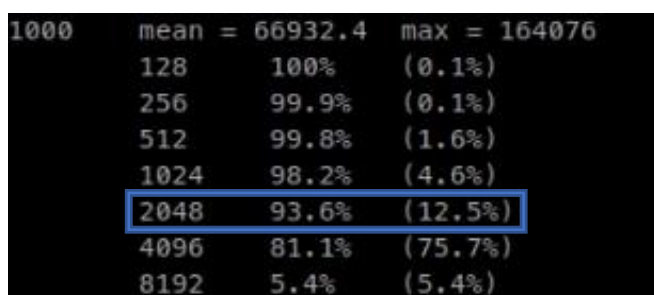
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float exact = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = move.reward() + exact - estimate(move.before_state());
        debug << "update error = " << error << " for after state" << std::endl << move.after_state();
        exact = update(move.before_state(), alpha * error);
    }
}

```

在 `update_episode()` 函數中，我們需要從路徑的最後一個狀態更新到第一個狀態。根據 TD 學習的公式，即  $V(s) \leftarrow V(s) + \alpha(r + V(s') - V(s))$  我們將在每個時間步驟更新前狀態的值  $V(s)$ 。公式中的  $r$  是當前狀態的獎勵， $V(s')$  是下一個前狀態的估計值， $V(s)$  是當前前狀態的估計值。我們將使用這些信息來計算 TD 誤差並更新前狀態的值。

## 5. Other discussions or improvements.

在訓練的過程中，僅使用了 100000 次的 epoch，且最終的成果為下圖，



1000	mean = 66932.4	max = 164076
128	100%	(0.1%)
256	99.9%	(0.1%)
512	99.8%	(1.6%)
1024	98.2%	(4.6%)
2048	93.6%	(12.5%)
4096	81.1%	(75.7%)
8192	5.4%	(5.4%)

2048 的成功率為 93.6%，沒有更好成果的原因推斷如下：

feature 在設計時，可以增加 3-tuple 的形式，或增加多一點斜向的 feature。

將 epoch 數量增加更多，因為在訓練的過程中可以發現，整體的成功率還有上升的趨勢，因此若能訓練久一點，效果會更好。

Learning rate 可以在後面的 epoch 中，降低一點，好幫助模型收斂。