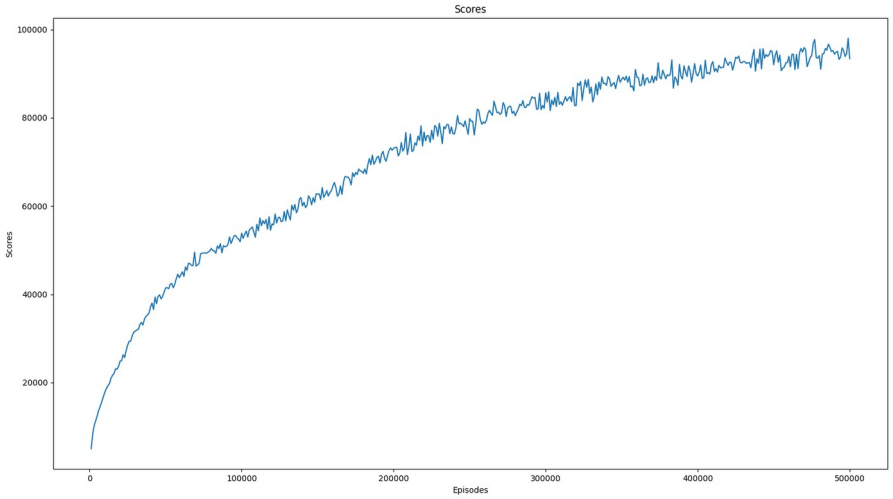
Lab 2 Report
309551064 張凱翔

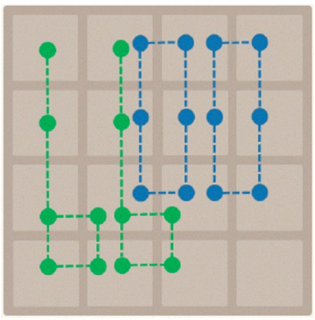
■ A plot shows episode scores of at least 100,000 training episodes



■ Describe the implementation and the usage of n-tuple network

2048 總共有 16 個格子，每一個格子可能的數字為 17 種，因此總共有 $17^{16} \sim= 10^{20}$ 種可能的狀態，需要的記憶體空間也會很大，因此改以 N-tuple 的形式來計算分數。以 4 個 6-tuple(此作業用法)為例，記憶體空間只須 $4 * 16^6 = 2^{26} \sim= 256MB$，大大降低所需的記憶體空間。

每一種 tuple 都會 rotate 和 reflection 來產生 8 種 Isomorphism，這 8 種共用同一個 weight table。將盤面上的數字轉換為 2 進位的次方，依照 tuple 的順序將這些數字編成 index 從 weight table 找值，再將 4 種 tuple 的值加起來便是這個狀態的分數。使用的 6-tuple 是由 **Multistage Temporal Difference Learning for 2048-Like Games** 這篇 paper 提出，tuple 的取法如下圖。



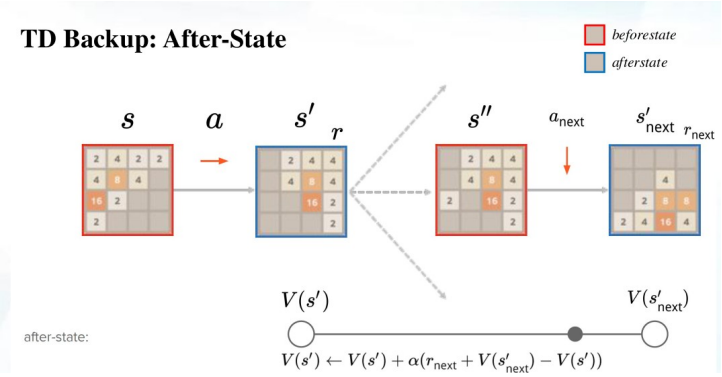
■ Explain the mechanism of TD(0)

TD(0)的公式如右：$V(S_t) = V(S_t) + \alpha(r_{t+1} + V(S_{t+1}) - V(S_t))$

理想上 $r_{t+1} + V(S_{t+1})$ 的值應該要等同於 $V(S_t)$，而他們之間的差值可以視為 predict error，TD(0)想要做的就是調整 V 對於 S_{t+1} 和 S_t 的估計值。

■ Explain the TD-backup diagram of V(after-state)

after-state 是找下一個時間點的 reward 加上 afterstate 的值，與前一個時間點的 afterstate 的值計算差值，乘上 learning rate 後更新 value table 的值。



■ Explain the action selection of V(after-state) in a diagram

在選擇 action 時，會去計算每一種 action 所帶來的 reward 和這個 action 所帶來盤面結果的分數，將兩種加起來後，選擇能獲得最高分數的 action。

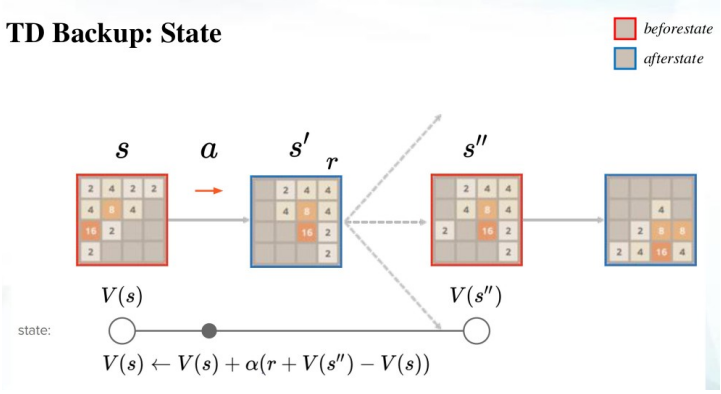
function EVALUATE(s, a)

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

  return $r + V(s')$

■ Explain the TD-backup diagram of V(state)

計算經過 action 並 popup 後的盤面和一開始盤面的差值，乘上 learning rate 後更新 value table 的值。



■ Explain the action selection of V(state) in a diagram

在選擇 action 時，每一種 action 造成的狀態都可以再衍生出很多種，這是因為在做完一個 action 後，遊戲會在盤面空的位置隨機加上 1 或者 2 的數字。V(state)就會去計算每一種衍生盤面的期望分數再加上 action 所帶來的 reward 當作所作 action 的分數，然後選出分數最高的 action。

function EVALUATE(s, a)

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

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

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

■ Describe your implementation in detail

- 這個 function 負責去計算一個特定 pattern 和其他 isomorphic 在當前盤面獲得的分數，可以更改 iso_last 來改變 isomorphic 數量(允許 rotate or reflection)。
  這些分數都是透過查詢共同 weight table 得到的。

```
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;
}
```

- 在 TD-backup 時要更新 4 個 feature 的 weight table。u 是已經乘上 learning rate 的 td-error，在將其平均給 isomorphic 的數量。找到 weight table 的值後，對其更新並回傳更新後特定 pattern 和其他 isomorphic 在當前盤面獲得的分數。

```
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;
}
```

- 要從 table 找值首先要知道該從 table 的哪個 index 取值，這個 function 就是在找這個 index。 在做的事情是將盤面上的數字轉換為，依照 tuple 的順序將這些數字編成 index，並回傳。

```
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;
}
```

- 總共有 4 種 action(**after[4]**)，會針對每一種 action 去判斷。
  以 **if(move->assign(b))**去判斷 action 的可行性
  可行：
  去模擬在經過這個 action 後再 popup 的所有盤面的可能性。
  首先會判斷經過這個 action 後盤面的格子是否為空，如果為空的話就會有 0.9 的積率生成 1、0.1 的機率生成 2，計算兩種可能盤面的分數乘上機率。
  最後加總除以總空格子的數量並加上這個 action 的 reward 去跟最佳(**best**)的 value 做比較，如果較高便將最佳替換成當前的 action。
  不可行：
  賦予 value 一個很大的值
  最後回傳最佳 action 的 before_state、 after state、 reward 和 value

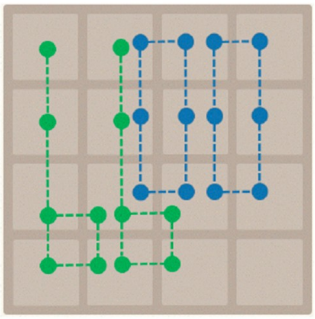
```
state select_best_move(const board& b) const {
    state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
    state* best = after;
    for (state* move = after; move != after + 4; move++) {
        if (move->assign(b)) {
            // TODO
            int empty_block = 0;
            float value = 0;
            for(int i = 0; i < 16; i++) {
                if (move->after_state().at(i) == 0) {
                    board tmp(move->after_state());
                    tmp.set(i, 1);
                    value += 0.9 * estimate(tmp);
                    tmp.set(i, 2);
                    value += 0.1 * estimate(tmp);
                    empty_block++;
                }
            }

            value /= empty_block;
            move->set_value(move->reward() + value);
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

- 從 path 的最後往前更新，用 exact 來紀錄上一次更新後狀態的 value。
  依據公式去計算 error 並乘上 learning rate 去更新 weight table。

```
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());
        exact = move.reward() + update(move.before_state(), alpha * error);
    }
}
```

■ Other discussions or improvements

1. 將 tuple 的取法變為下圖的樣子。



在固定 random seed 的情況下，相較於 sample code 上的取法，training 的前 10000 的 episode 都能取得較好的結果。

Before:

```
10000    mean = 16815.5   max = 40440
         128     100%     (0.1%)
         256     99.9%    (3.5%)
         512     96.4%    (17.6%)
         1024    78.8%    (60%)
         2048    18.8%    (18.8%)
```

After:

```
10000    mean = 17334.7   max = 43984
         128     100%     (0.4%)
         256     99.6%    (2.7%)
         512     96.9%    (17.6%)
         1024    79.3%    (58.8%)
         2048    20.5%    (20.5%)
```

2. 將所能獲得的 reward 限制在 4096 以下，意思就是不讓兩個 2048 合起來，我認為像是在設定 2048 這個學習目標，能更好的學習如何做出 2048。在沒有這個設定之前都在 91-93%之間震盪，加上之後能突破 95%。

```
int move_left() {
    uint64_t move = 0;
    uint64_t prev = raw;
    int score = 0;
    lookup::find(fetch(0)).move_left(move, score, 0);
    lookup::find(fetch(1)).move_left(move, score, 1);
    lookup::find(fetch(2)).move_left(move, score, 2);
    lookup::find(fetch(3)).move_left(move, score, 3);
    raw = move;
    return (move != prev && score < 4096) ? score : -1;
}
int move_right() {
    uint64_t move = 0;
    uint64_t prev = raw;
    int score = 0;
    lookup::find(fetch(0)).move_right(move, score, 0);
    lookup::find(fetch(1)).move_right(move, score, 1);
    lookup::find(fetch(2)).move_right(move, score, 2);
    lookup::find(fetch(3)).move_right(move, score, 3);
    raw = move;
    return (move != prev && score < 4096) ? score : -1;
}
int move_up() {
    rotate_right();
    int score = move_right();
    rotate_left();
    return (score < 4096) ? score : -1;
}
int move_down() {
    rotate_right();
    int score = move_left();
    rotate_left();
    return (score < 4096) ? score : -1;
}
```

```
4000     mean = 51824.8   max = 71392
         128     100%     (0.2%)
         256     99.8%    (0.1%)
         512     99.7%    (0.1%)
         1024    99.6%    (4.1%)
         2048    95.5%    (95.5%)
```