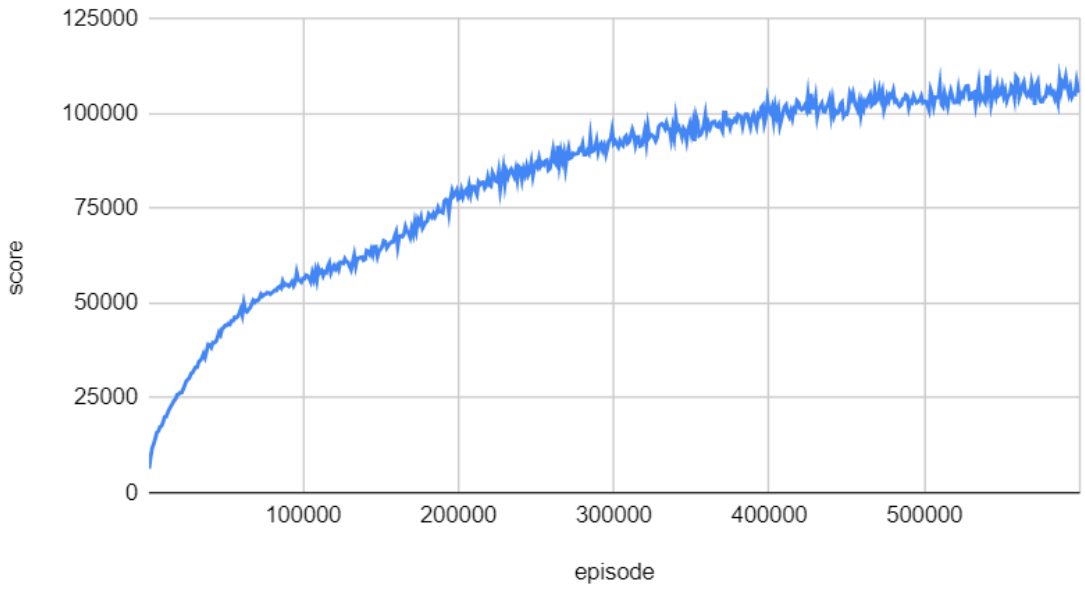
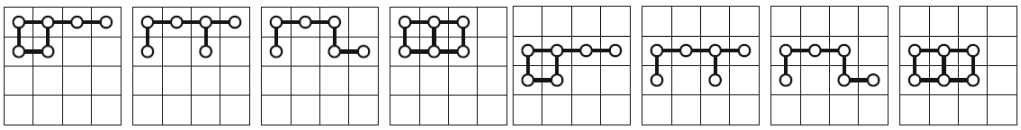
1. A plot shows episode scores of at least 100,000 training episodes



2. Describe the implementation and the usage of n-tuple network.
參考 *Oka, Kazuto, and Kiminori Matsuzaki . "Systematic selection of n tuple networks for 2048." International Conference on Computers and Games. Springer International Publishing, 2016.*這篇論文的作法，選出他測量出排名前幾的 tuple 圖案，加上圖案往下平移共 8 種 tuple。



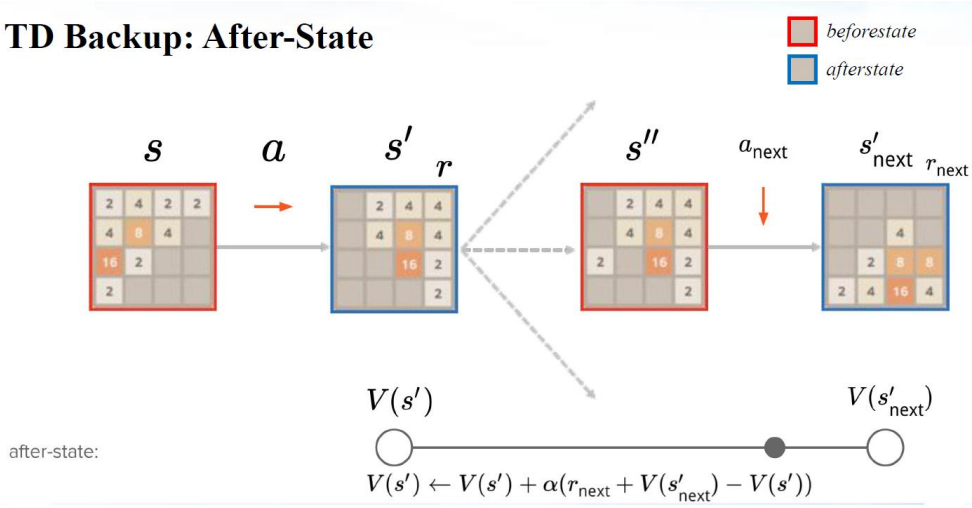
```
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 5 }));
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 6 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 6, 7 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));

tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 9 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 10 }));
tdl.add_feature(new pattern({ 4, 5, 6, 8, 10, 11 }));
tdl.add_feature(new pattern({ 4, 5, 6, 8, 9, 10 }));
```

3. Explain the mechanism of TD(0).
先對目前的 state (S)進行一個 action，針對該 action 得到一個 reward (r)和下一個 state (S')。先計算下一個 state 的 value 加上該次得到的 reward 再和 S 的 value 計算誤差，使用誤差乘上 learning rate 更新對 value 的評估方法。在 2048 中，因為在移動後會 popup (隨機在一個空位生成 2 或 4 的

tile)，所以在計算 state 會分成 before-state (還沒做 action 時的版面) 和 after-state (做 action 但還沒 popup 時的版面) 討論。

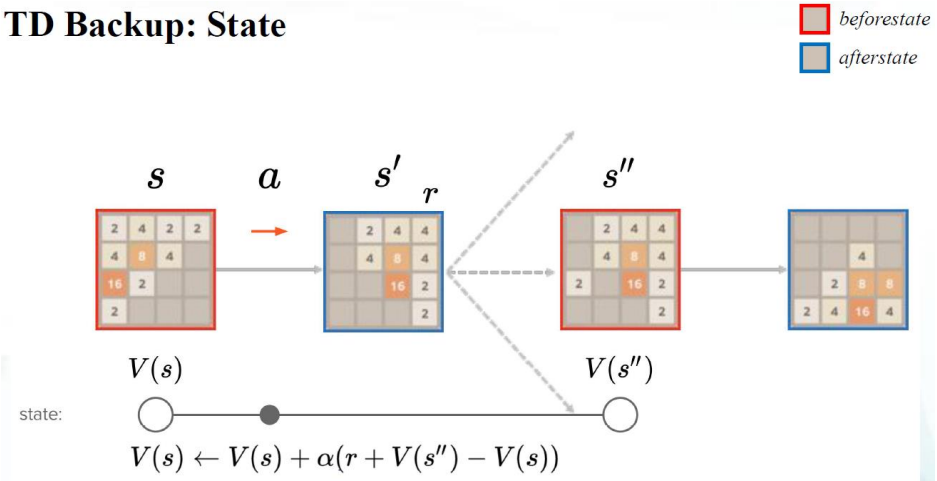
4. Explain the TD-backup diagram of V(after-state).



取在 popup 前的 state (即 S') 的 value，然後在 popup 後的 state 進行一個 action 得到 S'next 和 reward，計算 reward 與 S'next value 的和再與 S'的 value 計算差，乘上 learning rate 對 weight 進行更新。在 after-state，因為要計算 value 的 state 是移動後但還沒 popup 的版面，所以直接計算 value 即可，不用考慮在哪裡隨機生成 tile 的問題。

5. Explain the action selection of V(after-state) in a diagram.
根據當前的 S''，選擇能產生最大 reward 的 action，即在 S''能得到最高 reward 的那個 action。

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



對一開始的版面 S 進行一個 action 得到 S'與 reward，再 popup 變成 S''。update 時先計算 S''的 value 加上 reward，再拿去和 S 的 value 計算差，乘

上 learning rate 對 weight 進行更新。在 before-state 要注意的是，S'會 random 選一個位置生成 tile 變成 S''，所以 S''有多種可能性。這裡要計算所有 S''的 value，並與該 S''會出現的機率相乘。

7. Explain the action selection of V(state) in a diagram.
   根據當前的 S，選擇能產生最大 reward 的 action，即在 S 能得到最高 reward 的那個 action。

8. Describe your implementation in detail.
   estimate 的部分沒有更動，同樣是使用 indexof 去算出目前 tuple 中的值來算出對應的 index，並用那個 index 對應的位置當作 value。

```
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;
}
```

update 的部分也沒有更動，把 u (= error * weight) 除以總 tuple 數，然後再拿算出來的值去更新 index 對應的 value 值。

```
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;
}
```

indexof 的部分也沒有更動。使用 tuple 假設是{A, B, C, D, E, F}，然後對應的位置的值分別是 2^a, 2^b, 2^c, 2^d, 2^e, 2^f，則 index 是 abcdef 轉成二進位接在一起再轉成十進位。

```
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;
}
```

select_best_move 試了上下左右的 action，在每個 action 中，對 after-state 進行 popup。在一個版面上有空的第一個位置給了 2 或 4 的 tile，然後在有空的第二個位置給了 2 或 4 的 tile，以此類推。在算出該盤面的 value，然後乘上該盤面出現的機率，最後把他們加總，再加上 reward 當作調整的 target。如果算出來的值是四個 action 中值最大的，就採用那個 action，並進行下一個盤面。

```
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
            float tmp_value = 0;
            int space[16], num = 0;
            for (int i = 0; i < 16; i++){
                if (move->after_state().at(i) == 0)
                    space[num++] = i;
            }
            for(int i=0; i<num; i++){
                for(int j=1; j<=2; j++){
                    board tmp_board = move->after_state();
                    tmp_board.set(space[i], j);
                    tmp_value += (estimate(tmp_board) * (1.0 / num) * (-0.8 * j + 1.7));
                }
            }
            move->set_value(move->reward() + tmp_value);
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

update_episode 中 exact 設為 0 的意思是表示在最後一盤死局中該盤面已經沒有 value，所以設成 0。用 0+reward 然後再跟 V(S)算 difference，然後再乘上 learning rate 去 update。在下一個 iteration 中 S 變成 S’’，error = V(S’’) – (V(S) + r)符合算式，然後將算到的 difference 乘上 learning rate 去更新 weight，以此類推。

```
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 = exact - (estimate(move.before_state()) - move.reward());
        debug << "update error = " << error << " for before state" << std::endl << move.before_state();
        exact = update(move.before_state(), alpha * error);
    }
}
```

9. Other discussions or improvements.

訓練到 500000~600000 次的時候分數會卡在 100000 分左右，這時候如果微調 learning rate 到 0.05 的話，就能上 110000 分。