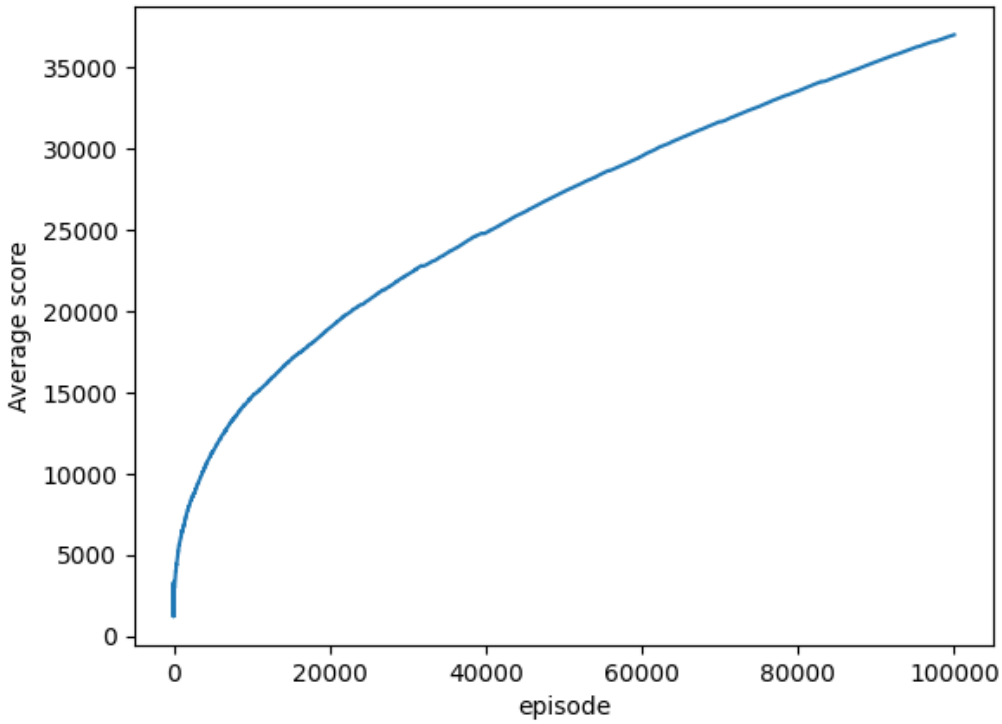
LAB 2 2048TD

謝宇恆 411551022

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



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

2048 的遊戲為 4*4 的版面，state 的總數為$16^{16} = 2^{64}$，若是要記錄所有 V(s)，需要極大的記憶體。因此需要使用 N-tuple Network 以降低其所需的記憶體。在實驗中，我們使用到了 4*6-tuple network，這樣的話我們只需要$4 \times 6^{16} = 2^{26}$種 state，約等於 256MB。每一種 pattern 會有 8 個 isomorphism，並且這 8 個 isomorphism 會使用同一個 weight table。

3. Explain the mechanism of TD(0)

TD(0)的其式子如下: $V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$，其中$V(S_t)$是對狀態$S_t$估計能獲得最終結果的獎勵，並且依據實際獎勵值$R_{t+1}$以及下一個狀態$S_{t+1}$估計能獲得最終結果的獎勵值調整$V(S_t)$。因此 TD(0)只需要考慮$V(S_{t+1})$即可更新$V(S_t)$。

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

a) TODO estimate:

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

Function estimate 是為了計算當前 board 中 Value 的期望值。該值等於將每一個 feature 對應的 8 個 isomorphic 加總出來的值。

b) TODO update

```
virtual float update(const board& b, float u) {
    // TODO
    float updated_value = 0;
    for(const std::vector<int> iso: isomorphic) {
        size_t index = indexof(iso, b);
        weight[index] += u;
        updated_value += weight[index];
    }

    return updated_value;
}
```

Function update 是更新 weight table 的 function，其中 u 為該原始 weight table 的 TD error 乘上 learning rate。

c) TODO indexof

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

Function indexof 是為了將 board 中每一個 pattern 儲存成 6*4bit 的值，這邊將 board 中每一個 pattern 上的值轉換成 4bit 的數字，並將其合併存入至變

數 index 中。

d) TODO update_episode

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float next_state_value = 0;
    for(int i = path.size() - 2; i >= 0; i--) {
        float TD_target = path[i].reward() + next_state_value;
        float TD_error = TD_target - (this -> estimate(path[i].before_state()));
        next_state_value = this -> update(path[i].before_state(), TD_error * alpha);
    }
}
```

Function update_episode 是為了更新 V 這個函數。因此在每一個 episode 都會計算 TD_target 以及 TD_error，並利用 TD_target 以及 TD_error 進行 V 的更新。

e) TODO in select_best_move

```
if (move->assign(b)) {
    // TODO
    float immediate_reward = move->reward();
    float finalest_reward = 0;
    int empty_squares = 0;
    for(int i = 0; i < 16; i++) {
        board after_state = move->after_state();
        if(after_state.at(i) == 0) {
            empty_squares++;
            //計算after_state產生4方塊的期望值
            board board_with_4tile = after_state;
            board_with_4tile.set(i, 2);
            finalest_reward += 0.1 * (this -> estimate(board_with_4tile));
            //計算after_state產生2方塊的期望值
            board board_with_2tile = after_state;
            board_with_2tile.set(i, 1);
            finalest_reward += 0.9 * (this -> estimate(board_with_2tile));
        }
    }

    if(empty_squares != 0)
        move->set_value(immediate_reward + finalest_reward / empty_squares);

    // update the cur best move, value() returns the esti
    if (move->value() > best->value())
        best = move;
```

Function select_best_move 會選擇 V(S_t)最大的值並進行移動。在這邊我們需要得知 V(S_{t+1})才能計算 V(S_t)，所以我們會先進行 next state 的期望值計算，而 next state 包含三個可能性: popup 一個 4 的 block(有 0.1 的可能性)、popup 一個 2 的 block(有 0.9 的可能性)以及沒有空間可以動(也就是 empty_square 等於 0)。在 empty_square 不等於 0 時都需要計算。

f) TD-backup diagram