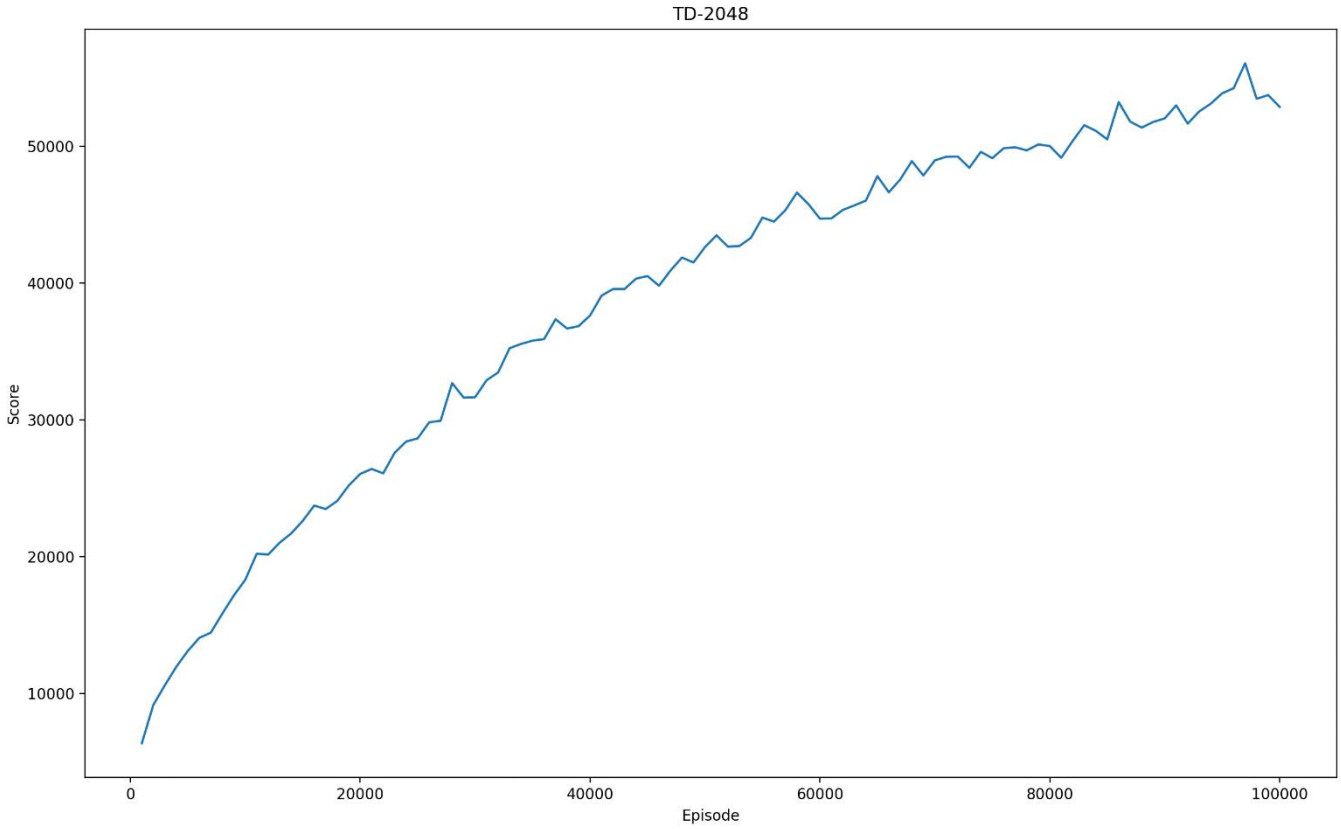
Experiment Report

- A plot shows scores (mean) of at least 100k training episodes



- Describe the implementation and the usage of *nn*-tuple network.

我們可以透過n-tuple所算出來的值，來代表當前狀態的value。這樣除了可以節省記憶體，還可以加快運算。N-tuple是透過設幾個index是我們想觀察的值，將前面所選的一組index 以及對應的isomorphic，一共會有8個狀態，一起當作一個feature。那我們可能會有4個feature，我們在每次計算board value時，都會去計算每一個feature，以及裡面每一個isomorphic所算出來的value，做相加總後，當作是這個board的value。

```
// 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 }));
```

在sample code已經訂好了 n-tuple的格式，我也沒有再特別更動。

- Explain the mechanism of TD(0).

TD(0)是會根據我們選擇動作後，所產生的下一個狀態來去更新自己的weights 或者是 value。而蒙地卡羅是TD(1)，他是用遊戲結束的最後一個狀態的value，來去更新前面所有的狀態的weights。

- Describe your implementation in detail including action selection and TD-backup diagram.
  - Action Selection:

```
if (move->assign(b)) {
    // TODO
    //每個state都會有對應的value，我們會透過計算下一個state的期望值，來去更新我們move的value
    //期望值 = 格子空格的地方出現2的value相加之後*0.9 + 格子空格的地方都出現4的value相加之後*0.1
    //E.g. 假如有三格空格, 期望值 = 3個格子各自填2對應的value相加之後乘上 0.9 + 3個格子各自填4對應的value相加之後乘上 0.1

    board temp = move->after_state();
    int num = 0;
    float total_value = 0;
    float expact_value = 0;
    for (int i = 0; i < 16; i++){
        if (temp.at(i) == 0) {
            num++;
            board temp1 = move->after_state();
            board temp2 = move->after_state();

            temp1.set(i , 1);
            total_value = total_value + estimate(temp1)*0.9;
            temp2.set(i , 2);
            total_value = total_value + estimate(temp2)*0.1;
        }
    }
    expact_value = total_value/num;
    move->set_value(move->reward()+expact_value);
    if (move->value() > best->value())
        best = move;
} else {
    move->set_value(-std::numeric_limits<float>::max());
}
```

上面 TODO 我自己撰寫的程式碼。

1. 我們會先去針對每個動作算出他的 reward 值
2. 再針對 action 後的 board(S'_t)，使用窮舉法，把每個空格出現 2 或 4 的 board(S_{t+1})，全部算出來之後，取出他的 value，乘上相對應的機率(2->0.9, 4->0.1)，全部加起來之後，再除以格子數。
3. 透過步驟 2 所算出的值，我們當作期望值，我們會拿 reward+期望值，去更新目前 board(S_t)還沒有 action 的 value。最後選擇出哪一個 action，所製造出來的 value 是最大的，我們就選擇哪個 action。

- TD-backup:

```
// TODO
float target = 0;
for(path.pop_back() , befor_action.pop_back(); path.size(); path.pop_back(),befor_action.pop_back() ){
    state& move = path.back();
    board no_action = befor_action.back();
    float error = move.reward() + target - estimate(no_action);
    target =  update(no_action, alpha * error);
}
```

上面 TODO 我自己撰寫的程式碼。

要更新 weight ，就是要用 self value + learning rate *(reward + next value – self value)。

我們透過 recursive 從尾巴(最後一個狀態)去不斷的更新 weight。我把最後一個狀態的 value 當作是 0，也就是遊戲結束的那個狀態，因為它是一個死路，所以將他設成 0 是為了讓 model 不要在進來這個會結束的狀態。而我們後面的 next value 是拿到已經更新好狀態的 self value(這是上一輪更新的 self value 被當成這一輪的 next value)，再來是 self value 以及 reward，我們都可以拿到，就可以更新 weights 了。

至於 code 裡面的 befroe_action，是我把沒有經過 select_best_move 的 board 存起來，傳入這個 funcction 裡面，讓他當成我們 self value 的 borad 來源。