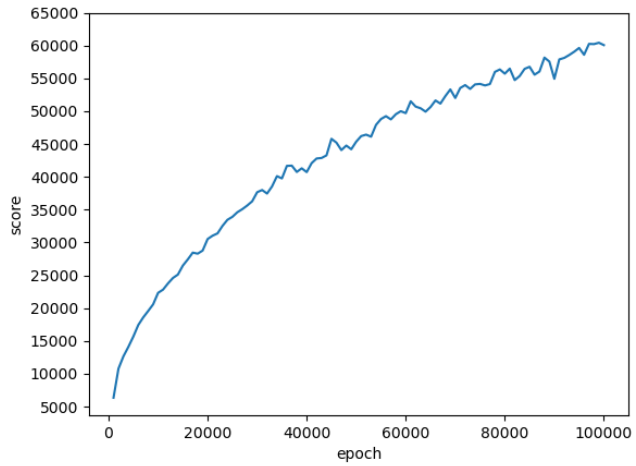
311551148 Lab2 Report

1. A plot shows scores (mean) of at least 100k training episodes
   train with before state TD(0)




```
356000  mean = 97879.6  max = 246252
        256     100%    (0.5%)
        512     99.5%   (1.3%)
        1024    98.2%   (4.2%)
        2048    94%     (10.5%)
        4096    83.5%   (41.2%)
        8192    42.3%   (41.8%)
        16384   0.5%    (0.5%)
6-tuple pattern 012345 is saved to best.bin
6-tuple pattern 456789 is saved to best.bin
6-tuple pattern 012456 is saved to best.bin
6-tuple pattern 45689a is saved to best.bin
```

   訓練 356000 episodes 時 2048 winrate 有達到 94%

2. Describe the implementation and the usage of n-tuple network.
   實作 TD(0) with n-tuple network，根據 n-tuple network 選擇最好的 move，把每一步的 state 存起來，等到 episode 結束，使用存起來的 state 更新 n-tuple network，套用 TD(0)的公式，因為要更新的是 before state 的 n-tuple network weight，所以套用 V(s) = V(s) + 0.1(r+ V(s'') – V(s))，其中 s''是做完 action 後的盤面再 pop 出新的 tile 的盤面，所以他的 value 要算所有可能盤面的 value 的期望值

```
state select_best_move(const board& b, std::string mode) 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 s_after = move->after_state();

            // Iterate all possible s''
            std::vector<int> empty_idx = s_after.get_empty_index();
            float value = 0;
            float prob_2 = 9 / (10. * empty_idx.size());
            float prob_4 = 1 / (10. * empty_idx.size());

            for (int i = 0; i < empty_idx.size(); i++) {
                board s_prime2_2 = s_after; // The one to popup 2
                board s_prime2_4 = s_after; // The one to popup 4

                s_prime2_2.popup_specific(empty_idx[i], 1); // 2^1 = 2
                s_prime2_4.popup_specific(empty_idx[i], 2); // 2^2 = 4

                value += prob_2 * estimate(s_prime2_2);
                value += prob_4 * estimate(s_prime2_4);
            }


            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;
}
```

estimate(board)會根據 n-tuple network weight 計算 board 的 value，select_best_move 會回傳上下左右四個 action 所對應的 s''中期望值最高的那個。

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float value_s_prime2 = 0; // Save for backup
    for (; path.size(); path.pop_back()) {
        state &move = path.back(); // Get the latest state
        float error = move.reward() + value_s_prime2 - estimate(move.before_state());
        value_s_prime2 = update(move.before_state(), alpha * error);
    }
}
```

訓練的部分使用 backward 的方式，從 path.back()直到 path 的地一個 state，TD error 會等於 reward + estimate(s'') – estimate(s)，因為 update 會回傳 update 後的 n-tuple value，所以可以直接當作下一個迴圈的 estimate(s'')，因為 estimate(s(t+1)) = estimate(s(t)'')

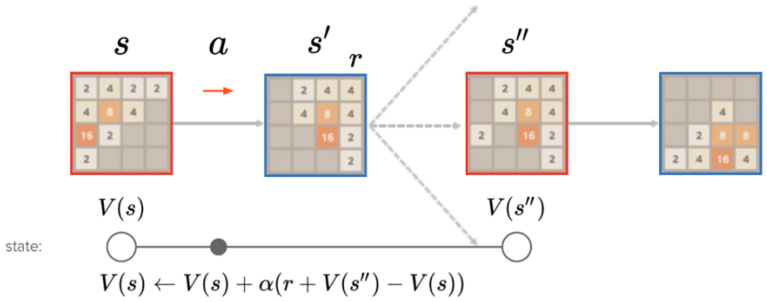
3. Explain the mechanism of TD(0)

TD(0)是一種基於時間差分的學習方法，用於評估在一個環境中採取某個動作後所獲得的即時獎勵和未來獎勵的期望值。TD(0)僅考慮了一個時間步的未來獎勵（即 0 步之後），而沒有考慮更長的時間範圍。TD(0)的機制可以包含下列幾個步驟

1. 隨機初始化 value function
2. 根據目前的狀態，選擇一個 action
3. 獲得這個 action 帶來的 reward 以及進到下一個 state
4. 更新 value function，根據公式 V(s_t)=V(s_t) + lr*(reward+V(s_t+1)-V(s_t))

重複 2~4 步直到收斂，隨著訓練進行，value function 會越來越厲害，越來越能正確的預估各個 action 的價值，所以根據 value function 來訓則 action 會越來越好。

這次 lab 是使用 before state 來訓練 n-tuple network，所以需要計算 pop up 後所有可能盤面的 value 去算期望值，根據下圖的訓練方式。



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

```
while (true) {
    debug << "state" << std::endl << b;
    state best = tdl.select_best_move(b, mode);
    path.push_back(best);

    if (best.is_valid()) {
        debug << "best " << best;
        score += best.reward();
        b = best.after_state();
        b.popup();
    } else {
        break;
    }
}
debug << "end episode" << std::endl;
```

每一個訓練的 episode 會持續到找不到 valid move，若有多個 valid move 會使用 select best move 挑選出最高 value 的 move，選出 move 後把 best move 的 state push back 到 path 裡，之後 backward 訓練時會用到

select best move 就是去計算上下左右四種 action 做完之後再 pop up 一個 tile 的那個 state 的期望值，因為 after state 上每個空格都有 0.9 的機率會 pop up 2 以及有 0.1 的機率會 pop up 4，所以使用迴圈把每一種可能的 board 都生出來，分別計算他們的 board value，乘上各自出現的機率相加起來後就是這個 action 的 value，使用 estimate 來計算 board value

```
float estimate(const board& b) const {
    debug << "estimate " << std::endl << b;
    float value = 0;
    for (feature* feat : feats) {
        value += feat->estimate(b);
    }
    return value;
}
```

```
virtual float estimate(const board& b) const {
    // TODO

    float value = 0;
    for(int i=0;i<iso_last;i++){
        value += operator[](indexof(isomorphic[i], b));
    }
    return value;
}
```

```
state select_best_move(const board& b, std::string mode) 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 s_after = move->after_state();

            // Iterate all possible s''
            std::vector<int> empty_idx = s_after.get_empty_index();
            float value = 0;
            float prob_2 = 9 / (10. * empty_idx.size());
            float prob_4 = 1 / (10. * empty_idx.size());

            for (int i = 0; i < empty_idx.size(); i++) {
                board s_prime2_2 = s_after; // The one to popup 2
                board s_prime2_4 = s_after; // The one to popup 4

                s_prime2_2.popup_specific(empty_idx[i], 1); // 2^1 = 2
                s_prime2_4.popup_specific(empty_idx[i], 2); // 2^2 = 4

                value += prob_2 * estimate(s_prime2_2);
                value += prob_4 * estimate(s_prime2_4);
            }


            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;
}
```

board value 的計算會是加總所有 tuple 對應到這個 board 的 index 對應到的 weight，index 會是走訪 tuple 所對應的位置，把那個位置的 tile 用 4bit 表示，把每個 tuple 所對應的數值 concat 起來得到 index

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float value_s_prime2 = 0; // Save for backup
    for (; path.size(); path.pop_back()) {
        state &move = path.back(); // Get the latest state
        float error = move.reward() + value_s_prime2 - estimate(move.before_state());
        value_s_prime2 = update(move.before_state(), alpha * error);
    }
}
```

當 episode 結束，使用 update_episode 來更新 n-tuple network weight，套用 TD(0)公式，因為 update 會回傳 update 後的 n-tuple value，所以可以直接當作下一個迴圈的 estimate(s'')，因為 estimate(s(t+1)) = estimate(s(t)'')，

```
float update(const board& b, float u) const {
    debug << "update " << " (" << u << ")" << std::endl << b;
    float u_split = u / feats.size();
    float value = 0;
    for (feature* feat : feats) {
        value += feat->update(b, u_split);
    }
    return value;
}
```

```
virtual float update(const board& b, float u) {
    // TODO
    float update_eq = u / iso_last;
    float updated_v = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t index = indexof(isomorphic[i], b);
        operator[](index) += update_eq;
        updated_v += operator[](index);
    }
    return updated_v;

}
```

update 的部分會把 TD target 平分給各個 tuple，各個 tuple 的 update 也會把 target 平分給 8 個 isomorphic tuple，更新的部分就是先算出這個 board 對應這個 isomorphic tuple 的 index，然後直接把 target value 夾到他的 weight 裡，會傳這個 index 更新後的 value。

5. Experiment result

```
(base) shlu2240@eva-All-Series:/eva_data2/shlu2240/NCTU_DLP/lab2$ ./2048_d0 demo
alpha = 0.1
total = 100000
seed = 1034361863
6-tuple pattern 012345, size = 16777216 (64MB)
6-tuple pattern 456789, size = 16777216 (64MB)
6-tuple pattern 012456, size = 16777216 (64MB)
6-tuple pattern 45689a, size = 16777216 (64MB)

Demo...
Loading pre-trained model...
6-tuple pattern 012345 is loaded from best_d0.bin
6-tuple pattern 456789 is loaded from best_d0.bin
6-tuple pattern 012456 is loaded from best_d0.bin
6-tuple pattern 45689a is loaded from best_d0.bin

Start playing 1000 episodes
1000    mean = 121823   max = 290216
        1024    100%    (1.2%)
        2048    98.8%   (3.3%)
        4096    95.5%   (31.1%)
        8192    64.4%   (62.6%)
        16384   1.8%    (1.8%)
(base) shlu2240@eva-All-Series:/eva_data2/shlu2240/NCTU_DLP/lab2$
```

train with after state TD(0)，with depth 1 expectimax

若使用 before state 來訓練，結果為 94% winrate

```
356000  mean = 97879.6  max = 246252
        256     100%    (0.5%)
        512     99.5%   (1.3%)
        1024    98.2%   (4.2%)
        2048    94%     (10.5%)
        4096    83.5%   (41.2%)
        8192    42.3%   (41.8%)
        16384   0.5%    (0.5%)
6-tuple pattern 012345 is saved to best.bin
6-tuple pattern 456789 is saved to best.bin
6-tuple pattern 012456 is saved to best.bin
6-tuple pattern 45689a is saved to best.bin
```