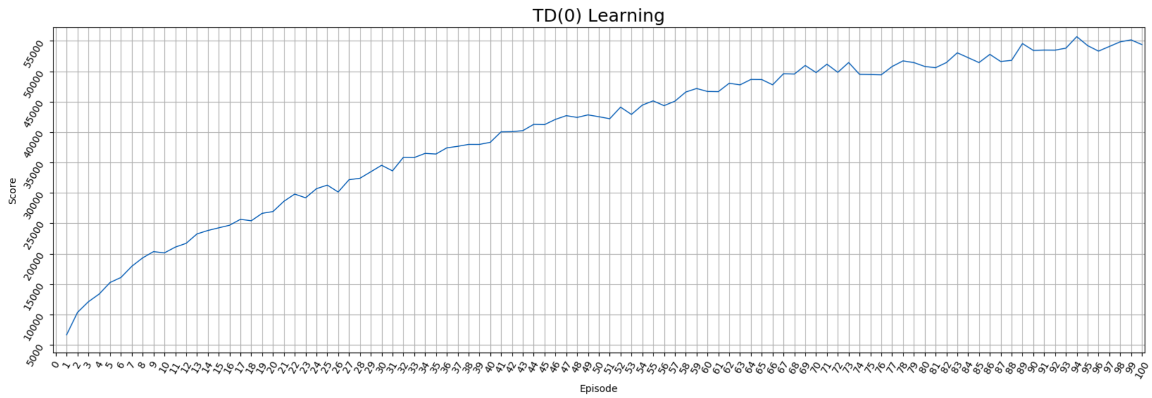
Lab7 Temporal Difference Learning

0516054 劉雨恩

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



取每1000 episode的mean score。

2. Explain the mechanism of TD(0)

TD(0) 直接從experience經歷中學習，而不需要等到最終結果才更新模型，它可以基於每個state的估算值來更新估計值，如此可以得到每個state最好的action。

3. Describe how to train and use a V(state) network

TD(0)-state

function EVALUATE(s,a)
  $s',r \leftarrow$ COMPUTE AFTERSTATE(s,a)
  $S'' \leftarrow$ ALL POSSIBLE NEXT STATES(s')
  return $r + \Sigma_{s''\in S''} P(s,a,s'')V(s'')$

function LEARN EVALUATION(s,a,r,s',s'')
  $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$

依照 presudo code 來完成 V(state) network。function EVALUATE對照的是function select_best_move，function LEARN EVALUATE對照的是function update_episode。

- **select_best_move**

```
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)) {
            /* TD(0)-state */
            float up_value = 0, p;
            board b_before = move->after_state();
            float p_tile[2] = {0.9, 0.1};
            int tileN = 0;
            std::vector<int> blankIndex;

            b_before.blankN(tileN, blankIndex);

            for(int i=0; i<tileN; i++){
                for(int j=0; j<2; j++){
                    b_before = move->after_state();
                    b_before.set(blankIndex[i], (j+1)*2);
                    p = p_tile[j] / tileN;
                    up_value += (p*estimate(b_before));
                }
            }
            move->set_value(move->reward() + up_value);
            /***************/

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

其目的是評估所有的action value，選出最好的下一步action(上、下、左、右)。而V(state) network的算法是總和所有可能出現的next state乘以出現機率，加上reward當作這個action的value。因此，實作方法便為：

1. 算出所有目前afterstate(做了一個action)空格的數量和index。

```cpp
void blankN(int& num, std::vector<int>& index) {
    index.clear();
    for (int i = 0; i < 16; i++){
        if (at(i) == 0) {
            num++;
            index.push_back(i);
        }
    }
}
```

2. 總和所有可能出現的情況算出value並乘以其機率。

   - 可能出現的情況數量：空格數量x2(tile 2 or 4)。

   - 機率：0.9(tile 2) 或 0.1(tile 4) 除以空格數量。

3. 選出最好value的action。

- **update_episode**

```cpp
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    /* TD(0)-state */
    float exact = 0, error = 0;
    board next_state = path.back().before_state();

    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = exact - (estimate(move.before_state()) - move.reward());
        exact = update(move.before_state(), alpha * error);
    }
}
```

其目的是每個episode結束時更新beforestate value function。而V(state) network的算法是計算next state value減去beforestate value加reward最後再乘以alpha。因此，實作方法便為：

1. 將episode的每一個state從terminal到最一開始一一pop出來。

2. 將每個迴圈的前一個state的value存下來，為當前迴圈的next state的value。

3. 評估next_state和beforestate的value計算更新公式，並對

beforestate做更新。

4. Describe how to train and use a V(after-state) network

TD(0)-afterstate

function EVALUATE(s, a)

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

return $r + V(s')$

function LEARN EVALUATION(s, a, r, s', s'')

$$a_{next} \leftarrow \underset{a' \in A(s'')}{\text{argmax}}\, EVALUATE(s'', a')$$

$$s'_{next}, r_{next} \leftarrow COMPUTE\ \ AFTERSTATE(s'', a_{next})$$

$$V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$$

依照presudo code來完成V(state) network。function EVALUATE對照的是function select_best_move，function LEARN EVALUATE對照的是function update_episode。

- **select_best_move**

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

其目的是評估所有的action value，選出最好的下一步action(上、下、左、右)。而V(after-state) network的算法是評估afterstate的value，加上reward當作這個action的value。因此，實作方法便為選出最好value的action。

- **update_episode**

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    float exact = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = exact - (move.value() - move.reward());
        debug << "update error = " << error << " for after state" << std::endl << move.after_state();
        exact = move.reward() + update(move.after_state(), alpha * error);
    }
}
```

其目的是每個episode結束時更新afterstate value function。而V(after-state) network的算法是計算next state afterstate true value減去afterstate value加下一個state reward最後再乘以alpha。因此，實作方法便為：

1. 將episode的每一個state從terminal到最一開始一一pop出來。

2. 將每個迴圈的前一個reward+update過的value存下來，為當前迴圈的next state afterstate true value。

3. 計算更新公式，並對afterstate做更新。

5. Describe how the code work (the whole code)

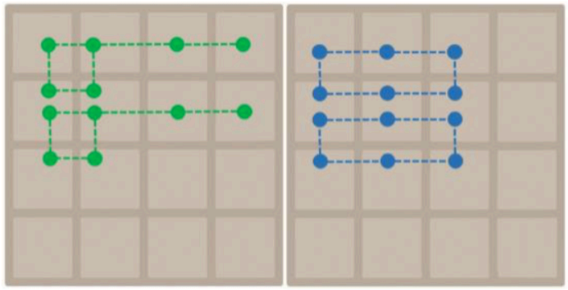
1. set the learning parameters

設定alpha、total episode數和random seed。

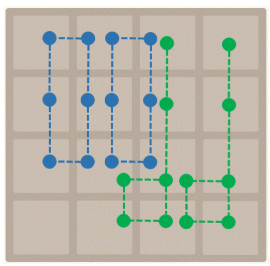
```
float alpha = 0.1;
size_t total = 100000;
unsigned seed;
__asm__ __volatile__ ("rdtsc" : "=a" (seed));
info << "alpha = " << alpha << std::endl;
info << "total = " << total << std::endl;
info << "seed = " << seed << std::endl;
std::srand(seed);
```

2. initialize the features

將用來評估value的n-tuple種類都先存好在feats裡。



```
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 }));
```



(多新增四種)

```
tdl.add_feature(new pattern({ 0, 1, 4, 5, 8, 9 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6, 9, 10 }));
tdl.add_feature(new pattern({ 2, 6, 9, 10, 13, 14 }));
tdl.add_feature(new pattern({ 3, 7, 10, 11, 14, 15 }));
```

3. train the model

i. play an episode

在每個state都選出最好的action，並得到reward、afterstate value和下一個state，將每個(before state, after state, action, reward)都一一存入path裡。

ii. update by TD(0)

使用V(state)或V(after-state) network方法來更新value function。

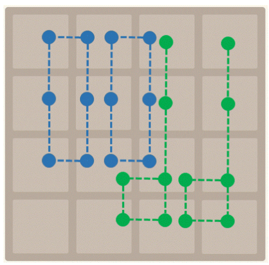
6. More you want to say

- **比較V(state) network和V(after-state) network**

  V(state) network相比V(after-state) training performance上升較慢。其原因可能為在計算best action時，需要計算每個可能發生的next state，造成計算量龐大。

- **提升training performance**

  - **增加n-tuple**



```
tdl.add_feature(new pattern({ 0, 1, 4, 5, 8, 9 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6, 9, 10 }));
tdl.add_feature(new pattern({ 2, 6, 9, 10, 13, 14 }));
tdl.add_feature(new pattern({ 3, 7, 10, 11, 14, 15 }));
```

多新增四種n-tuple後，performance上升速度較只有四種的還要來得快。

7. Strength

- {C/C++ version} The 2048-tile win rate in 1000 games.

```
mean = 98953    max = 286740
256     100%    (0.2%)
512     99.8%   (0.7%)
1024    99.1%   (3.6%)
2048    95.5%   (18.2%)
4096    77.3%   (33.6%)
8192    43.7%   (42.4%)
16384   1.3%    (1.3%)
```