DLP HW2

數據所 311554019 宋沛潔

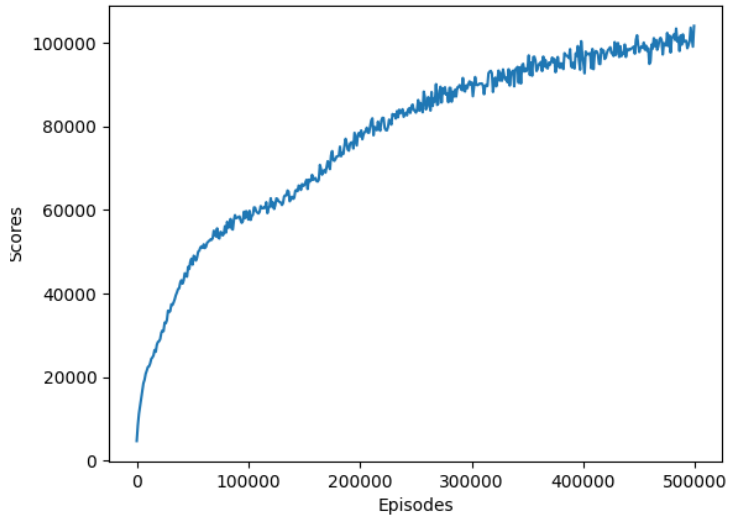
1. A plot shows scores (mean) of at least 100k training episodes:

左邊是跑 500000 次的結果，2048 有達 95.1%；右邊是 test，2048 達 92.1%

```
500000  mean = 104111   max = 188552
        64      100%    (0.1%)
        256     99.9%   (0.3%)
        512     99.6%   (1.1%)
        1024    98.5%   (3.4%)
        2048    95.1%   (10.1%)
        4096    85%     (32.9%)
        8192    52.1%   (52.1%)
```

```
1000    mean = 100316   max = 279736
        64      100%    (0.2%)
        512     99.8%   (1.1%)
        1024    98.7%   (6.6%)
        2048    92.1%   (10.6%)
        4096    81.5%   (31.6%)
        8192    49.9%   (49.8%)
        16384   0.1%    (0.1%)
```

↓ 下面的圖是訓練跑 500000 次的平均分數



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

在這次的作業中要實作 2048，對於 2048 這樣的遊戲 *nn*-tuple 可以用來計算期望值，作為選擇下一步的評估標準，又 2048 states 的空間很大，可以利用 *nn*-tuple 來解決空間問題。

2.1. 首先會先定義 n tuple，每個 n tuple 是在 4x4 遊戲版上的一組 n 個 tile 位置。設置整個 board 的 index，以計算出 8 個 isomorphic 各自在 board 上的 index。

```
for (int i = 0; i < 8; i++) {
    board idx = 0xfedcba9876543210ull;
    if (i >= 4) idx.mirror();
    idx.rotate(i);
    for (int t : p) {
        isomorphic[i].push_back(idx.at(t));
    }
}
```

2.2. 再來會去估計 isomorphic pattern 的權重，透過 key 尋找的方式去找到當下遊戲版上 isomorphic 的值，將所有 n-tuple 的值加總起來，就是當下遊戲版上的估計值。

```
virtual float estimate(const board& b) const {
    // TODO
    float est_value = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t idx = indexof(isomorphic[i], b);
        est_value += operator[](idx);  // weight -> the value of estimate
    }
    return est_value;
}
```

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

2.3. 之後在 training 中，會使用 TD-Learning 的方式去將每個 n tuple 更新權重。

```
/**
 * update the value of a given board, and return its updated value
 */
virtual float update(const board& b, float u) {
    // TODO
    float value = 0;
    for(int i = 0; i < iso_last; i++) {
        size_t idx = indexof(isomorphic[i], b);
        operator[](idx) += u / iso_last ;
        value += operator[](idx);
    }
    return value;
}
```

3. Explain the mechanism of TD (0):

TD (0)可以再給定的 policy 讓 agent 與環境互動時去更新 value function。使用兩個前後時間點的狀態和回饋的 reward 來取得新的 value 值。

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float realValue = 0;
    for(path.pop_back(); path.size(); path.pop_back()) {
        state &move = path.back();
        //TD Error
        float TD_Error =  move.reward() + realValue - estimate(move.before_state());
        realValue = move.reward() + update(move.before_state(), (alpha * TD_Error));
    }
```

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

4.1. Estimate the value of given board:

對於遊戲版上的值進行計算，透過 isomorphic 的 key 來找到他相對應的權重，並把他們進行加總，est_value 就是估出的值。

```
virtual float estimate(const board& b) const {
    // TODO
    float est_value = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t idx = indexof(isomorphic[i], b);
        est_value += operator[](idx);  // weight -> the value of estimate
    }
    return est_value;
}
```

4.2. Update the value of a given board:

這部分是在訓練中要更新權重時要呼叫的 function，同樣是透過 isomorphic 的 key 來找到他相對應的權重來進行更新。

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

4.3. Index tuple:

透過 key 形成索引，來對應 n-tuple network 中 isomorphic 相對應的權重，這邊每個 tile 都是 4 bits 表示。

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

4.4. Select best action move:

會先找到現在 game board 上的哪一些位置是空的 tile，然後去計算出所有空的位置分別放 2、4 (兩者出現機率不同，因此要乘出現的機率)算出期望值。再將這個 move 的 value 設成 before state 的期望值，最後比較四種 action（上、下、左、右）的評估分數，並選出最高的作為 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)) {
            // TODO-
            int game[16], num = 0;
            for (int i = 0; i < 16; i++) {
                if (move->after_state().at(i) == 0) game[num++] = i;
            }
            float e = 0.0;

            for(int i = 0; i< num; i++) {
                    board temp = move->after_state();
                    temp.set(game[i], 1);
                    e += estimate(temp) * 0.9 / num;
                    temp = move->after_state();
                    temp.set(game[i], 2);
                    e += estimate(temp) * 0.1 / num;
                }

            move->set_value(move->reward() + e);

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

4.5. TD-backup diagram:

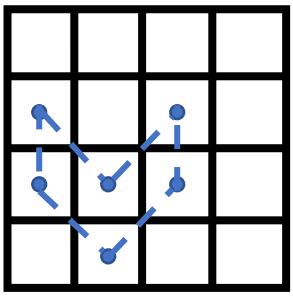
因為最後一個 path 紀錄的是 terminal，所以先 pop 掉。並開始計算 TD error，計算出這個 move 和下一個 move 的 before state 的 error，並加上該 move 的 reward。再用這個算好的 TD error 去更新現在 state 的期望值，得到 TD target ，再繼續更新。

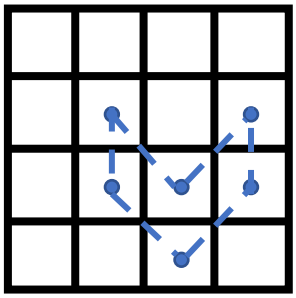
$$V(s) = V(s') + \alpha(\mathcal{R} + V(s'') - \mathrm{V}(\mathrm{s}))$$

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float realValue = 0;
    for(path.pop_back(); path.size(); path.pop_back()) {
        state &move = path.back();
        //TD Error
        float TD_Error =  move.reward() + realValue - estimate(move.before_state());
        realValue = move.reward() + update(move.before_state(), (alpha * TD_Error));
    }
```

5. Discussion:

關於 feature 的設計，除了原本的範例的四個還額外加兩種。





```
// 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 }));
tdl.add_feature(new pattern({ 4, 6, 8, 9, 10, 13 }));
tdl.add_feature(new pattern({ 4, 7, 9, 10, 11, 14 }));
```