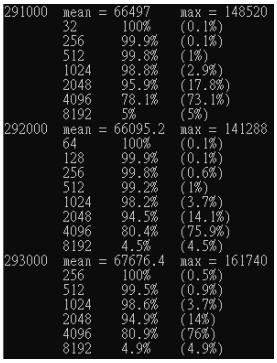
Deep Learning and Practice Lab 2

309552007

袁鈺勛

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



上圖為訓練到 291000、292000 和 293000 episodes 時呈現的結果。

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

因為要利用所有 pattern 的總值來代表當下 board 的期望值，所以會用 n-tuple network 來將各 pattern 的 8 個 isomorphic pattern 的 weight 值加總得到這個 pattern 的值。

```
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));
    }
}
```

在上圖會取得 pattern 的所有 isomorphic pattern 各自在 board 上的 index。

```
size_t indexof(const std::vector<int> &patt, const board &b) const {
    // TODO
    // Return b.at(patt[len - 1]) | b.at(patt[len - 2]) | ... | b.at(patt[1]) | b.at(patt[0])
    // as a value
    size_t index = 0;
    for (size_t i = 0; i < patt.size(); i++)
        index |= b.at(patt[i]) << (4 * i);
    return index;
}
```

藉由上圖的 function 可以取得 isomorphic pattern 在當下 board 得出來的一個值，並把這個值當成一個 key。

```
float &operator[](size_t i) { return weight[i]; }

float operator[](size_t i) const { return weight[i]; }
```

在上圖便會用從 isomorphic pattern 得到的 key 來取得他對應的 weight，也就是這些 key 在 n-tuple 的 input 為 1，所以要取得他們的 weight。

```
/**
 * estimate the value of a given board
 */
virtual float estimate(const board &b) const {
    // TODO
    // Sum up the values of all isomorphic patterns
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t index = indexof(isomorphic[i], b);
        // Operator will return the value of the given pattern.
        value += operator[](index);
    }
    return value;
}
```

藉由這 8 個 isomophic pattern 得到的 key 所對應到的 8 個 weight，便可以如上圖經過 n-tuple network 加總成 pattern 在當下 board 所代表的值。

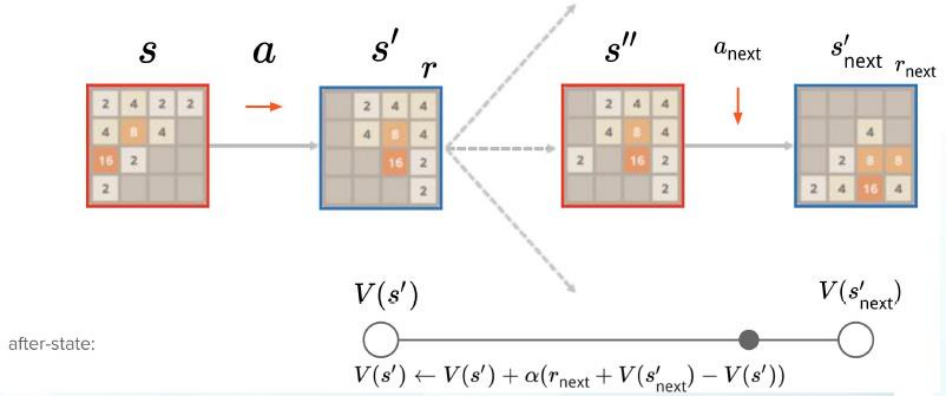
C. Explain the mechanism of TD(0).

```
void update_episode(std::vector <state> &path, float alpha = 0.1) const {
    // TODO
    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.before_state(), alpha * error);
    }
}
```

在 TD(0)會從這個 episode 所走過的 path 的倒數第二個 move 開始，計算出這個 move 和下一個 move 的 before state 的 error，並且以這個 error 來更新對應的 state 的期望值，然後再算出 TD target，也就是 exact 來讓上一個

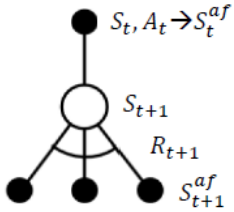
move 可以繼續更新。

D. Explain the TD-backup diagram of V(after-state).



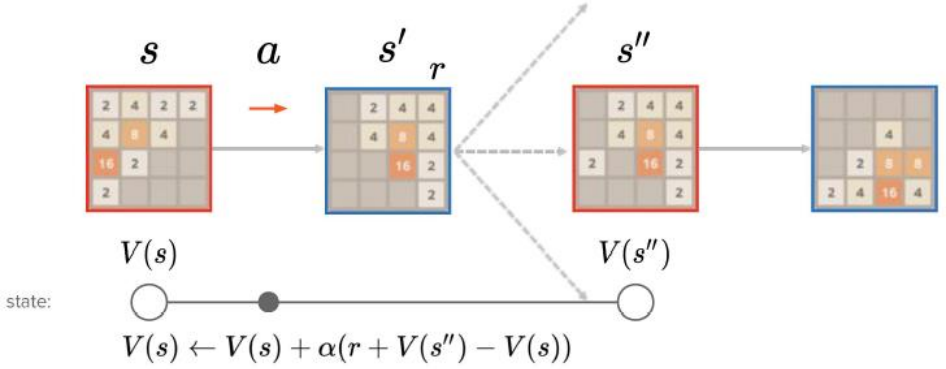
在 after state 的方式下，會以每個 move 的 after state 作為 value function，所以會以下一個 move 的 reward 加上 after state 的期望值作為 TD target 來更新現在這個 move 的 after state 的期望值。

E. Explain the action selection of V(after-state) in a diagram.



在上圖中要在 S_t 選出最好的 action，會認為下一 move 的 before state 就是一個固定的節點，而下一個 move 在選 action 也是一樣的看法，所以只要選出可以造成 $V(S_t^{af})$最大的 action 即可。

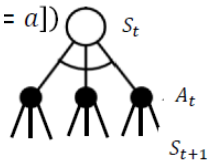
F. Explain the TD-backup diagram of V(state).



在 before state 的方式下，會以每個 move 的 before state 作為 value function，所以會以下一個 move 的 reward 加上 before state 的期望值作為

TD target 來更新現在這個 move 的 before state 的期望值。

G. Explain the action selection of V(state) in a diagram.



在上圖中要在 S_t 選出最好的 action，會因為模擬一個 action 後，board 會隨機生成一個 2 或 4 的 tile，所以會有多個 S_{t+1}，因此要計算出模擬各 action 的期望值，會需要將所有可能的 S_{t+1} 做加權平均，也就是每個 S_{t+1} 的 $V(S_{t+1})$都要乘上 2 或 4 的 tile 出現的機率以及除以空 tile 的數量。

H. Describe your implementation in detail.

```
/**
 * estimate the value of a given board
 */
virtual float estimate(const board &b) const {
    // TODO
    // Sum up the values of all isomorphic patterns
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t index = indexof(isomorphic[i], b);
        // Operator will return the value of the given pattern.
        value += operator[](index);
    }
    return value;
}
```

上圖會取得每個 isomorphic pattern 在當下 board 得到的 key，並以這個 key 來取得對應的 weight，再將這些 weight 加總來得到 pattern 在當下 board 所代表的值。

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

上圖會將 pattern 應該修正的 error 平均分給 8 個 isomorphic pattern，所以會先取得各 isomophic pattern 在當下 board 所代表的 key，並去更新這個 key 所對應的 weight，並且將更新過後的所有 weight 加總回傳。

```
size_t indexof(const std::vector<int> &patt, const board &b) const {
    // TODO
    // Return b.at(patt[len - 1]) | b.at(patt[len - 2]) | ... | b.at(patt[1]) | b.at(patt[0])
    // as a value
    size_t index = 0;
    for (size_t i = 0; i < patt.size(); i++)
        index |= b.at(patt[i]) << (4 * i);
    return index;
}
```

上圖是取得給定的 isomorphic pattern 在當下 board 代表的 key，所以會取出 pattern 的 index 所對應到的值，並把這些值經過 shift 組合得到 key。

```
state select_best_move(const board &b) const {
    state after[4] = {0, 1, 2, 3}; // up, right, down, left
    state *best = after;
    float best_total = -std::numeric_limits<float>::max();
    for (state *move = after; move != after + 4; move++) {
        if (move->assign(b)) {
            // TODO
            // Find empty tile
            board after_state = move->after_state();
            int space[16], num = 0;
            for (int i = 0; i < 16; i++)
                if (after_state.at(i) == 0) {
                    space[num++] = i;
                }

            // Set value
            float total = move->reward();
            for (int i = 0; i < num; i++) {
                board* temp = new board(uint64_t(after_state));
                temp->set(space[i], 1);
```

```
                total += 0.9f * estimate(*temp) / num;
                temp->set(space[i], 2);
                total += 0.1f * estimate(*temp) / num;
                delete temp;
            }
            move->set_value( v: estimate(move->before_state()));

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

在上面兩張圖中，會先 move 的 after state 中找到所有空的 tile，然後在所有空 tile 上放上 2 或 4 並計算出 board 的期望值並且乘上加權，而且將這個 move 的 value 設為 before state 的期望值，最後比較走這個 move 是否可以得到 4 個 move 中最高的 value。

```
void update_episode(std::vector <state> &path, float alpha = 0.1) const {
    // TODO
    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.before_state(), alpha * error);
    }
}
```

在上圖會從這個 episode 所走過的 path 的倒數第二個 move 開始，計算出這個 move 和下一個 move 的 before state 的 error，並且以這個 error 來更新對應的 state 的期望值，然後再算出 TD target，也就是 exact 來讓上一個 move 可以繼續更新。

I. Other discussions or improvements.

在以原先 code 裡面的兩種 6-tuple pattern 來訓練的情況底下能達到 2048 的比例大約只能到 90%左右，所以我加上了 3 個不同的 pattern。

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

第一和第二行是缺一腳的 x，第三行是十字架，第四行是類似く的圖案，加上這三種 pattern 後，達到 2048 的比例最高有到 97%左右。