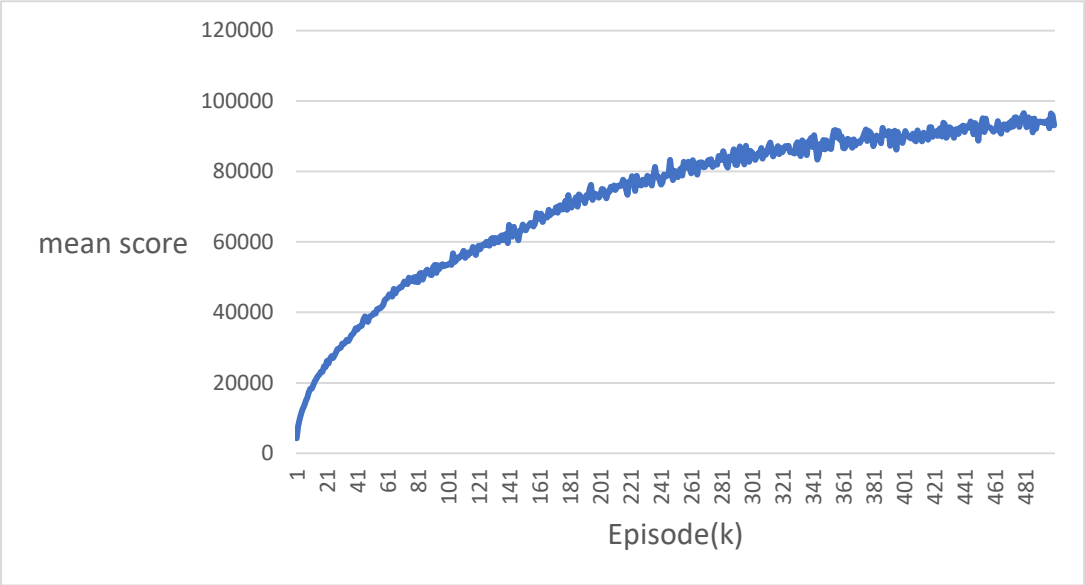
Selected Topics in Reinforcement Learning

Lab1：Temporal Difference Learning

312552026

蔡濟謙

A plot shows scores (mean) of at least 100k training episodes (20%)



Bonus: (20%)

- **Describe the implementation and the usage of n-tuple network. (5%)**

要計算所有 pattern 的總和來計算當前 board 的期望值，

Board 有 16 個格子，每個格子有 16 種狀態(沒數字、2^1、2^2 … 2^{15})，因此如果儲存 16^{16} 種可能的組成，需要使用非常大的記憶體空間。為了解決這個問題，使用 n-tuple network，只取版面上 n 個格子的特徵，來降低記憶體使用以及有更好的結果。

- 取得 pattern 所有同構圖形(isomorphic pattern)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));
    }
}
```

- 計算當前 board 在 pattern 對應 index 的數值。

```
size_t indexof(const std::vector<int> &patt, const board &b) const {
    // TODO
    size_t index = 0;
    // 將 board 數值轉換成 value，第 16 格在最高位，第一個在最低位
    for (int i = patt.size() - 1; i >= 0; i--) {
        index = index * 16 + b.at(patt[i]);
    }
    return index;
}
```

- 計算 8 的同構圖形的 index 並取得對應 weight，加總作為當前 board 代表的值。

```
virtual float estimate(const board &b) const {
    // TODO
    // 計算所有同構 pattern 值總和
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        // 取得 board 以第 i 種結構組成的 index
        size_t index = indexof(isomorphic[i], b);
        // operator 會回傳該 index 的 pattern 數值
        value += operator[](index);
    }
    return value;
}
```

- operator[]會回傳 pattern 對應 index 的 weight。

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

- **Explain the mechanism of TD(0). (5%)**

在一個 episode 中，TD(0)會從移動路徑(path)倒數第二個 move 開始，計算當前 move 與下一個 move 的 before state 的差值(error)，透過差值來更新 state 的 weight。然後再計算 TD target，讓再上一個 move 可以更新。

```
void update_episode(std::vector<state> &path, float alpha = 0.1) const {
    // TODO
    // 將一次移動的狀態更新到 weight
    float target = 0;
    for (path.pop_back(); path.size(); path.pop_back()) {
        state &move = path.back();
        // 計算差值
        float err = target - (estimate(move.before_state()) - move.reward());
        // 更新差值*比例到 weight
        target = move.reward()+update(move.before_state(), err * alpha);
    }
}
```

- **Describe your implementation in detail including action selection and TD-backup diagram. (10%)**

1. 取得每個同構的 pattern 對應的 index，並得到其 weight，將這些 weight 加總作為當前 board 的值。

```
virtual float estimate(const board &b) const {
    // TODO
    // 計算所有同構 pattern 值總和
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        // 取得 board 以第 i 種結構組成的 index
        size_t index = indexof(isomorphic[i], b);
        // operator 會回傳該 index 的 pattern 數值
        value += operator[](index);
    }
    return value;
}
```

2. 計算到的 error 要平均分配到 8 個同構的 pattern，調整值要除 8。除了將調整值更新到每個 pattern 對應 index 的 weight，更新後也將新的 weight 回傳。

```
virtual float update(const board &b, float u) {
    // TODO
    // 使用平均 value 更新所有同構 pattern
    float adjust = u / iso_last;
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        // 取得 board 以第 i 種結構組成的 index
        size_t index = indexof(isomorphic[i], b);
        // 將調整值加入到對應 pattern
        operator[](index) += adjust;
        // 新的 pattern 值累加至 value
        value += operator[](index);
    }
    return value;
}
```

3. 計算當前 board 在 pattern 對應 index 的數值。

```
size_t indexof(const std::vector<int> &patt, const board &b) const {
    // TODO
    size_t index = 0;
    // 將 board 數值轉換成 value，第 16 格在最高位，第一個在最低位
    for (int i = patt.size() - 1; i >= 0; i--) {
        index = index * 16 + b.at(patt[i]);
    }
    return index;
}
```

4. action selection：要在 S_t 選出一個最好的 move，因此會模擬下一個 move(S_{t+1})，在移動上下左右其中一步後，在 board 上空的位置生成一個 2 或 4 的 tile，期望值以 2 與 4 出現的比例做加權，並計算所有空格的期望值總和再除格子數

取平均，作為該 move 的 before state，最後最高的平均值作為 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
            float val = 0;
            int cnt = 0;
            board now = move->after_state();
            for (int i = 0; i < 16; i++) {
                // 2 和 4 出現的比例 9:1，因此出現 2 與出現 4 的 board 值以 9:1 的加權累計
                if (now.at(i) == 0) {
                    cnt++;
                    now.set(i, 1);
                    val += 0.9 * estimate(now);
                    now.set(i, 2);
                    val += 0.1 * estimate(now);
                    now.set(i, 0);
                }
            }
            // cnt=0 代表沒空格，因此不必計算
            if (cnt != 0) {
                // 因 board 可能有多個空格，用 cnt 去紀錄有幾個空格，全部累計後再取平均
                move->set_value(move->reward() + val / cnt);
            }
            if (move->value() > best->value()) best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

5. TD-backup diagram：每個 move 的 before state 的 value，加上下一個 move 的 reward 作為當前 move 的 before state 的期望值。在一個 episode 中，會從移動路徑(path)倒數第二個 move 開始，計算當前 move 與下一個 move 的 before state 的差值(error)，透過差值來更新 weight。以及計算 TD target，讓上一個 move 可以更新。

```
void update_episode(std::vector<state> &path, float alpha = 0.1) const {
    // TODO
    // 將一次移動的狀態更新到 weight
    float target = 0;
    for (path.pop_back(); path.size(); path.pop_back()) {
        state &move = path.back();
        // 計算差值
        float err = target - (estimate(move.before_state()) - move.reward());
        // 更新差值*比例到 weight
        target = move.reward()+update(move.before_state(), err * alpha);
    }
}
```