Lab2-TD

張千祐
Department of Computer Science
National Yang Ming Chiao Tung University
qianyou.cs11@nycu.edu.tw

1 Learning curve

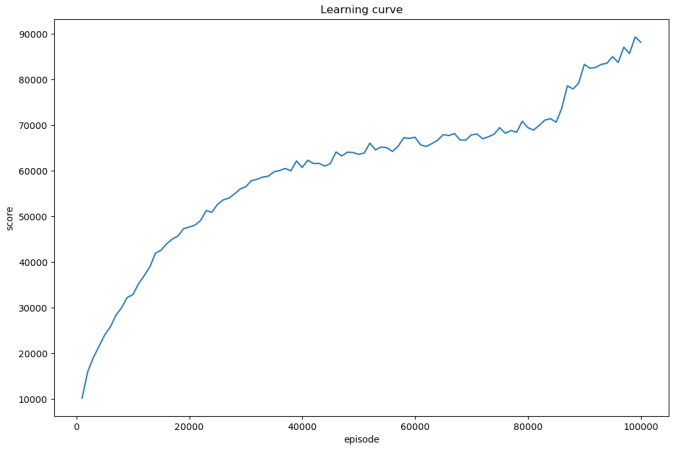

Figure 1: The scores in 100k training episodes

2 Implementation and the usage of n-tuple network

2.1 Implementation of n-tuple network

在實作的部分，首先可以從 index 講起，每一種 n-tuple 的 n 個位置數字的組合可以計算出一個獨特的 index，這也就是說，只要當任何一個 n-tuple 的位置或數字不同，那麼計算出來的 index 也會不同，因此，實作上我們就可以使用這種 index，透過 lookup table 的方式，讓每一種 n-tuple 的狀態有自己的權重，詳細計算 index 的方式如程式原始碼裡的 indexof 函式，如 Listing 1：

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
    size_t index = 0;
    for(int i=0;i<patt.size();i++){
        index |= b.at(patt[i]) << (4*i);
    }
    return index;
}
```

Listing 1: Function indexof()

接著，我們透過旋轉和鏡像的方式轉換盤面，算出 n-tuple 裡面所有的 isomorphism，將所有 isomorphism 的權重相加即可得到某個盤面下的 n-tuple 分數預測值，如 listing 2：

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

Listing 2: Function estimate()

2.2 Usage of n-tuple network

我認爲使用 n-tuple 的原因有兩大用處，第一是原始問題的 state 數量太多，我們知道 2048 每一格可以有 17 種不同的數字，總共有 16 格，因此 state 數量高達 17^{16} 種，若是同樣用上面 lookup table 的方式，由於記憶體的限制，目前是不可行的，因此 n-tuple 成了很好的解決方案，使得 state 數量因爲拆成多組 n-tuple 有了減省。第二是 n-tuple 可以根據自己的經驗選擇 pattern，也就是那些對於做決策更重要的 pattern，這樣子有助於更快的訓練速率。

3 Machanism of TD(0)

若要用一言以蔽之，TD(0) 可以用下面這個式子來描述 (假設將 γ 設置爲 1 的情況)：

$$V(s_t) \leftarrow V(s_t) + \alpha\big[R_{t+1} + V(s_{t+1}) - V(s_t)\big]$$

其中可以單獨的將兩樣東西提出來看：

$$\text{TD target} \coloneqq R_{t+1} + V(s_{t+1})$$

$$\text{TD error} \coloneqq R_{t+1} + V(s_{t+1}) - V(s_t)$$

如同字面上的意思，TD target 是我們希望 $V(s_t)$ 接近的目標，而我們當前的誤差就可以用 TD target 和 $V(s_t)$ 兩者的差來表達，也就是我們的 TD error，因此透過調節適當的 step size α，我們可以使得 $V(s_t)$ 的預測愈加準確。

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

4.1 Implementation

上面已經有帶過 indexof 和 estimate 的 implementation，下面將依序介紹剩下的部分

4.1.1 update

update 的 step size 會是原始的 step size/isomophism 個數，因此很簡單的將每個 isomorphism 的 index 計算出來後更新權重即可。

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

Listing 3: Function update()

4.1.2 select_best_move

根據 best move 的定義，這邊我們必須選擇 $a_t = \arg\max \mathbb{E}[V(s_{t+1})]$ where s_{t+1} is influenced by a_t。根據這個定義，我們必須走過上下左右四種 action，之後計算所有可能的 s_{t+1} 的 value，最後取擁有最大 value 的 action。

```
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)) {
            // check every possible empty tile for new tile in s'
            float value = 0;
            unsigned int numOfEmpty = 0;
            for(int i=0;i<16;i++){
                if(move->after_state().at(i) == 0){
                    board board_pop_two = move->after_state();
                    board board_pop_four = move->after_state();
                    board_pop_two.set(i, 1);
                    board_pop_four.set(i, 2);
                    value += POP_TWO_PROB * estimate(board_pop_two) +
    POP_FOUR_PROB * estimate(board_pop_four);
                    numOfEmpty++;
                }
            }

            value /= numOfEmpty;
            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;


}
```

Listing 4: Function select_best_move()

4.1.3 update_episode

這邊則是根據 TD(0) 的機制，由後往前的依序更新每個 state 的 value 值，更新方式如同前面 TD(0) 機制的介紹，這邊會由後往前更新是因爲要先計算 next_value 值，也就是 $V(s_{t+1})$。

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const
    {
    float next_value = 0;

    for(path.pop_back();path.size();path.pop_back()){
        state &s = path.back();
        next_value = update(s.before_state(), alpha * (s.reward() +
    next_value - estimate(s.before_state())));
    }
}
```

Listing 5: Function update_episode()

4.2 TD-backup diagram

首先要釐清 beforestate 和 afterstate 的區別，beforestate 是還沒執行 action 的盤面，而 afterstate 是執行完 action 但還沒有 popup 出新的 tile 時的盤面，如 Figure 2。

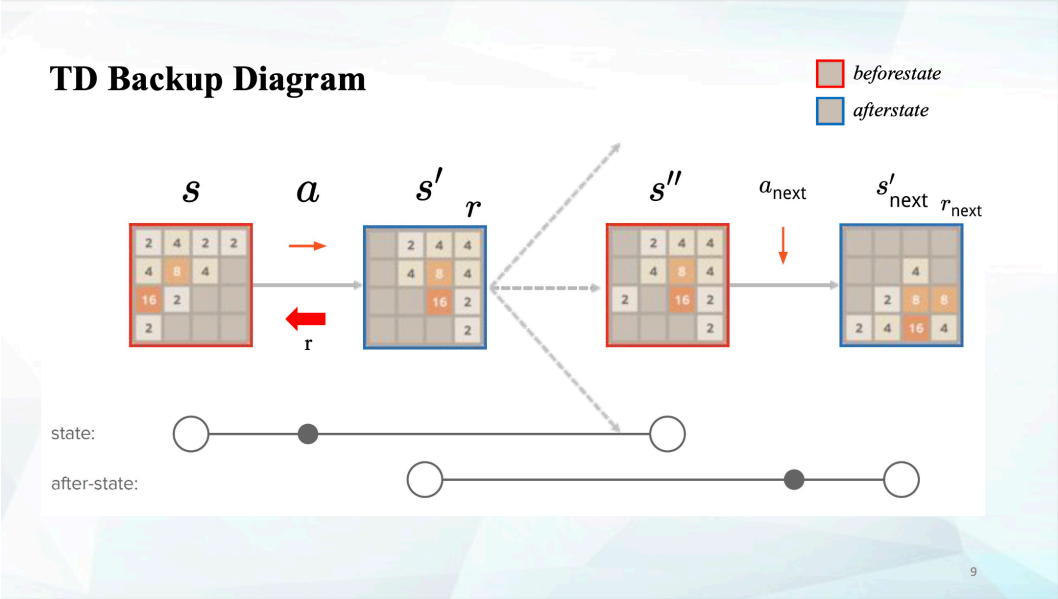

Figure 2: TD-backup diagram

而我們這次做的是 beforestate 的方式，因此在選 best move 時才會需要分別計算執行上下左右四個 action 後的 s'' 的 value 期望值

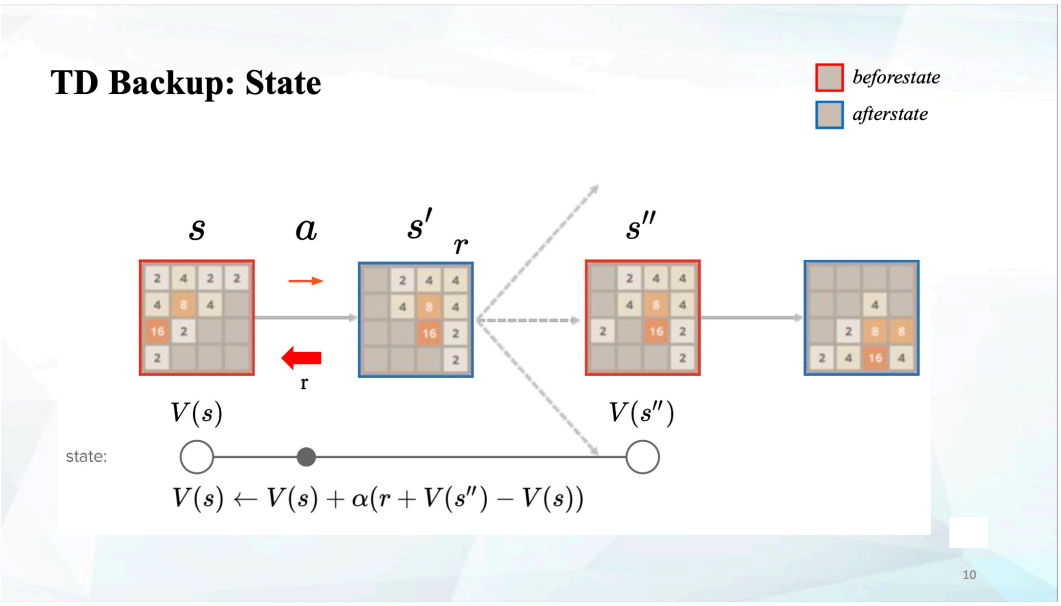

Figure 3: TD-backup: beforestate