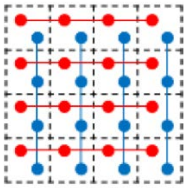
Theory of Computer Games 2022 – Project 2

311551069 余忠旻

Network Design

我試了兩種 N-tuple network 方法

第一種，依照老師上課教的 8 x 4-tuple network，也就是四直四橫來操作



第二種，可以看到下圖是依照 4 x 6-tuple network 並且有 8 種 isomorphic patterns 需要去儲存(也就是旋轉 0°, 90°, 180°, 270° 以及水平鏡射後的旋轉 0°, 90°, 180°, 270°)

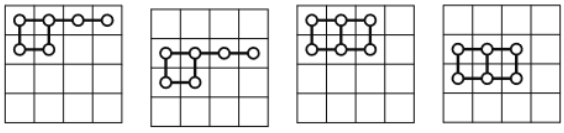


Fig. 2. The four 6-tuples by Wu et al. [11]

Method Used

Take action

We use immediate rewards and afterstate values.

```
virtual action take_action(const board& before) {
    int bestOP = -1;
    int bestReward = -1;
    float bestValue = -100000;
    board bestAfterstate;
    for(int op : opcode) {
        board afterstate = before; // use to store state after sliding
        int reward = afterstate.slide(op);
        if(reward == -1) continue;
        float value = valueEstimate(afterstate);
        if(reward + value > bestReward + bestValue){
            bestReward = reward;
            bestValue = value;
            bestOP = op;
            bestAfterstate = afterstate;
        }
    }
    if(bestOP != -1){
        replayBuffer.push_back({bestReward, bestAfterstate});
    }
    return action::slide(bestOP);
}
```

TD-learning

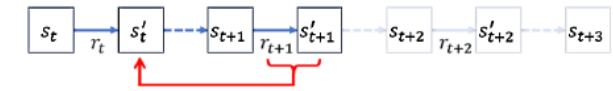
We update the weight by the difference between estimated value of current afterstate and the estimated value of the next afterstate plus reward.

實際操作部分，我們會在每個 episode 中用 replayBuffer 去紀錄 afterstate 以及 reward，接著用 backward method updates 的方式從後面更新回來(TD target of last afterstate should be 0)

Step 1: after game over (s_{t+3}), update the last state (s'_{t+2})

Step 2: update the previous *afterstate* (s'_{t+1})

Step 3: update the previous *afterstate* (s'_t)



ValueEstimate:

The value function $\boldsymbol{V}(\boldsymbol{s}t')$ is calculated as $\boldsymbol{\Theta}_1[\boldsymbol{\phi}_1(\boldsymbol{s}t')]+\cdots+\boldsymbol{\Theta}_8[\boldsymbol{\phi}_8(\boldsymbol{s}t')]$.

ValueAdjust:

These feature weights are adjusted with the same TD error

$\boldsymbol{\Theta}[\boldsymbol{\phi}(\boldsymbol{st}')]\leftarrow\boldsymbol{\Theta}[\boldsymbol{\phi}(\boldsymbol{st}')]+\boldsymbol{\alpha}(\boldsymbol{rt}+\mathbf{1}+\boldsymbol{V}(\boldsymbol{st}+\mathbf{1}')-\boldsymbol{V}(\boldsymbol{st}'))$.

```
int featureExtract(const board& after, int a, int b, int c, int d, int e, int f) const {
    //for 8*4-tuple
    //return after(a) * 16 * 16 * 16 + after(b) * 16 * 16 + after(c) * 16 + after(d);

    //for 4*6-tuple
    return after(a) * 16 * 16 * 16 * 16 * 16 + after(b) * 16 * 16 * 16 * 16 + after(c) * 16 * 16 * 16 + after(d) * 16 * 16 + after(e) * 16 + after(f);
}
```

```
float valueEstimate(const board& after) const {
    float value = 0;

    //for 8*4-tuple
    /*
    value += net[0][featureExtract(after, 0, 1, 2, 3)];
    value += net[1][featureExtract(after, 4, 5, 6, 7)];
    value += net[2][featureExtract(after, 8, 9, 10, 11)];
    value += net[3][featureExtract(after, 12, 13, 14, 15)];
    value += net[4][featureExtract(after, 0, 4, 8, 12)];
    value += net[5][featureExtract(after, 1, 5, 9, 13)];
    value += net[6][featureExtract(after, 2, 6, 10, 14)];
    value += net[7][featureExtract(after, 3, 7, 11, 15)];
    */

    //for 4*6-tuple
    board state = after;
    for(int r = 0; r < 4; r++) {
        value += net[0][featureExtract(state, 0, 1, 2, 3, 4, 5)];
        value += net[1][featureExtract(state, 4, 5, 6, 7, 8, 9)];
        value += net[2][featureExtract(state, 0, 1, 2, 4, 5, 6)];
        value += net[3][featureExtract(state, 4, 5, 6, 8, 9, 10)];
        state.rotate_clockwise();
    }
    state.reflect_horizontal();
    for(int r = 0; r < 4; r++) {
        value += net[0][featureExtract(state, 0, 1, 2, 3, 4, 5)];
        value += net[1][featureExtract(state, 4, 5, 6, 7, 8, 9)];
        value += net[2][featureExtract(state, 0, 1, 2, 4, 5, 6)];
        value += net[3][featureExtract(state, 4, 5, 6, 8, 9, 10)];
        state.rotate_clockwise();
    }

    return value;
}
```

```
void valueAdjust(const board& after, float TDtarget) {
    float currentV = valueEstimate(after);
    float TDerror = TDtarget - currentV;
    float adjustment = alpha * TDerror;

    //These 8 feature weights are adjusted with the same TD error.
    /*
    net[0][featureExtract(after, 0, 1, 2, 3)] += adjustment;
    net[1][featureExtract(after, 4, 5, 6, 7)] += adjustment;
    net[2][featureExtract(after, 8, 9, 10, 11)] += adjustment;
    net[3][featureExtract(after, 12, 13, 14, 15)] += adjustment;
    net[4][featureExtract(after, 0, 4, 8, 12)] += adjustment;
    net[5][featureExtract(after, 1, 5, 9, 13)] += adjustment;
    net[6][featureExtract(after, 2, 6, 10, 14)] += adjustment;
    net[7][featureExtract(after, 3, 7, 11, 15)] += adjustment;
    */

    //These is 4*6-tuple for eight isomorphic patterns.(32 features)
    board state = after;
    for(int r = 0; r < 4; r++) {
        net[0][featureExtract(state, 0, 1, 2, 3, 4, 5)] += adjustment/8;
        net[1][featureExtract(state, 4, 5, 6, 7, 8, 9)] += adjustment/8;
        net[2][featureExtract(state, 0, 1, 2, 4, 5, 6)] += adjustment/8;
        net[3][featureExtract(state, 4, 5, 6, 8, 9, 10)] += adjustment/8;
        state.rotate_clockwise();
    }
    state.reflect_horizontal();
    for(int r = 0; r < 4; r++) {
        net[0][featureExtract(state, 0, 1, 2, 3, 4, 5)] += adjustment/8;
        net[1][featureExtract(state, 4, 5, 6, 7, 8, 9)] += adjustment/8;
        net[2][featureExtract(state, 0, 1, 2, 4, 5, 6)] += adjustment/8;
        net[3][featureExtract(state, 4, 5, 6, 8, 9, 10)] += adjustment/8;
        state.rotate_clockwise();
    }
}
```

而使用 4 x 6-tuple network 時，要注意的是 valueEstimate 和 valueAdjust 要把 8 種 isomorphic patterns 計算進去

Training Process

We reduce the learning rate α only if the network is converged.

我以 200000 episode 為單位慢慢把 learning rate 調低(一開始 learning rate=0.1 開始訓練)，並且同時把 n-tuple network weights 寫到 weights.bin，執行結果寫到 train.log

以下是 4 x 6-tuple network 最終訓練完的結果(會比 8 x 4-tuple network 的效果好):

```
[chungminyu@tcglinux7 311551069]$ ./threes-judge --load pj2/stats.txt --judge version=2
Threes! Judge: ./threes-judge --load pj2/stats.txt --judge version=2

1000    avg = 174965, max = 561159, ops = 807830 (466175|4259039)
        48      100%    (0.3%)
        96      99.7%   (1%)
        192     98.7%   (1%)
        384     97.7%   (5.2%)
        768     92.5%   (7.1%)
        1536    85.4%   (18.3%)
        3072    67.1%   (66.7%)
        6144    0.4%    (0.4%)

Judging the actions... Passed
Judging the speed... Passed, expected 39761 ops
Assessment: 95.5 points
```