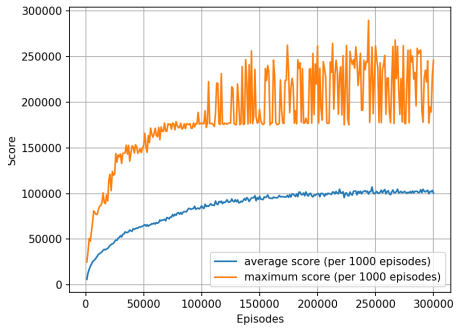
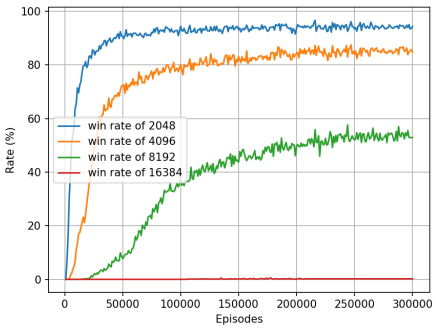
Lab2: Temporal Difference Learning

Hsiang-Chun Yang
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University
yanghc.cs09g@nctu.edu.tw

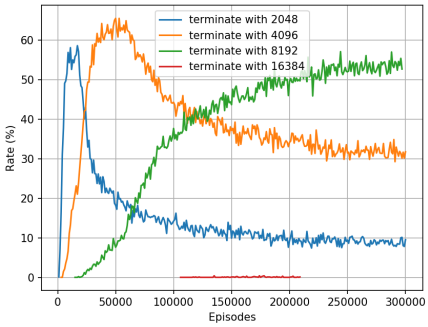
1 Plotting



(a) Score



(b) Win rate



(c) Termination rate

Figure 1: Result after 300000 episodes

1.1 After 100000 episodes

```
100000  mean = 86475.6  max = 177340
        256     100%    (0.6%)
        512     99.4%   (1.3%)
        1024    98.1%   (3.7%)
```

```
        2048    94.4%   (14.3%)
        4096    80.1%   (44.3%)
        8192    35.8%   (35.8%)
```

1.2 After 300000 episodes

```
300000  mean = 100645  max = 246004
        128     100%    (0.1%)
        256     99.9%   (0.6%)
        512     99.3%   (1.8%)
        1024    97.5%   (3.2%)
        2048    94.3%   (9.6%)
        4096    84.7%   (31.8%)
        8192    52.9%   (52.7%)
        16384   0.2%    (0.2%)
```

2 n-tuple network

用少量的格子來估計狀態的好壞，可作為選擇下一步的評估標準，程式中的 `pattern` 物件即為 n-tuple 的實作，當中的 `weight` 是用來儲存單一 tuple 所有可能狀態所對應的權重，對於一個由 n 個格子組成的 tuple，`weight` 是一個長度為 2^{4n} 的 floating point array，並利用 `indexof` 函數計算該狀態下 tuple 的值，並作為 `weight` array 的索引。

Figure 2 是我選用的 tuple。

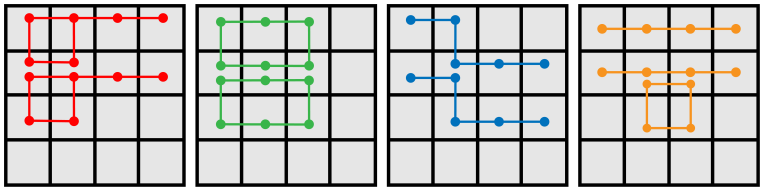


Figure 2: Design of n-tuple

3 Mechanism of TD(0)

TD(0) 指的是 one-step TD，只需要兩個相鄰時間點的狀態以及 reward 就能更新估計值。

4 TD-backup diagram of V(after-state)

$$V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$$

示意圖請參考 Figure 3，n-tuple network 儲存的是 after-state 的時候 tuple 的權重，更新權重的時候也是針對 after-state 的 tuple 做更新。估計誤差的計算方式為將下個時間點的 after-state 估計分數減去目前的 after-state 的估計分數，並加上下個時間點的 state 到下一個 after-state 之間實際獲得的 reward，再將最後的結果乘上 learning rate α。

5 Action selection of V(after-state)

評估 after state 的公式為：

$$r + V(s')$$

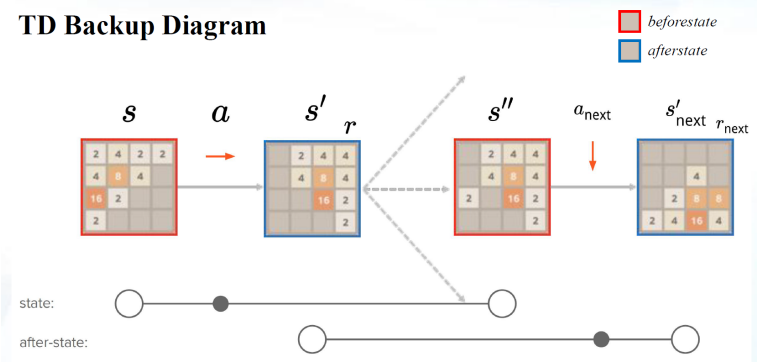


Figure 3: TD backup diagram

r 為目前的 state 做完 action 之後獲得的 reward，s' 為做完 action 之後產生的 after-state。然後比較四種 action（上、下、左、右）的評估分數，並選出最高的作為 best move。

6 TD-backup diagram of V(state)

$$V(s) \leftarrow V(s') + \alpha(r + V(s'') - V(s))$$

示意圖同樣參考 Figure 3，n-tuple network 儲存的是 state 的時候 tuple 的權重，更新權重的時候是針對 state 的 tuple 做更新。估計誤差的計算方式為將下個時間點的 state 的估計分數減去目前 state 的估計分數，並加上目前的 state 到 after-state 之間實際獲得的 reward，再將最後的結果乘上 learning rate α。

7 Action selection of V(state)

在知道目前的 state 轉變為下個時間點的 state 的機率的情況下，評估目前 state 的公式為：

$$r + \Sigma_{s'' \in S''} P(s, a, s'') V(s'')$$

s 為目前的state，P 為 state 轉變為下個時間點所有可能的 state 的機率，r 為目前的 state 做完 action 之後獲得的 reward。然後比較四種 action（上、下、左、右）的評估分數，並選出最高的作為 best move。

8 Implementation

8.1 Estimate the value of given board

```
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;
}
```

將一個或多個（isomorphic）tuple 的值作為索引找到相對應的 weight 並加總，`iso_last` 是該 pattern 中的 tuple 數量，如果是 isomorphic 就會有八個 tuple。

8.2 Update the value of a given board

```
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;
}
```

將一個或多個（isomorphic）tuple 的值作為索引找到相對應的 weight，並更新。

8.3 Encode a tuple into a index

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

把 tuple 內各個格子的值組合成一個索引值，用來找 n-tuple network 中相對應的 weight。

8.4 Select a 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)) {
            int num = 0;
            float value = 0;
            board tmp;
            for (int i = 0; i < 16; i++) {
                if (move->after_state().at(i) == 0) {
                    tmp = board(move->after_state()); tmp.set(i, 1);
                    value += 0.9 * estimate(tmp);
                    tmp = board(move->after_state()); tmp.set(i, 2);
                    value += 0.1 * estimate(tmp);
                    num++;
                }
            }
            move->set_value(move->reward() + value / num);
            if (move->value() > best->value()) {
                best = move;
```

```
            }
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
    }
    return *best;
}
```

按照 Section 7 所說明的方式估計四種 action 的分數，並選擇分數最高的作為 best move。

8.5 Update the tuple network by an episode

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

由最終狀態往前（最初狀態）更新 n-tuple network，照著 Section 6 說明的方式來計算估計誤差並更新網路。

9 Discussion

我原本認為 tuple 越大表現就會越好，後來用六個 6-tuple 與四個 6-tuple 加兩個 4-tuple 相比，發現後者的表現反而比較好，而且記憶體使用量也少很多，改變我原先認為 tuple 越大越好的想法，雖然這可能跟我使用的 6-tuple 如何設計有關，但我認為經過仔細設計之後的 4-tuple 與 5-tuple，是有機會達到比 6-tuple 更好的表現，同時使用的記憶體也比較少。