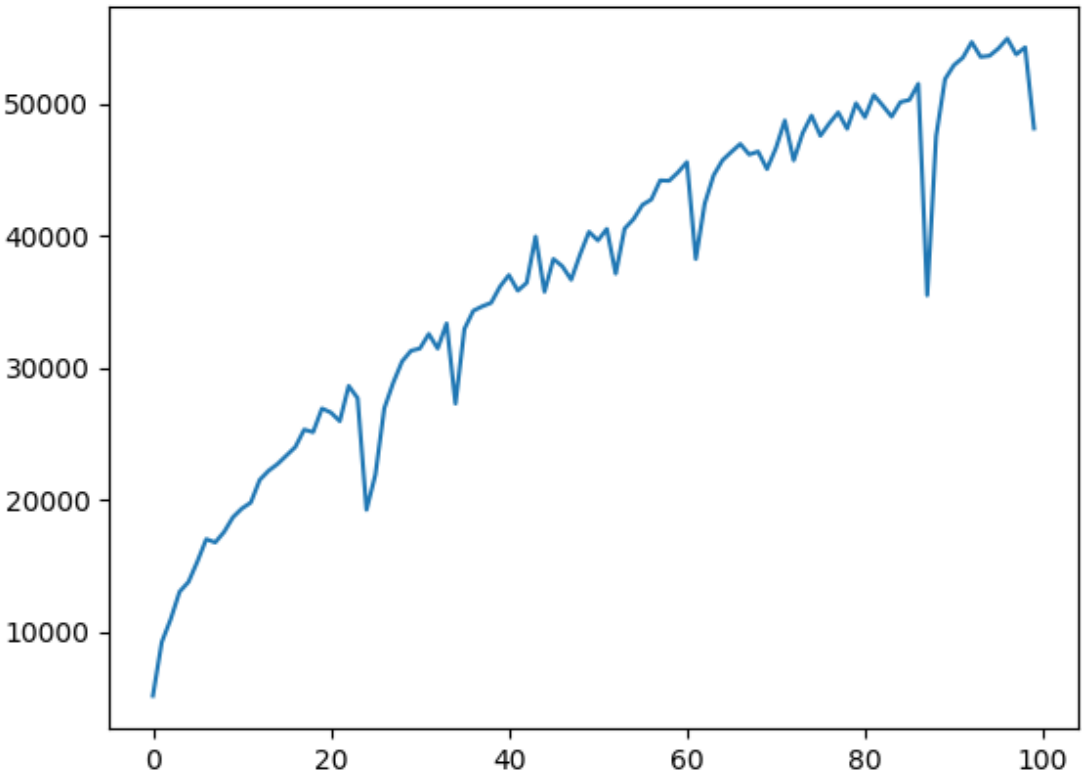
DLP HW2

309553008

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




```
99000   mean = 54284.4   max = 170828
    128 100%    (0.2%)
    256 99.8%   (0.6%)
    512 99.2%   (3.2%)
    1024    96% (15.2%)
    2048    80.8%   (30.9%)
    4096    49.9%   (41.7%)
    8192    8.2%    (8.2%)
100000  mean = 48141.4   max = 177768
    32  100%    (0.1%)
    128 99.9%   (0.6%)
    256 99.3%   (1.7%)
    512 97.6%   (4.6%)
    1024    93% (17.4%)
    2048    75.6%   (33.7%)
    4096    41.9%   (37%)
    8192    4.9%    (4.9%)
```

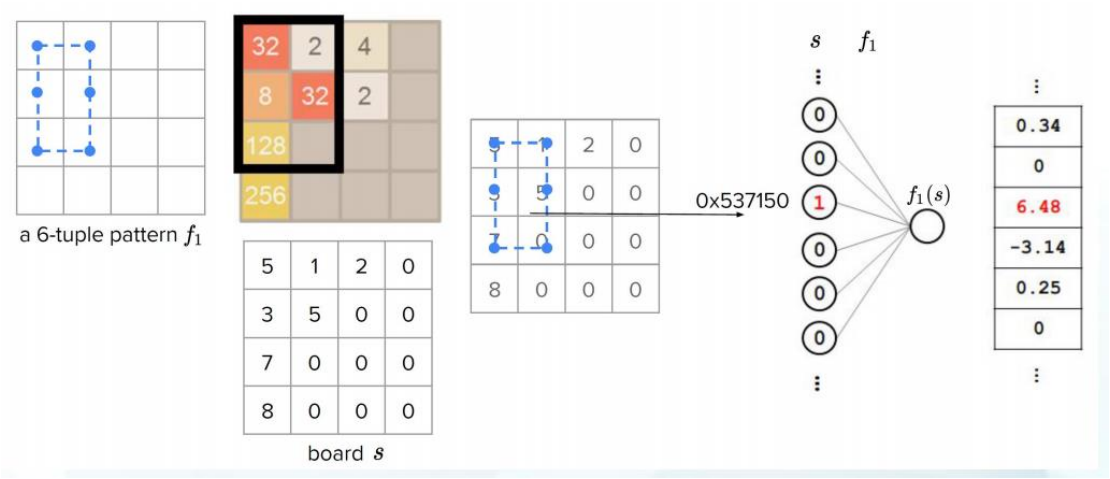
10 萬筆後突然又下降了 qq

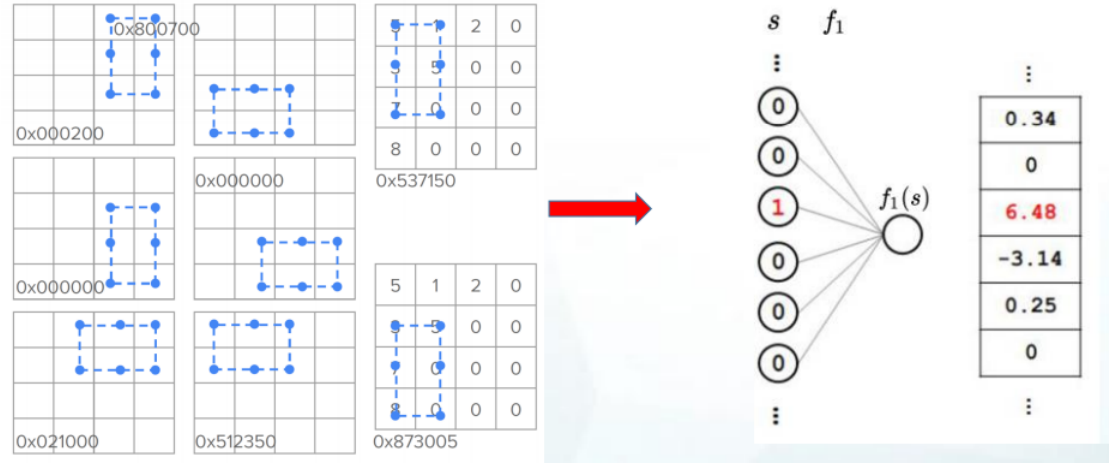
- Describe the implementation and the usage of n-tuple network.

首先先設定 tuple 的 index

```
// initialize the features
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 5 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 9 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));
tdl.add_feature(new pattern({ 4, 5, 6, 8, 9, 10 }));
```

然後就可以用這些 tuple(feature)來表示盤面(利用這些 feature 旋轉、翻轉等操作後蓋在盤面能得到不同的 value，藉由不同的 value 組合代表不同的盤，這樣就能使用更少的空間代表不同的盤面 (2^16)^16 -> (2^16)^n*8，n 為 tuple 數)





- Explain the mechanism of TD(0).

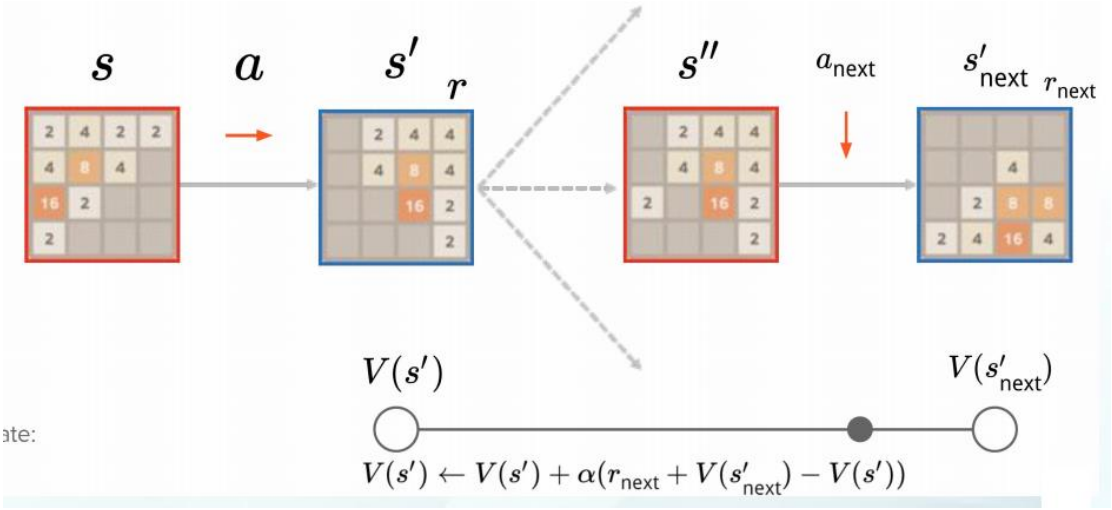
$$V(S_t) \Leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

利用觀測到的 reward R_{t+1} 和估計的 $V(S_{t+1})$對 $V(S_t)$進行更新

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

Exact 最開始是 0(因為最終目標是會更新到 0)，藉由 move.value() - move.reward()得到下一個 t step 的 value，所以就可以比對估計出來的與實際值的差異(error)，然後拿去更新 weight、exact。

```
float exact = 0;
for (; path.size(); path.pop_back()) {
    state& move = path.back();
    float error = exact - (move.value() - move.reward());
    exact = move.reward() + update(move.after_state(), alpha * error);
}
```



function LEARN EVALUATION(s, a, r, s', s'')

$$a_{next} \leftarrow \underset{a' \in A(s'')}{\text{argmax}}\, EVALUATE(s'', a')$$

$$s'_{next}, r_{next} \leftarrow COMPUTE\ \ AFTERSTATE(s'', a_{next})$$

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

- Explain the action selection of V(after-state) in a diagram.
  計算該盤面(b)做完動作(上下左右)後的 value，選擇做完動作後 value 最大的動作。

```
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)) {
            move->set_value(move->reward() + estimate(move->after_state()));
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

其中 after state 的 value 是根據下面公式所算

function EVALUATE(s, a)

$s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)

return $r + V(s')$

- Explain the TD-backup diagram of V(state).
  與 after state 差不多，不過初始的 exact 不是 0，因為最終狀態的 value 是 0，而最終狀態的 before state 會差了一個 reward(差一個 action)，所以我把最後一個 reward 的值加了回去當作初始值。

```
path.pop_back(); // terminal
state& move = path.back();
float exact = 0 + move.reward(); // s''
//float exact = 0;
for (; path.size(); path.pop_back()) {
    state& move = path.back();
    float error = exact - (move.value() - move.reward());
    exact = move.reward() + update(move.before_state(), alpha * error);
}
```

function LEARN EVALUATION(s, a, r, s', s'')

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

- Explain the action selection of V(state) in a diagram.
  與 after state 差不多，主要是差在 value 的計算方式

function EVALUATE(s, a)

$s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)

$S'' \leftarrow$ ALL POSSIBLE NEXT STATES(s')

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

若該盤面有空格就分別補上 2 與 4，在計算更新後的盤面的 value 分別乘上會出現的機率，在乘上(1/空格數)就可以得到 Σ P(s, a, s'')V(s'')

```
float value = 0;
int space_num = 0;
for(int i = 0; i < 16; i +=1)
{
    if (move->after_state().at(i) == 0)
    {
        // appear 2, 90%
        move->after_state().set(i, 2);
        value += 0.9 * estimate(move->after_state());
        // appear 4, 10%
        move->after_state().set(i, 4);
        value += 0.1 * estimate(move->after_state());

        move->after_state().set(i, 0);
        space_num += 1;
    }
}
value /= space_num;
move->set_value(move->reward() + value);
```

- Describe your implementation in detail.
  先決定 feature(tuple)

```
// initialize the features
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 5 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 9 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));
tdl.add_feature(new pattern({ 4, 5, 6, 8, 9, 10 }));
```

然後決定最佳動作，把這個 state 存下來(更新前後的盤、盤更新前後的 value、更新的動作、此動作的 reward)

```
while (true) {
    debug << "state" << std::endl << b;
    state best = tdl.select_best_move(b);
    path.push_back(best);
```

然後看這個盤面有沒有 terminal，若沒有就更新盤面，然後對更新後的盤面再做 popup，得到下一輪的 before state

```
if (best.is_valid()) {
    debug << "best " << best;
    score += best.reward();
    b = best.after_state();
    b.popup();
```

最後根據 path 更新 weight

```
tdl.update_episode(path, alpha);
```

- Other discussions or improvements.
  我的理解是 state 版本的 update_episode()裡面的 exact 更新中 move.after_state()其實應該要是 move.before_state()

```
exact = move.reward() + update(move.after_state(), alpha * error);
```

因為依照 pdf 中應該是從 s''跳到 s，也就是兩個都是 before state，可是如果是用 move.before_state()，performed 就會很爛，甚至跑了 10000 episodes 連 1024 都沒有出現，不太懂會甚麼是用 move.after_state()