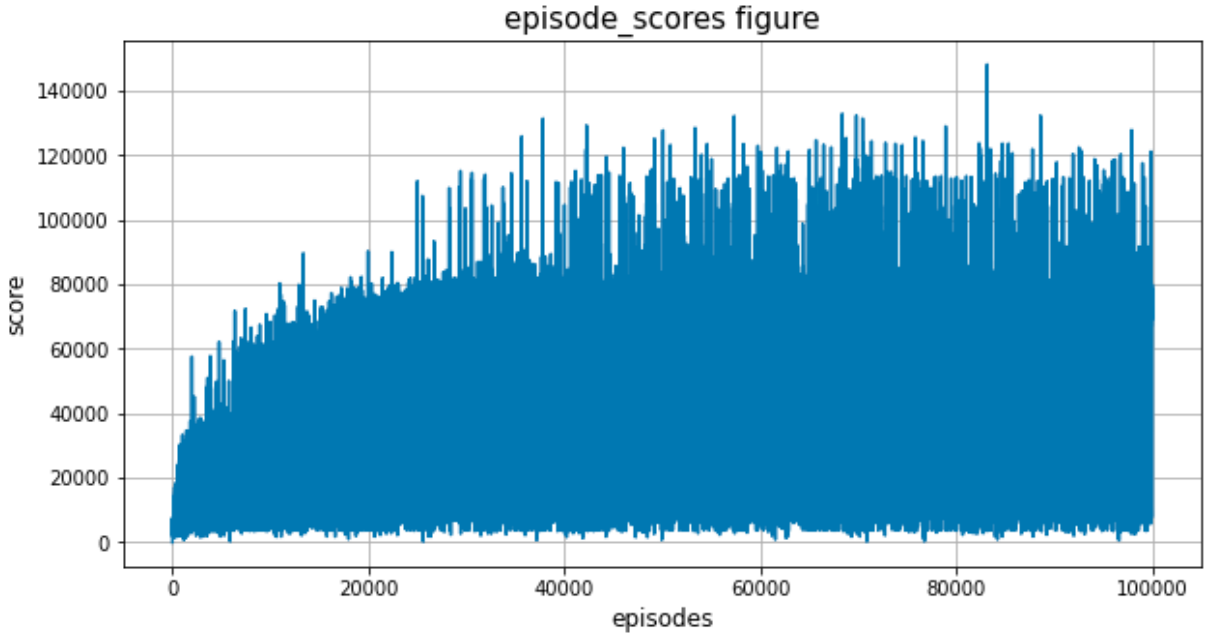
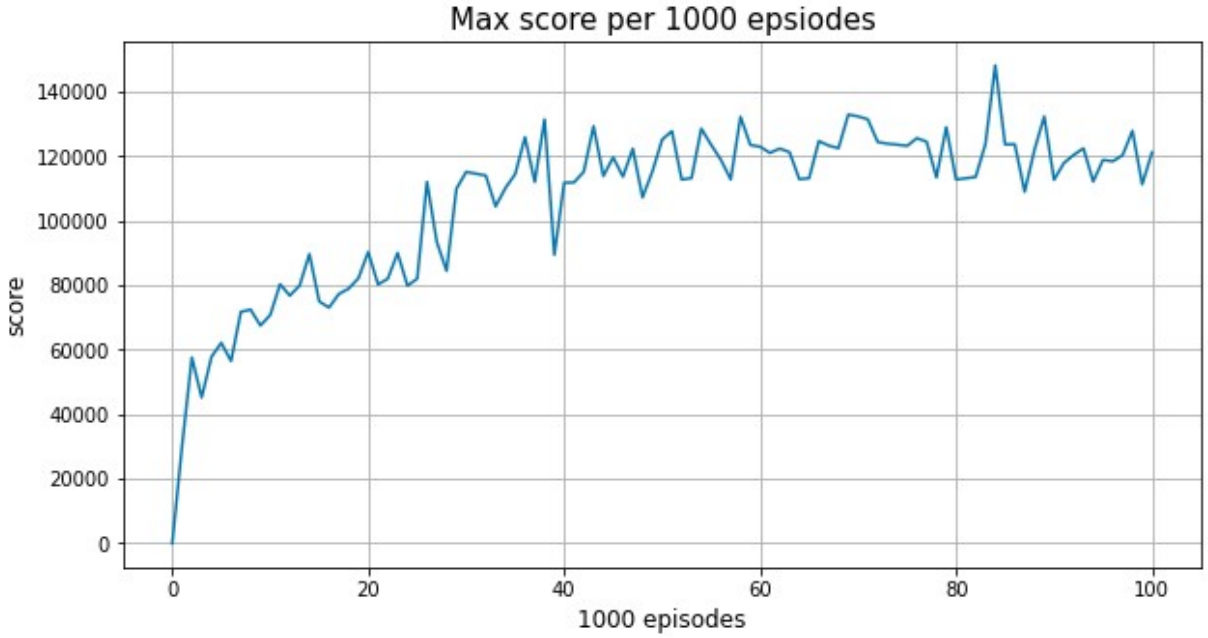
DLP Lab2 Temporal Difference Learning

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



（圖 1）每個 episode 對應的遊戲分數



（圖 2）每 1000 場 episodes 中分數最高的分數

由上面兩張圖可看出，愈後面的 episode 分數越高，也代表 model 訓練的愈來愈好。

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

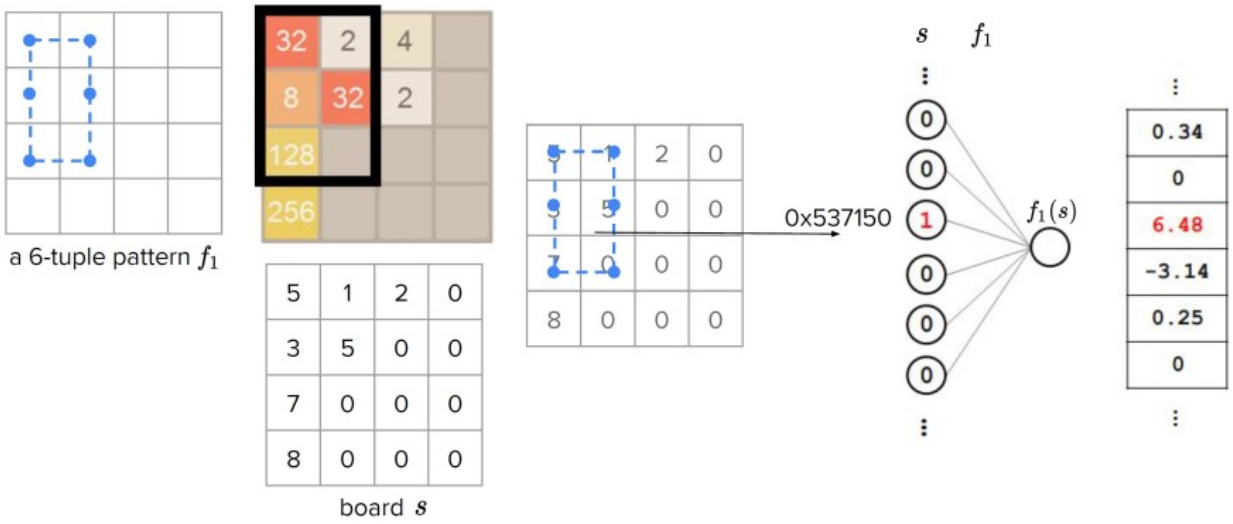
   n-tuple network (a.k.a. RAM-based neural network) is a type of artificial neural network.

   - A large number of input nodes.
   - Input values are either 1 or 0.
   - Input is a sparse vector.

- No hidden layers.
- Only 1 output node.

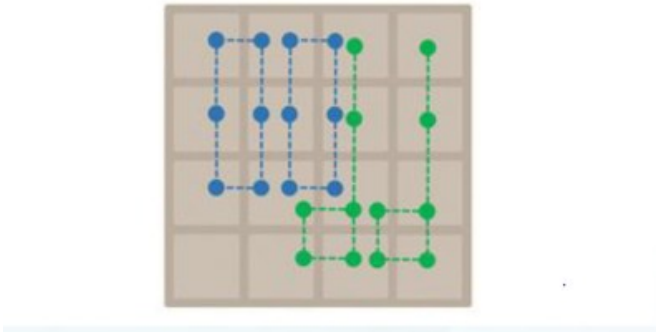
它是用盤面中的一些特徵(feature)來代表當前狀態，不直接在表中查出當前盤面多少分，而是查出每個特徵多少分，再把這些特徵加起來代表整個盤面的分數。這樣的好處是不必將所有盤面資訊都做計算，可以提高效率並節省記憶體的使用率。

下圖為單一個 6-tuple 的使用實例:



也可以設計多個 n-tuple 來計算整個盤面的分數:

$$V(s) = f_1(s) + f_2(s) + f_3(s) + f_4(s)$$



3. Explain the mechanism of TD(0).

TD 方法只需要等到下一個時刻。在 t+1 時刻，TD 方法立刻形成一個目標，並利用觀測到的 reward R_{t+1} 以及估計的 $V(S_{t+1})$做出有效的更新。最簡單的 TD 方法的更新方式如下:

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

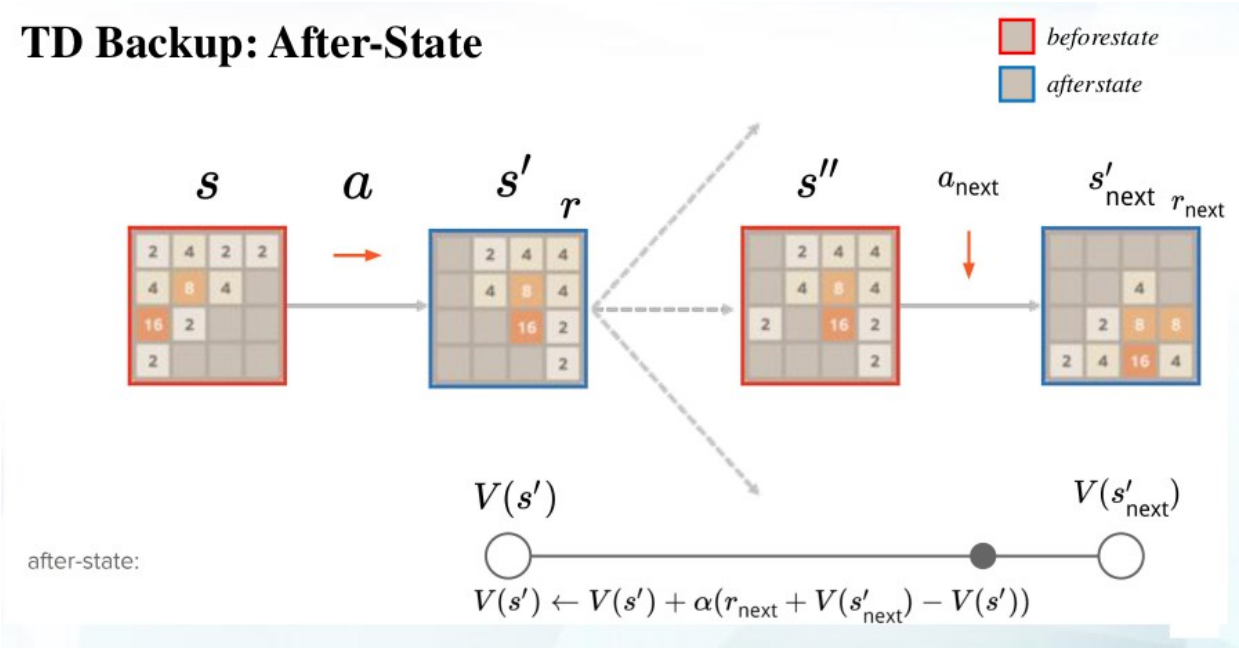
TD 方法一旦轉 $R_{t+1} + \gamma V(S_{t+1})$ 移到 S_{t+1} 並且得到了 reward R_{t+1} 就立刻進行更新。實際上，TD 方法更新的目標是。 。 這種 TD 方法被稱為 one-step TD 或 TD(0)。它是 TD(λ)和 n-step TD 的一種特殊情形。

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
    $S \leftarrow S'$
  until S is terminal

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



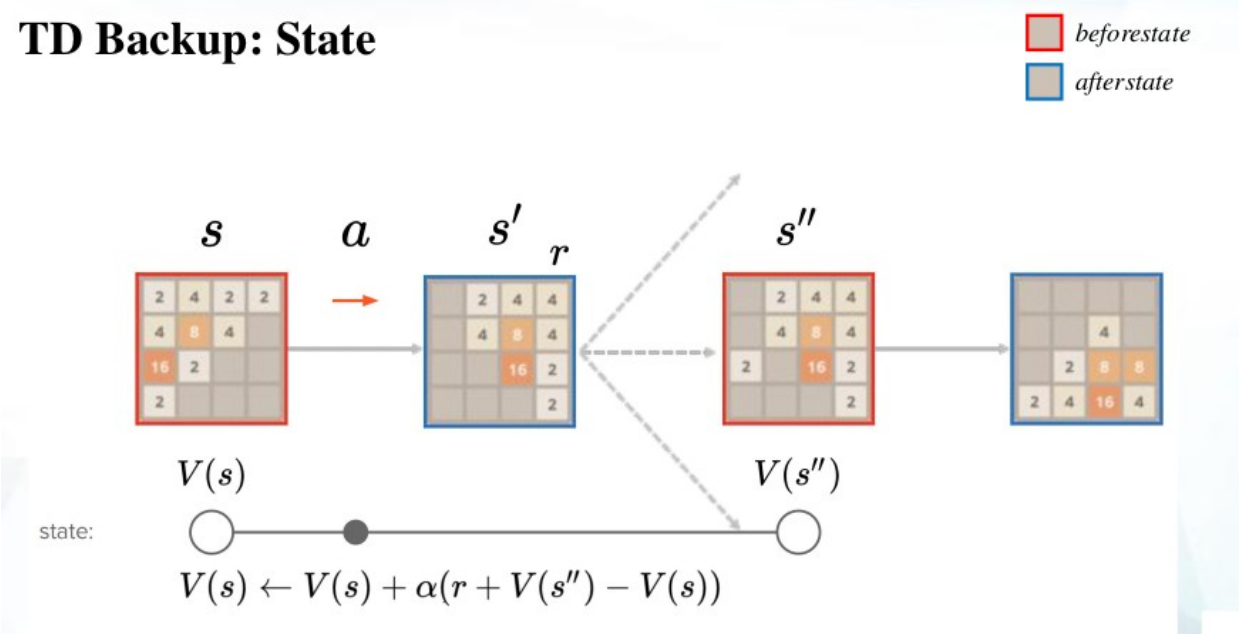
由上圖可看出，TD-backup(after-state)計算 value 時考慮的 board 是做完 action 後的 board，也就是圖中用藍色邊框起來的 S'和 S'$_{next}$，而兩個 board 之間的差異是，S'$_{next}$是 S'隨機在 board 中 popup 出一個 2 或 4，然後經由 $V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$公式更新舊 value，並找出最佳的 action 執行後的結果。

5. Explain the action selection of V(after-state) in a diagram.

function EVALUATE(s, a)
  $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
  return $r + V(s')$

after-state 選擇最佳動作的作法是，將 popup 後的 board 上下左右四個動作先各做一次，並得出個別對應的 reward 和做完 action 後的 board，然後計算 board 的 value 並加上對應的 reward，看哪個動作做完後此值最大，就選擇使用那個動作。

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



由上圖可看出，TD-backup(state)計算 value 時考慮的 board 是尚未做 action 的 board，也就是圖中用紅色邊框起來的 S 和 S''，而兩個 board 之間的差異是，S'' 是 S 經由 $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$ 公式更新舊 value，找出最佳 action 並執行，然後隨機在 board 中 popup 出 2 或 4 的結果。

7. Explain the action selection of V(state) in a diagram.

function EVALUATE(s, a)

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

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

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

state 選擇最佳動作的作法是，將 board 上下左右四個動作先各做一次，得到個別對應的 reward，然後去計算新的 board 的 value，最後將兩者相加，看做哪個動作後這個值最大，就執行那個動作。這邊計算 value 的方式是，加總所有可能出現的新 board 的 value，也就是在每個空位都 popup 一次 2 和 4 並計算 board 的 value 並加總。

8. Describe your implementation in detail.

為了將 after-state 改成 state，原本的 code 裡有 5 個 TODO 要做，這邊有三個 TODO 並未做更改，因為 after-state 和 state 都需要使用到這三個 function。

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

```
virtual float update(const board& b, float u) {
    // TODO
    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;
}
```

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

為了能加總所有可能出現的新 board 的 value，select_best_move 我這邊做的更動是，先找出空位以及空位數，並將所有空位都 popup 一次 2 和 4，並計算當下 board 的 value，且逐一加總到 esti_value，以利選出擁有最高 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)) {
            // TODO
            int esti_value = 0;
            int num_empty = move->after_state().popup(-1, 0);
            for (int i = 0; i < num_empty; i++){
                board tmp1 = move->after_state();
                board tmp2 = move->after_state();
                tmp1.popup(1, i);
                tmp2.popup(2, i);
                // info<<"[pop]"<<tmp2<<std::endl;
                esti_value += (1.0/num_empty) * 0.9 * estimate(tmp1) + (1.0/num_empty) * 0.1 * estimate(tmp2);
            }
            move->set_value(move->reward() + esti_value);
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

此外，為了能在特定位置 popup 2 或 4，我也對 popup 這個 function 做了一些更動，當 k 給 0 時就是隨機在空位 popup，k 給-1 時則會回傳空格數(num)，而 k 給 1 或 2 代表會在某個特定 space[n]這個位置 popup 2 或 4。

```
int popup(int k, int n) {
    int space[16], num = 0;
    for (int i = 0; i < 16; i++)
        if (at(i) == 0) {
            space[num++] = i;
        }
    if (num){
        if (k == 0){
            set(space[rand() % num], rand() % 10 ? 1 : 2);
        }else if (k == -1){
            ;
        }else
            set(space[n], k);
    }
    return num;
}
```

update_episode 這邊則是先讀入 S 和 S’，並按照公式 $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$ 計算 TD error，然後 update 成新的 value 再帶入下一次做計算。

```
void update_episode(std::vector<board>& path_s, std::vector<state>& path_ss, float alpha = 0.1) const {
    // TODO
    float exact = 0;
    float error = 0;
    for (path_s.pop_back() /* terminal state */; path_s.size(); path_s.pop_back()) {
        path_ss.pop_back();
        board& before = path_s.back();
        state& move = path_ss.back();
        error = move.reward() + exact - estimate(before);
        debug << "update error = " << error << " for after state" << std::endl << move.after_state();
        exact = update(before, alpha * error);
    }
}
```

9. Other discussions or improvements.

多增加一些 3-tuple 和 4-tuple 會發現 2048 的勝率比起原本預設只有 4 個 6-tuple 的勝率還高出許多，因為 2048 要勝出的很重要關鍵是，要針對某個角落攻擊，因此我增加的 tuple 基本上都針對左上角那個角落。

```
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 5 }));
tdl.add_feature(new pattern({ 0, 1, 4, 5, 8, 12 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));
tdl.add_feature(new pattern({ 0, 1, 4, 5, 8, 9 }));
tdl.add_feature(new pattern({ 0, 1, 4, 5 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6 }));
tdl.add_feature(new pattern({ 4, 5, 8, 9 }));
tdl.add_feature(new pattern({ 5, 6, 9, 10 }));
tdl.add_feature(new pattern({ 0, 1, 2, 3 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4 }));
tdl.add_feature(new pattern({ 0, 1, 2, 5 }));
tdl.add_feature(new pattern({ 0, 1, 4, 8 }));
tdl.add_feature(new pattern({ 0, 4, 5, 8 }));
tdl.add_feature(new pattern({ 0, 4, 7, 10 }));
tdl.add_feature(new pattern({ 0, 2, 5, 8 }));
tdl.add_feature(new pattern({ 0, 1, 5, 6 }));
tdl.add_feature(new pattern({ 0, 4, 5, 9 }));
tdl.add_feature(new pattern({ 0, 5, 10, 15 }));
tdl.add_feature(new pattern({ 0, 1, 4 }));
tdl.add_feature(new pattern({ 0, 1, 2 }));
tdl.add_feature(new pattern({ 0, 4, 8 }));
tdl.add_feature(new pattern({ 0, 5, 10 }));
tdl.add_feature(new pattern({ 2, 5, 8 }));
tdl.add_feature(new pattern({ 1, 4, 5 }));
tdl.add_feature(new pattern({ 0, 1, 6 }));
tdl.add_feature(new pattern({ 0, 4, 9 }));
```

將 learning rate (alpha) 調低也會讓 model 訓練的更好，勝率也會提高，但若調的太低，則會無法在 10 萬次內收斂出最佳結果，因此 learning rate 和 total episodes 需要做搭配。

```
// set the learning parameters
float alpha = 0.08;
size_t total = 100000;
```