

# 深度學習 HW2

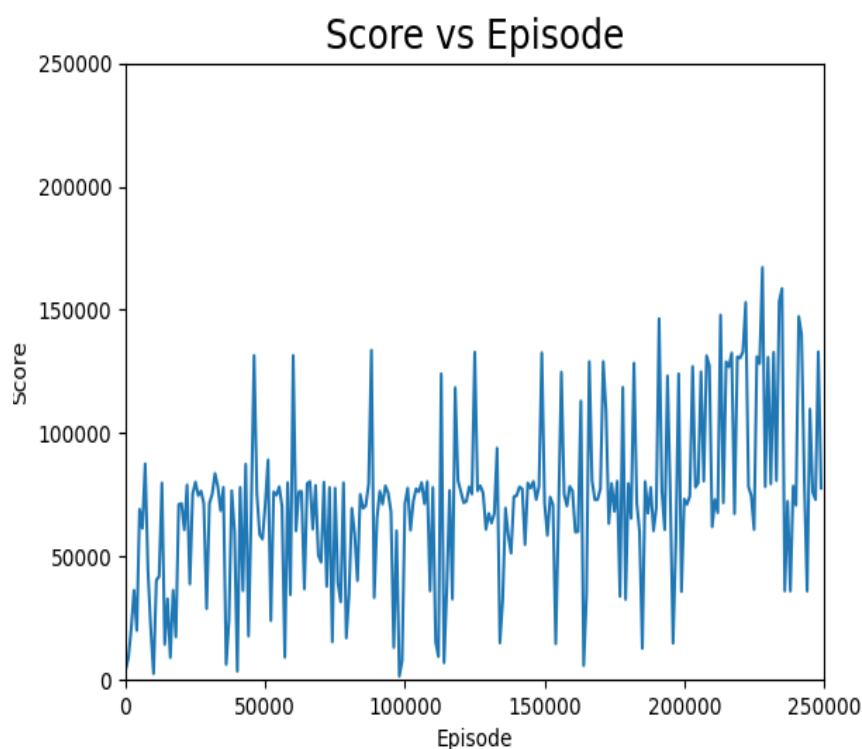
機器人學程 李啟安 3010605015

tags: 深度學習

Q1. A plot shows episode scores of at least 100,000 training episodes

```
8000    mean = 93757.7    max = 176412
512      100%    (0.3%)
1024     99.7%    (1.1%)
2048     98.6%    (6.6%)
4096     92%      (53.1%)
8192     38.9%    (38.9%)
```

- In episodes 208000, we can get the best 2048 score for 98.6%



- Score plot for 250000 episodes (trained for 18 hours)
  - Change the learning\_rate from 0.1 -> 0.01 after 200000 episodes

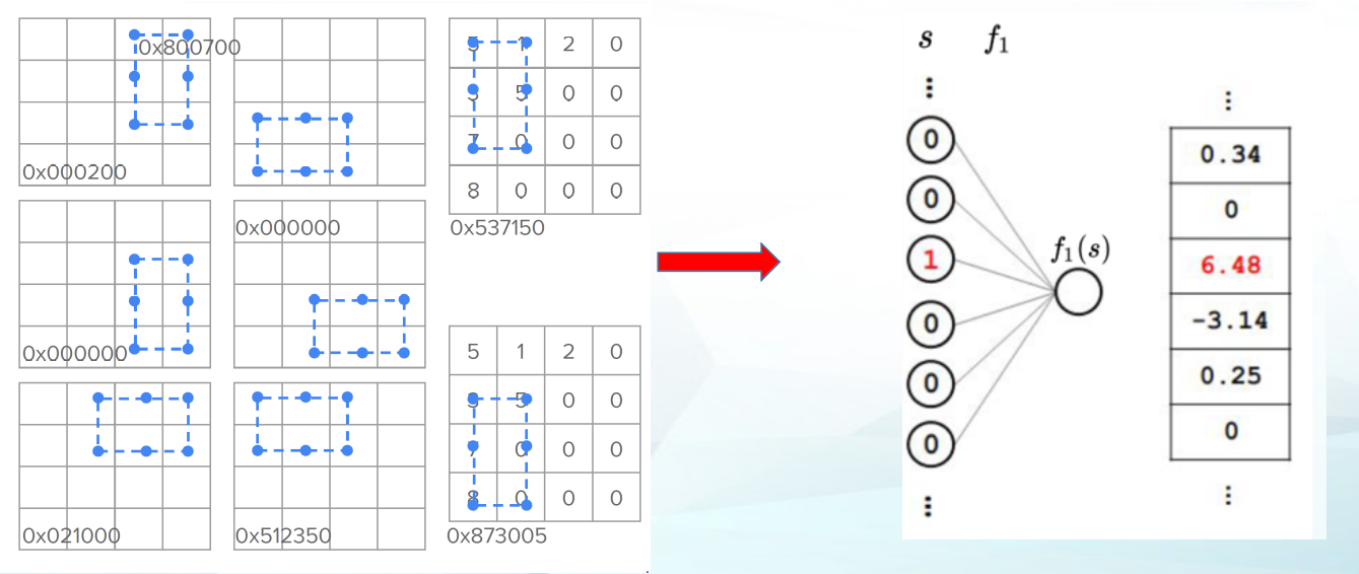
Q2. Describe the implementation and the usage of n-tuple network

Every single position can have the probability to be

$[null, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, \dots]$  . Total will be at least  $12^{16}$  combinations. It is impossible to store all these possibilities in computer.

Compared to recording all the state of boards, **n-tuple network can use multiple small tuples for caculation** because a tuple can present the characteristic of board. It is also easy for caculations and saves

a lot of spaces.



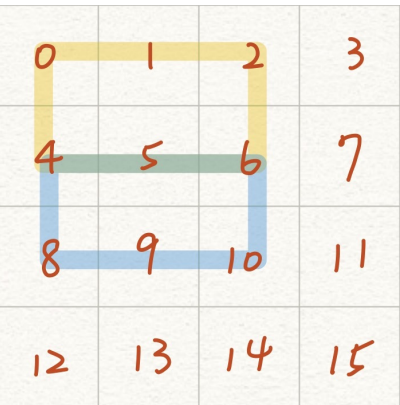
可以看到我們宣告了一個 6-tuple 然後根據鏡像以及旋轉，可以得到八種不同的結果。六個位置可以對應到一組 index，然後可以從 network 中找到對應的 value，例如：**fig.3** 右側的網路就是以第三種版面為例子，所以可以看到第三種版面為 1，其餘都是 0，我們即可拿到第三種版面的權重，相乘就可以得到估計值，因此八種版面估計值加總，就可以成為此版面的價值了！

我們可以在多取一點 feature，助教的 sample code 有四種，那我又自己加上三種，總共有七種 feature。

fig.3 新增 窗形 2 x 3 feature

```
// ---  
// ---  
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));  
tdl.add_feature(new pattern({ 4, 5, 6, 8, 9, 10 }));
```

fig.4 黃色與藍色分別代表窗形 feature 的兩種情況。



### Q3. Explain the mechanism of TD(0).

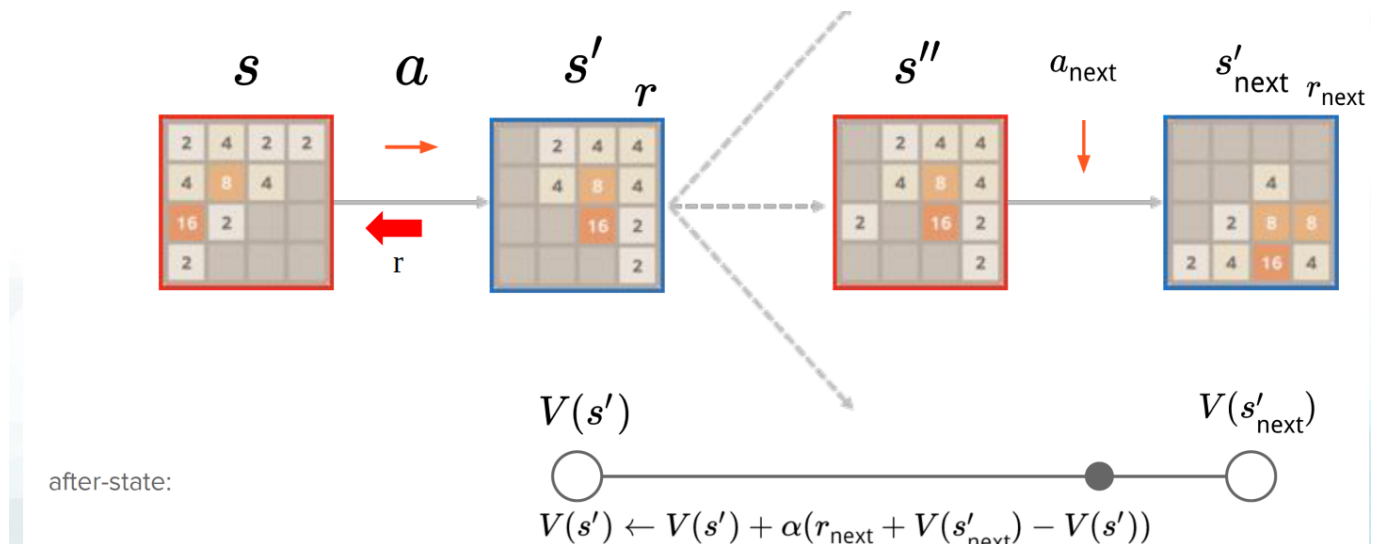
TD(0) is special case of TD( $\lambda$ ), it will look one step ahead. Unlike the MC, we will get the immediate reward plus the discount estimate value of 1 step ahead.

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

In the project, the discount factor is set to 1, We will keep update our value function in each state. The transition is  $s \rightarrow s''$ . We wait until arrive the next time step  $s''$  and then **combine immediate reward  $r$  in state  $s$  and prediction  $V(s'')$  to update our  $V(s)$** .

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

Q4. Explain the TD-backup diagram of  $V(\text{after-state})$ .



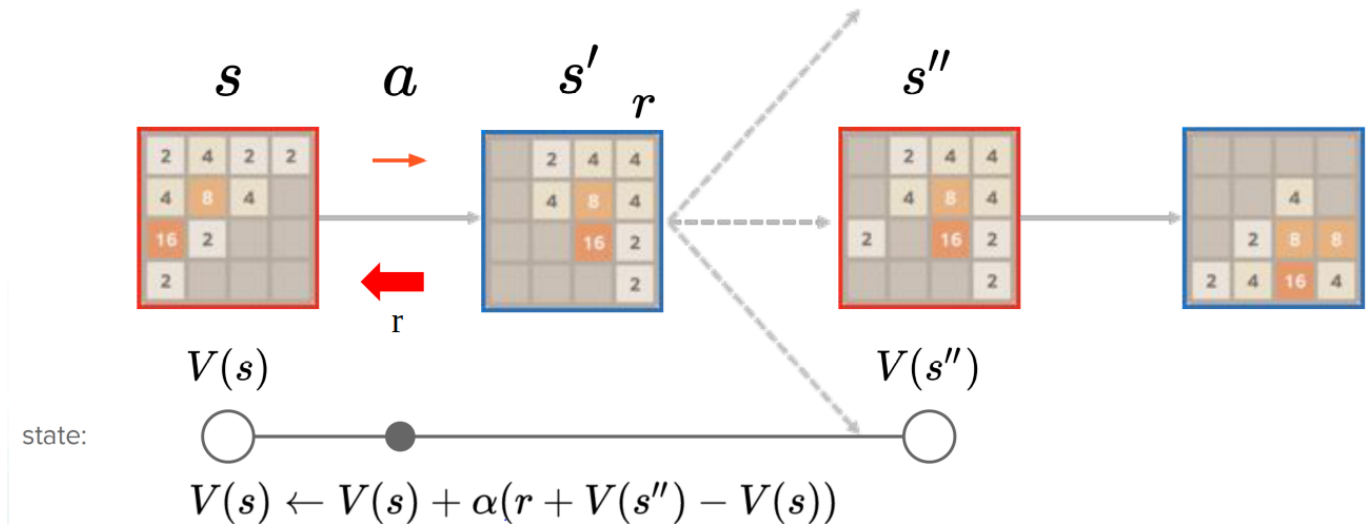
The after-state method only cares about the after state. We want to update  $V(s')$  by selecting  $a_{next}$  and choosing TD target as  $r_{next} + V(s'_{next})$ . Then we can get TD error by  $(r_{next} + V(s'_{next}) - V(s'))$ . Finally update  $V(s')$  by multiplying the learning rate  $\alpha$ .

Q5. Explain the action selection of  $V(\text{after-state})$  in a diagram.

To select the best action, it will calculate all possible actions and select the action that can go to state with highest value function.

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

Q6. Explain the TD-backup diagram of  $V(\text{state})$ .



The before-state method cares about the state "after the transition" or "before the action". We want to update  $V(s)$  by selecting  $a$  and choosing TD target as  $r + V(s'')$ . Then we can get TD error by  $(r + V(s'') - V(s))$ . Finally update  $V(s)$  by multiplying the learning\_rate  $\alpha$ .

## Q7. Explain the action selection of $V(\text{state})$ in a diagram.

Action selection will take all possible action (up, down, left, right) and go through all the possible transition to next state  $s''$ . Then we can choose the best action with the highest state-action value function.

```
function EVALUATE( $s, a$ )
 $s', r \leftarrow$  COMPUTE AFTERSTATE( $s, a$ )
 $S'' \leftarrow$  ALL POSSIBLE NEXT STATES( $s'$ )
return  $r + \sum_{s'' \in S''} P(s, a, s'') V(s'')$ 
```

## Q8. Describe your implementation in detail.

In this lab, I change the following function :

- select\_best\_move :
  - To select best action, build a for loop that will take all actions and build an inner for loop to calculate the expectation of choosing that action
- 1. define a vector to store the empty position on board

```
// check all possible s'' will occur (empty position)
std::vector<int> empty_position;

// record all empty position
for (int i = 0; i < 16; i++){
    if (move->after_state().at(i) == 0){
        empty_position.push_back(i);
    }
}
```

2. Run a for loop to run through all possible action. **Choose one possible action and run through all possible transition to predict the expectation of choosing that action.**

(Remember: the probability of popup 4-tile is 10% but 2-tile is 90%)

```
for (int i =0; i < empty_count ; i++){

    // initialize the future state
    S_future = move->after_state();
    // if popup 2 in empty position
    S_future.set(empty_position[i],1);
    // the chance is 0.9/empty_position_count
    val += estimate(S_future) * 0.9 / empty_count;

    // initialize the future state
    S_future = move->after_state();
    // if popup 4 in empty position
    S_future.set(empty_position[i],2);
    // the chance is 0.1/empty_position_count
    val += estimate(S_future) * 0.1 / empty_count;
}
```

3. Return the best action with the highest value function

```
// value is immediate reward and the expectation of all possible value in next state
move->set_value(move->reward() + val);
```

- update\_episode :

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

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

```
path.pop_back();
state& move_final = path.back();
// Terminal state
// v(s) <- v(s) + alpha(r + V(s'') - V(s))
float previous_state = 0 - estimate(move_final.before_state());

// loop through all path
for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
    state& move = path.back();
    float error = previous_state - estimate(move.before_state()) + move.reward();
    previous_state = update(move.before_state(), alpha * error);
}
```

1. Caculate the value function of terminal state
2. Loop through the path by updating the value function
3. Then you can get the new value function of the whole path!

- main :
  - 跑程式
  - 自己加上新的 feature

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

// ---
// -
tdl.add_feature(new pattern({ 0, 1, 2, 5}));
tdl.add_feature(new pattern({ 4, 5, 6, 9}));

// --
// --
tdl.add_feature(new pattern({ 0, 1, 4, 5}));
tdl.add_feature(new pattern({4, 5, 8, 9}));
tdl.add_feature(new pattern({1, 2, 5, 6}));
tdl.add_feature(new pattern({5, 6, 9, 10}));

// -
// ---
tdl.add_feature(new pattern({0, 4, 5, 6}));
tdl.add_feature(new pattern({1, 5, 6, 7}));
tdl.add_feature(new pattern({4, 8, 9, 10}));
tdl.add_feature(new pattern({5, 9, 10, 11}));

//--
// --
tdl.add_feature(new pattern({0, 1, 5, 6}));
tdl.add_feature(new pattern({1, 2, 6, 7}));
tdl.add_feature(new pattern({4, 5, 9, 10}));
tdl.add_feature(new pattern({5, 6, 10, 11}));

// ----
// --
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 }));
```

## Q9. Other dicussions or improvement

1. We can adjust the learning rate after 100000 episodes, turn 0.1 to 0.01 can get a better performance
2. To draw the score and episode graph, define a function to automatically save score in txt file so we can plot from python or matlab!

```
void write_score(size_t episode, int score){

    std::ofstream file;
    file.open("./record_final.txt", std::ios::app);
    file << "ep: " << episode << " , score: " << score << std::endl;

}
```