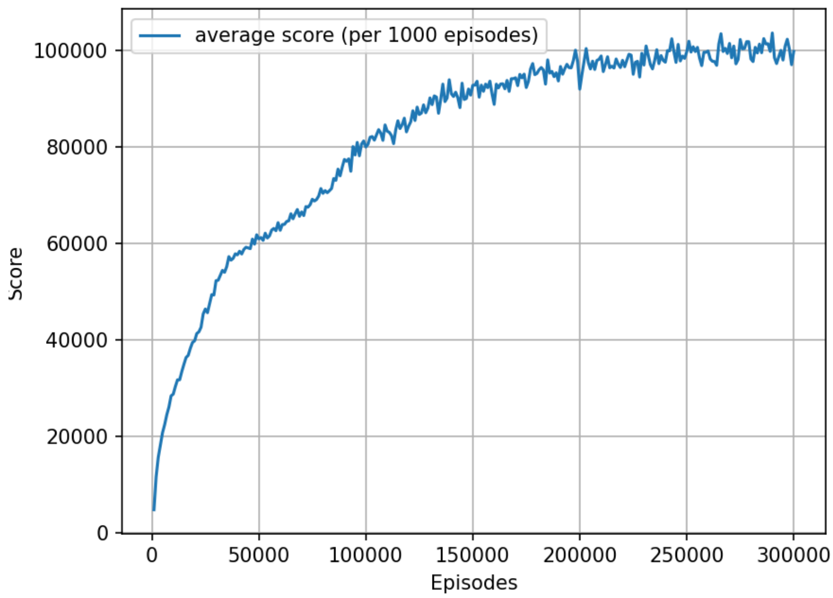
# Lab2: 2048 Temporal Difference Learning

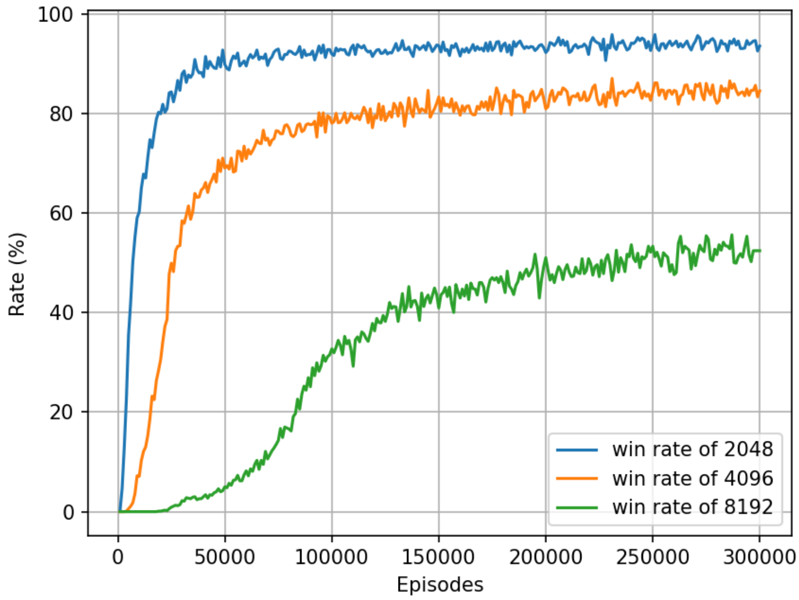
Wei-Yun Hsu
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University

## 1. Plotting

### A. Shows scores (mean) of at least 100k training episodes




### B. Shows win rate of 2048, 4096 and 8192




## C. 2048-tile should appear within 10k episodes

```
2000      mean = 11658.5   max = 33204
          64        100%     (0.1%)
          128       99.9%    (0.5%)
          256       99.4%    (7.6%)
          512       91.8%    (44.9%)
          1024      46.9%    (42.2%)
          2048      4.7%     (4.7%)
```

```
10000     mean = 28789.2   max = 80704
          128       100%     (0.3%)
          256       99.7%    (1.5%)
          512       98.2%    (6.1%)
          1024      92.1%    (31.9%)
          2048      60.2%    (53.1%)
          4096      7.1%     (7.1%)
```

## D. After 100k episodes

```
100000    mean = 80008.6   max = 223728
          128       100%     (0.1%)
          256       99.9%    (0.5%)
          512       99.4%    (2.3%)
          1024      97.1%    (5.2%)
          2048      91.9%    (13.9%)
          4096      78%      (47.8%)
          8192      30.2%    (30.1%)
          16384     0.1%     (0.1%)
```

## E. After 300k episodes

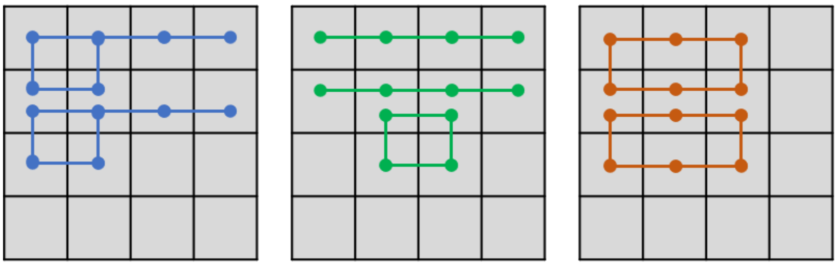
```
298000  mean = 100248   max = 177368
        128     100%    (0.1%)
        256     99.9%   (0.2%)
        512     99.7%   (1.6%)
        1024    98.1%   (3.5%)
        2048    94.6%   (9%)
        4096    85.6%   (34%)
        8192    51.6%   (51.6%)
299000  mean = 97067.2  max = 217112
        256     100%    (0.2%)
        512     99.8%   (2.5%)
        1024    97.3%   (4.8%)
        2048    92.5%   (9.2%)
        4096    83.3%   (33.1%)
        8192    50.2%   (50.1%)
        16384   0.1%    (0.1%)
300000  mean = 99809.8  max = 176904
        128     100%    (0.4%)
        256     99.6%   (0.2%)
        512     99.4%   (2.1%)
        1024    97.3%   (3.8%)
        2048    93.5%   (9%)
        4096    84.5%   (32.1%)
        8192    52.4%   (52.4%)
```

## 2. *n*-tuple network

We need to calculate the isomorphic patterns by board, and then add the corresponding weights together, which is expected value of the current board.

Why has 8 isomorphisms? Because board has 4 possible 90-degree rotations (0, 1, 2, 3), each of these rotations, there are two possible mirror reflections (horizontal and vertical).

The design of my n-tuple network are shown below: $4 \times$ 6-tuple + $3 \times$ 4-tuple.



## 3. Mechanism of TD(0)

TD(0) starts from the terminal state and move backwards to the initial state. For each episode, we will calculate the difference (error) between the immediate reward obtained after taking the last action and the estimated value for the previous state. Then, we will use this error to update the expected value of given state, after the series of action, we can get the TD target (called “exact” in my code). Taking the action mentioned above repeat to update the last move.

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {  // alpha is learning rate
    // TODO
    float exact = 0;
    // starting from the terminal state and moving backwards to the initial state
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = move.reward() + exact - estimate(move.before_state());
        debug << "update error = " << error << " for before state" << std::endl << move.before_state();
        exact = move.reward() + update(move.before_state(), alpha * error);
    }
}
```

# 4. My implement in detail

## A. Action selection

### A-1. State

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

When we want to select the best move and perform an action, the board will randomly popup a tile (2: 90%, 4: 10%) for the current empty cell. The mechanism of action selection will calculate the expected value of all possible actions and find out the highest value of them. It is more difficult than the action selection of after-state, because we don't know which one is the next before-state.

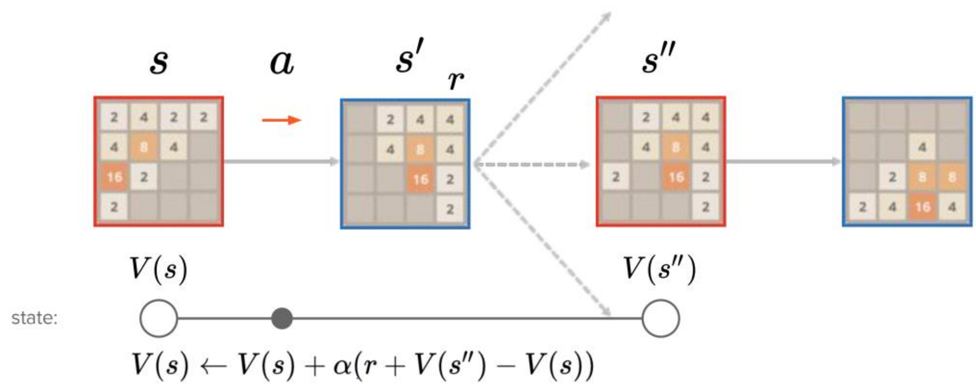
### A-2. After-state

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

This process is simple. When we perform an action, the board only need to select a move which has the highest expected value, that means the selected after state is the best successor of before-state.

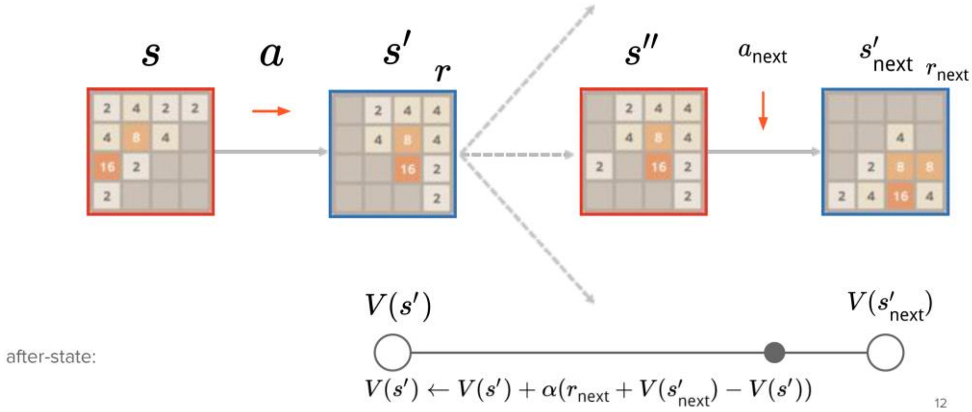
## B. TD-backup diagram

### B-1. State




State means that the action isn't executed, that is before-state. The *n*-tuple network stores the weight of the tuple in the before-state, and also updates the weight for the tuple in the before-state. The error is calculated by subtracting the estimated value of the current state from the estimated value of the next before-state, and adding the immediate reward obtained after taking the last action. Then, multiply the obtained value above by a learning rate.

**B-2. After-state**




After-state means that no tile has been popped up after the action is executed. The *n*-tuple network stores the weight of the tuple in the after-state, and also updates the weight for the tuple in the after-state. The error is calculated by subtracting the estimated value of the current after-state from the estimated value of the next after-state, and adding the reward obtained between the current after-state and the next after-state, and then multiply the obtained value above by a learning rate.