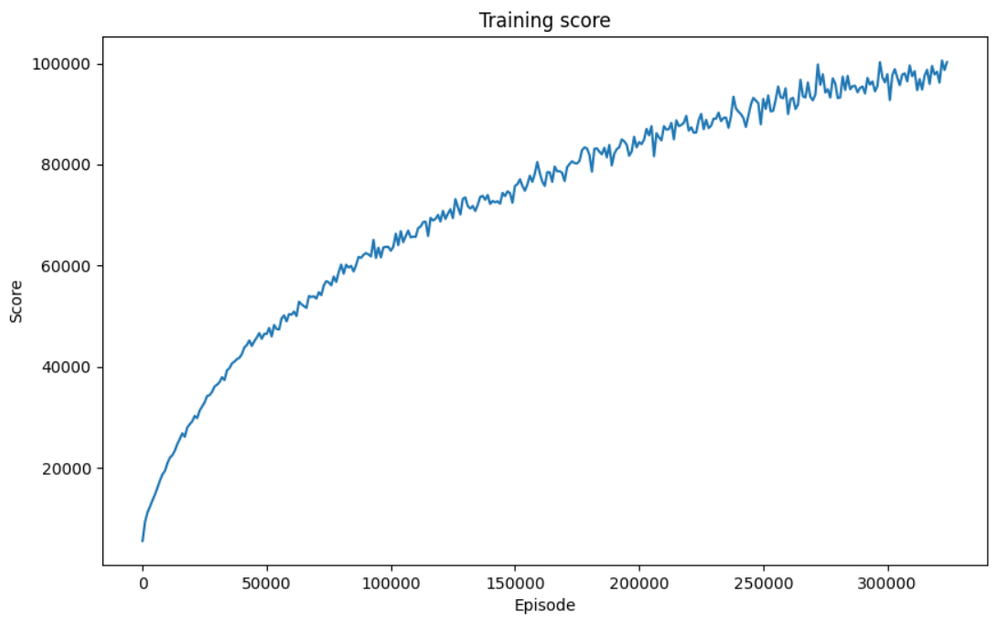
Deep Learning Lab 2

311552004 李明倫

A plot shows scores (mean) of at least 100k training episodes

我總共訓練了 325000 個 episodes，最佳的 2048 win rate 在第 323000 個 eposide 產生，mean score 為 100587， win rate 為 95.9%。

```
323000  mean = 100587   max = 277424
        256     100%    (0.4%)
        512     99.6%   (0.6%)
        1024    99%     (3.1%)
        2048    95.9%   (18.7%)
-- saved new wieght
        4096    77.2%   (29.5%)
        8192    47.7%   (47.4%)
        16384   0.3%    (0.3%)
```

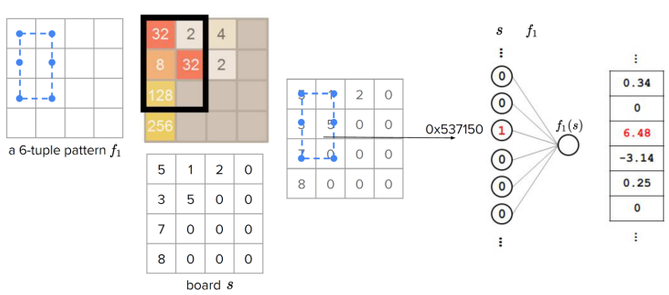


Describe the implementation and the usage of n-tuple network

Usage

Value function *V(s)* 幫助我們去評估一個 state (盤面)的 ”前途”，如果 value function 評估出來的值越大，代表目前的 state 越好，如果 value function 評估出來的值越小，也就代表目前的 state 相對來說沒有那麼好。

但若要完整的列出 2048 所有可能的 state，會需要 17^16 個空間去儲存每一個 state 的值，因此選擇採用 n-tuple network 的方式，有點類似對目前的盤面取各式各樣的 feature，然後讓這些 feature 的加總代表盤面好壞的一個分數。



Implementation

- Indexof

  透過這個 function，讓我們去拿到目前所有 n-tuple feature patterns 在盤面上所對應到的值。

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

- Estimate

  因為 2048 是全對稱的遊戲，所以要把每一個 feature 做 rotation 跟 reflection 去取得所有 isomorphism。接著就透過上面的 indexof 依序把每一個 feature 的值所對應到的 weight 拿出來做加總，代表目前這個盤面當前的分數。

```
virtual float estimate(const board& b) const {
    // TODO
    // 跑過某一個 feature 的所有 isomorphic，然後把每一個 isomorphic 的 weight (value) 加起來，代表這個 state (盤面) 的 value
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
      size_t index = indexof(isomorphic[i], b);
      value += operator[](index);
    }
    return value;
  }
```

- Update

  這個 update funciton 會在每個 episode 跑完要做 value function update 的時候用到，目的是為了調整上面每一個 feature 的值，讓 value function 在做每個 state 的評估分數上能夠更準確。

```
virtual float update(const board& b, float u) {
    // TODO
    // 更新某一個 feature 的每一個 isomorphic 的 weight (value) 值
    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;
  }
```

Explain the mechanism of TD(0)

TD(0) 也是最簡單的 temporal-difference learning algorithm，他會根據當前做完某個 action 拿到的 reward 加上下一個時間點的 estimated value 來更新當前時間點的 estimated value。

- TD target: $R_{t+1} + \gamma V(S_{t+1})$
- TD error: $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

- Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

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

TD target 是我們的目標，而 TD error 則是把 TD target 減掉當前這個 state 的 estimated value，就可知道我們離實際值差多遠，也就是我們算出來的誤差。在更新 value 的時候，我們會再乘上一個 learning rate 去做調整。

如此一來，當我們結束某個 episode 之後，我們就可以從最後一個時間點所拿到的 reward 往前更新到第一個 state，經過很多次的 episode 之後，我們 value function 評估出來的值也會越準確。

Describe your implementation in detail including action selection and TD-backup diagram.

- Feature selection

  除了一開始助教給的四個 features 之外，我有去參考文獻 “*Systematic selection of N-tuple networks with consideration of interinfluence for game 2048*” 裡面參考作者提出的一些 features，從裡面挑了另外四個 6-tuple 的 pattern 去做訓練。

```
// 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}));

  tdl.add_feature(new pattern({ 0, 1, 2, 5, 9, 10 }));
  tdl.add_feature(new pattern({ 0, 1, 5, 9, 13, 14 }));
  tdl.add_feature(new pattern({ 0, 1, 5, 8, 9, 13 }));
  tdl.add_feature(new pattern({ 4, 0, 1, 2, 6, 10 }));
```

- Action selection

  在這個部分，我們要做的事情就是從所有可能的 action 當中選一個最好的 action 出來 (up, down, left, right)。那這個 action 要怎麼挑，我們就利用我們的 value function，也就是去看我們當前這個 state，做完某個 action 之後可以得到的期望值，然後挑可以獲得最大期望值的那個 action 當作我們目前要選的 best action。

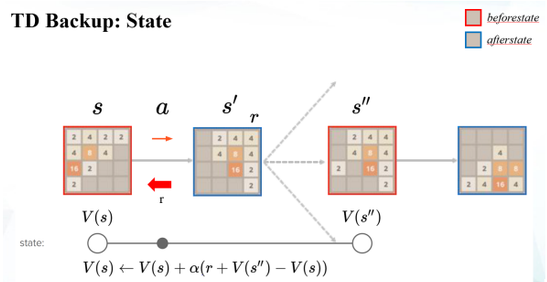
  我們會需要跑一個 for loop，跑過每一個可能的 action，在每一個 action 裡面，我們會先需要知道做完當前 action 的盤面長怎樣 (afterstate)，接著從 afterstate 裡面去計算可以獲得的期望值。

  因此我們會先需要找出在 afterstate 裡面有哪些格子是空的，並且透過 “90% → 2 tile, 10%→ 4 tile” 的機率，下去計算如果盤面 pop tile 出來情形的 estimated value，算出可以得到的期望值是多少，同時也把當前 state (beforestate) 的 estimated value 設進去，最後就挑可以獲得最大期望值的那個 action 作為我們當前的 action。

```
state select_best_move(const board& b) const {
    state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
    state* best = after;

    float best_total = -std::numeric_limits<float>::max();
    for (state* move = after; move != after + 4; move++) {
      if (move->assign(b)) {
        // TODO
        board afterstate = move->after_state();
        float value = move->reward();

        // 找現在剩餘的空間
        int space[16], num = 0;
        for (int i = 0; i < 16; i++) {
          if (afterstate.at(i) == 0) {
            space[num++] = i;
          }
        }

        // 計算所有可能的期望值 (所有可能的 next state)
        for (int i = 0; i < num; i++) {
          board* tmp_board = new board(afterstate);
          tmp_board->set(space[i], 1);
          value += 0.9 * estimate(*tmp_board) / num;
          tmp_board->set(space[i], 2);
          value += 0.1 * estimate(*tmp_board) / num;
          delete tmp_board;
        }

        move->set_value(estimate(move->before_state()));

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

- TD-backup

  在這個部分我們要做的就是去更新我們的 estimated value，因為是玩完一個 episode，所以我們可以從最後一個 state 往前更新回去。最後一個 state 得到的 reward 會是 0，因此我們就可以透過上面 TD learning 的方式，計算 state 之間的 TD error，然後乘上 learning rate alpha，去更新每個 state 的 estimated value。




```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float td_error = 0;
    float next_value = 0;
    for (int i = path.size() - 2; i >= 0; i--) {
      td_error = (path[i].reward() + next_value) - path[i].value();
      next_value = update(path[i].before_state(), alpha * td_error);
    }
}
```