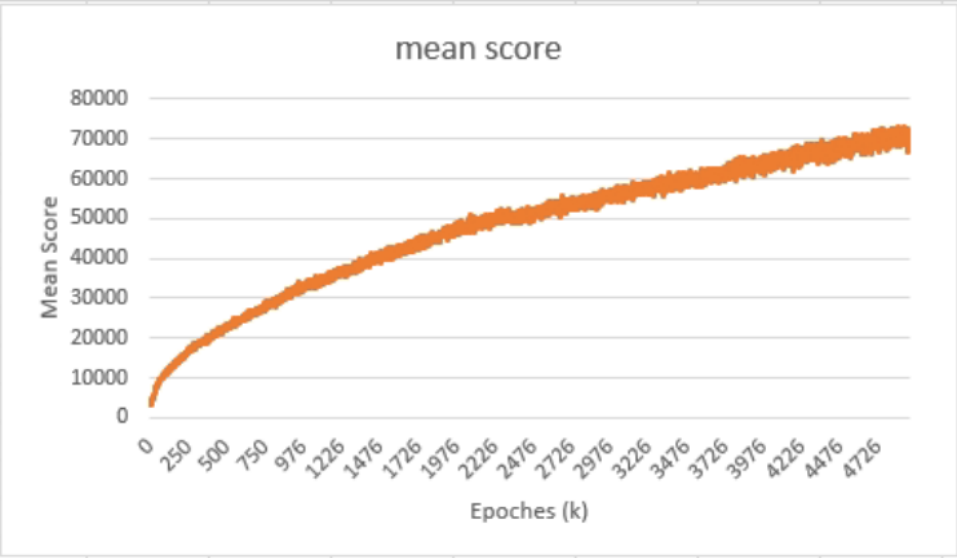
A plot shows episode scores of at least 100,000 training episodes (10%)



根據 4900k 次訓練結果，發現訓練還是有上升的空間，但是漸漸收斂。這裡顯示的是 mean score，意味著最大的分數能更高，但需要訓練的時間更久。當中我截取到最大分數位於 4854epoch 的 72788.8 有 89.8%為 2048 分。

Describe the implementation and the usage of n-tuple network. (10%)

N-Tuple Network 主要有三個大特點：

1. 提供大量的特徵
   N-Tuple Network 可以用一組 tuple 對應的旋轉、鏡面來取得不一樣的特徵，藉以最大化特徵的利用，從而減少直接從整個版面獲取特徵時的龐大數量。
2. 容易更新
   在更新權重時只需要根據 tuple 的對應權重做更新，無需更新其他權重，用一種的權重去表達更多的版面，學習上也更快。
3. 節省儲存空間
   在 2048 遊戲中每一格都有著不同數字的可能，在 16 格中的排列組合就更為龐大，要儲存這大量的特徵所需要的空間就會超出預期，假設每一格的數字可能有 12 種(由空格到 2048)，使用 4 個 6-tuple 其大小為 $4 \times 12^6 \times 4$ ，相對於儲存 16 格特徵(12^{16})大大節省了許多空間。

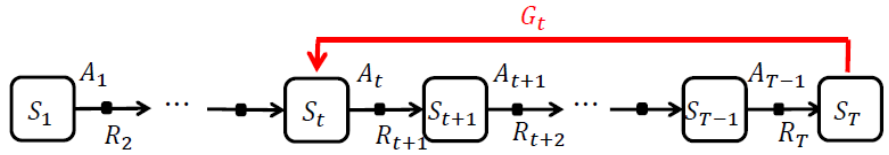
在這次實驗中，我們採用 4x6 tuple。

Explain the mechanism of TD(0). (5%)

在了解 TD(0)之前我們要先了解 Monte-Carlo (MC)和 Temporal Difference (TD) 兩種不同的強化學習

Monte-Carlo (MC):

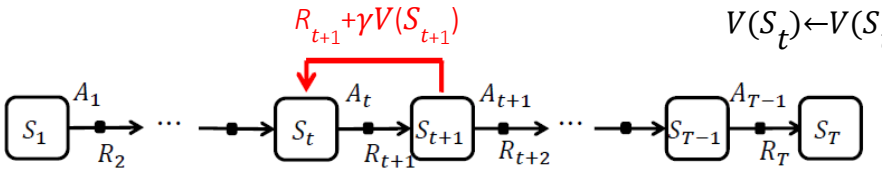
MC 是以實際的最終結果S_T得到的 G_T來更新之前的參數，其中 G_T為之前每一步 Reward 的總和，用實際得到的值用 backpropagation 的方式向前更新，從而讓機器學習最好的 Action



$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Temporal Difference (TD):

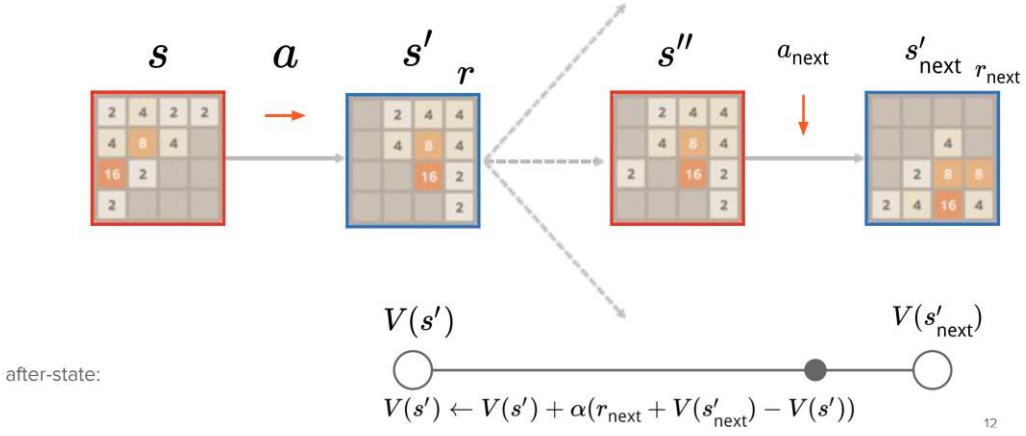
TD 是以估計形式去估算當前 State 在進行 Action 時，哪一個能賺取最大的 Reward，根據以往相同 State 的 Reward 來更新哪一個 Action 後的 Reward 較大，從而達到以每一步來更新的目的。



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

事實上，TD 還有一個參數 λ 通常會表示為 TD(λ)，其中 λ 可以理解成更新估算的步長，TD(1)其實就是 MC，只考慮最終實際得到的 Reward，而 TD(0)則是標準的 TD 只考慮下一步得到的 Reward。

Explain the TD-backup diagram of V(after-state). (5%)

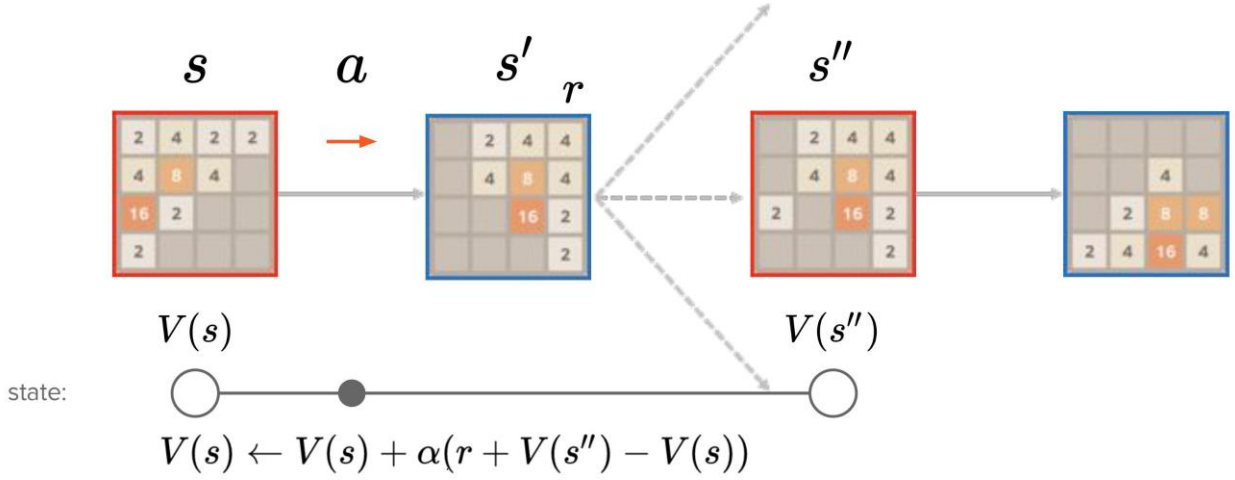


After-state 是根據 action 後的 state(S'_{next})的值V(S'_{next})和 reward 來更新之前的 state(S')的值V(S')，只考慮每個 action 後的 state。當中，α 可以理解成 learning rate。

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

根據論文和公式上的理解，after-state 會以 state(S')來決定最好的 action，並用V(S'_{next})和 reward 與V(S')的差來更新參數，當中 state(S'')和隨機出現的數字結合並反映在V(S'_{next})，以此來決定下一個 action。

Explain the TD-backup diagram of V(state). (5%)



State 是考慮了每一個 action 後的 reward 和隨機生成數字的 state(S'')的值V(S'')，來更新之前的 state(S)的值V(S)，這當中用 action 得到的 reward 和V(S'')與原來的V(S)的差來更新V(S)。

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

State 相對 after state 擁有較為獨立的生成隨機數考量，所以需要加入隨機數生成的機率(Policy)作為預測的根據，每一個 action 的預測值結合隨機生成的數字來選擇要進行哪一個 action，相對 after-state 得到的值會較為準確，但需要知道 policy 和會更費時間。

Describe your implementation in detail. (10%)

Indexof 輸入為 patt 和 board，主要是根據輸入的 pattern，計算該 pattern 在 board 上面的 index 為多少，以利後面的運算。

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

estimate 輸入為 board 是用來獲取當前 index 的值，根據 tuple 輸入的 board 並將之旋轉和水平翻轉得出的 8 個不同 isomorphic 作為 indexof 的輸入，來除得 8 組不同的 index，並依據 index 的位置獲取數值。

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

update 輸入為 board 和 u，這裡 u 為$\alpha(R_{t+1}+\gamma V(S_{t+1})-V(S_t))$，亦稱為 TD error，這個部份主要是把 weight 進行更新，加總後回傳。

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

Select_best_move 輸入為 board，顧名思義要找出最優的 move 作為下一步。通過不同的 4 種 action 計算出 reward 和 policy 的隨機數來作為各算 action 的期望值，最後選擇出最大的值來作為下一個 action。

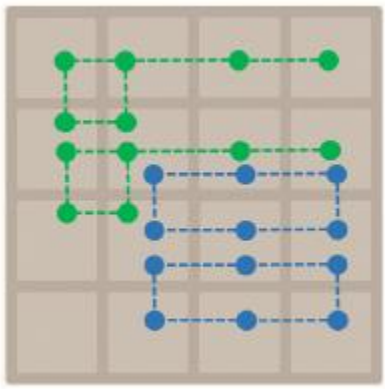
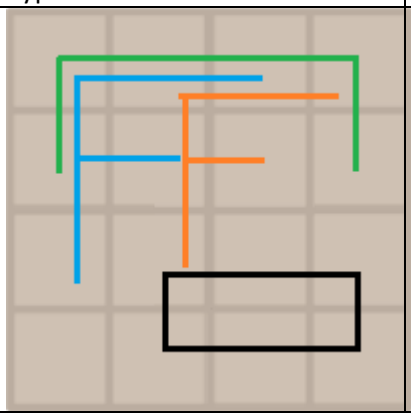
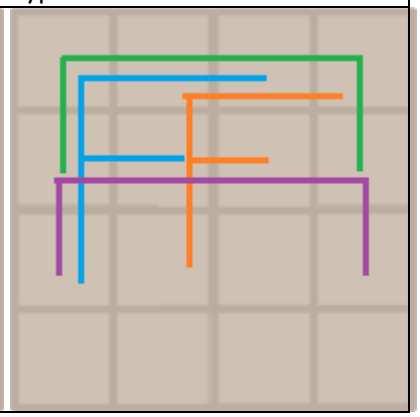
```
  state select_best_move(const board &b) const {
    state after[4] = {0, 1, 2, 3}; // up, right, down, left
    state *best = after;
    for (state *move = after; move != after + 4; move++) {
      if (move->assign(b)) {
        // TODO
        move->set_value(move->reward() + expect_value(move-
>after_state()));  //calculate corresponding reward and popup
        if (move->value() > best->value())
          best = move;
      } else {
        move->set_value(-std::numeric_limits<float>::max());
      }
      debug << "test " << *move;
    }
    return *best;  //return the best move
  }
  float expect_value(const board &b) const {  //consider the popup tile
    float total = 0;
    int count = 0;
    for (int n = 0; n < 16; n++) {
      if (!b.at(n)) {
        board tmp(b);
        tmp.set(n, 1);
        total += 0.9 * estimate(tmp);
        tmp.set(n, 2);
        total += 0.1 * estimate(tmp);
        count += 1;
      }
    }
    return total / count;
  }
```

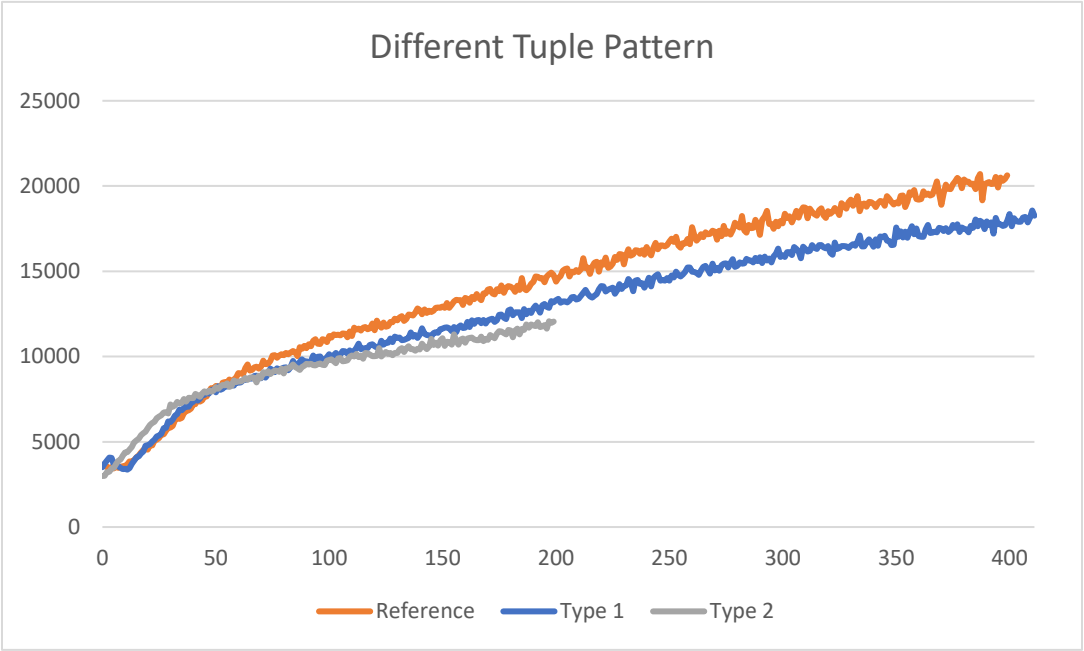
Update episode 輸入為 path，是根據每一個 episode 的倒數第二個 action 開始計算出 action 之間的 error，並以這個 error 來更新 state 的期望值，然後一步步逆向推出之前的每個 action 的期望值。

```
  void update_episode(std::vector<state> &path, float alpha = 0.1) const {
    // TODO
    float target = 0;
    for (path.pop_back(); path.size(); path.pop_back()) {
      state &move = path.back();
      float error = target - estimate(move.before_state());
      target = move.reward() + update(move.before_state(), alpha * error);
    }
  }
```

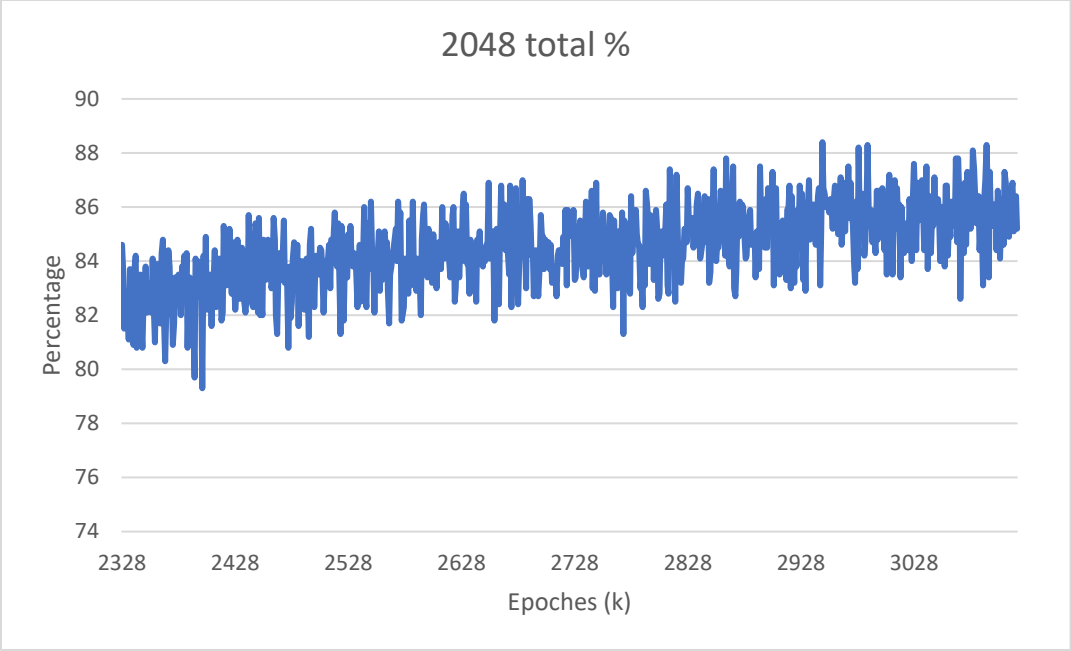
Other discussions or improvements. (5%)

看了一些研究和 paper，不同 tuple 的數量和 pattern 會對 training 有影響，有試著根據個人玩法運用不同的 tuple pattern，一開始效果不錯，但到後面就未如理想，因為時間關係，最後還是採用 reference 的 pattern。

Reference	Type1	Type2
		



在後面的訓練中，我以 2048 出現的 percentage 來做了個走向圖，可以發現雖然上升緩慢，但這個 2048 的比例還是上升當中。其中可以發現 2048 的比率和 mean score 不一定成正比，在 4820epoch 的 2048 有高達 91.5%的佔比，但 mean score 才 71702.7，不是最高的分數，可以想像 2048 佔比高並不一定會有最高的分數，還可能會因為和較小的數字互相阻礙遊戲發展而導致分數不能上升，延續遊戲的生存較合成數字重要。



本次作業需要大量的訓練次數才能夠達到真正的收斂，在我的實驗中，很明顯還沒有完全收斂，還有可以進步的空間，至於收斂的點在哪可能要花更多的時間來證明。在論文中提及分三個 state 來替互 feature，但沒有具體說哪幾個 feature，而我也沒有時間再繼續嘗試。