Lab 02 Temporal Difference Learning

邱以中

311551040

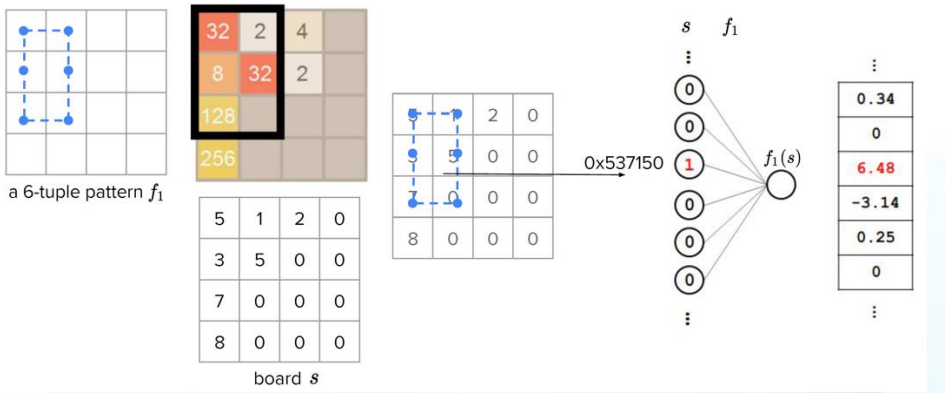
1. Introduction

在這次的作業當中，我們需要使用 reinforce learning 的方法去學習玩 2048 的遊戲，目標是盡可能讓每次遊玩都能讓 2048 的方塊出現。

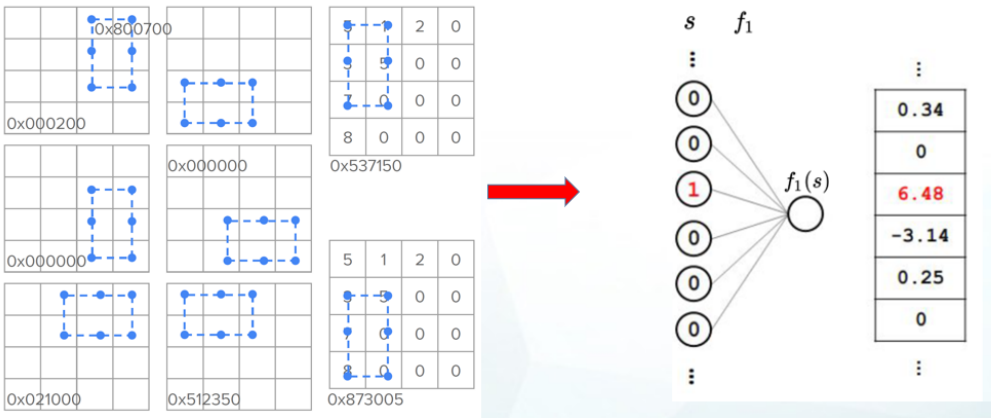
2. N-tuple Network

在計算每個 board 的分數估計值時，我們可以發現若是我們要去計算所有盤面的值的話，假設一格有 15 種可能的數字，我們就需要有 15^16 的儲存空間，這是我們無法負擔的，因此我們就會使用 N-tuple network 來估計盤面，以此來解決這個問題。

在 2048 中 N-tuple network 的用法就是只看一小部分的盤面來進行預測，我們會提取盤面中在 pattern 範圍內的值，並將他們組成一個 index，之後再根據這個 index 得到相對應的 weight 值。

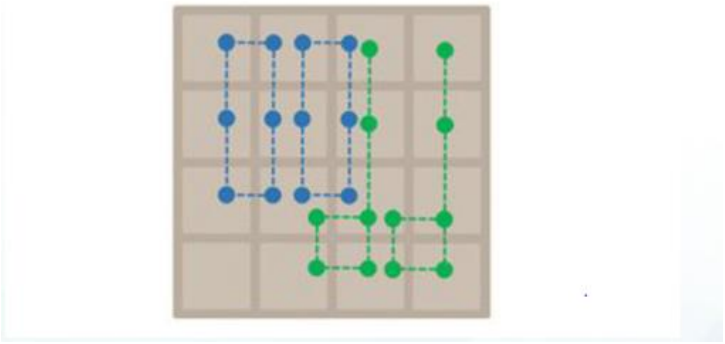


因為盤面經過 rotate/reflect 代表的其實是同一種盤面，所以我們也會對 pattern 做 rotate/reflect 得到 8 個 isomorphic，最後再將它們對應的 weight 相加，代表這個盤面的估計值。



因為只使用一個 pattern 可能代表性不足，沒辦法很好的得到整個盤面的資訊，因此我們也會使用多個 pattern 來計算綜合的盤面估計值。

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



在這裡我們會先取得這個 pattern 所有 isomorphic 在 board 上的位置

```
for (int i = 0; i < 8; i++) {
    board idx = 0xfedcba9876543210ull;
    if (i >= 4) idx.mirror();
    idx.rotate(i);
    for (int t : p) {
        isomorphic[i].push_back(idx.at(t));
    }
}
```

然後在 indexof 根據當前的 pattern 與 board 得到 weight 的索引值

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

利用索引值得到對應的 weight

```
float& operator[] (size_t i) { return weight[i]; }
float operator[] (size_t i) const { return weight[i]; }
```

用上述的方法計算所有的 isomorphic 的 value 並進行相加，這樣就可以得到這個 pattern 在當前 board 的 value

```
/**
 * estimate the value of a given board
 */
virtual float estimate(const board& b) const {
    // TODO
    debug << "estimate " << std::endl;

    float value = 0.0;
    for(int i=0;i<iso_last;i++){ //所有旋轉/鏡像
        // 用這個isomorphic在board 上得到的weight 索引值
        size_t index = indexof(isomorphic[i],b);
        // 用index 得到這個board 的value，並加總所有可能的旋轉/鏡像
        value += operator[](index);
    }
    // 回傳這個 pattern 在這個board上所有旋轉/鏡像的加總得分
    return value;
}
```

最後再對所有 pattern 的得分進行相加，得到當前盤面的估計值

```
float estimate(const board& b) const {
    debug << "estimate " << std::endl << b;
    float value = 0;
    // n tuple 的模板，預設四個，將每個模板的旋轉/鏡像加總得分，代表這個board預期分
    for (feature* feat : feats) {
        value += feat->estimate(b);
    }
    return value;
}
```

3. TD(0)

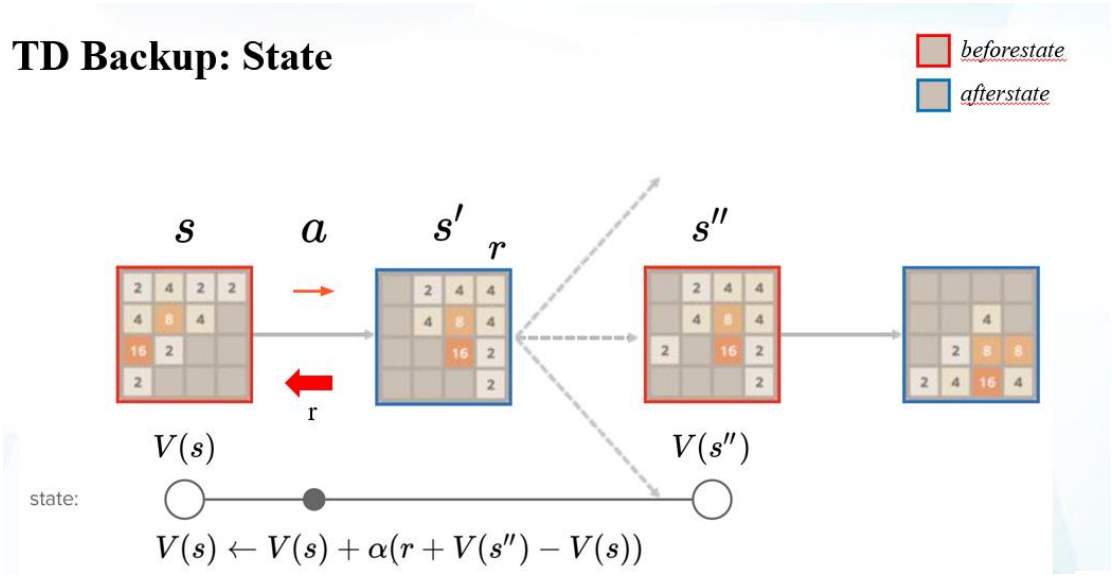
Tabular TD(0) for estimating v_π

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'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

TD(0) 就是每次只走一步就更新，相較於 MC 需要執行到最終狀態才進行更新，TD(0) 的方法具有更小的 variance 與更大的 bias。

在實作當中，我們會去計算每一步 before state 的估計值 V(S)，與下一步 before state 的估計值 V(S'') + reward 的差 (error)，然後再將 error 乘上 learning rate (alpha) 去更新 n-tuple network 的權重。




```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float after_value = 0.0;
    for(int i=path.size()-2 ; i>=0 ;i--){
        float update_weight = alpha * (path[i].reward()+after_value-estimate(path[i].before_state()));
        after_value = update(path[i].before_state(),update_weight);
    }
    debug << "update_episode " << std::endl;
}
```

4. Implementation

- Estimate

  在 estimate function 當中我們要做的是去計算 board 在這個 feature 的得分，因為我們會對 pattern 做 rotations/refection 的動作，並去計算總共的分數，所以會跑一個 for 迴圈遍歷每個 isomorphic，之後使用 indexof 根據 isomorphic 與 board 取得 weight 的 index，最後會將得到所有分數加總回傳。

```
/**
 * estimate the value of a given board
 */
virtual float estimate(const board& b) const {
    // TODO
    debug << "estimate " << std::endl;

    float value = 0.0;
    for(int i=0;i<iso_last;i++){
        size_t index = indexof(isomorphic[i],b);
        value += operator[](index);
    }
    return value;
}
```

- Update

  在 update function 當中，我們會去根據 error (u) 去更新這個 feature 的 weight，因為這個 feature 有 8 個 isomorphic，所以會將 u 平均分給所有 isomorphic，之後使用 indexof 取得 weight 的 index，並使用 u 更新 weight，最後再將更新完的 value 回傳。

```
/**
 * update the value of a given board, and return its updated value
 */
virtual float update(const board& b, float u) {
    // TODO
    debug << "update " << std::endl;

    u = u/iso_last;
    float value = 0.0;

    for(int i=0;i<iso_last;i++){
        size_t index = indexof(isomorphic[i],b);
        operator[](index) += u;
        value += operator[](index);
    }

    return value;
}
```

- Indexof

  在 indexof function 中，我們會根據給定的 pattern 得到 board 對應位置的值，並經由 shift 操作組成 weight 的 index。

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

- Select_best_move

  在 select_best_move function 當中，我們要做的就是選出最好的下一步動作，看是要執行上、下、左、右哪個動作。首先會先取得執行動作後得到的 reward，接著會去計算所有可能的盤面的 value，並進行平均，代表執行完動作後的盤面的期望 value，根據出現 2 機率 90%、4 出現機率 10% 的規則，我們會對所有空白的區域填入 2 或 4，並使用 estimate function 計算相應的 value，如果填入 2 value 就乘 0.9，填入 4 就乘以 0.1，之後將加總的分數除上空白的格子數量再加上 reward，就是當前盤面執行這個動作的得分，最後再取得分最高的動作就好。

```
711     state select_best_move(const board& b) const {
712         state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
713         state* best = after;
714 
715         // 看選擇上下左右後哪個最後得分會最高
716         for (state* move = after; move != after + 4; move++) {
717             if (move->assign(b)) {
718                 // TODO
719                 debug << "select best move " << std::endl;
720 
721                 // 執行動作後得到的reward
722                 float r = move->reward();
723 
724                 // 移動後對空白的區塊 pop 2(90%) 4(10%)
725                 // 對所有可能情況計算平均得分
726                 int blank=0;
727                 float total_value=0.0;
728 
729                 board af = move->after_state();
730                 for(int i=0;i<16;i++){
731                     if(af.at(i) == 0){ // 空白區域
732                         // 計算新場面的value
733                         af.set(i,2);
734                         total_value = total_value + 0.1*estimate(af); // 出現4機率 1/10
735 
736                         af.set(i,1);
737                         total_value = total_value + 0.9*estimate(af); // 出現2機率 9/10
738 
739                         blank++;
740                         af.set(i,0);
741                     }
742                 }
743 
744                 // 移動之後的期望得分
745                 total_value = r+total_value/blank;
746                 move->set_value(total_value);
747 
748                 if (move->value() > best->value())
749                     best = move;
750             } else {
751                 move->set_value(-std::numeric_limits<float>::max());
752             }
753             debug << "test " << *move;
754         }
755         return *best;
756     }
```

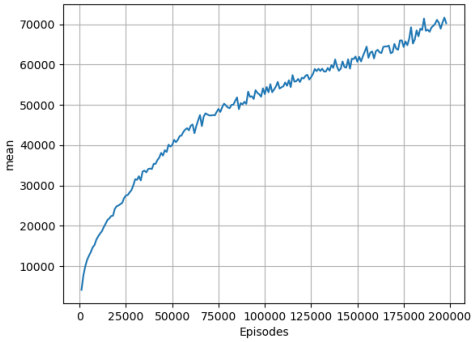
- Update episode

  在 update_episode function 當中，我們會去計算 V(S) + reward 與 V(S'') 的差距 (error)，並將 error 乘上一個 learning rate 拿去更新 V(S)，之後再拿更新完的 V(S) 當作 target 去更新上一步的值。

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float after_value = 0.0;
    for(int i=path.size()-2 ; i>=0 ;i--){
        float update_weight = alpha * (path[i].reward()+after_value-estimate(path[i].before_state()));
        after_value = update(path[i].before_state(),update_weight);
    }
    debug << "update_episode " << std::endl;
}
```

5. Scores plot

下圖為跑了大約 200K episode 的 mean 值變化



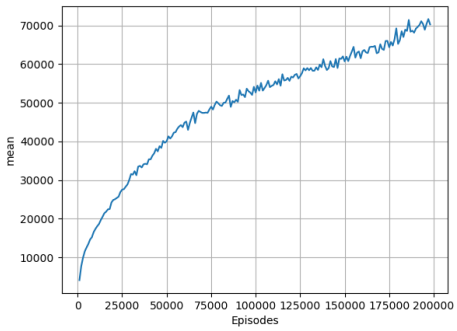
6. Extra

在這次的實作當中，除了原先的 4 個 feature 以外，我也有嘗試增加新的 feature，分別是 2 組 4 個相連的直線與一組十字形狀的 pattern。

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

然後可以發現使用更多的 feature，可以用更少的 episode 達到相同的分數，原因是使用更多不同的 pattern 可以得到更多當前 board 的資訊，因此可以訓練的更好，缺點就是需要使用到更多的 memory 去儲存 n-tuple network 的 weight，並且因為每次查看的 feature 變多，所以每個 episode 所需計算的時間也會變長。

- Origin



- Add_feature