Deep Learning Lab2 – 2048TD Report

智能所 311581006 林子淩

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

Figure 1 展示了 2048TD agent 訓練 420,000 episodes 後的結果。使用的 N-tuple pattern 為預設的四個 6-tuple，learning rate 為 0.1。Figure 1 中，當次的 mean, max 和 1000 場遊戲平均 2048 win rate 分數分別為 100,916、262,776 和 93.9%。Figure 2 則是整個訓練過程中，每 1000 場遊戲的 mean, max 以及 2048 win rate 的 learning curve，其中 2048 win rate 最高可以達到 94.5%。

```
420000  mean = 100916   max = 262776
        128     100%    (0.1%)
        256     99.9%   (0.5%)
        512     99.4%   (2.1%)
        1024    97.3%   (3.4%)
        2048    93.9%   (10.9%)
        4096    83%     (36.4%)
        8192    46.6%   (46.4%)
        16384   0.2%    (0.2%)
```

Figure 1　訓練 420,000 episodes 後，1,000 場遊戲的平均分數(Type=original, learning rate=0.1)

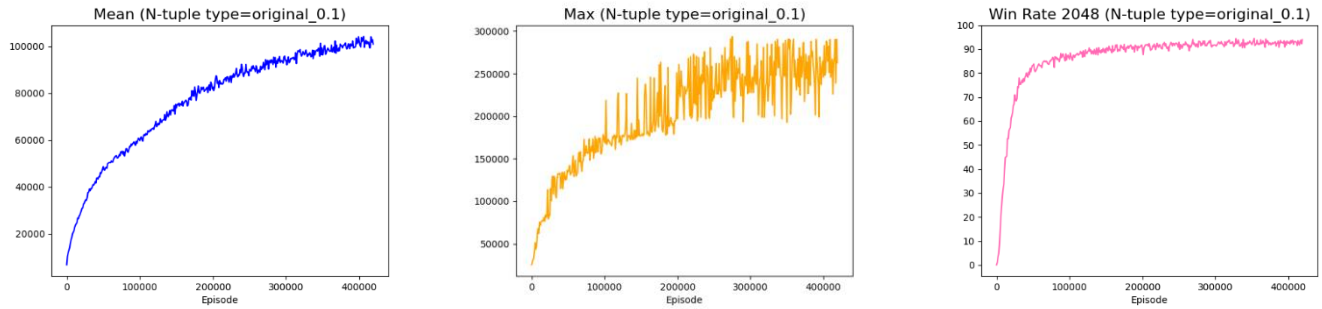

Figure 2　Learning Curve of Mean, Max and 2048 win rate (Type=original, learning rate=0.1)

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

N-tuple Network

N-tuple network 是一種常見訓練 2048 agent 的方法，作為一種 value function approximator 可以用來估計整個 2048 遊戲盤面的潛在價值，即預估出現某種盤面後，還能再賺多少分數遊戲才會結束。考量到如果要儲存每一種 4*4=16 格遊戲盤面狀態(board)與其對應的潛在價值(board value)需要消耗巨量的記憶資源，使用不同形狀的 N-tuple 分割並作為 feature 代表整個遊戲盤面可以在保留大量特徵之餘，大量減少總共需要記憶的狀態(states)數量，因為這些 N-tuple features 可以不同組合重複出現在不同的遊戲盤面中。在針對每一個 board，通過將盤面上 N-tuple 的潛在價值(N-tuple value)加總，即可得到整體 board value。不過在考量 N-tuple 的組合成一個 board 的狀態時，需要考量到當格子數字分布相同，但方向不同的情況(上下左右、水平垂直翻轉)，此時總盤面是等價的，其 N-tuple value 總和也應該等價。因此針對每一個 N-tuple 需要考慮 4 種 rotation 與 2 種 reflection，總共 4*2=8 種 isomorphic features。綜合以上，每一個 board (state s)的對應 board value 可以通過下方公式計算：

$$V(s) = \sum_{i=1}^{k} \sum_{j=1}^{8} f_{ij}(s)$$

公式中，f_{ij}為第 i 個 N-tuple 的第 j 個 isomorphic feature 的 value function，k 為 N-tuple 的數量。在給定 state s 的情況下，加總所有 N-tuple isomorphisms 的 value 可以得到 board value V(s)。

Implementation of N-tuple

在本次 lab 中，我嘗試了不同的 N-tuple 組合，N-tuple 的記錄如 Table 1 所示。Group 為 N-tuple 編號，N-tuple 則以在 2048 board 中的 index 表示。Group Original 為一開始預設的 4 個 6-tuple。其他 Group 則包含各種自行設計的 n-tuple patterns。

Group	N-tuple
Group Original	{0, 1, 2, 3, 4, 5}, {4, 5, 6, 7, 8, 9}, {0, 1, 2, 4, 5, 6}, {4, 5, 6, 8, 9, 10}
	4 個 6-tuple pattern
Group 1	{0, 1, 2, 5}, {1, 2, 5, 6, 9}
	1 個 4-tuple、1 個 5-tuple pattern
Group 2	{1, 5, 6}, {0, 1, 2, 5}, {0, 1, 2, 5, 9}, {0, 4, 5, 8, 9, 10}
	1 個 3-tuple、1 個 4-tuple、1 個 5-tuple、1 個 6-tuple pattern
Group 3	{0, 1, 2, 5}, {0, 1, 2, 5, 9}, {1, 2, 5, 6, 9}, {6, 10, 11, 13, 14},
	1 個 4-tuple、3 個 5-tuple pattern
Group 4	{0, 2, 5, 10}, {4, 6, 9, 14}, {1, 4, 9, 14}, {1, 4, 5, 6, 9, 13}
	3 個 4-tuple、1 個 6-tuple pattern

Table 1　N-tuple groups，每一個 group 包含不同 N-tuple patterns 和其對應的 board index

在實作過程中，我將不同 N-tuple group 組合使用，組合有四種如 Table 2 所示。Type Original 為預設的 4 個 6-tuple patterns，即取 Group Original 單做一個組合。Type 1, Type 2, Type 3 分別在 Type Original 的基礎上，加入新的 N-tuple Group 1, Group 2, Group 3, Group 4。Type 1 從原組合 Type Original 有的 4 個 pattern 增加到擁有 6 個 pattern，Type 2, 3, 4 則增加到擁有 8 個 pattern。

Type	Combination
Type Original	Group Original
Type 1	Group Original + Group 1
Type 2	Group Original + Group 2
Type 3	Group Original + Group 3
Type 4	Group Original + Group 4

Table 2　不同 N-tuple Type，由 N-tuple Group 組合而成

3. Explain the mechanism of TD(0)

TD(0)是 **T**emporal **D**ifference Learning 中最簡單的 case。首先介紹 TD learning 與 Monte-Carlo Method，TD learning 可以根據每一個 action 執行後得到的 reward r 和下一個 state (先不論 before-state 或 after-state)代入 value function approximator 估計出來的 board value $V(s')$，去更新當下 step 的 board value V(s)，整個過程稱作 bootstrapping；而 Monte-Carlo Method 則是要等到遊戲最後的結果出來，得到實際得分(actual return)才去一一更新每個 step 的 board value。兩相比較之下，TD learning 由於兩個 step 之間的 board value 不會相差很大，更新目標的 variance 較低，相對穩定；Monte-Carlo 每次都需要等到遊戲結束得到實際賺分，不同盤遊戲的得分 variance 較高，value 的更新浮動。然而以更新準確度(error)來看，TD learning 由於每次更新的目標是從 value function approximator 估計得

出，即 update toward a guess，準確度較低；Monte-Carlo Method 則是針對實際遊戲結束後的最終得分作為更新目標，即 update toward actual return，準確度較高。
TD(λ)中的 λ 代表以多少比例考慮實際賺分(actual return)作為 value 更新目標，其範圍介於 0 到 1 之間。TD(0)代表完全不考慮 final actual return，只考慮下一個 state。Monte-Carlo Method 則可以看作 TD(1)，即完全考慮 actual return 的一種特殊的 TD learning case。

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

第四部份首先介紹 TD-backup diagram 和 action selection，分成針對 before-state 和針對 after-state 兩種方法(本次 lab 採用 before-state 方法)，再對應到實作細節，最後討論不同 N-tuple 的實驗數據。

TD-backup diagram

① Before-state：

Figure 3 為針對 before-state 的 TD-backup diagram，圖中的紅色格子代表 before-state (環境生成完畢，但尚未執行 action 的狀態)，藍色格子代表 after-state (action 執行完畢，新環境尚未生成的狀態)。假設遊戲從 before-state s 開始，經過執行 action a 後可以得到 after-state s'和對應的 reward r。接著盤面會跳出新的 tile (2 或 4)，此時新環境生成完畢，得到新的 before-state s"。針對每個 state，其 board value 為 V(state)。
在 TD backup 過程中，通過 backward 更新到每一個 state 的 board value V(state)稱作一個 episode。以更新 before-state s 的 board value V(s)為例子，其更新目標 TD target 為經過一個 action a 後得到的 reward r 加上下一個 before-state s"的 value 估計值$V(s'')$。而更新數值 TD error 為 TD target $r + V(s'')$ 減掉當下 before-state s 的 value 估計值V(s)，再將 TD error 乘上 learning rate α 即為實際更新數值。

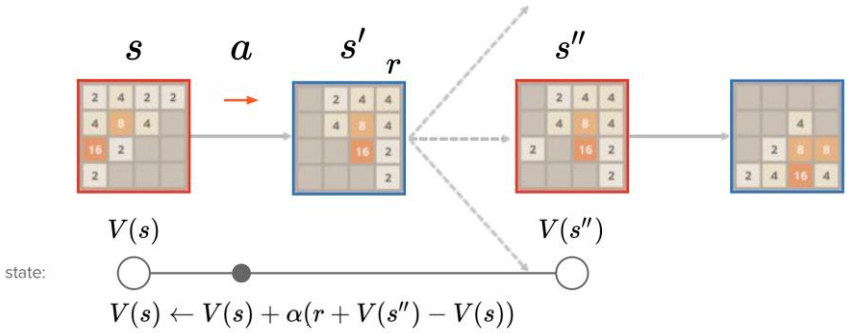

Figure 3　針對 before-state 的 TD-backup diagram

② After-state：

Figure 4 為針對 after-state 的 TD-backup diagram，圖上的符號與定義都跟 Figure 3 一樣。重複 before-state s 到 after-state s'的執行步驟，從 before-state s"開始執行 action a_{next}，又可以得到新的 after-state s'_{next} 和其對應的 reward r_{next}。針對每個 state，其 board value 為 V(state)。
以更新 after-state s'的 board value V(s′)為例子，其更新目標 TD target 為經過下一個 action a_{next} 後得到的 reward r_{next} 加上下一個 after-state s'_{next} 的 value 估計值$V(s'_{next})$。而更新數值 TD error 為 TD target $r_{next} + V(s'_{next})$減掉當下 after-state s'的 value 估計值$V(s')$，再將 TD error 乘上 learning rate α 即為實際更新數值。

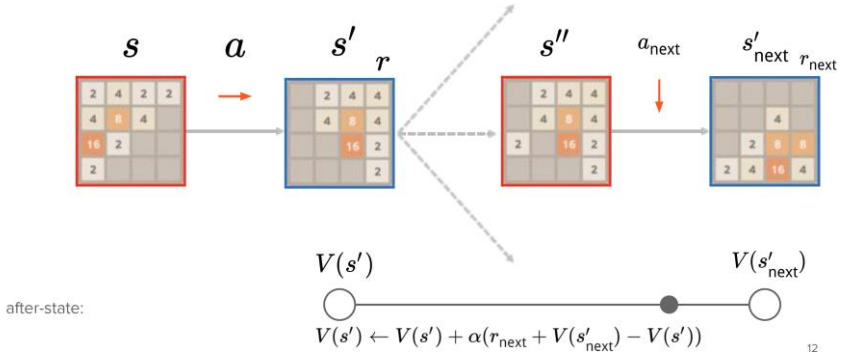

Figure 4　針對 after-state 的 TD-backup diagram

Action Selection

① **Before-state：**

在做 before-state 的 action selection 時，由於 agent 和環境(environment)互相獨立，agent 不會知道下一個 tile 的生成位置與實際數值，因此會需要額外列出並考量下一個 before-state s"的所有可能 cases。通過列舉出執行不同 action (上下左右滑)後，所有可能產生的 before-state s" cases，可以得到產生某個特定 case 的機率與這個 case 的對應 estimated state value。此時一個 action 的價值可以用所有執行該 action 衍伸出來 state cases 的 value weighted sum 表示，由產生該 case 的機率作為 weight，state value 則為 weighted sum 的主體。estimate$[V]$ (action value)的公式如下：

$$\text{estimate}[V] = \sum_{i=1}^{n} p_i \times estimate[V(\text{s}_\text{i}'')]$$

公式中，假設選擇特定 action 會衍伸出 n 個 possible states，可以把這些 state cases 的衍伸機率p_i作為權重，加總每個 case 的 state value $V(\text{s}_\text{i}'')$，得到estimate$[V]$當作選擇某個 action 的預估衍伸價值。另外，除了考量 action 衍伸出來的 estimate state value，還應該將執行該 action 能夠得到的 reward r 加入 action 總價值來衡量。因此最終的 action selection 目標應為最大化 r + estimate$[V]$。

$$\text{Before-state action selcection objective} = \mathbf{Max}(\mathbf{r} + \mathbf{estimate}[\mathbf{V}])$$

② **After-state：**

相較於 before-state 的 action selection，after-state 的 action selection 不需要考慮新環境生成的不同可能，較為簡單。在 after-state 的情況下，只需要考慮執行每一種可能的 action 後，能夠得到的 reward 和下一個 after-state 的 value，即 after-state TD-backup diagram 中所描述的 TD target。最終的 action selection 目標為最大化 TD target $\text{r}_\text{next} + \text{V}(\text{s}'_\text{next})$。

$$\text{After-state action selcection objective} = \mathbf{Max}\left(r_{next} + V(\text{s}'_\text{next})\right)$$

Implementation detail

這次 lab 實作中，sample code 裡共有五個 TODO。前三個是實作 `class pattern` 裡的 function，`class pattern` 則是 `class feature` 的 subclass，用來處理 N-tuple features 的相關功能。後兩個則是 `class learning` 中的 function，負責 TD learning 訓練過程。另外，最後也附上所有實驗的結果與討論作為實作細節的一部分。

①**TODO 1**：Figure 5 為 `estimate` function 的程式碼實作，功能為加總一個 N-tuple 所有八個 isomorphism 的 value，來得到計算給定遊戲盤面的 board value。

```
500     virtual float estimate(const board& b) const {
501         // TODO
502 
503         float value = 0;
504 
505         /* value = sum of 4(rotate)*2(flip, mirror) isomorphics' weights */
506         for (int i=0; i<iso_last; i++){
507             // get idx of isomorphic pattern
508             size_t idx = indexof(isomorphic[i], b);
509             // add weight (find by idx) to total board value
510             // operator[] : returns weight[i]
511             value += operator[](idx);
512         }
513 
514         return value;
515 
516     }
```

Figure 5　TODO 1，實作估計給定遊戲盤面的 board value

②**TODO 2**：Figure 6 為 `update` function 的程式碼實作，功能為更新 N-tuple feature weight，並回傳更新後的盤面新的 board value `new_value`，以提供後續 TD target 計算使用。注意 total update value `u` 為整個 board state 要更新的 value，必須平均分攤到每個 N-tuple isomorphism 上，因此一開始要先將 total update value 分成 8 份。

```
521     virtual float update(const board& b, float u) {
522         // TODO
523 
524         /* split board total update value u to each isomorphic */
525         float update_isomorphic = u / iso_last;
526         float new_value = 0;
527 
528         for (int i=0; i < iso_last; i++){
529             size_t idx = indexof(isomorphic[i], b);
530             // weight of idx (index of isomorphic pattern) increase by (update_each)
531             operator[](idx) += update_isomorphic;
532             // calculate new value of the whole board
533             new_value += operator[](idx);
534         }
535 
536         return new_value;
537 
538     }
```

Figure 6　TODO 2，實作更新 N-tuple isomorphism weight

③**TODO 3**：Figure 7 為 `indexof` function 的程式碼實作，功能為找到 N-tuple pattern 在 board 上的 index 位置，並組合出儲存該 pattern 對應的 weight index 位置 `idx`，再回傳以用來得到該 pattern 對應的 weight value。

```
573     size_t indexof(const std::vector<int>& patt, const board& b) const {
574         // TODO
575 
576         size_t idx = 0;
577 
578         /* get pattern idx in board b */
579         for (size_t i=0; i < patt.size(); i++){
580             // b.at(patt[i]) = pattern idx in board b, shift left by 4 bits for a blank in board
581             // idx = b.patt[n] | b.patt[n-1] | ... | b.patt[1] | b.patt[0] (4 bits for a index)
582             idx |= b.at(patt[i]) << (4*i);
583         }
584 
585         return idx;
586     }
```

Figure 7　TODO 3，實作找出不同 N-tuple pattern 的 weight index

④**TODO 4**：Figure 8 為 `select_best_move` function 的程式碼實作，功能為選擇最好的 action (考慮 before-state 的情況)。跟前面 before-state action selection 描述的一樣，需要嘗試所有可能的 action (上下左右滑動)，並列出所有可能的結果。首先通過我自己定義的 function `get_empty_idx` 可以得到執行 action 後，產生的 after-state 中仍然空白的 tile 在 board 上的 index。接著新生成的 tile 按照 0.9 : 0.1 的機率分別可能跳出 2 或 4，且可能在任何一格空白 tile 中生成新 tile，由此得到某一特定 tile 會跳出 2 跟跳出 4 的機率(`prob_2, prob_4`)。接著就循環每一種接下來會產生的 before-state case，計算每一種 before-state 的 board value (以 `estimate` function 計算)，再乘上產生該 before-state 的機率，以 weighted sum 的形式加總所有 before-state case 的 estimate value。最後，選擇執行 action 得到的 reward 加上所有執行該 action 後可能的 before-state value 最大的 action 作為實際遊戲決策。

```
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

            // s = board after move
            board s = move->after_state();

            /* consider new tile in next board */
            std::vector<int> empty_idx = s.get_empty_idx();
            // sum up value of all possible new boards with probability as weight
            float value = 0;
            // probability of a new tile (2/4) popup at specific empty_idx
            float prob_2 = 0.9f / empty_idx.size();
            float prob_4 = 0.1f / empty_idx.size();


            for (int i=0; i<empty_idx.size(); i++){
                // two possible boards after s, popup 2/4 at specific empty_idx i
                board s_next_2 = s;
                board s_next_4 = s;

                // s_next_2 and s_next_4 popup 2/4 at empty_idx[i], respectively
                s_next_2.popup_specific(empty_idx[i], 1);
                s_next_4.popup_specific(empty_idx[i], 2);

                // sum up possible new boards with probability as weight
                value += prob_2 * (estimate(s_next_2));
                value += prob_4 * (estimate(s_next_4));
            }

            // value of this move = reward of this move + prob weighted sum of new boards' value
            move->set_value(move->reward() + value);

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

Figure 8　TODO 4，實作 before-state 方法下的 action selection

⑤**TODO 5**：Figure 9 為 `update_episode` function 的程式碼實作，功能為以 backward 的方法由後向前更新每一個 step 中的 before-state value，更新數值為 learning rate α * TD error (同 before-state TD-backup diagram 中描述)。注意每一次的 next before-state value 要換成已經在上一個 loop 經過更新並重新計算的 `value_s_next`。

```cpp
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO

    /* update each V(s) with TD error, s=before_state
    * V(s) <- V(s) + alpha * {[reward' + V(s')] - V(s)}
    * TD target = [reward + V(s')]
    * TD error = {[reward + V(s')] - V(s)}
    * V(s_last) always equals to 0, initial value_s_next=0
    */

    // save for update, valud_s_next = new value of V(s') that has been updated
    float value_s_next = 0;
    while(!path.empty()){
        // step = the last (before state, after state, action, reward) element in <vector> &path
        state &step = path.back();
        // td error = reward + V(s') - V(s)
        float td_error = (step.reward() + value_s_next) - estimate(step.before_state());
        value_s_next = update(step.before_state(), alpha*td_error);
        // pop current step from the <vector> &path
        path.pop_back();
    }

}
```

Figure 9　TODO 5，實作 backward 更新每一個 step 的 before-state value

Experiment Result of different N-tuple patterns

Table 3 為使用不同 N-tuple patterns 之下，demo 模式跑 1,000 episodes 的結果(讀取經過訓練 420,000 episodes 後儲存出現最佳 2048 win rate 的 model weight，訓練皆固定 learning rate=0.1)。每個 Type 皆跑十次 demo，並選取出現最高 2048 win rate 的那次結果放入 Table 3 比較，完整的十次 demo 結果如 Table 4 所示。從 Table 3 中可以看出，Type 2 的 N-tuple 組合可以達到最好的訓練結果，Mean、Max 和 2048 Win Rate 都高於其他組合。整體排序為 Type 2>Type 3=Type 4>Type 1>Type Original。這個結果反映了當 n-tuple pattern 越多且多樣化，會讓訓練效果更好，可以達到更高的 2048 Win Rate。

Type	Combination	Mean	Max	2048 Win Rate
Type Original	Group Original	93121.1	290016	93.70%
Type 1	Group Original + Group 1	96514.6	177424	94.50%
Type 2	Group Original + Group 2	**134787**	**320708**	**98.00%**
Type 3	Group Original + Group 3	114528	226008	96.60%
Type 4	Group Original + Group 4	65820.1	148332	96.60%

Table 3　不同 N-tuple Type 的 demo 結果(1,000 episode)

Table 4 中包含不同 N-tuple Type 完整的十次 demo 結果，第一欄為該次 demo 使用的 random seed，第二到四欄則分別為 mean, max, 2048 win rate 數據。由於 random seed 不同，數據會稍微浮動，但幅度沒有很大。另外，最佳 2048 win rate 出現的該次結果以粗體藍色底標示，並被選入 Table 3 比較。

type original	mean	max	2048 rate
78185174	**93121.1**	**290016**	**93.70%**
3037811302	92350.3	233908	91.60%
2007858262	92838.8	246236	91.70%
3202459731	94527.4	249160	93.10%
1952287557	92609.0	288300	92.30%
3270456862	94453.0	246576	92.90%
350970174	94097.5	226656	92.90%
287462308	90935.7	264652	91.00%
2846475523	94129.2	240608	91.90%
2344792614	89872.6	276588	92.20%

type 1	mean	max	2048 rate
910317249	92948.8	177780	93.40%
3593144916	94435.4	177124	93.30%
150094854	92539.8	239792	93.40%
2459945441	91337.9	205000	93.00%
3936539249	94781.0	178760	93.40%
2147144702	94989.5	176900	94.10%
2217256478	96620.3	178700	93.80%
3192691361	96033.3	178132	94.20%
2078086163	93324.9	188064	94.00%
303627953	**96514.6**	**177424**	**94.50%**

type 2	mean	max	2048 rate
622202512	130901	288464	97.00%
1092160061	129577	291156	97.00%
596836343	128704	290144	97.10%
121684100	127695	285836	97.10%
1056720956	**134787**	**320708**	**98.00%**
3684820726	131244	290476	97.50%
3808534764	131380	289916	97.10%
4064675905	130794	286344	96.90%
2588019256	130795	310664	97.10%
1327484023	127148	313052	97.10%

type 3	mean	max	2048 rate
1727221591	**114528**	**226008**	**96.60%**
941918874	113806	255484	96.30%
1512264078	113079	246868	96.20%
557504599	111719	178024	95.80%
504680620	114198	247872	95.70%
4144081986	112281	246212	96.40%
2996508325	114130	181808	96.00%
1063038197	112800	279372	95.70%
3227568024	114866	246080	95.90%
1434812513	114072	239608	96.20%

type 4	mean	max	2048 rate
2218702476	**65820.1**	**148332**	**96.60%**
3059832209	66794.2	148584	95.90%
2644871152	65327.9	135812	96.40%
1125524035	65526.0	143384	95.60%
1378279715	66153.5	142196	96.50%
1155839372	65583.9	137936	96.00%
2187784204	65676.7	148252	96.30%
3980035393	66390.4	144392	96.00%
1566769555	65913.7	135176	95.20%
251728691	64995.0	142268	95.10%

Table 4　不同 N-tuple Type 的 demo 十次 demo 結果

Figure 10 為不同 N-tuple Type 的 learning curve (Mean, Max, 2048 win rate)。從圖中可以看出 Type 4 的 Mean 和 Max 皆明顯低於其他 Type。然而對於這次實驗的觀察重點 2048 win rate 來說，不同 Type 之間的差異不大，這點也可以從 demo 結果看出，五種 Type 皆可以達到 90%以上的 2048 win rate。另外，Type 2,3,4 的收斂速度相對較快，2048 win rate 收斂略快於 Type original, 1。

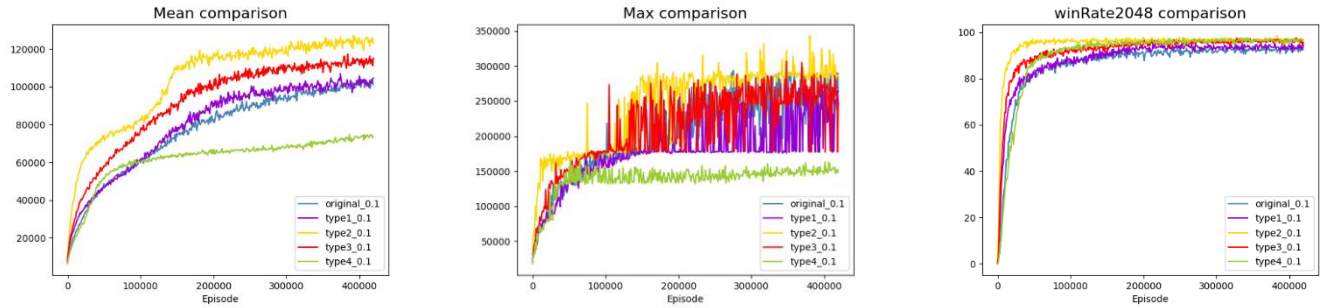

Figure 10　不同 N-tuple Type 的 learning Curve of Mean, Max and 2048 win rate (learning rate=0.1)