Deep Learning and Practice

Lab3 : Temporal Difference Learning

310605009
吳公耀

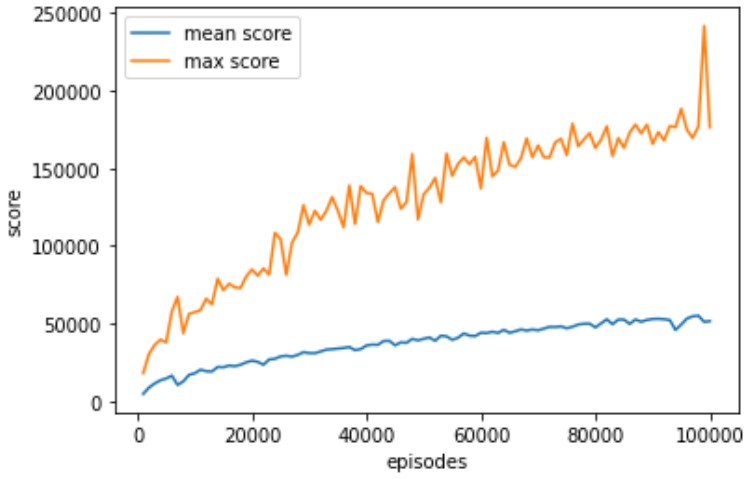
A. A plot shows episode scores of at least 100,000 training episodes

下圖為訓練到100,000 episodes時呈現的結果，且透過圖表來看成長幅度，橫軸為episodes，縱軸為每1000個episodes的平均分數以及最大分數。

Winrate為中間的趴數，即表示可以達到這個分數的機率

最後面的趴數則為最後到這個分數的趴數

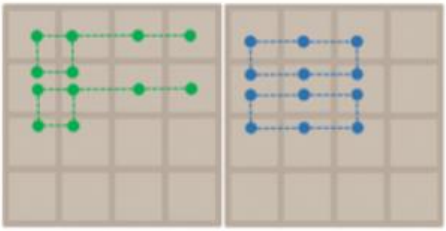
```
99000   mean = 50941     max = 241668
        64       100%    (0.2%)
        128      99.8%   (0.3%)
        256      99.5%   (1.1%)
        512      98.4%   (3.2%)
        1024     95.2%   (16.7%)
        2048     78.5%   (33.2%)
        4096     45.3%   (38.5%)
        8192     6.8%    (6.7%)
        16384    0.1%    (0.1%)
100000  mean = 51349.1   max = 176240
        128      100%    (0.4%)
        256      99.6%   (1.3%)
        512      98.3%   (3%)
        1024     95.3%   (18.1%)
        2048     77.2%   (30.8%)
        4096     46.4%   (40.4%)
        8192     6%      (6%)
```



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

TD Learning 中，相較於紀錄全部共 17^{16}（大約 10^{20}）可能的盤面，會因耗用記憶體過大而無法評估所有的 value 值，所以透過使用 n-tuple network 除了可以大幅降低需要用於紀錄的記憶體空間，也可以製作出大量有用的 features，以一條 4-tuple 為例，只需要紀錄 16^4 種可能，即使考慮所有的 isomorphisms（包含旋轉和鏡射），也只需紀錄 $8x16^4$ 種可能。

實作的部分主要是參考助教提供的 sample code，只需要實做出 feature 的添加提取等過程，且考慮所有的 isomorphisms，我們只需依照 N-tuple 對應到的 tile 去計算他們的 index，得到對應的 index 就可以進行 feature 中的 estimate 還有 update 來偵測這個 tuple 的值以及更新這些 index 對應 weight 的值。使用的 tuple 為助教 sample code 上的範例，並且一些助教 PPT 上顯示的範例，將會在下面附上實驗解果以及使用的 tuple。



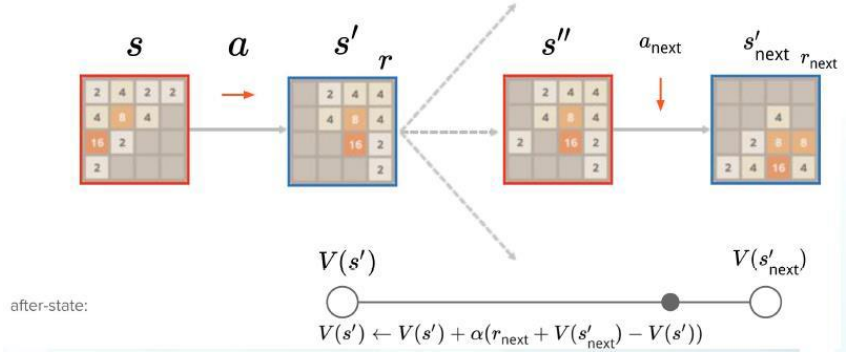
C. Explain the mechanism of TD(0)

TD(0)會在每一步函數取得的值都更新為下一個狀態的值之後，並一路獲得 r，這種觀察到的 r 在足夠數量的採樣後收斂的關鍵因素。TD(0)會從這個 episode 所走過的所有路經中的倒數第二個 action 開始計算出這個 action 和下一個 action 的 before state 的 error，並且以這個 error 來更新對應的 state 的期望值，然後再算出 TD target，也就是 exact 來讓上一個 move 可以繼續更新。

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float exact = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = exact - (move.value() - move.reward());
        debug << "update error = " << error << " for after state" << std::endl << move.after_state();
        exact = move.reward() + update(move.after_state(), alpha * error);
    }

}
```

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

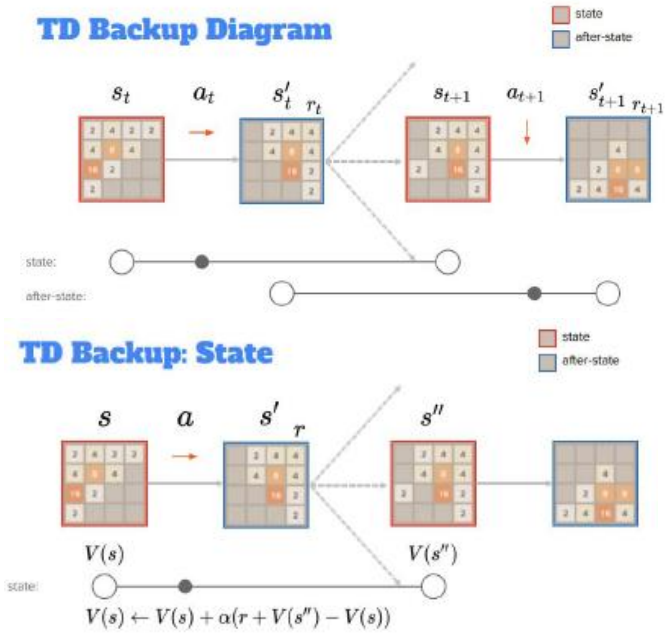


在 after state 的方式下，會以每個 action 的 after state 作為 value function，所以會以下一個 action 的 R 加上 after state 的期望值作為 TD target 來更新現在這個 action 的 after state 的期望值。更新的方法為已知 S'跟 S''，所以讓 S''去找出最好的 action，得到 Snext'，再來去計算 V(S')跟 rnext 加上 V(Snext')的誤差，對 V(S')進行更新，希望 V(S')跟 V(Snext')越來越接近

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

將下一 move 的 before state 視為一個固定的節點，而下一個 move 在選 action 也是一樣的看法，所以只要對 4 個方向所產生的 s'分別去計算分別去計算 reward 加上 V(s')，選擇結果最大的 action 進行移動。

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



TD-backup 主要就是想找出目前 V(St)的分數跟現在這個 state 所做出的預測，找出可能得到最高分數的動作後，實際得到的結果(r + V(St+1))，兩個應該要相同，那如果不相同的話不是 V(St)太低就是 V(St+1)太高，則須將 V(St)進行更新，更新的方式就是把 V(St)跟(r + V(St+1))的誤差以 alpha 對 V(St)進行調整，希望 V(St)跟 V(St+1)要越來越接近。

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

在上圖中要在 St 選出最好的 action，會認為下一 move 的 before state 就是一個固定的節點， 而下一個 move 在選 action 也是一樣的看法。選擇方式為將 St 分別對 4 個不同 action 進行評估，而移動後的 St'是可以計算的，但之後須針對隨機跳出的 2 或 4 也就是 S" 計算 V(S")，期望值，找出 reward 加上期望值最大的 action 進行移動。

H. Describe your implementation in detail.

大部分的實作是參考助教提供的 sample code 然後將 TODO 的部分補上。下列分別為各個缺少的部分講解實作內容。第一個是 feature 中的 estimate function ，這個函式是要將輸入的盤面依照每個 tuple 在 8 個(旋轉 4 次+鏡像) isomophic pattern 得到的 index 所對應到的 8 個 weight，在依照 index 累加，最後經過 n-tuple network 加總成 pattern 在當下盤面所代表的值

```
virtual float estimate(const board& b) const {
    // TODO
    float value = 0;
    // isomorphic
    for (int i = 0; i < iso_last; i++) {
        size_t index = indexof(isomorphic[i], b);
        // return the weight of the given pattern.
        value += operator[](index);
    }
    return value;

}
```

而下列的 update function 是將 pattern 應該修正的 error 平均分給 8 個 isomorphic pattern，會先取得各 isomophic pattern 在當下 board 所代表的 index，並去更新這個 index 所對應的 weight，因為累加的時候是 4 個旋轉+鏡像，所以在更新時有平均更新，然後也有把更新後的值累加後回傳，作為下一次更新時的 S”，節省還要再去計算一次的過程。

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

其中上面兩個 function 中所使用的的 Indexof function 就是為了實現前面兩個計算 index 的功能，是取得給定的 isomorphic pattern 在當下 board 所對應的位置還有盤面，將盤面上的這些位置依照順序進行左移(shift)，最後就可以得到一個 10 進位的 index，來代表 isomorphic pattern 對應到的磚塊內容。

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
    // TODO
    // Return b.at(patt[len - 1]) | b.at(patt[len - 2]) | ... | b.at(patt[1]) | b.at(patt[0])
    // as a value
    size_t index = 0;
    for (size_t i = 0; i < patt.size(); i++)
        index |= b.at(patt[i]) << (4 * i);
    return index;
}
```

再來是 select_best_move function，所以這部分就是計算每個 action 後得到的 reward 加上所有可能產生的 s”盤面的 value 去計算期望值，找出最高的值當成這個 state 的 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() + estimate(move->after_state()));
            if (move->value() > best->value())
                best = move;
        }
        else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

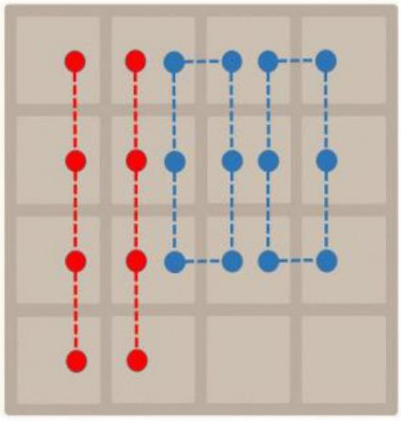
最後一個是update_episode function ，將這個episode所有經過的state還有alpha，從state的最後開始進行更新，計算V(s)跟S做action後得到的reward加上實際得到的V(S’’)，也就是TD target之間的差異，之後將這個值乘上alpha去update，update前面有提到會回傳更新後的值，把這值加上reward就可以作為下一次的TD target，避免重複計算，可以加速執行。

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float exact = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = exact - (move.value() - move.reward());
        debug << "update error = " << error << " for after state" << std::endl << move.after_state();
        exact = move.reward() + update(move.after_state(), alpha * error);
    }

}
```

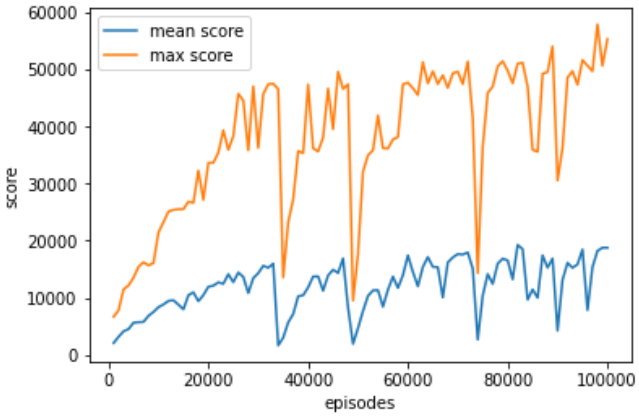
I. Other discussions or improvements

我也有試著用其他不同 n-tuple 來做訓練，可以看到第一個的效果沒有其他兩組來的好。

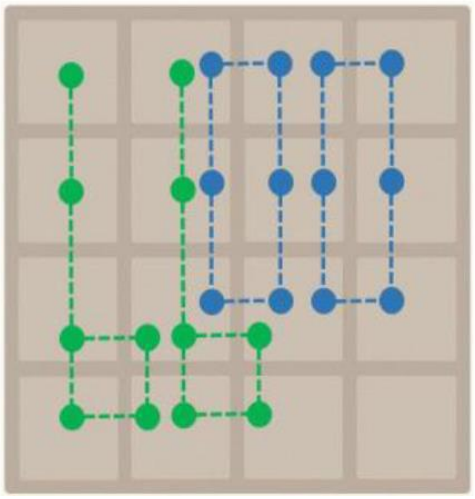


```
// initialize the features
tdl.add_feature(new pattern({ 0, 4, 8, 12 }));
tdl.add_feature(new pattern({ 1, 5, 9, 13 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6, 9, 10 }));
tdl.add_feature(new pattern({ 2, 3, 6, 7, 10, 11 }));
```

```
99000   mean = 18777.2  max = 50640
        16      100%    (0.3%)
        32      99.7%   (0.8%)
        64      98.9%   (0.6%)
        128     98.3%   (3.9%)
        256     94.4%   (5.1%)
        512     89.3%   (15.9%)
        1024    73.4%   (29%)
        2048    44.4%   (43.3%)
        4096    1.1%    (1.1%)
100000  mean = 18772.7  max = 55316
        8       100%    (0.1%)
        16      99.9%   (0.1%)
        32      99.8%   (0.3%)
        64      99.5%   (1.2%)
        128     98.3%   (2.1%)
        256     96.2%   (7.9%)
        512     88.3%   (17.2%)
        1024    71.1%   (26.6%)
        2048    44.5%   (43.1%)
        4096    1.4%    (1.4%)
```

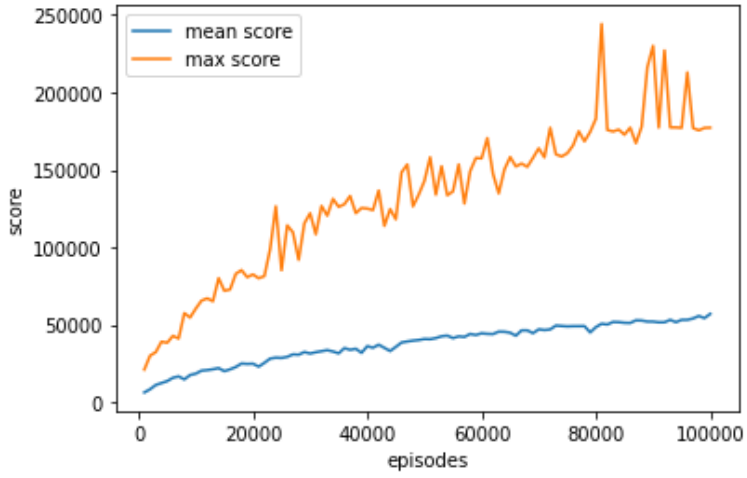


另外一種是可以看出這個效果非常好



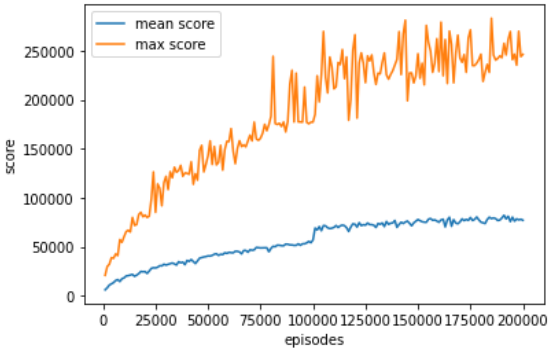
```
// initialize the features
tdl.add_feature(new pattern({ 0, 4, 8, 9, 12, 13 }));
tdl.add_feature(new pattern({ 1, 5, 9, 10, 13, 14 }));
tdl.add_feature(new pattern({ 1, 2, 5, 6, 9, 10 }));
tdl.add_feature(new pattern({ 2, 3, 6, 7, 10, 11 }));
```

```
99000   mean = 54789     max = 177384
        32      100%     (0.1%)
        64      99.9%    (0.1%)
        256     99.8%    (0.9%)
        512     98.9%    (3.2%)
        1024    95.7%    (15%)
        2048    80.7%    (31.5%)
        4096    49.2%    (39.7%)
        8192    9.5%     (9.5%)
100000  mean = 56205.5   max = 235284
        256     100%     (1%)
        512     99%      (3.6%)
        1024    95.4%    (14.3%)
        2048    81.1%    (30.9%)
        4096    50.2%    (40.3%)
        8192    9.9%     (9.8%)
        16384   0.1%     (0.1%)
```



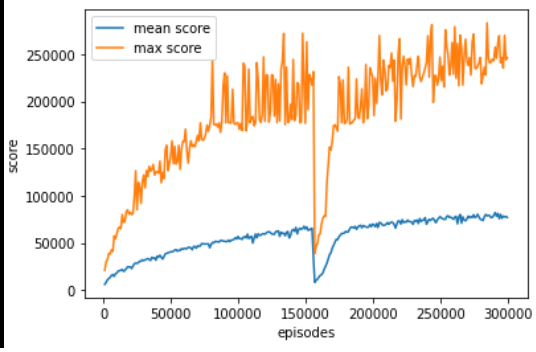
以及繼續做更多的訓練，將之前的結果存起來並每次 load 繼續做 100,000 次訓練，以下為 200,000 的結果。

```
99000   mean = 67259.2  max = 230296
        64      100%    (0.1%)
        128     99.9%   (0.1%)
        256     99.8%   (1%)
        512     98.8%   (1.9%)
        1024    96.9%   (11.7%)
        2048    85.2%   (24.1%)
        4096    61.1%   (40.8%)
        8192    20.3%   (20.2%)
        16384   0.1%    (0.1%)
100000  mean = 70024.9  max = 225652
        128     100%    (0.2%)
        256     99.8%   (0.6%)
        512     99.2%   (1.7%)
        1024    97.5%   (10.4%)
        2048    87.1%   (25.7%)
        4096    61.4%   (39.6%)
        8192    21.8%   (21.7%)
        16384   0.1%    (0.1%)
```



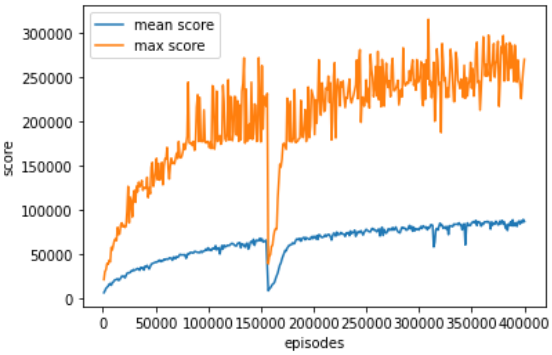
以下為 300,000 的結果

```
99000   mean = 78089.4  max = 244000
        128     100%    (0.1%)
        256     99.9%   (0.3%)
        512     99.6%   (1.1%)
        1024    98.5%   (10.4%)
        2048    88.1%   (21.9%)
        4096    66.2%   (36.2%)
        8192    30%     (29.6%)
        16384   0.4%    (0.4%)
100000  mean = 77078.5  max = 246212
        64      100%    (0.1%)
        128     99.9%   (0.2%)
        256     99.7%   (0.1%)
        512     99.6%   (1.9%)
        1024    97.7%   (9.2%)
        2048    88.5%   (23.8%)
        4096    64.7%   (35.6%)
        8192    29.1%   (28.9%)
        16384   0.2%    (0.2%)
```



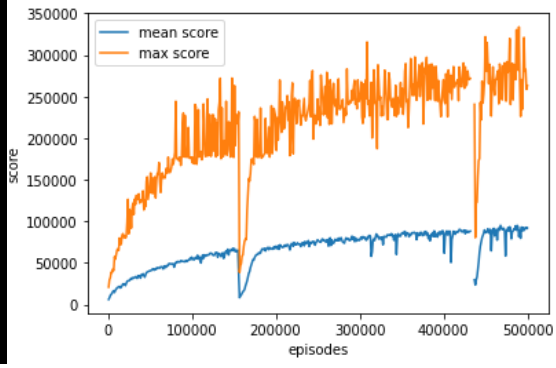
以下為 400,000 的結果

```
99000   mean = 88926.1  max = 259024
        128     100%    (0.3%)
        256     99.7%   (0.2%)
        512     99.5%   (1.1%)
        1024    98.4%   (9.7%)
        2048    88.7%   (17.6%)
        4096    71.1%   (31.8%)
        8192    39.3%   (38.8%)
        16384   0.5%    (0.5%)
100000  mean = 86538    max = 270164
        128     100%    (0.1%)
        256     99.9%   (0.4%)
        512     99.5%   (1%)
        1024    98.5%   (10.2%)
        2048    88.3%   (18.1%)
        4096    70.2%   (32.5%)
        8192    37.7%   (37.1%)
        16384   0.6%    (0.6%)
```



以下為 500000 的結果

```
99000   mean = 91142.8  max = 258576
        128     100%    (0.1%)
        256     99.9%   (0.1%)
        512     99.8%   (1.3%)
        1024    98.5%   (9.1%)
        2048    89.4%   (17%)
        4096    72.4%   (31.5%)
        8192    40.9%   (40.1%)
        16384   0.8%    (0.8%)
100000  mean = 92124.5  max = 263224
        128     100%    (0.1%)
        256     99.9%   (0.1%)
        512     99.8%   (1.3%)
        1024    98.5%   (8%)
        2048    90.5%   (17.4%)
        4096    73.1%   (32.7%)
        8192    40.4%   (39.6%)
        16384   0.8%    (0.8%)
```



可以看出隨時著 episodes 增加各個分數的 win rate 都有持續進步雖然過程有時候會有動盪。雖然過程中有時候 2048 的 win rate 有到 90% 了。