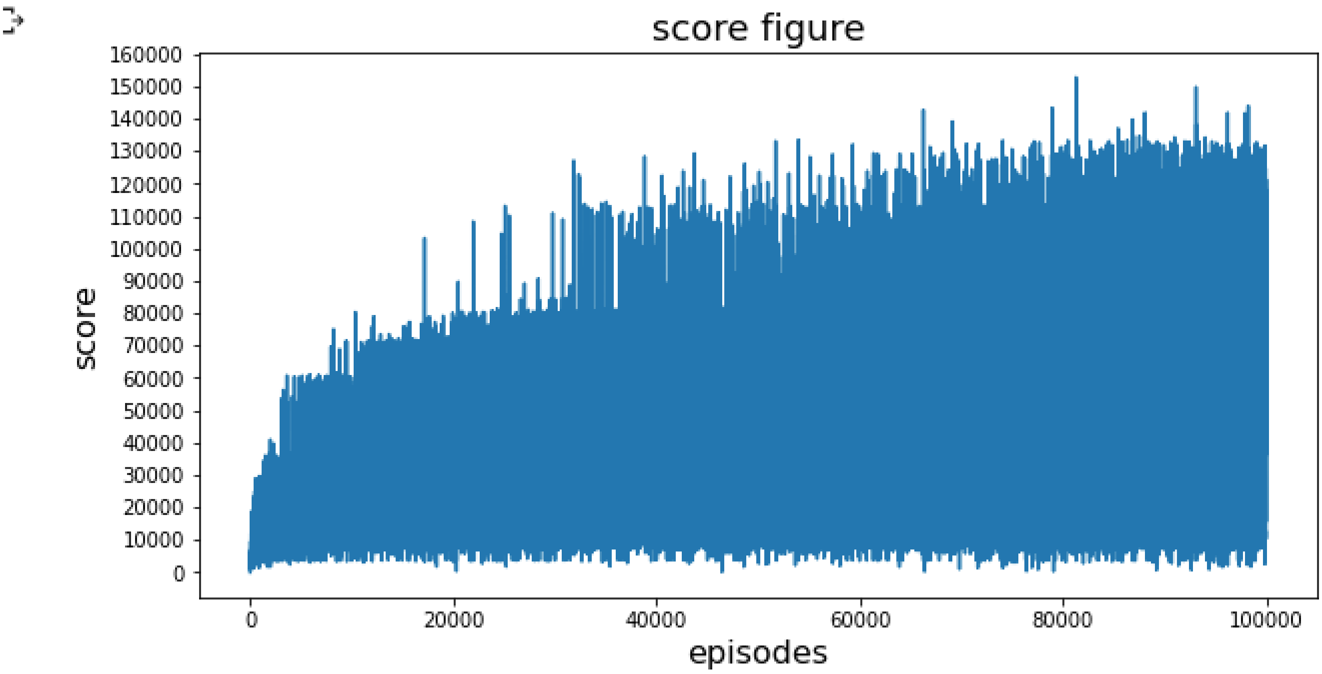
HW2 Report 2048

tags: `DL and Practice`

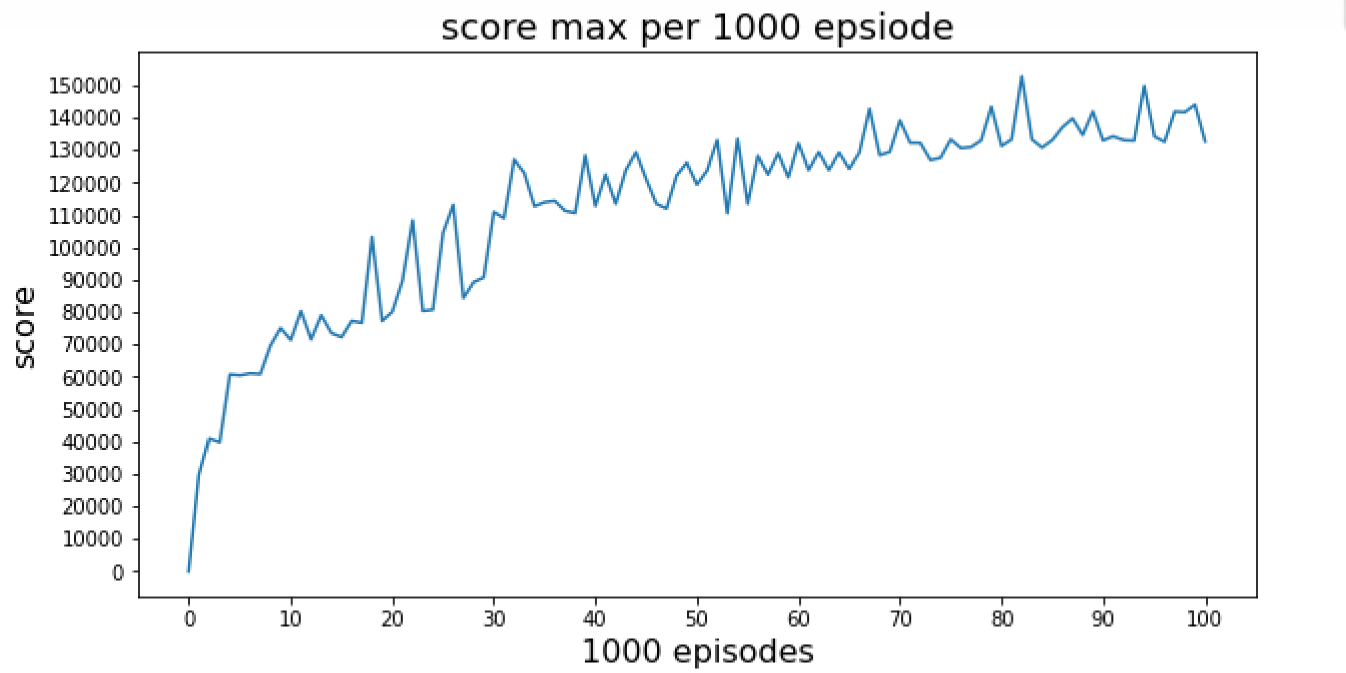
電控碩一 黃柏叡 309512074

Report

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



圖(1):每個episode的score圖表，最後我的訓練結果大約可以得到95%的勝率，由圖中也可以看出score有隨著episodes越來越高



圖(2):從每1000個episode取其中最大值的max score圖表，，由圖中也可以看出max score會越來越高

Describe the implementation and the usage of n-tuple network

n-tuple network(RAM-based neural network)

- large number of input nodes
- Input value are either 1 or 0
- Input is a sparse vector
- no hidden layer
- only 1 output node

選擇用n-tuple是因為整個棋盤大約有13^16^個state(2^0^~2^12^)，如此多的state會比較適合使用RAM-based neural network. 在玩2048時，有發現在執行action時集中往一個角落進攻會有比較高的score，所以在選取feature時我是選取左下角的feature，選擇3-tuple和4-tuple是因為它們在訓練時有很好的表現

```
// 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({ 8, 9, 12, 13}));
tdl.add_feature(new pattern({ 12, 13, 14, 15}));
tdl.add_feature(new pattern({ 5, 8, 9, 12}));
tdl.add_feature(new pattern({ 4, 8, 12, 13}));
tdl.add_feature(new pattern({ 10, 12, 13, 14}));
tdl.add_feature(new pattern({ 8, 12, 13}));
tdl.add_feature(new pattern({ 5, 9, 12}));
```

Explain the mechanism of TD(0)

Tabular TD(0) for estimating v_π

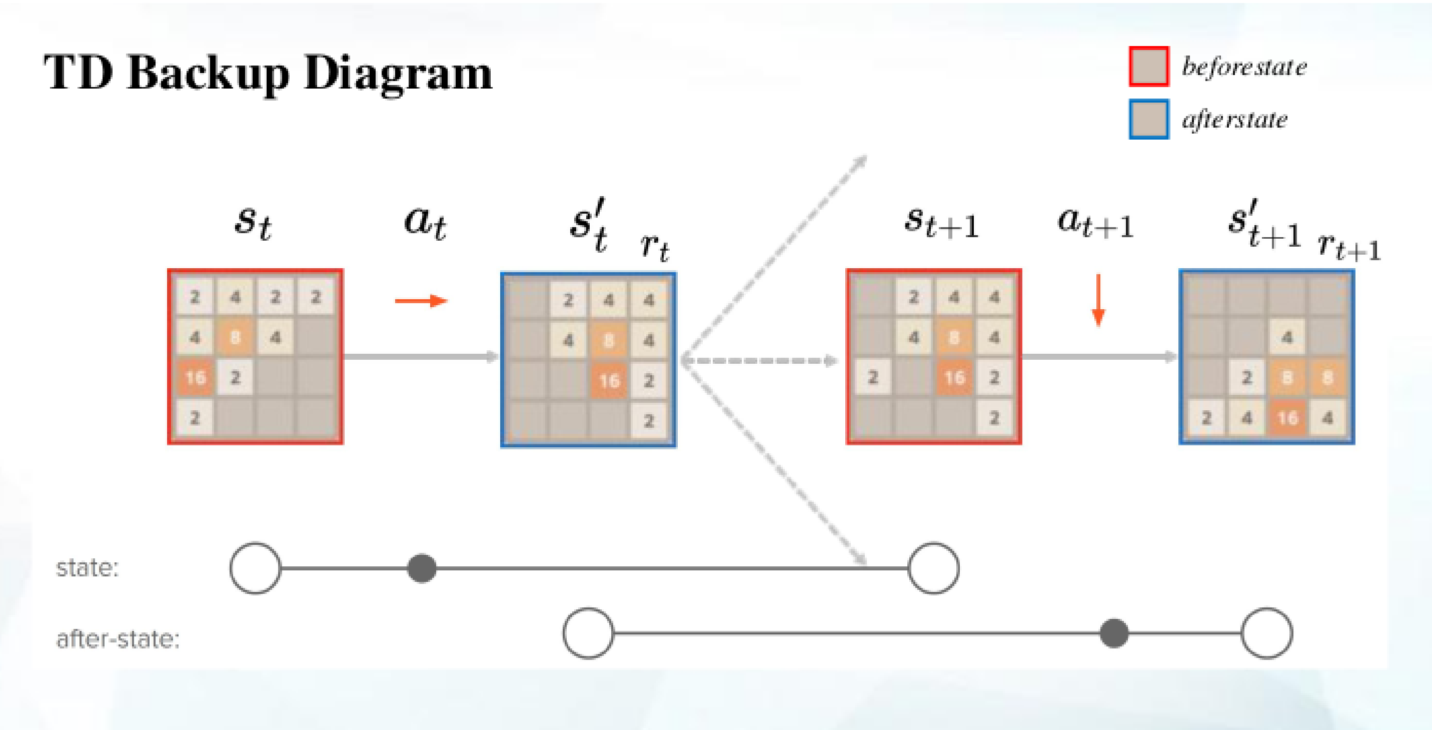
Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

TD(0)one-step TD，就是每走一步便利用觀測到的獎勵值R~t+1~和現有的估計值V(S~t+1~)來更新一次V(S)

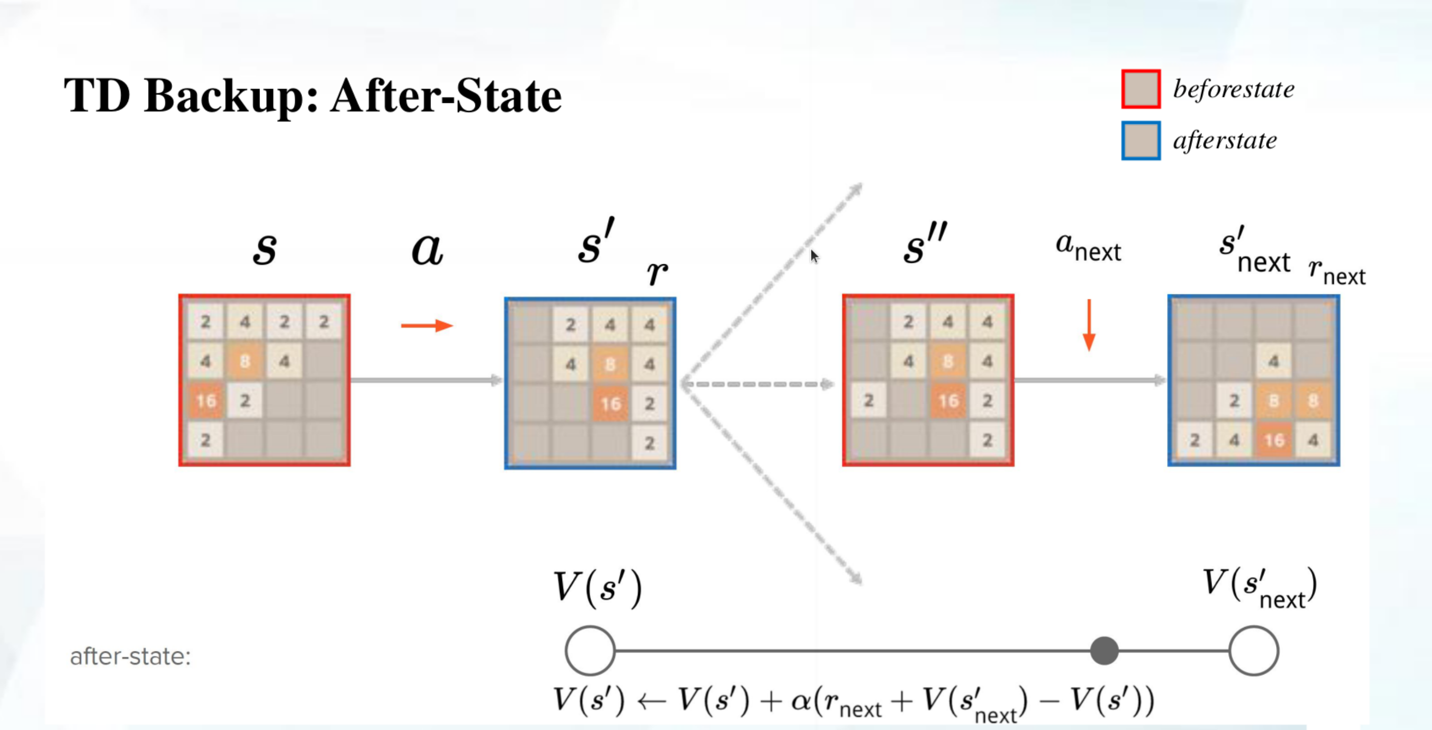
Explain the TD-backup diagram of V(after-state)

Explain the TD-backup diagram of V(state)



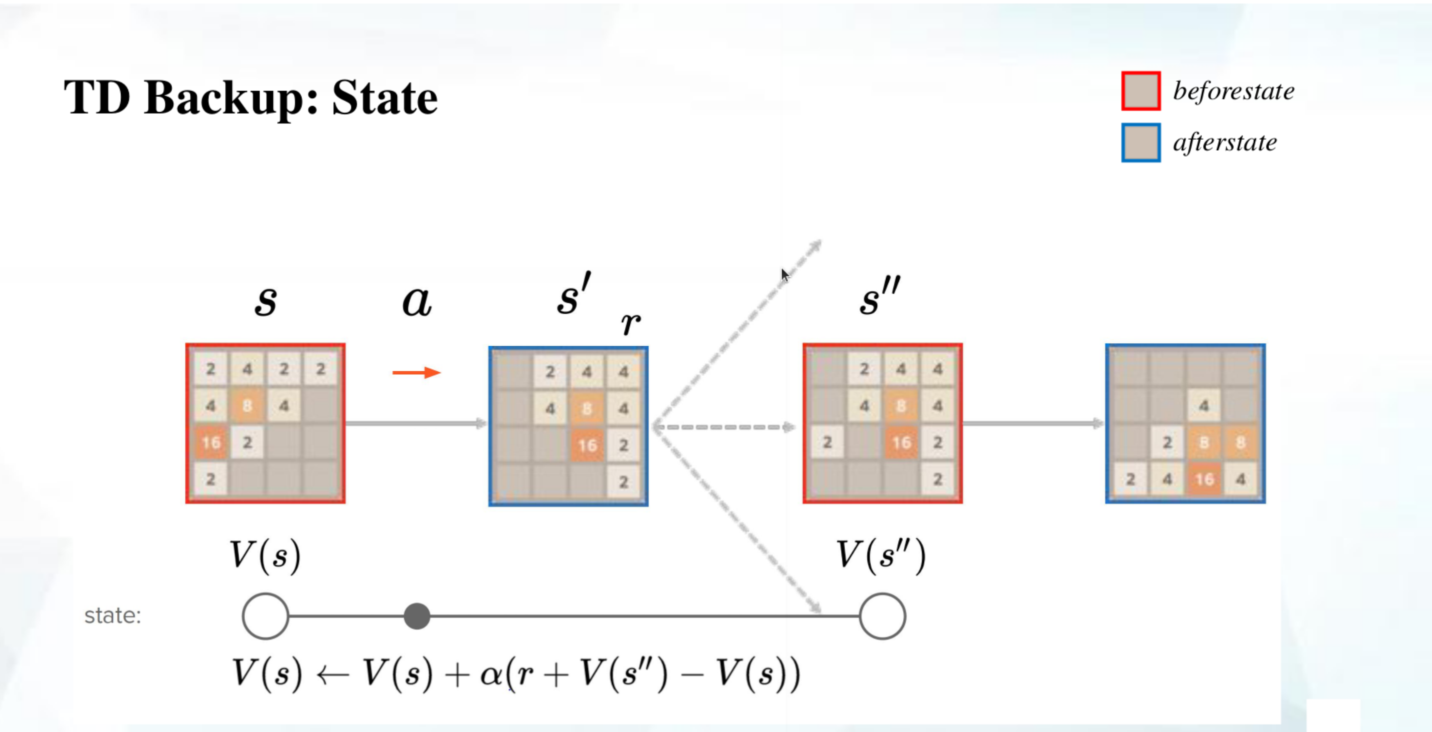
V(after-state):after-state是考慮藍色的board，也就是執行完action後產生的board V(state):state是考慮紅色的board，也就是popup之後的board

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



afterstate不用考慮2或是4popup的位置，它會直接執行V(s')最大期望值的action

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



state在執行action會考慮到all possible next states, 將所有的可能性加總做estimate,接著會判斷執行哪個action會得到最大的value期望值

Describe your implementation in detail

popup:窮舉出所有的可能性

```
int popup(int val, int n) {
    int space[16], num = 0;
```

```
    for (int i = 0; i < 16; i++)
        if (at(i) == 0) {
            space[num++] = i;
        }
    if (num){
        if(val == 0){
            set(space[rand() % num], rand() % 10 ? 1 : 2);}
        else if(val == -1){
            ;
        }
        else{
            set(space[n], val);
        }
    }
    return num;
}
```

將所有可能性的value加總後，去計算做哪個action會得到最大的value期望值

```
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
            float estimate_value=0;
            int num_empty = move->after_state().popup(-1, 0);
            // info << "after state: "<< estimate(move->after_state()) <<std::endl;
            for(int i=0; i<num_empty; i++)
            {
                board tmp2 = move->after_state();
                board tmp4 = move->after_state();
                tmp2.popup(1,i);
                tmp4.popup(2,i);
                estimate_value += (1.0/num_empty)*0.9*estimate(tmp2) + (1.0/num_empty)
            }
            // info << "estimate total:" << estimate_value <<std::endl;
            move->set_value(move->reward() + estimate_value);
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

更新episode, error:TD error, exact:更新V(S)

```
void update_episode(std::vector<board>& s, std::vector<state>& s1, float alpha = 0.1)
        // TODO
        float exact = 0;
        float error;
        for(s.pop_back(); s.size(); s.pop_back())
        {
            s1.pop_back();
            board& before_s = s.back();
            state& after_s1 = s1.back();
            error = after_s1.reward()+exact-estimate(before_s);
            debug << "update error = " << error << " for after state" << std::endl <<
            exact = update(before_s, alpha*error);
        }

    }
```

Other discussions or improvements

在訓練的過程中，我認為要讓勝率提高有兩個地方需要調整 1.feature: 這個最為重要，選取好的feature才能有效提高勝率

- 3-tuple or 4-tuple
- 集中一個角
- 斜的feature

2.learning rate: 當learning rate設為0.1時，大約在50000 episodes後便收維持在約95%的勝率 所以將learning rate調整至0.08, 在75000到100000 episodes之間可以達到約97%的勝率