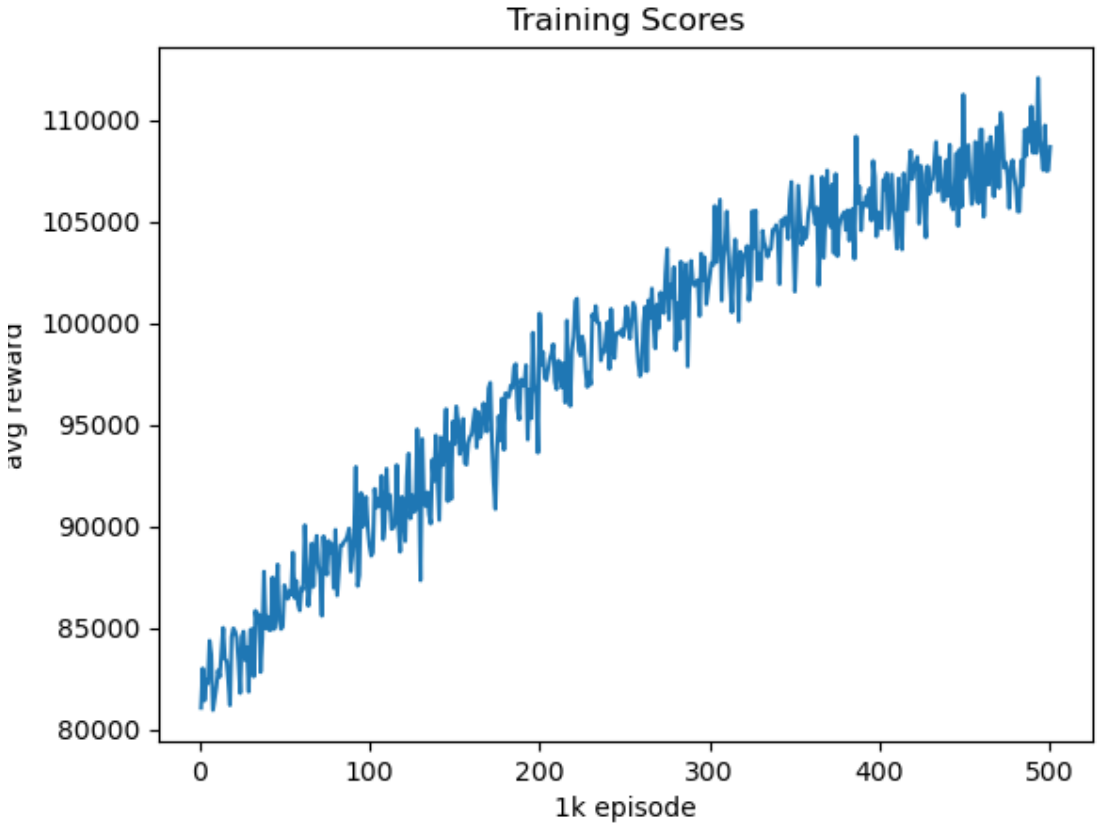
Deep Learning and Practice

Lab 2 309505018 郭俊廷

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



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

2048總共有24個格子, 我使用4個 6-tuple Network 去紀錄整個board, 這樣state數量會從 16^{16} 縮減到 4×16^{6} 種states, 一種pattern的8個isomorphism共享同一個weight table, 我們可以用 alloc()來初始化weight table, 一個weight table約 16^{6} 的大小, 這個weight table在做value iteration的時候可以用來紀錄及更新value function的值

Performance:

episode=500000時, win rate可以到 94.8%

```
500000  mean = 108698   max = 271140
        512     100%    (0.8%)
        1024    99.2%   (4.4%)
        2048    94.8%   (7.3%)
        4096    87.5%   (34.5%)
        8192    53%     (52.6%)
        16384   0.4%    (0.4%)
```

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

model-free的value iteration有兩種方法, Monte-Carlo和Temporal Difference, Monte-Carlo比較適合用在有限的steps, 而Temporal Difference是利用 S_t 和 S_{t+1} 來更新state的value

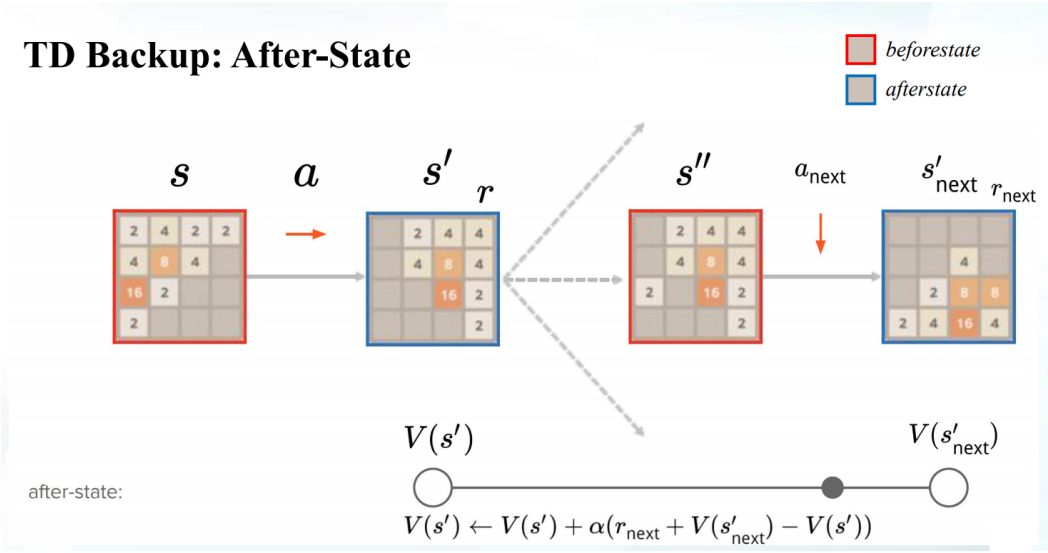
TD: $V(s_t) \leftarrow V(s_t) + \alpha(R_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

優點是可以讓訓練的結果有low variance, 且不需要整個episode都跑完就能得到結果

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

與TD(0)作法類似, 但這裡我們更新的是V(s'), 並且用state evaluation s'所產生的afterstate s''來更新V(s'), 更新公式如下

$$V(s') \leftarrow V(s') + \alpha(R_{next} + V(s'_{next}) - V(s'))$$

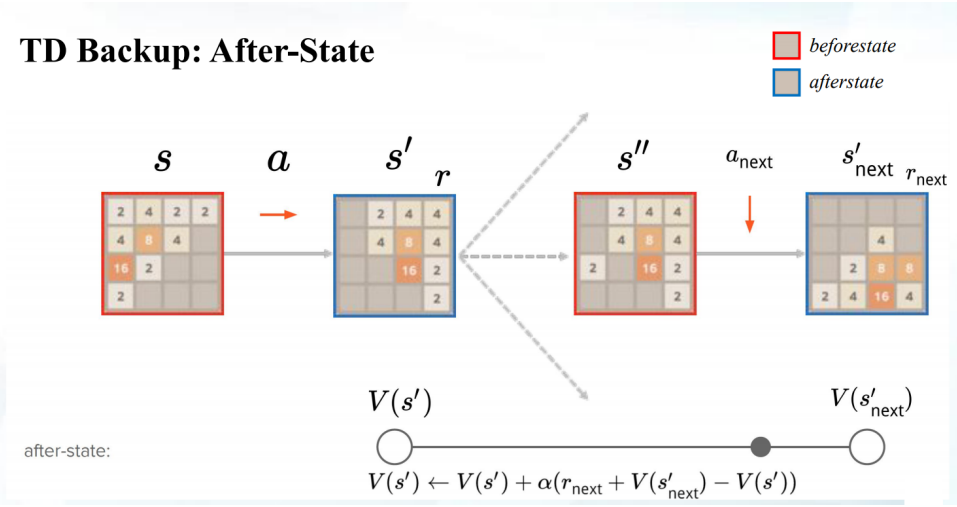


5. Explain the action selection of V(after-state) in a diagram. (5%)

Agent用來更新policy公式如下

$$\pi(s) = arg\max_{a(s)}[R(s,a) + V(T(s,a))]$$

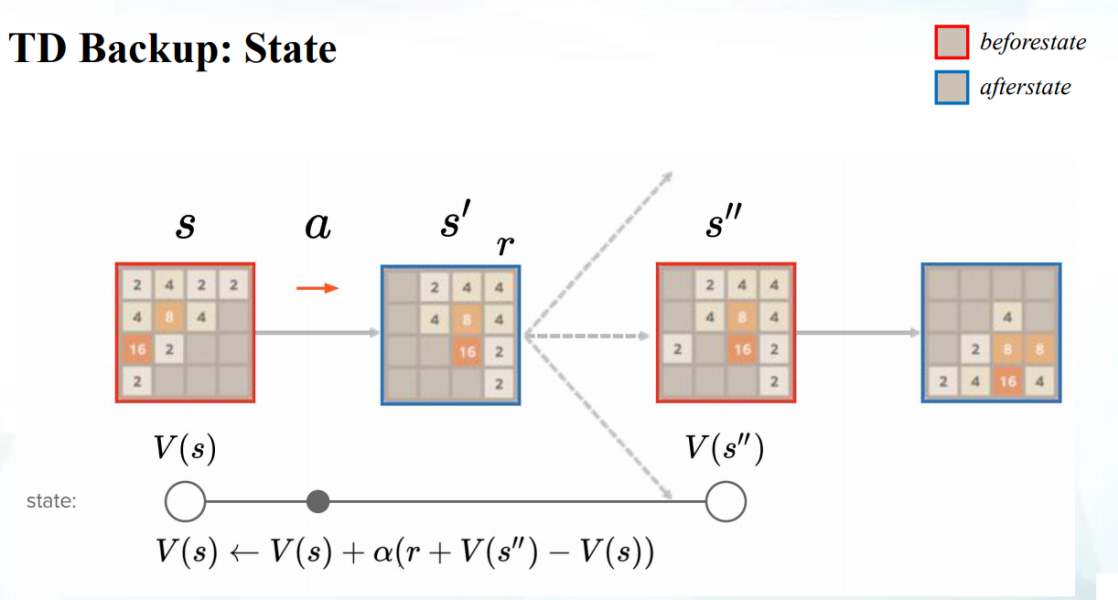
where T(s,a) denote the mapping from state s and action a to the resulting afterstate s'. Agent會選擇state value預估值最大的action



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

根據TD learning, 我們要用下個state的value加上當前action帶來的value與現在state value的差來更新value, 不過這裡並非使用afterstate, 而是從其前一步s”著手, 計算TD error: $r + V(s'') - V(s)$, 我們的更新公式如下

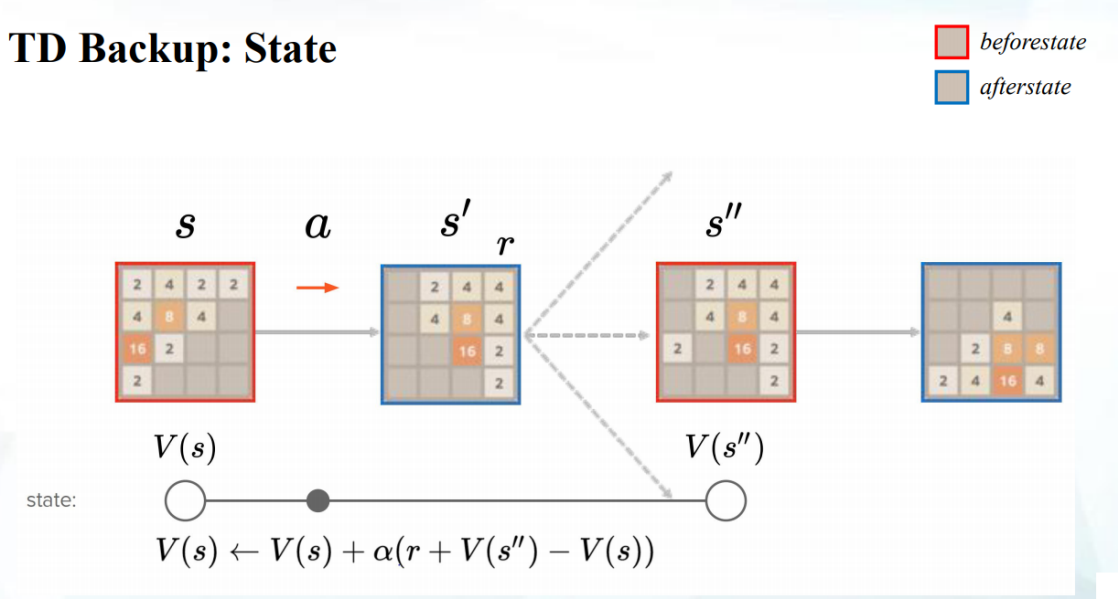
$$V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$$



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

與第5點的action selection類似，但這個方法中需要知道environment model，也就是 需要知道reward function R和transition function P。如果都能知道，那agent便能用下面的公式更新policy:

$$\pi(s) = arg \max_{a \in A(s)} [R(s,a) + X_{s'' \in S} P(s,a,s'') V(s'')]$$



8. Describe your implementation in detail. (10%)

有五個TODO

1) 在estimate function中, 透過查weight table得到各個value, 並且計算當前board中特並的pattern和pattern的8個isomorphism value的和

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

2) 在Temporal-Difference 中, 要去更新feature pattern的weight table, 這裡我們將u平分給每個isomorphism, 對每個isomorphism查lookup table, 找到對應的value之後再給他們加上分配到的value做更新

```
/**
 * update the value of a given board, and return its updated value
 */
virtual float update(const board& b, float u) {
    // TODO
    float u_split = u / iso_last;
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t index = indexof(isomorphic[i], b);
        operator[](index) += u_split;
        value += operator[](index);
    }
    return value;
}
```

3) Get the index of a given pattern, 對於給定的pattern，回傳在表上對應到的index。 在b上面，patt[i]的位置對應到的值再左移4 bits(除以16)，便能以十六進位去對 其位置重新編碼，得到我們所要的index。

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

4) 選擇使reward+V(s)最大的action
這裡參考pseudo code上的EVALUATE function來實作。
這裡我們需要找到每個有可能的next state，才能去做action selection。 根據規則，0.9的機率會出現2，0.1的機率會出現4，我們用這個規則去把所有 可能的next state找出來，然後根據下面的pseudo code來做更新

TD-state

function EVALUATE(s, a)
  $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
  $S'' \leftarrow$ ALL POSSIBLE NEXT STATES(s')
  return $r + \Sigma_{s'' \in S''} P(s, a, s'') V(s'')$

function LEARN EVALUATION(s, a, r, s', s'')
  $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$

```
float all_possible_next_states(const board& b) const {
    size_t num_empty_block = 0;
    float sum_possible_state = 0;

    for (size_t i = 0; i < 16; ++i) {  // simulate popup
        if (b.at(i) == 0) {
            board tmp(b);
            tmp.set(i, 1);
            sum_possible_state += 0.9 * estimate(tmp);
            tmp.set(i, 2);
            sum_possible_state += 0.1 * estimate(tmp);
            num_empty_block++;
        }
    }
    return sum_possible_state / num_empty_block;
}

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() + all_possible_next_states(move->after_state()));

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

5) 最後，在每個episode結束後，呼叫update_episode，對存起來的trajectory中每 個state從後面back propagate error到前面:

$$\Delta V = (R_{t+1} + V(S_{t+1} - V(S_t))) = (y_t - V(S_t))$$

```
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 - (estimate(move.before_state()) - move.reward());
        exact = move.reward() + update(move.before_state(), alpha * error);
    }
}
```

9. Other discussions or improvements. (5%)

Learning rate 我試過 0.0125, 0.00625, 0.003125, 後來發現learning rate設成0.003125(0.1/32)會有比較好的結果, 而原始paper中也有提到不同learning rate對勝率的影響，但礙於時間的關係，我沒有跑完所有的實驗，不過從paper的結果看來，調整learning rate的確能提升勝率，而在TD-AFTERSTATE使用α = 0.0025能得到最好的結果。