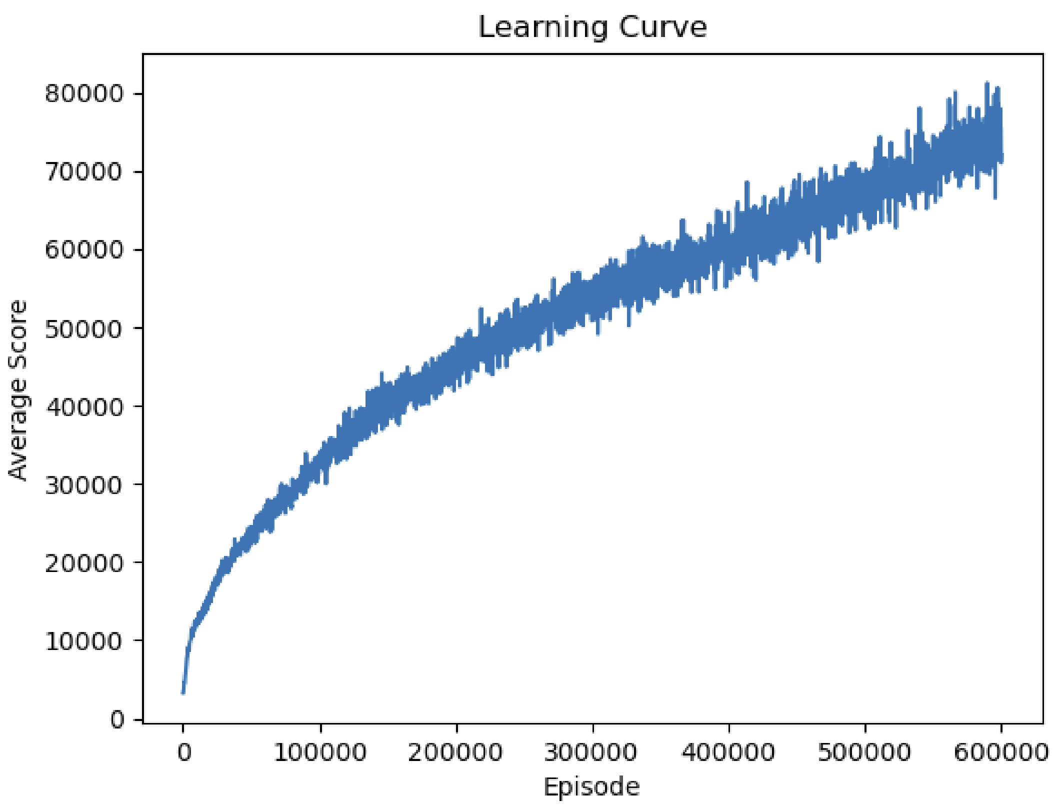
DLP lab2 report

309553005 賴佑家

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



Describe the implementation and the usage of n-tuple network

在2048這個case當中，state的數量太多了，我們不可能窮舉所有的states，因此我們需要一個value function approximator，也就是n-tuple netwrok

而在程式中n-tuple network，會用一個一個的pattern來代表，並且一個pattern會有8個ismorphism，與一個weight table，利用pattern與weight table的乘加，就可以代表board也就是V(s)

Explain the mechanism of TD(0)

TD(0)為最簡單的Temporal Difference Learning，TD(0)具有low variance但有bias的特性，TD(0)定義了兩個值TD target和TD error

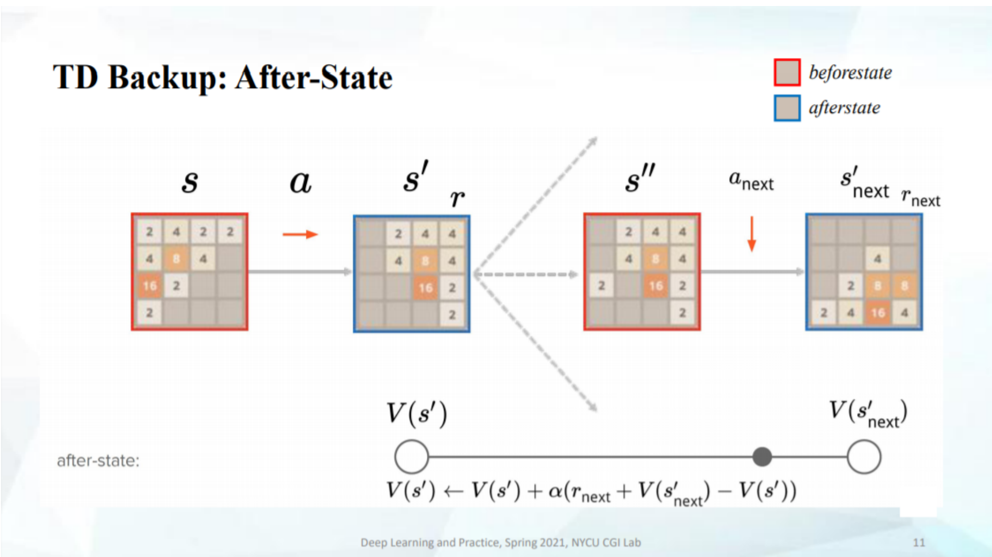
- TD target在evaluate時會用來更新$V(S_t)$
- 結束episode時，TD error會用來進行TD backup

TD target: $R_{t+1} + \gamma V(S_{t+1})$

TD error: $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

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

利用episode紀錄的結果來算出TD target => $r_{next}+V(s'_{next})$
再由TD target來算出TD error => $r_{next}+V(s'_{next})-V(s')$來更新after state $V(s')$
alpha是learning rate



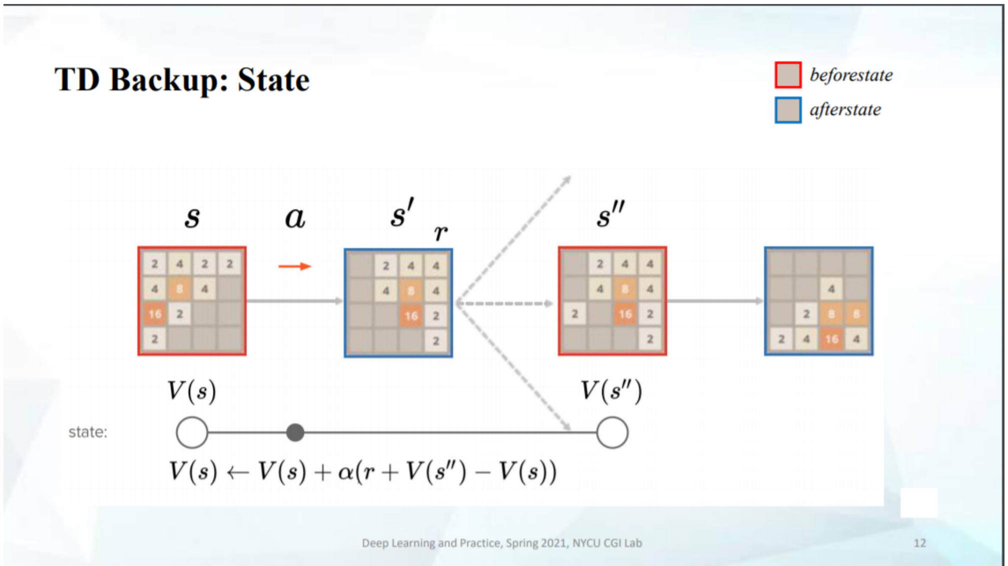
Explain the action selection of V(after-state) in a diagram

在選擇action時，會有四種可能的after state s'（上下左右），哪個action會有最大的reward + V(s')，便選擇那個action

function EVALUATE(s, a)
  $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
  return $r + V(s')$

Explain the TD-backup diagram of V(state)

利用episode紀錄的結果來算出TD target => r+V(s”)
再由TD target來算出TD error => r+V(s”)-V(s)來更新V(s)
alpha是learning rate



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

因s’可以產生多種可能的s”，所以需要模擬所有可能的s”（2跟4會pop在哪個位置）， 並且要乘上transition probability，最後看哪個action會對reward + E[V(s”)] (對於所有可能的s”)所造成的期望值最高，便選擇為最佳的action

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

Making a decision (based on value).

$$\pi(s) = argmax_a(R_{t+1} + \mathbb{E}[V(S_{t+1}) \mid S_t = s, A_t = a])$$

Describe your implementation in detail

1. estimate()

透過weight table查表，計算當前board與對應pattern 8個ismorphism值的和，而這個8個值的和就可以代表一個board

```
/**
 * estimate the value of a given board
 * The sum of the eight values can represents the board
 */
virtual float estimate(const board& b) const {
    // TODO
    float value=0;
    for(int i=0;i<iso_last;i++)
        value+=(*this)[indexof(isomorphic[i],b)];    //we
    return value;
}
```

2. update()

用乘上alpha的TD error將4個pattern的weight table裡的權重更新
並回傳更新後的值
float u即為update value也就是乘上alpha的TD error

```
virtual float update(const board& b, float u) {
    // TODO
    float value=0;
    for(int i=0;i<iso_last;i++){
        value+=((*this)[indexof(isomorphic[i],b)]+=u);
    }
    return value;
}
```

3. indexof()

回傳當下board所對應到weight table哪個index，而決定index的方法為將pattern對應到board上的值加總OR起來

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

4. select_best_move()

首先要計算所有的after state的可能會產生哪些V(s”)，也就是2跟4可能會pop在哪個格子上，得到V(s”)後還要再乘上P(s,a,s”) transition probability(2為0.9, 4為0.1)

選擇action時，就看哪個action會對我們算的這項值產生最大的期望值，就選為當作最佳的action

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

Making a decision (based on value).

$$\pi(s) = argmax_a(R_{t+1} + \mathbb{E}[V(S_{t+1}) \mid S_t = s, A_t = a])$$

```
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
            int space[16], num = 0;
            for (int i = 0; i < 16; i++)
                if (move->after_state().at(i) == 0) {
                    space[num++] = i;
                }
            float E = 0.0;
            if (num){
                for(int i = num-1; i>=0; i--){
                    board temp_board;
                    temp_board = move->after_state();
                    temp_board.set(space[i], 1);
                    E = E + estimate(temp_board)*0.9;
                    temp_board.set(space[i], 2);
                    E = E + estimate(temp_board)*0.1;
                }
            }
            E/=num;
            move->set_value(move->reward() + E);

            if (move->value() > best->value())
                best = move;
```

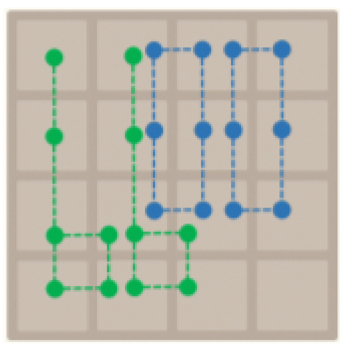
5. update_episode()

在結束一個episode時，會執行update_episode()來進行TD backup，從後面的往前面更新，TD error為 r + V(s’) - V(s)，之後便利用TD error對V(s)進行更新($V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$)

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    state& move=path.back();
    update(move.before_state(), alpha * -estimate(move.before_state())); // update the terminal state
    for (path.pop_back() /* terminal state, already updated */; path.size(); path.pop_back()) {
        state& move = path.back();
        // td_error = r[t+1] + V(s[t+1]) - V(s)
        // td_error = r + V (s'') - V (s)
        float error = move.value() - estimate(move.before_state());
        // V (s) ← V (s) + α(r + V (s'') - V (s))
        update(move.before_state(), alpha * error);
    }
}
```

Other discussions or improvements

我有嘗試著使用不同的n-tuple network下去train看看，圖(a)是使用範例程式預設的pattern，而圖(b)是使用*Multi-Stage Temporal Difference Learning for 2048-like Games*所使用的pattern



最終的結果，使用其他的pattern在train想同的episodes的情況下，比原本的預設的pattern，performance有稍微好一點點

```
600000  mean = 73715.3  max = 173432
        256     100%    (0.1%)
        512     99.9%   (1.4%)
        1024    98.5%   (7.3%)
        2048    91.2%   (19.3%)
        4096    71.9%   (52.2%)
        8192    19.7%   (19.7%)
```

圖(a)

```
600000  mean = 76504.1  max = 178056
        128     100%    (0.3%)
        256     99.7%   (0.3%)
        512     99.4%   (0.8%)
        1024    98.6%   (6.7%)
        2048    91.9%   (17.5%)
        4096    74.4%   (51.2%)
        8192    23.2%   (23.2%)
```
圖(b)