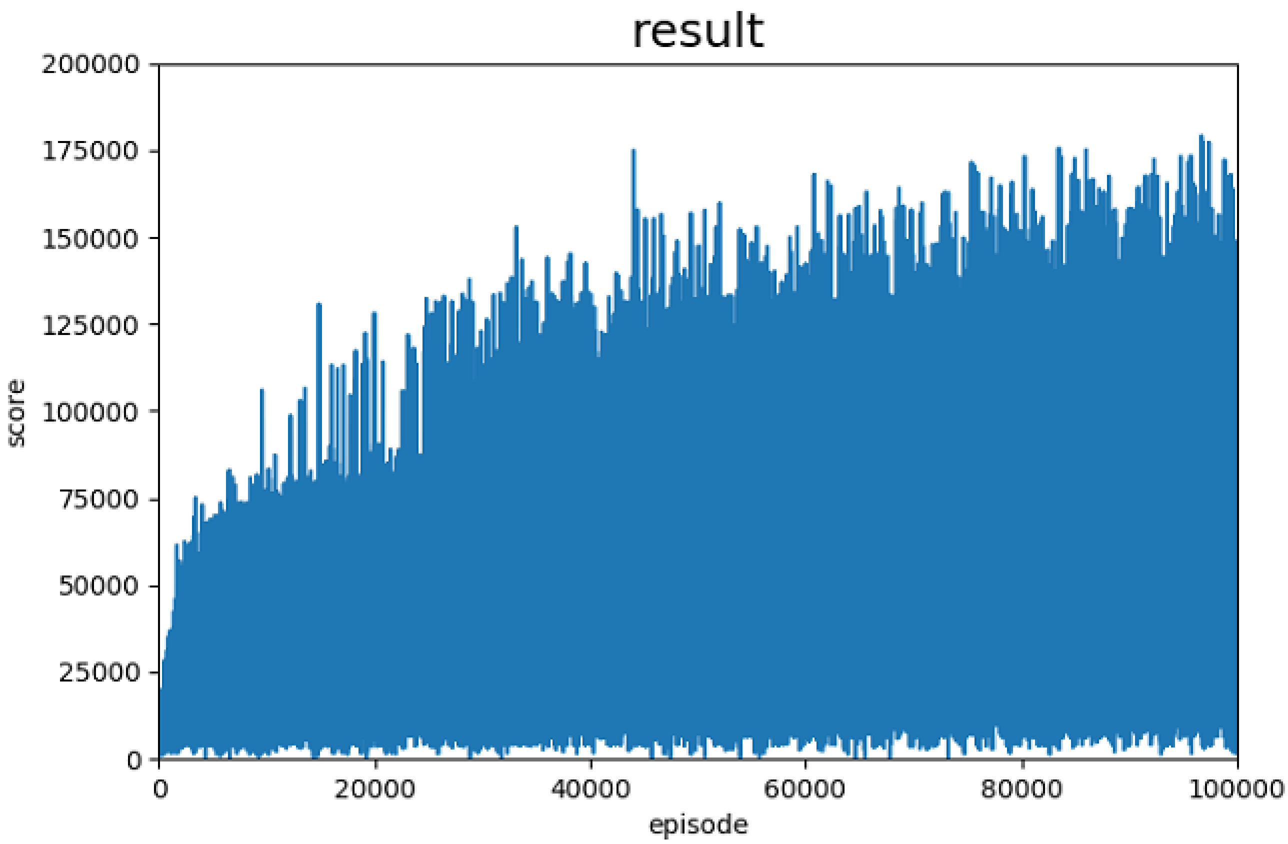
DLP_LAB3_310605001_王盈驊

LAB3

tags: `deep` `learning`

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



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

When we calculate the value state function, we must know the value on the board. Each board has 16 chunks. If the largest tiles is $2^{15} = 32768$, we have $16^{16} = 1.6 * 10^{9}$ state, which include posibility of empty space. It will consume lots of momory space. So we need to us n-tuple to reduce parameter we need to store.

Base on the past reaserch, we can only pick some chunks to be the useful feature we evaluate the state. On the other hand, the memory problem can also be solve. If we only pick 6-tuple as our features, we only need to maintain only $12^{6} = 3 * 10^{6}$ memory space.

Also we will aply the feature to the clockwise rotated board, and that we can have different isomprphic features.

The index of :

0	**1**	**2**	**3**
4	5	6	7
8	9	10	11
12	13	14	15

If we pick {0,1,2,4,5,6} for example:

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

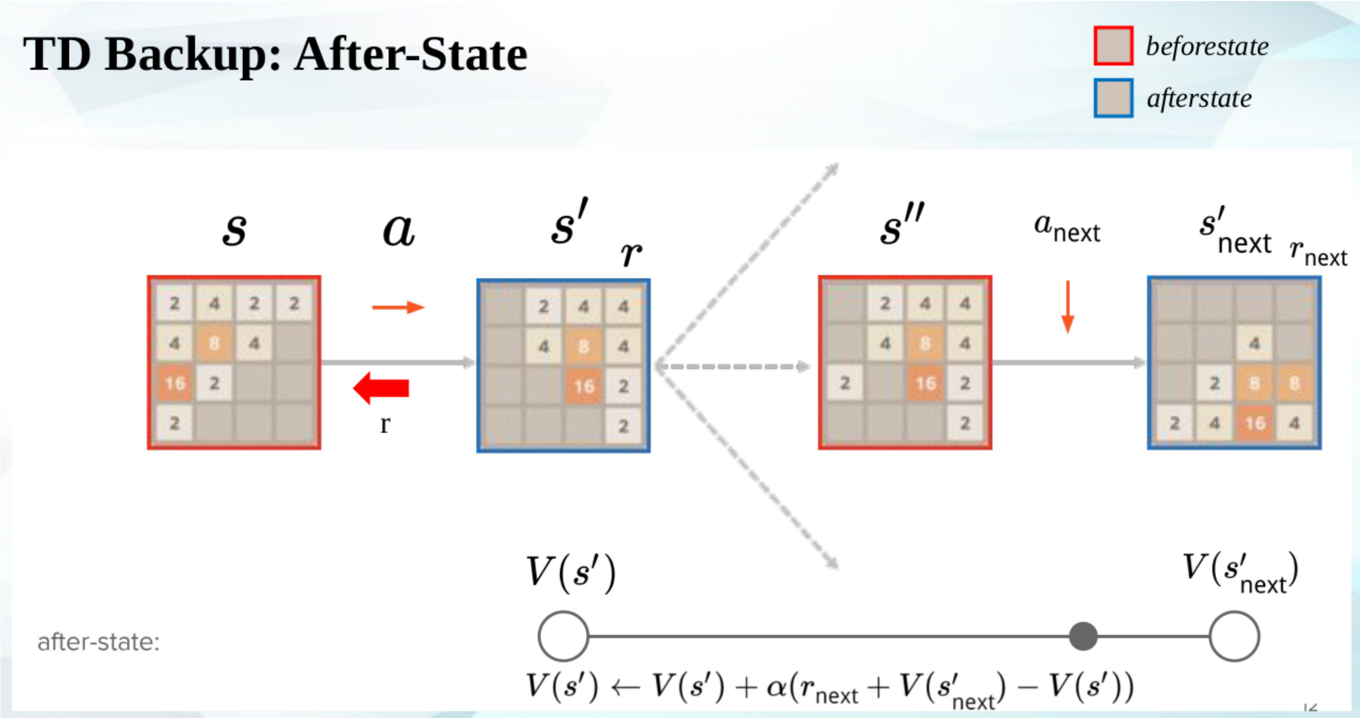
Temporal difference (TD) learning refers to a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function. These methods sample from the environment, like Monte Carlo methods, and perform updates based on current estimates, like dynamic programming methods.

While Monte Carlo methods only adjust their estimates once the final outcome is known, TD methods adjust predictions to match later, more accurate, predictions about the future before the final outcome is known.

TD(0) is the spacial case of TD(λ), and also means it will look ahead one step.

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

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



This is the state transition of 2048. s mean the borad state at now, $s^{'}$ represent the state sfter state s do a action. $s^{''}$ represent $s^{'}$ add random tile at the one of empty chunk on the board. r is the value after we take a action at the state s. r_{next} is the value after we take a_{next} action at the state s''. After state mean we evaluate the reward by after state $V(s^{'})$, r_{next} and the $V(s^{'}_{next})$ to determine the learning score. The formula is show on figure.

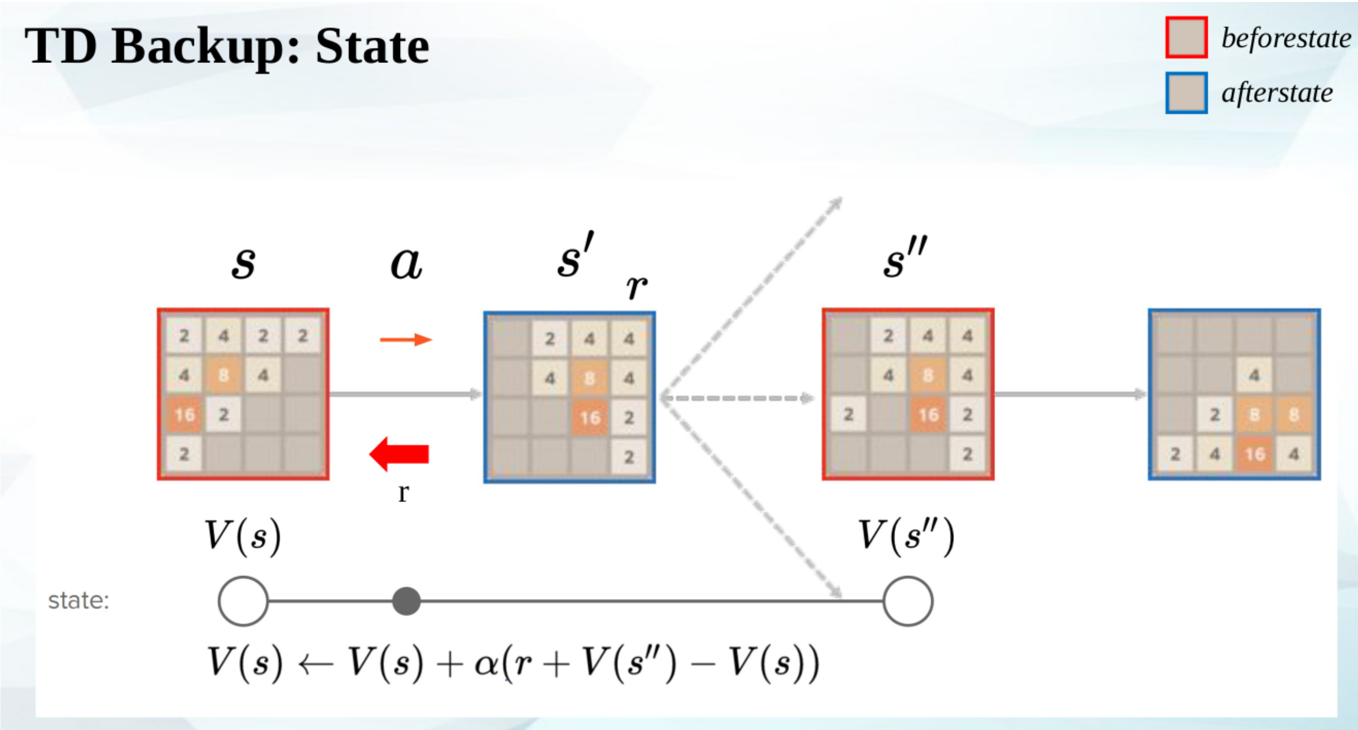
Explain the action selection of V(after-state) in a diagram. (5%)

To find the best action, agent will select the action by the following formula.

$$\pi(s) = \underset{a \in A(s)}{\operatorname{argmax}}[R(s,a) + V(T(s,a))]$$

First, we will go through every possible action and evaluate the value of the state after this action. We will combine the value of action $R(s,a)$ and value of state $V(T(s,a))$, where $T(s,a)$ mean the state $V(s^{''})$ after do a from previous state $V(s^{'})$, and find the largest one to determine the action we do.

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



State mean we evaluate the reward by state $V(s)$, r and the $V(s^{''})$ to determine the learning score. The formula is show on figure.

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

To find the best action, agent will select the action by the following formula.

$$\pi(s) = \underset{a \in A(s)}{\operatorname{argmax}}[R(s,a) + \sum_{s'' \in S} P(s,a,s'')V(s'')]$$

First, we will go through every possible action and evaluate the value of the state after this action. Also, we will consider the possibility of model transition from state $s^{'}$ to $s^{''}$, which mean we will consider every possible situation of the new tile being added.

We will combine the value of action $R(s,a)$ and value of state $\sum_{s'' \in S} P(s,a,s'')V(s'')$, where $P(s,a,s'')$ mean the posible state $V(s^{''})$ after the new tile add by model from previous state $V(s^{'})$, and find the largest one to determine the action we do.

Describe your implementation in detail. (10%)

select_best_move

This is the TODO part in select_best_move. Because in TD backup (state) we consider transition posibility of model, so I add line 7 - 11. The transition function can calculate with environment model, generating 4-tile and 2-tile is 1:9, and divide by the number of empty chunk.

```
/*return reward + sum */
std::vector<int> e = move->after_state().find_empty(); // check every
chunks
int mid = e.size(); // number of empty chunk
float val = 0;
for(int i = 0; i < mid; i++){
    board S_next = move->after_state();
    S_next.set(e[i], 1); // generate 2-tile
    val += estimate(S_next) * 0.9 / mid;
    S_next = move->after_state(); // generate 4-tile
    S_next.set(e[i], 2);
    val += estimate(S_next) * 0.1 / mid;
}
move->set_value(move->reward() + val);
```

find_empty

We need to check every empty chunk to find the transition function.

```
std::vector<int> find_empty() {
    std::vector<int> result;
    for (int i = 0; i < 16; i++)
        if (at(i) == 0) {
            result.push_back(i);
        }
    return result;
}
```

feature I add

```
tdl.add_feature(new pattern({ 0, 1, 2, 3}));
tdl.add_feature(new pattern({ 0, 1, 5, 6}));
tdl.add_feature(new pattern({ 0, 1, 2, 5}));
tdl.add_feature(new pattern({ 0, 1, 4, 5}));
tdl.add_feature(new pattern({ 1, 2, 5, 6}));
tdl.add_feature(new pattern({ 4, 5, 6, 7}));
tdl.add_feature(new pattern({ 0, 1, 2, 5, 9}));
tdl.add_feature(new pattern({ 1, 2, 3, 6, 10}));
tdl.add_feature(new pattern({ 4, 5, 6, 9, 10}));
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({ 0, 4, 5, 8, 9, 10 }));
```

Other discussions or improvements. (5%)