Lab 3: TD 2048

310551083 許嘉倫

1 Lab objective

In this lab, we will learn temporal difference learning (TD) algorithm by solving the 2048 game using an n-tuple network.

1.1 Game Environment

- Introduction: 2048 is a single-player sliding block puzzle game. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048.
- Actions: Up, Down, Left, Right
- Reward: The score is the value of new tile when two tiles are combined.

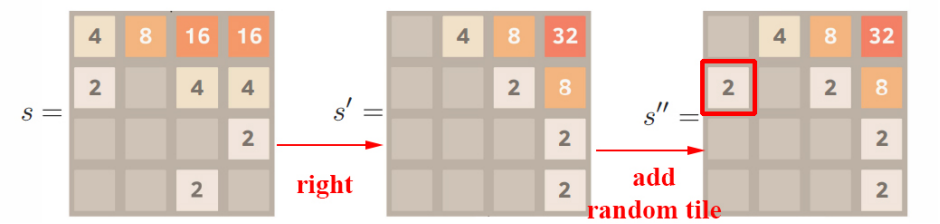
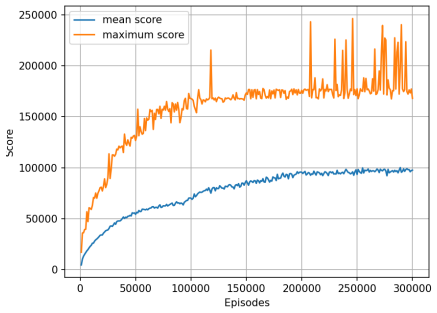
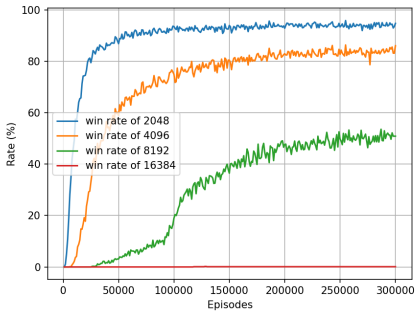


Figure 1: A sample of two-step state transition

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



(a) score in 300K episode



(b) win rate in 300K episode

Figure 2: score of 2048 game

3 n-tuple network

Because this board has 16 tiles and each tile has 16 kind of value included $0, 2, ...2^{15}$, we need very large memory to store all possibilities. To avoid this situation, n-tuple network can help us to extract feature in current board and this feature is composed of n tiles. Through this way, we can effectively reduce memory usage and get better result than before. We can use feature as index to find corresponding weights in look up table and summarize all isomorphic weight value. I design four 6-tuple network and two 4-tuple network to play the 2048 game. The figure 3 is my 6-tuple and 4-tuple network.

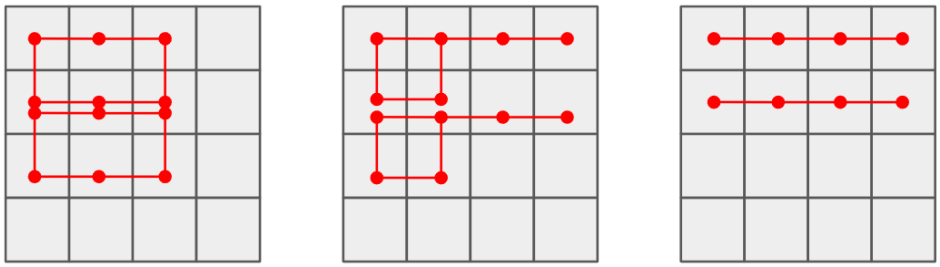


Figure 3: Design of 6-tuple and 4-tuple network

4 Mechanism of TD(0)

TD(0) learning need two adjacent state and reward to update its estimated value. So TD(0) learning dose not need the estimated value of other future state. Compared with Monte-Carlo learning, TD(0) learning has bias but get lower variance. We can see the mechanism in Figure 4. And the formula is shown below.

- Update value $V(S_t)$ toward estimated return $R_{t+1} + V(S_{t+1})$
  $V(S_t) = V(S_t) + \alpha(R_{t+1} + V(S_{t+1}) - V(S_t))$
- TD target: $R_{t+1} + V(S_{t+1})$
- TD error: $R_{t+1} + V(S_{t+1}) - V(S_t)$
- α : learning rate

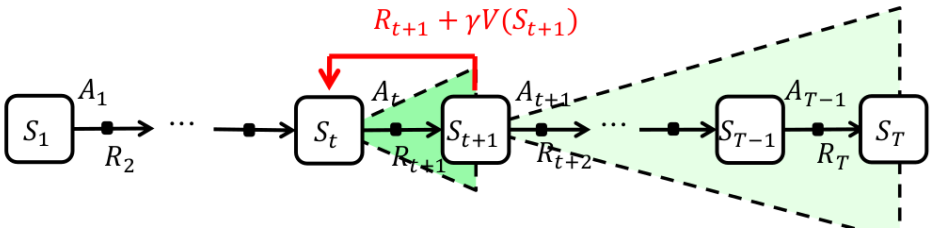


Figure 4: The mechanism of TD learning

5 TD-backup diagram of V(after-state)

Refer to figure 5 for a schematic diagram. The n-tuple network store weight of after state, so we need estimated value of two adjacent after state and reward to update weight of after state. The formula of TD-backup after state is shown below:
$V(S') = V(S') + \alpha(r_{next} + V(S'_{next}) - V(S'))$

- S': First after state which generate by an action a.

- $S^{'}_{next}$: Second after state which generate by an action a_{next}.
- TD target: $r_{next} + V(S_{next})$
- TD error: $r_{next} + V(S_{next}) - V(S^{'})$
- α : learning rate

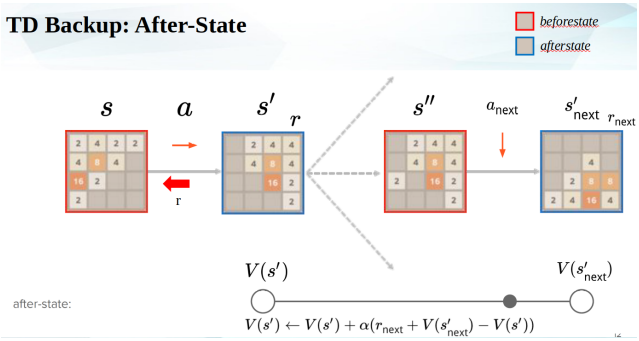


Figure 5: TD backup: After State

6 Action selection of V(after state)

We should compute which action (up, down, left, right) can get the maximum estimated value of current state by this formula $r + V(S^{'})$. And select an action which has the highest value as best move.

- r: A reward which the current state complete an action and get.
- $S^{'}$: An after state which generates by completing an action.
- $V(S^{'})$: estimated value of after state.

7 TD-backup diagram of V(state)

Refer to figure 6 for a schematic diagram. The n-tuple network store weight of before state, so we need estimated value of two adjacent before state and reward to update weight of before state. The formula of TD-backup state is shown below:
$V(S) = V(S) + \alpha(r + V(S^{''}) - V(S))$

- S: First before state.
- $S^{''}$: Second before state which environment popup new tile.
- TD target: $r + V(S^{''})$
- TD error: $r + V(S^{''}) - V(S)$
- α : learning rate

8 Action selection of V(state)

We should compute which action (up, down, left, right) can get the maximum estimated value of current state by this formula $r + mean(\sum V(S^{''}))$. And we need to consider all possible estimated value that environment popup new tile (90%: 2, 10%: 4) in each empty tile.

- $V(S^{''})$: A new state which environment popup new tile.
- $mean(\sum V(S^{''}))$: mean of all possible estimated value of state which environment popup new tile.

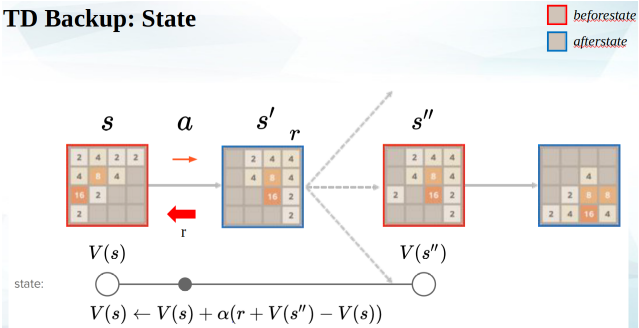


Figure 6: TD backup: State

9 Implementation

9.1 Value function

```
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;
}
```

Estimate the value of a given board. Summarize weights of the feature, a feature has 8 isomorphic at most. This function is calculate the weight sum of those isomorphic.

9.2 Update function

```
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;
}
```

Update the value of a given board, and return its updated value. u means the learning rate and iso_last means the the number of isomorphic. Look up the weight of isomorphic in table, then update its value.

9.3 Index of pattern

```
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;
}
```

Encode the pattern of a given board to index. Get the value of tile and then encode each value to 4-bit. Finally, use little endian to represent it.

9.4 Select best move

```
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 value_sum = 0;
      int num = 0;
      board after_state_board = move->after_state();
      for (int i = 0; i < 16; i++){
        if (after_state_board.at(i) == 0) {
          // 90% popup 2
          after_state_board.set(i, 1);
          value_sum += 0.9 * estimate(after_state_board);
          // 10% popup 4
          after_state_board.set(i, 2);
          value_sum += 0.1 * estimate(after_state_board);
          after_state_board.set(i, 0);
          num++;
        }
      }
      move->set_value(move->reward() + value_sum / num);
      if (move->value() > best->value())
        best = move;
      } else {
        move->set_value(-std::numeric_limits<float>::max());
      }
    debug << "test " << *move;
  }
  return *best;
}
```

We should compute which action (up, down, left, right) can get the maximum estimated value of current state by this formula $r + mean(\sum V(S^{''}))$. And we need to consider all possible estimated value that environment popup new tile (90%: 2, 10%: 4) in each empty tile.

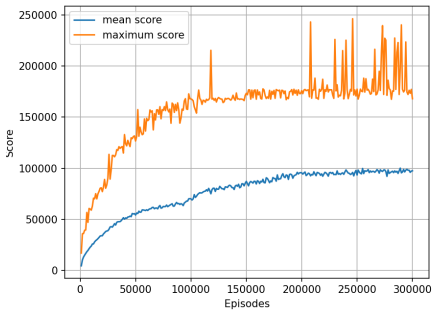
9.5 Update episode function

```
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());
    debug << "update error = " << error << " for before state" << std::endl << move.before_state
    exact = move.reward() + update(move.before_state(), alpha * error);
  }
}
```

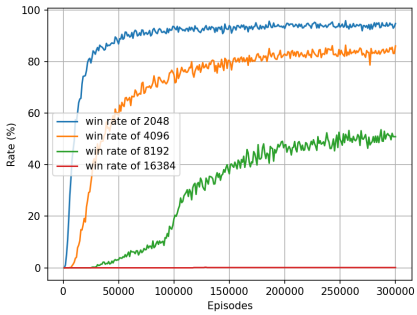
First, we need to pop the terminal state because the estimated value of terminal state is 0. Then we follow the formula below that we describe in section 7.
$V(S) = V(S) + \alpha(r + V(S^{''}) - V(S))$

10 Discussion

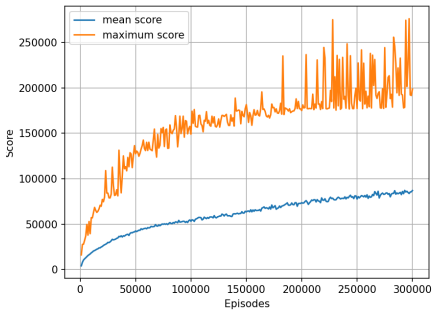
Compare with only four 6-tuple network, my n-tuple network as shown in figure 3 cost almost one day to train but get better result. This is because more pattern can learn more information in current board, so it need more time to train. And it also need more memory to store the weight of n-tuple network. The figure 7 is comparison of only 6-tuple network and my n-tuple network.



(a) score of four 6-tuple and two 4-tuple network



(b) win rate of four 6-tuple and two 4-tuple network



(c) score of only four 6-tuple network

(d) win rate of only four 6-tuple network

Figure 7: score of 2048 game