Deep Learning Practice Lab 7

0856056 Yu-Wun Tseng

June 2020

1 Introduction

The goal of this lab is to implement a temporal difference learning algorithm to solve 2048 game by using n-tuple network. 2048 is a single-player sliding block puzzle game and there are four actions the player can choose, which are up, down, left and right [1]. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048.

2 Experiment Setups

2.1 Implement Details

This lab is done by using c++ and using the code provided by TA [3].The model has five classes, including board, feature, pattern, state and learning.

The board class stores the board information and implements the move, popup and transformation functions. Each board information is stored using a unsigned long long integer, such as 0x4312752186532731ull. Each tile is stored using a hexadecimal integer, i.e., four bits. In addition, the tile doesn't store the exact number, but the power of two. Thus, the maximum number of each tile is 2^{16}, which is 65536. Since the information is not stored in array, the information can't be fetched directly by using index. Therefore, the bitwise operations are used to obtain the tile values [2].

The feature class is the parent class of pattern, it overloads the operators to get corresponding weight of the feature. There are three virtual functions, including estimate, update and name. The pattern class inherits the feature class, and overwrites the estimate, update and name functions. Each pattern will has eight isomorphic patterns, including the patterns after four rotations and one horizontal reflection. The estimate function calculates the values of all isomorphic patterns and returns the summation of all. Given the value to be updated, the update function will update the weights of all isomorphic patterns.

The state class wraps the states in episodes, each state has its before state, after state, value, score and opcode. The after state refers to the state after moving according to the opcode, and the score refers to the reward for moving according to the opcode. The assign function of state class applies the state to the board and establishes the corresponding attributes.

The learning class contains the training functions and has three attributes, including features, scores and maxtiles. The estimate function calculates the values of all features and returns the summation of all. Given the value to be updated, the update function will update the weights of all features. The select_best_move function establishes the states of the current board according to the four opcodes, and selects the state with highest value as the best state. The update_episode is the most important function, it calculates the TD error, updates the after state value according to the TD errors and finally calculates the expected value. The equation is as below:

$$\begin{aligned} error &= R_{t+1} + exact - V'(s_t) \\ V'(s_t) &= V'(s_t) + \alpha \times error \\ exact &= R_{t+1} + V'(s_t) \end{aligned}$$

The R_{t+1} refers to the reward for moving according to the opcode, the $V'(s_t)$ refers to the after state, the α refers to the learning rate and the *exact* refers to the expected value.

In this lab, the n-tuple network is used to represent features. There are four patterns in this lab, as shown in Figure 1 and the indices of features are shown in Table 1. Each pattern has its own isomorphic patterns, and each feature has its weight. The value of the state is the sum of all features.

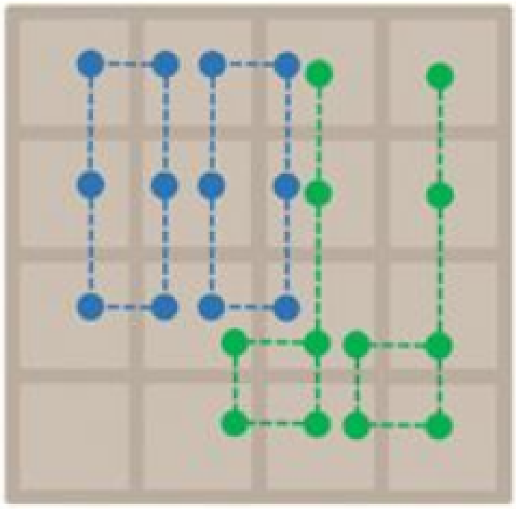


Figure 1: N-tuple Network

pattern 1	0	1	2	3	4	5
pattern 2	4	5	6	7	8	9
pattern 3	0	1	2	4	5	6
pattern 4	4	5	6	8	9	10

Table 1: N-tuple Patterns

2.2 TD Learning

In TD learning, the model doesn't need to finish the game and use the final reward to update the model, it adjust predictions to match the later state.

The V(state) is updated by the error between V(state) and V(next state). The V(next state) is the state after the action and the new values pop up on zero value tiles. The action is selected based on the V(next state) with the highest score. The equation is as below:

$$V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s)) \tag{1}$$

The V(after-state) is updated by the error between V(after-state) and V(next after-state). The V(next after-state) is the after-state after the action. The action is selected based on the V(next after-state) with the highest score. The equation is as below:

$$V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s')) \tag{2}$$

The TD learning is off-policy, the next action choice depends on the values of next state, not the policy. In addition, the TD-update performs bootstrapping, it uses $V(s_t)$ to update the $V(s_t)$.

3 Results

The training mean and max scores curves are shown in Figure 2, the scores are calculated every 1000 episodes and presented in log scale.

The 2048 win rate in 10000 episodes is 28.8% as shown in Figure 3, and the highest win rate of 2048 is 92.6% as shown in Figure 4.

4 Discussion

This lab using the code provided by TA. In the beginning, I wanted to write the code myself, but I found that I was not familiar with c++ code. Thus, I decided to understand the code first. After I understood it, I thought if the best state selection was made at a deeper layer, e.g., next next state of current state, it would have better performance. Consider the time and Lab 8, I will implement it later.

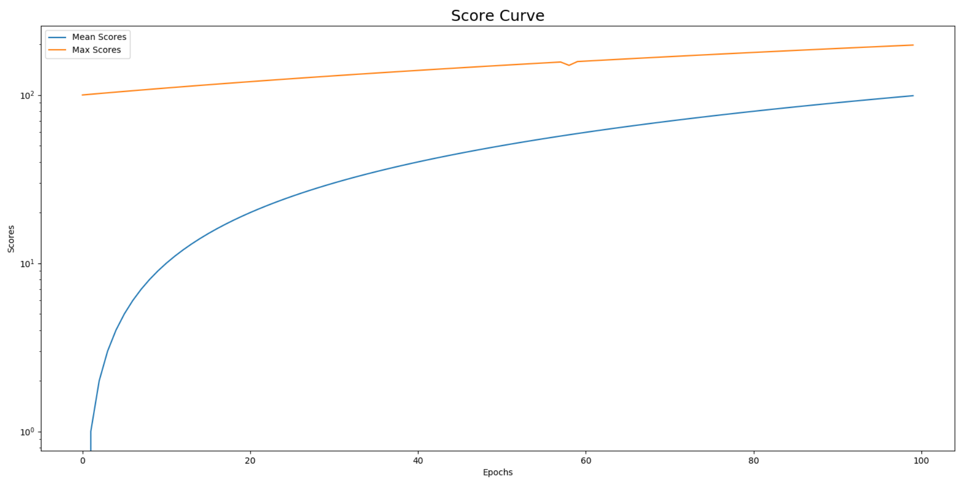
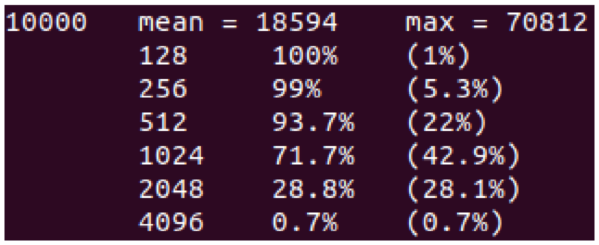
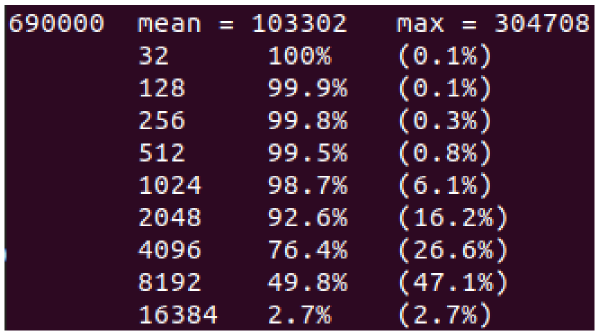


Figure 2: Score Curve



```
10000    mean = 18594     max = 70812
         128      100%     (1%)
         256      99%      (5.3%)
         512      93.7%    (22%)
         1024     71.7%    (42.9%)
         2048     28.8%    (28.1%)
         4096     0.7%     (0.7%)
```

Figure 3: Win rates in 10000 episode



```
690000   mean = 103302    max = 304708
         32       100%     (0.1%)
         128      99.9%    (0.1%)
         256      99.8%    (0.3%)
         512      99.5%    (0.8%)
         1024     98.7%    (6.1%)
         2048     92.6%    (16.2%)
         4096     76.4%    (26.6%)
         8192     49.8%    (47.1%)
         16384    2.7%     (2.7%)
```

Figure 4: The highest win rate of 2048

References

[1] *2048*. URL: `https://en.wikipedia.org/wiki/2048_(video_game)`.

[2] *Bitwise operation*. URL: `https://en.wikipedia.org/wiki/Bitwise_operation`.

[3] *TDL2048-Demo*. URL: `https://github.com/moporgic/TDL2048-Demo/`.