Double DQN-based AI for Sichuan Mahjong

MSBD5011 Advanced Statistics
Term I Report – Xiangyu Wang

1. INTRODUCE

Mahjong is a tile-based game that originated in China. Sichuan Mahjong, Japan Mahjong, and international Mahjong are regarded as the most representative versions. In Sichuan mahjong, the game is played with a set of 108 tiles based on Chinese characters and symbols. Compared to other versions, Sichuan mahjong omits some tiles such as Honors and Bonus to make it easier, so there are only three suits in Sichuan Mahjong: Dots, Bamboo, and Characters.

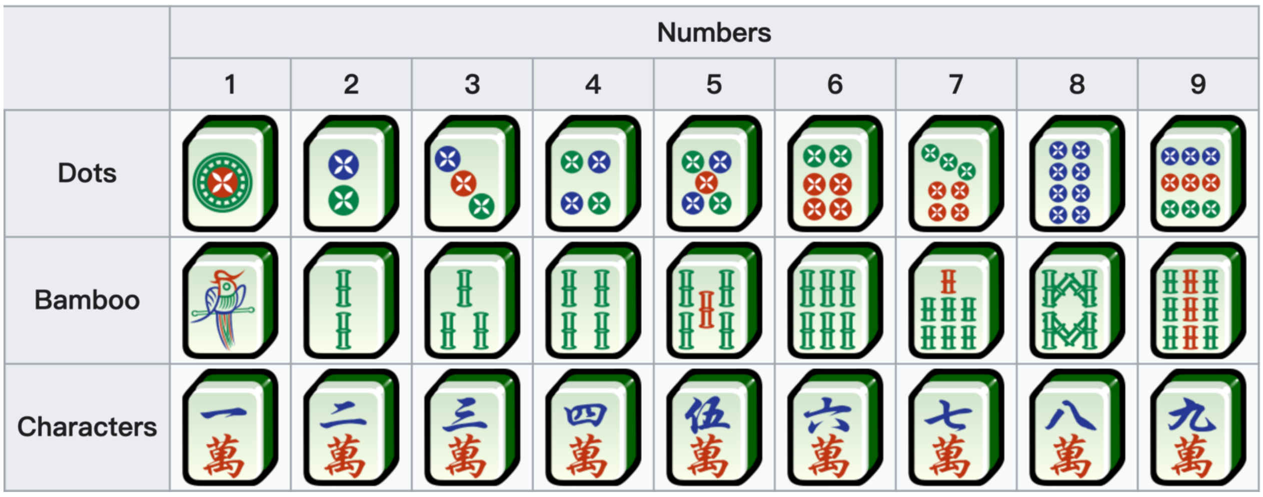

Figure 1: Unique Tiles in Sichuan Mahjong - Picture credit to Wikipedia

The 3 characteristics of Sichuan Mahjong are:

Battle to the bloody end: Hu does not end the game. Instead, the remaining players continue until there are three Hu or the remaining players left the round. All players' scores shall be computed at the end of the round. This way, the first winner may not get the most reward, and the player who fires the gun (Dian Pao) can turn over.

Voided Suit: All players have to void a suit immediately after dealing. During the game, the player cannot discard other tiles when he/she still has the tiles of the voided suit in his/her hand. If the player still has a tile of the voided suit at the end of the game, he/she will be penalized.

Chow claiming is not allowed: Only Pong and Kong claiming are allowed.

Unlike Go, Mahjong is a game with incomplete information. In other words, a player can only have partial information about opponents. Many common models, such as Monte Carlo Tree Search, do not give a satisfactory performance with such game design. This work attempts to develop a heuristic agent and use it to train a simple double Deep Q Network (DDQN) based agent. Then, we let the DDQN play with a heuristic agents and random policy agents to evaluate the performance.

2. RELATED WORK

Almost all games can be regarded as Markov Decision Processes (MDP)[1]. For a MDP, a deep Q network (DQN)[2] is a neural network that fits the optimal Q-function. So far, the most powerful DQN model for Japan Mahjong is Suphx[3] developed by Microsoft in 2019. Suphx uses 5-DQN model: Discard model, Riichi model, Chow model, Pong model, and Kong model, each of which is a complex Convolutional Neural Network (CNN) with 50+ layers. The rough idea of Suphx is shown as below:

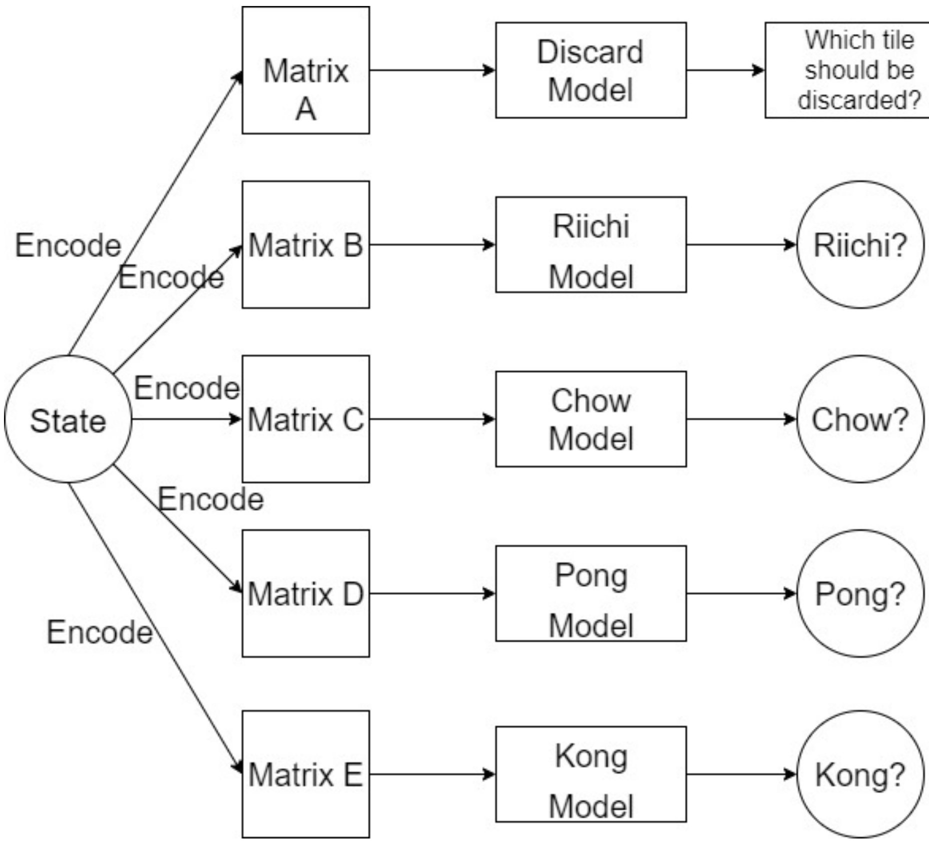

Figure 2: A rough overview of Suphx

The idea is simple, but the implement is very difficult and time consuming. As a beginner, we attempt to train a simplified model for Sichuan Mahjong, as follows:

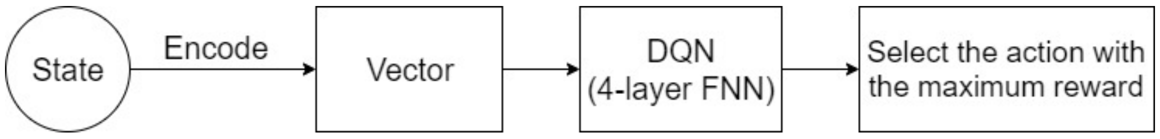

Figure 3: Overview of our DDQN Agent

3. METHODOLOGY

So far, few game data of Sichuan Mahjong can be found on the internet. Thus, we first build a naïve heuristic agent (rule-based) that behaves like a human. Next, we implement a Double DQN(DDQN) agent. To train the DDQN agent, a Mahjong Game with 1 DDQN agent and 3 heuristic agents is instantiated and repeated for 100,000 times. Finally, the DDQN agent is trained.

3.1. Heuristic Agent

The heuristic agent makes decisions like a human. The basic idea is: the more tile *t* in the hand, the more likely it is to be melded; the more tile *t* visible, the less likely you can get another tile t in the future. The rule of the Heuristic Agent is shown as below:

Voiding	After dealing, evaluates the possibility of being melded for each tile in the hand, then void the suit with the minimum total score.
Discarding	Evaluate the possibility of being melded for each tile in the hand, then discard the lowest scored one.
Pong/Kong	Do if possible.

Table 1: The rule of Heuristic Agents

3.1.1. The Probability of Being Melded

So far, there is no accurate method to calculate the probability that a tile in the hand can be melded. So, this work uses a probability score as the measure. It consists of 3 terms: the score of Pong, the score of Shun(顺子), and the score of future.

$$S(t) = S_p(t) + S_s(t) + S_f(t)$$

In the next sub-sections, the variables are defined as below:

$H(t)$: The number of tile *t* in the hand.
$V(t)$: The number of tile *t* visible, i.e. the sum of $H(t)$ and total number of tile *t* dropped/melded by all players.
$I(t)$: Is at least one tile *t* in the hand? Yes = 1, No = 0.

The score of Pong

$S_p(t)$ describes the probability that tile *t* can be melded into a Pong/Kong.

If $H(t) \geq 3$, tile *t* has already formed a Pong/Kong, so we define:

$$S_p(t) = I_p \times \frac{H(t)}{3},\ I_p \text{ is a given parameter}$$

If $H(t) = 2$, consider tile *t* as a potential Pong/Kong. As $V(t)$ increases, the probability that you can draw/Pong/Kong a tile *t* in the future decreases, so we define:

$$S_p(t) = I_p \times \frac{H(t)}{3} \times \frac{5 - V(t)}{3}$$

When $V(t) = 4$, it is impossible to draw/Pong/Kong in the future. However, the agent still gives a score of $\frac{2}{9}I_p$ to tile *t* because a Pair is valuable.

If $H(t) < 2$, the agent does not consider *t* as a potential Pong/Kong.

Combine these 3 cases:

$$S_p(t) = I_p \times I[H(t) \geq 2] \times \frac{H(t)}{3} \times max\left(\frac{5 - V(t)}{3}, I[H(t) \geq 3]\right)$$

The score of Shun

$S_s(t)$ describes the possibility that tile *t* can be melded into a Shun.

Tile *t* can be melded into at most 3 patterns: $(t-2, t-1, t)$, $(t-1, t, t+1)$, and $(t, t+1, t+2)$, so each of them contributes to $S_s(t)$.

Now consider pattern $(t-1, t, t+1)$: If $\sum_{i=t-1}^{t+1} I(i) = 3$, then it already exists, so we define:

$$S_s(t-1, t, t+1) = I_s \times \frac{\sum_{i=t-1}^{t+1} I(i)}{3},\ I_s \text{ is a given parameter}$$

If $\sum_{i=t-1}^{t+1} I(i) = 2$, then regard pattern $(t-1, t, t+1)$ as a potential Shun. For tile $t' \in \{t-1, t, t+1\}$ and $I(t') = 0$, the probability that pattern $(t-1, t, t+1)$ can be melded is negatively related to $V(t')$. When $V(t') = 4$, it is impossible that pattern $(t-1, t, t+1)$ can be formed. Thus, the formula should give a score of 0:

$$S_s(t-1, t, t+1) = I_s \times \frac{\sum_{i=t-1}^{t+1} I(i)}{3} \times \left(1 - \frac{V(t')}{4}\right)$$

Or:

$$S_s(t-1, t, t+1) = I_s \times \frac{\sum_{i=t-1}^{t+1} I(i)}{3} \times \prod_{j=t-1}^{t+1} max\left(I(j), 1 - \frac{V(j)}{4}\right)$$

If $\sum_{i=t-1}^{t+1} I(i) < 2$, the agent does not consider *t* as a potential Shun:

Combine these 3 cases:

$$S_s(t-1, t, t+1) = I_s \times I\left[\sum_{i=t-1}^{t+1} I(i) \geq 2\right] \times \frac{\sum_{i=t-1}^{t+1} I(i)}{3} \times \prod_{j=t-1}^{t+1} max\left(I(j), 1 - \frac{V(j)}{4}\right)$$

For pattern $(t-2, t-1, t)$ and $(t, t+1, t+2)$, we use the same method, so:

$$S_s(t) = \sum_{i=t-1}^{t+1} S_s(i-1, i, i+1)$$

$$= I_s \times \sum_{i=t-1}^{t+1} I\left[\sum_{m=i-1}^{i+1} I(m) \geq 2\right] \times \frac{\sum_{j=i-1}^{i+1} I(j)}{3} \times \prod_{k=i-1}^{i+1} max\left(I(k), 1 - \frac{V(k)}{4}\right)$$

The score of Future

To evaluate the discrete tiles, $S_f(t)$ is applied. The bigger $V(t)$ is, the less possible it is that *t* can be melded in the future, so we define:

$$S_f(t) = I_f \times \left(1 - \frac{V(t)}{4}\right),\ I_f \text{ is a given parameter}$$

Also, a lower $S_f(t)$ means more safety, i.e., a lower probability that *t* is Ponged/Konged by opponents.

3.1.2. The selection of parameters (I_p I_s and I_f)

When comparing two group of parameters P_1 and P_2, a Mahjong game with 4 players in which 2 are controlled by P_1 and others are by P_2 should be instantiated. The Game is repeated many times. If the average score of P_1 is positive, then P_1 is better. Otherwise, P_2 is better.

To select the suitable parameters, many games are played. In this work, each game is repeated 10,000 times. The results of experiments are shown as below.

P_1	P_2	Average score of P_1
$I_p = 1,\ I_s = 1,\ I_f = 1$	$I_p = 2,\ I_s = 1,\ I_f = 1$	-9.0
$I_p = 2,\ I_s = 1,\ I_f = 1$	$I_p = 3,\ I_s = 1,\ I_f = 1$	-7.0
$I_p = 3,\ I_s = 1,\ I_f = 1$	$I_p = 4,\ I_s = 1,\ I_f = 1$	-4.0
$I_p = 4,\ I_s = 1,\ I_f = 1$	$I_p = 5,\ I_s = 1,\ I_f = 1$	3.0
$I_p = 4,\ I_s = 1,\ I_f = 1$	$I_p = 4,\ I_s = 2,\ I_f = 1$	-8.0
$I_p = 4,\ I_s = 2,\ I_f = 1$	$I_p = 5,\ I_s = 2,\ I_f = 1$	-4.0
$I_p = 5,\ I_s = 2,\ I_f = 1$	$I_p = 6,\ I_s = 2,\ I_f = 1$	-3.0
$I_p = 6,\ I_s = 2,\ I_f = 1$	$I_p = 7,\ I_s = 2,\ I_f = 1$	-1.0
$I_p = 7,\ I_s = 2,\ I_f = 1$	$I_p = 8,\ I_s = 2,\ I_f = 1$	0.0
$I_p = 7,\ I_s = 2,\ I_f = 1$	$I_p = 7,\ I_s = 3,\ I_f = 1$	0.0
$I_p = 7,\ I_s = 2,\ I_f = 1$	$I_p = 7.5,\ I_s = 2.5,\ I_f = 1$	-1.0

Table 2: The comparison of Heuristic Agents of different parameters

Thus, the following parameters are selected:

$$\begin{cases} I_p = 7.5 \\ I_s = 2.5 \\ I_f = \ \ 1 \end{cases}$$

In the next experiments, the parameters of a Heuristic agent is set as above if no special instructions.

3.2. DDQN Based Agent

An ideal DQN θ should be:

$$\theta = arg\ min_w\ E[(r(s,a) + \gamma \times max(w(s')_{a'}) - w(s)_a)^2]$$

The gradient is not continuous because a' may change after each gradient descent (GD). As a result, the learning is unstable. To solve this problem, DDQN model keeps 2 DQNs θ and θ'. When learning, θ is used to selecting the optimal action a' and θ' is used to estimate the reward of action a' to calculate the gradient. In every time period, θ' is synchronized to θ.

When our DDQN agent behaves, it encodes the current state to a vector/matrix s and calculate $\theta(s)$, then it selects the legal action with the maximum reward.

3.2.1. Encoding and Layers

In this work, the state is encoded to a 294-dimension vector:

Term	Size
Suit dropped	3
Tiles in the hand	27
Tiles in the ground	28(27 tiles + Hu)
Tiles discarded	27
Suit dropped by the opponents	3*3
Tiles in the ground of opponents	28*3
Tiles discarded by others	27*3
Legal Actions	35(Action Size)

Table 3: The encoding of state

The DQN is a 4-layer Feedforward Neural Network (FNN) with 91875 parameters:

Layer	Units
0(Input)	294
1	192
2	128
3	64
4	35

Table 4: The structure of DQN

3.2.2. Training

The reward is the actual score obtained after taking an action. When training, experience replay is used to make the learning stable, which is shown as below:

```
$\gamma = 0.95$, $\epsilon = 1$, $\alpha = 0.001$
$b = 32$, memory size = 2000
Initialize two DQNs $\theta'$ and $\theta$
for $i = 1$: 10000
  reset the game
  while the game does not finish:
    $s$ = current state
    $r$ = random float between 0 and 1
    if $r < \epsilon$:
      randomly select an action $a$
    else:
      select action a = $argmax_a \theta(s)_a$
    get the corresponding score $r$ and enter new state $s'$
    add tripe ($s, a, r, s'$) to the memory
    sample a batch $B$ of $b$ triples from the memory
    for $(s_b, a_b, r_b, s'_b)$ in $B$:
      $a' = argmax_{a'} \theta(s_b)_{a'}$
      $\theta = \theta - \alpha \nabla_\theta \left( r(s_b, a_b) + \gamma \times \theta'(s'_b)_{a'} - \theta(s_b)_{a_b} \right)^2$
    if $\epsilon \geq 0.01$:
      $\epsilon = 0.999\epsilon$
  $\theta' = \theta$
```

4. EXPERIMENT

After completing the training, we conducted 3 different groups of experiment: 2 DDQN vs 2 Random, 2 DDQN vs 2 Heuristic, and 1 DDQN vs 2 Heuristic vs 1 Random. In each experiment, the same set of agents will be competing against each other for 10 000 games and the performance of each type of agent will be evaluated by the average scores obtained. The results are shown as below:

4.1. RESULT

2 DDQN vs **2 Random**

Agent	Average Score
DDQN	5.0
Random	-5.0

Table 5: The result of 2 DDQN vs 2 Random.

2 DDQN vs **2 Heuristic**

Agent	Average Score
DDQN	-12.0

Heuristic	12.0

Table 6: The result of 2 DDQN vs 2 Heuristic

1 DDQN vs **1 Heuristic** vs **1 Heuristic ($I_p = 1,\ I_s = 1,\ I_f = 1$)** vs **1 Random**

Agent	Average Score
DDQN	-7.0
Heuristic	22.0
Heuristic ($I_p = 1,\ I_s = 1,\ I_f = 1$)	-5.0
Random	-10.0

Table 7: The result of 1 DDQN vs 1 Heuristic vs 1 Heuristic ($I_p = 1,\ I_s = 1,\ I_f = 1$) vs 1 Random

4.2. DISCUSSION

The experiment shows that both Heuristic agent and DDQN agent perform better than the random agent. Given that the DDQN-based agent has been trained for 100,000 games against our heuristic agent. Such results are expected.

However, the experiment suggests that the DDQN agent does not give a satisfactory performance even though the training of the DDQN agent is done against 3 heuristic agents. As section 2 says, one of potential reasons for the DDQN agent giving poor performance in comparison with our heuristic agent is that our DQN model is so simple that it cannot fit the data well. Another one is that the reinforcement training process is too slow for our DDQN agent to improve enough in 10 000 games.

5. CONCLUSION AND FUTURE WORK

This work develops a heuristic agent and use it to train a simple DDQN agent. The result shows that our DDQN agent does not give a satisfactory performance. In contrast, the heuristics model works well. Still, enhancement works could still be done on both the DDQN-based agent and the heuristic agent.

For the heuristic agent, a potential improvement would be to give priority to the melds belonging to the same suit, which allow the player to obtain a higher score. Also, the agent should give the suit voided by more opponents a lower score to reduce the possibility of being Konged/Ponged.

As you see, this is a beginner level work. Our DQN model only contains a simple 4-layer FNN. To improve the performance, we plan to divide the model into 4 ones (i.e., discard/Pong/Kong/void Model) and use multiple complex networks such as CNN and LSTM instead in the future. Also, the DDQN agent could also be first trained through supervised learning to mimic the behavior of our heuristic agent to skip the lengthy training time for well-known playing techniques and focus on techniques which would better be explored through the reinforcement training procedures.

6. ACKNOWLEDGEMENTS

This work is implemented on Lisa's Sichuan Mahjong code. Some of DDQN code refer to Keon's blog[4].

7. REFERENCE

[1] Bellman, R. (1957). "A Markovian Decision Process". Journal of Mathematics and Mechanics. 6 (5): 679–684.

[2] van Hasselt, Hado; Guez, Arthur; Silver, David (2015). "Deep reinforcement learning with double Q-learning". AAAI Conference on Artificial Intelligence: 2094–2100.

[3] Li, Junjie; Koyamada, Sotetsu; Ye, Qiwei; Liu, Guoqing; Wang, Chao; Yang, Ruihan; Zhao, Li; Qin, Tao; Liu, Tie-Yan; Hon, Hsiao-Wuen(2020). "Suphx: Mastering Mahjong with Deep Reinforcement Learning". arXiv:2003.13590

[4] Keon (2017). "Deep Q-Learning with Keras and Gym":
https://keon.github.io/deep-q-learning/