INSTITUTE FOR INTERDISCIPLINARY INFORMATION SCIENCES, TSINGHUA UNIVERSITY

Deep Learning Project Report
Chuanchuan – An AI for Sichuan Mahjong

Zongtai Li, 2020010902
Yifeng Liu, 2020010909
Haocheng Xi, 2020010871
June 21, 2022

ABSTRACT

Games are abstractions of the real world, where artificial agents learn to compete and cooperate with other agents. Artificial Intelligence (AI) has achieved great success in various perfect- and imperfect- information games. Sichuan Mahjong is a popular multi-player imperfect-information game, especially in China. It has even become a culture, which is deeply ingrained in the Chinese community. It is a very challenging task for AI with competition, collaboration, imperfect information, large state space, and a massive set of actions space. In this work, we design an AI for Sichuan Mahjong, named "Chuanchuan", based on a deep Q network, action encoding, and hierarchical auxiliary Helper Agent System. Starting from scratch with only one GPU, Chuanchuan performed well after training for several hours. Our research shows that it may have learned something rational and useful by a short period of training, but it may need far longer training to beat the experienced.

1 INTRODUCTION

Games are regarded as benchmarks of AI since they are abstractions of many real-world problems. Lots of achievements have been made in perfect-information games in the past few decades, such as DeepBlue[2], AlphaGo[6], AlphaZero[7]. Recent research has evolved to more challenging imperfect-information games, where the agents compete or cooperate in a partially observable game environment. There are also a few encouraging works. OpenAI released OpenAI Five[5], learned by playing over 10000 years of games against itself, beat human world champions in 2019, which becomes the first AI to beat the world champions in an esports game. Brown and Sandholm developed a program, dubbed Pluribus[1], that learned how to play six-player no-limit Texas hold'em by playing against five copies of itself.

Sichuan Mahjong, a traditional Chinese game, is the first choice for the elderly in China for leisure. In each round, four players compete with each other towards the completion of a winning hand. Building an AI for Sichuan Mahjong is quite challenging. In Sichuan Mahjong, there are 108 tiles in total and each player only has 13 private tiles in his/her hand. There is a public discard pile, storing cards discarded by each player. All the information that a player knows is his/her hand tiles and the public discard pile. They cannot see the hands of the other players. As a result, it's really hard to guess other players' hands. It's quite difficult for a player to determine whether to attack or to defend since the best action highly depends on the hands of other players. What's more, there are lots of possible winning hands. Different winning hands are also worth different scores. Naturally, those winning hands which can get high scores are more difficult to achieve. Players also need to balance between conservative and aggressive strategies, to carefully choose what kind of winning hand to form.

J. Li et al. have proposed the famous Suphx[3] for Japanese Mahjong, which is quite similar but with different rules. However, due to the high difficulty, there is no famous public AI model for Sichuan Mahjong so far.

In our work, we build Chuanchuan, an AI model for Sichuan Mahjong. It is based on RLCard [10], a toolkit for Reinforcement Learning in card games. Chuanchuan adopts a deep Q network as its model. It is first trained through learning the actions of a fixed-policy model and then trained by self-play reinforcement learning. The model is trained for about twelve hours with only one GPU. Then the model is evaluated by battling some other fixed-policy agents. (We wrote several such agents and they were ranked according to their strength.) Although our model cannot beat the strongest one which is at the same level as human players, the model can really beat some weaker agents.

Through Chuanchuan, we find that small models with DQN can be used to deliver good results in large-scale and complex card games which need to balance between attack and defense over huge state and action spaces. We also notice that for such difficult games as Sichuan Mahjong, models need to be trained for weeks and billions of epochs. We think it could have a better performance after a longer training time.

2 MODEL

We have tried several models, and finally decide to use Dueling DQN in our project.

Compared to normal DQN[4], Dueling DQN [8], is a useful variation. Instead of simply calculating Q-value from a Deep Neural Network, Dueling DQN divide the Q-value into two parts, value and advantage. Value can be viewed as the score of the current state, and advantage can be viewed as a score for each action that sums up to zero. In mahjong, the piles in hand and the pile discarded are relatively separated, thus fitting the Dueling DQN structure.

2.1 Preproccessment of Information

Information of the current state is translated to a dictionary and the input of the estimator networks is a $6 \times 27 \times 4$ tensor to represent the visible information of the player. The encoding method is raised by Microsoft [3]. In Sichuan Mahjong, there are 27 kinds of different cards, each with 4 copies, which gives the second and third axis of the representation. The encoding of the third axis is suggested to have multiple 1s instead of one-hot encoding to get denser inputs rather than sparse ones. For example, if the hand pile has three "one dot", then the column should be $1-1-1-0$ instead of $0-0-1-0$. The first axis respectively represents the state of the current hand pile, the state of the hikawa(the discarded piles), the state of reward, the state after the action, possible legal actions, done_batch.

2.2 Structure of Dueling DQN

In the original DQN network, the Q value of the state s and action a is

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha),$$

where the V and A are 2 the value network and advantage network with shared parameters θ, and $alpha$ and β are parameters of the network.

In Dueling DQN, the advantage network A is replaced by a normalized one where the normalization means that the average of the advantage is set to zero. This will accelerate the convergence rate and achieve better performance. Formally the Q value of the Dueling DQN is

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + [A(s,a;\theta,\alpha) - \sum_{a'} \frac{A(s,a';\theta,\alpha)}{|A|}].$$

And the reward function is defined as the Sichuan Mahjong rule with discrete integral values, where the detailed rule is presented in Appendix A.

For each of the networks, it consists of 4 shared convolution layers and 2 exclusive fully-connected layers connected with a ReLU layer(See Figure 2.1).

The loss function is computed with a composition of the current state, action, final reward, next possible states, the set of legal_actions, and the finality signal(done), which is shown in Figure 2.2.

In detail, the Q-values of the next states after legal actions are computed by Q estimator which is a neural network, and the best action is taken by the arg-max of these Q-values. And the Q-values for the next target are estimated by another estimator, and the target is computed by either the payoffs at the end of the game or the Q-values estimated by the

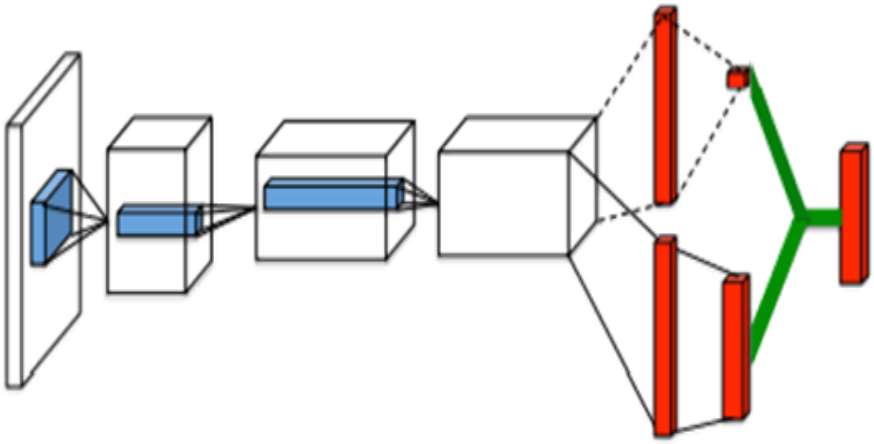

Figure 2.1: The structure of value and advantage networks .

model. And the total loss is the composition of the current state, the action, and the target by the Q estimator with simultaneous updates.

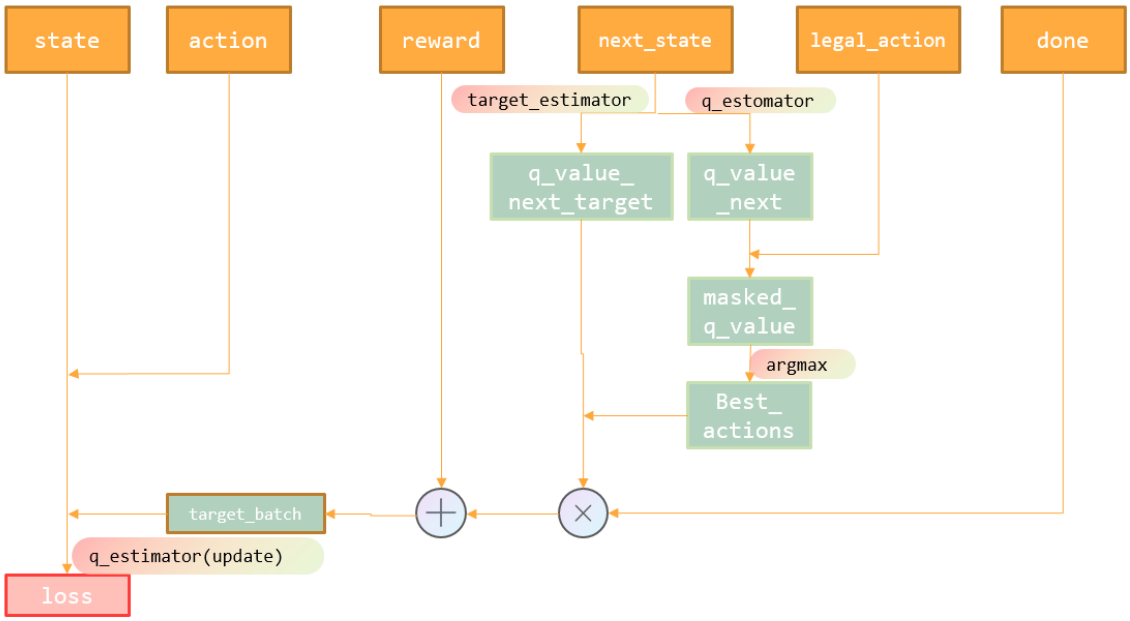

Figure 2.2: The structure of DQN model.

2.3 Helper Agent System

We have designed the Helper Agent System to assist in training. We have elaborated on 3 kinds of helper agents: Random Agent, Helper2, and Helper4. The random agent follows a pure random strategy: it discards each tile with the same probability and takes special actions(pong, gong, or hu) with a probability of 0.5; and the details of strategies of Helper2 and Helper4 are in Appendix B. And we have designed two successive parts of training procedures by using the Helper Agent System: in the first half of training, the records and payoffs of the games played among 4 helper agents are used to feed the model for initialization; and in the second half, the model plays games with 3 helper agents and in each epoch, we use either a winning round or a losing round for training alternatively to add to the flexibility of the model.

2.4 $\epsilon-$greedy Exploration Strategy

We applied the $\epsilon-$greedy strategy to balance exploration and exploitation. During the training process, we have probability $1-p$ to take the best action that gives the largest Q-value, and probability p to take random actions. The probability linearly decays from 1 to 0.1 through training.

In the first half of training, the model tries to learn the actions of an agent that is similar to a human. And in the second half, the model is trained to compete against itself to improve the result.

The Dueling DQN network can be chosen from a convolution or linear network, and the experiment indicates that linear neural network outperforms convolutional neural network, maybe because the input of the network does not have strong spatial information compared to pictures used in computer vision tasks.

3 Training and Visualization

3.1 Training process and training results

We trained the model for 60000 epochs with about 15 hours on one 3080Ti. The loss curve is shown in figure 3.1 and 3.2. The learning rate is set as 0.00005 and the batch size is 512.

We find that the loss may explode after about 3000 epochs, but we haven't found a feasible solution. What's more, larger or smaller learning rates are not good for model training.

We also find that the model can learn in the first half of training. However, in the second half, the model is trained from self-play but it seems that the performance sometimes becomes worse.

We tried to make the network more complicated. We added residual blocks in the network and also changed part of the structures. Adding residual blocks makes the training process slower and it doesn't improve the performance. We tried to reduce or add the number of parameters. There's a lot of variation in training time, but the model performance changes little. After balancing between training time and performance, we finally determine the final network structure.

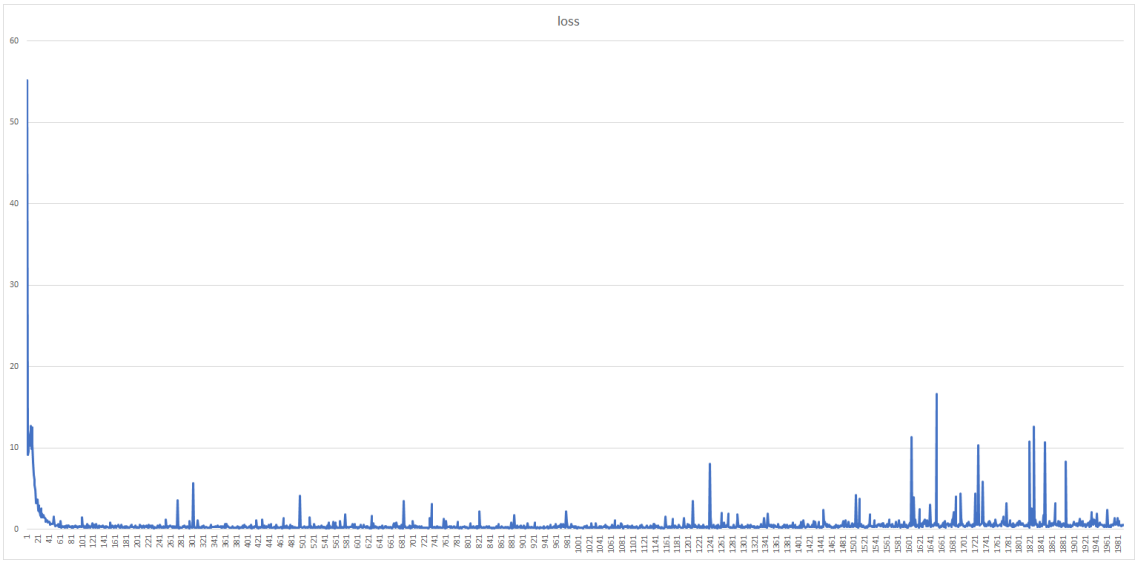

Figure 3.1: loss-timestep curve

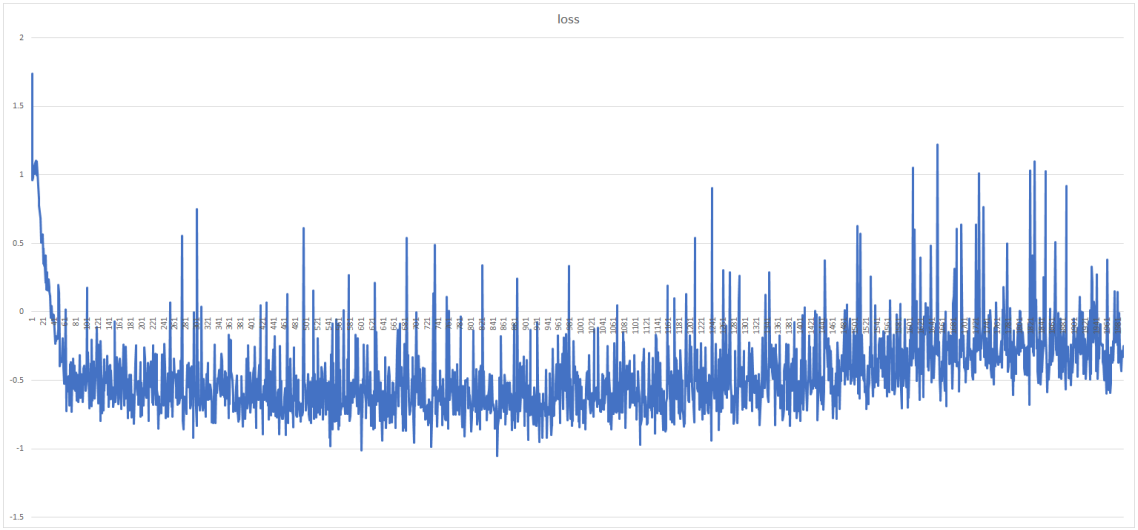

Figure 3.2: $\log_{10}$(loss)-timestep curve

Then we tried a variety of training patterns. The basic pipeline is to first train by the trajectories of some Helper agent and then by self-play. Since there are four players in a game, we need to decide which agent to use for each player. After exploring many possibilities, our final solution is as the following:

In the first half of the training process, HelperFour is playing against three RandomAgents. Our model learns from the actions of HelperFour. The reason to play against RandomAgents is simply to hu more. In the second half, four copies of our model play against each other. We separate the environment into a win_env and a lose_env. Win_env is to try playing until player 0 wins. Lose_env is to try playing until player 0 loses. The two environments alternate. The aim is to teach our model balancing between attacking and defending.

We have also written preprocessing Python programs and a Qt6 C++ program to visualize the results of games played between the trained model and our elaborated helpers with resources from Quehun Game, in which the detailed information of private tiles and folds of each player, the current player and the number of remaining cards are shown in the central widget, and the Figure 3.3 is a screenshot of our visualization result. All the programs are packed in the zip file and Qt6.1 is needed for visualization(executive file is visualization.zip/build-mahjongv-Desktop_Qt_6_1_2_MinGW_64_bit-Debug/debug/mahjongv.exe, and the preprocessed input file for the example of Section 3.2 is "example.txt" in the same directory of the executive file). This is for debugging, testing the model, and looking for what our model's strategy is.

Figure 3.3: The visualized result.(Resources from Quehun Game)

3.2 THE ANALYSIS OF MODEL BASED ON THE VISUALIZATION

With the help of visualization, we can analyze some specific games played between our model and the helper agents or even the humans to infer what our trained model may have learned. Figure 3.4 is a series of screenshots in a game between our model and Helper2. We can observe that from a to b, the model chooses to get out of the "dots" which is the least set of 3 sets, and then it got rid of the isolated "character-9"(c). After that it discarded "bamboos-6" and "bamboos-8"(d), aiming at setting "bamboos-7" as the sole couple. With the incoming third "bamboos-7", it chose to discard "character-6"(e) and tenpai(be waiting for winning). Finally, it wins(f). We can get that it seems to have learned something rational.

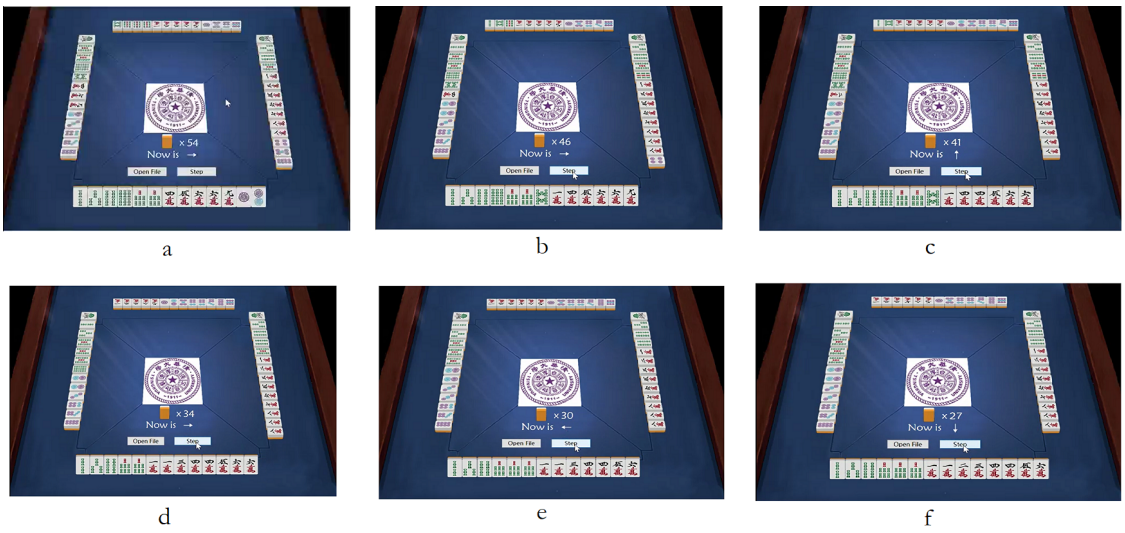

Figure 3.4: The visualized result of a specific game. The game starts from subfigure (a) and ends in subfigure (f).

4 EXPERIMENTS

The model is mainly evaluated via battling with some fixed-policy agents, named Helpers. These agents are ranked according to their strengths. For instance, the level 4 agent is designed similar to a human, while the level 2 simply follows a relatively trivial strategy. The details are listed in the appendix. Our model battled with each of the agents for 1000 independent random games.

4.1 RANDOMAGENT (HELPER0)

RandomAgent (Helper0) will randomly play some tile. When it can pong or gong, it will also choose randomly.

The results of battling with RandomAgent for 1000 rounds are shown in the table below.

Chuanchuan	RandomAgent	RandomAgent	RandomAgent
538	-142	-190	-206

Table 4.1: Average Result battling RandomAgent for 1000 rounds in 10 trials

Since randomly playing some tile can hardly win, so battling with RandomAgent only tests the ability to hu. From the result, we can see that our model can hu most of the time.

What's more, the result of Helper4 palying against RandomAgent is shown below.

Chuanchuan	RandomAgent	RandomAgent	RandomAgent
1836	-600	-600	-636

4.2 HELPER2

The strategy of Helper2 can be found in the appendix.

The results of battling with Helper2 for 1000 rounds are shown in the table below.

Chuanchuan	Helper2	Helper2	Helper2
303	-101	-101	-101

Table 4.2: Average Result battling Helper2 for 1000 rounds in 10 trials

Helper2 is quite a "clever" agent. We think it can achieve the level of a beginner. Our model beats Helper2 as well. Although this does not mean anything, we think our model has learned some of the logic of Sichuan Mahjong.

Chuanchuan	Helper4	Helper4	Helper4
-1668	755	1045	-132

Table 4.3: Average Result battling Helper4 for 1000 rounds in 10 trials

4.3 HELPER4

The strategy of Helper4 can be found in the appendix.

The results of battling with Helper4 for 1000 rounds are shown in the table below.

Yifeng Liu is from Chongqing and Sichuan Mahjong is quite popular there. He also plays Sichuan Mahjong very well. The strategy of Helper4 is copied from his logic. Thus, Helper4 can be seen as an experienced player. Our model loses, unfortunately.

4.4 TOGETHER

We also tried to put the four models together.

The results of battling for 1000 rounds are shown below.

Chuanchuan	RandomAgent	Helper2	Helper4
113	-515	-219	621

Table 4.4: Average Result in 10 trials

5 CONCLUSION AND FUTURE WORKS

We have designed an encoding method for Sichuan Mahjong and apply Dueling DQN to calculate the Q-value for state action pairs. We have also elaborated a hierarchical Helper Agent System and ϵ-greedy exploration strategy for auxiliary learning. And we also visualized the detailed situation of some specific games. Our trained model can defeat Helper2 Agents in most games but it can hardly defeat Helper4 Agents which is the essence of 15 years' experience of an old hand.

In the future, we can enhance our model through the following methods. Firstly, we can enlarge the feature numbers that feed the network and use larger, more intricate model structures. Secondly, We should also train for a longer time while keeping the training process stable. We may also apply a more delicate net structure rather than Dueling DQN, which is raised 6 years before. Moreover, we found some disadvantages of DQN structure on partially-observable games, including instability and inefficiency of training, and some other researchers have found other better structures like DouZero[9], which uses the Deep Monte-Carlo method. Thirdly, considering that Japanese Mahjong focuses on defending, we may also need to train the model to learn to defend(in other words, guess other players' hand pile and try not to fangchong.) given the discard information. Fourthly, more comparative experiments are needed for an exhaustive analysis of our model. We would like to know what strategy our model has learned. Fifthly, some human knowledge will make the model

stronger. In suphx, we can utilize human masters' games to train a model to judge the action. Sixthly, we can test our model on some platforms such as Happy Mahjong to test our model performance in the future.

REFERENCES

[1] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[2] IBM. Deep blue(https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/). 1997.

[3] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning, 2020.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[5] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[6] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[7] David Silver, Tomas Hubert, Julian Schrittwieser, and et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. 2016.

[8] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

[9] Daochen Zha, Xie. Jingru, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. Douzero: Mastering doudizhu with self-play deep reinforcement learning, 2021.

[10] Daochen Zha, Kwei-Herng Lai, Songyi Huang, Yuanpu Cao, Keerthana Reddy, Juan Vargas, Alex Nguyen, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A platform for reinforcement learning in card games. In *IJCAI*, 2020.

APPENDIX

A. THE RULE OF SICHUAN MAHJONG

The Sichuan Mahjong is a simplified version of standard mahjong, and the balance between agility and aspiration is the key to winning.

Detailedly, there are 3 sets, 27 kinds of tiles without wind, and honor tiles. The basic requirement for hu(winning) is to get rid of one set ("dots", "characters" or "bamboos") of tiles. The basic type of hu is to get a sole couple with multiple melds (triplets or 3-sequences), which values 1 point. And most of the other types are derived from it with some faans(the points are multiplied with powers of 2). The case that the melds are all triplets values 1 faan; the case that all tiles are in the same sets values 2 faans; the case of only one invisible tiles values 2 faans; the case when winning at the end of the pile wall values 1 faan; and each quadruplets values 1 faan. Especially, the type of seven couples values 2 faans. And the total number of faans is no more than 3, and the case when you get the last tile of hu by yourself values an additional point. And the payoffs of our model are strictly following the rules above.

B. THE STRATEGIES OF HELPER2 AND HELPER4

The strategy of Helper2 is that: firstly, it manages to pong or gong. Elsewise, it discards the one that has no neighbors in our hand piles(called isolated pile) or the card that has appeared on the table or the hand pile for 4 times(called dead pile). If they do not exist, it tries to discard a card that has only one neighbor while not a member of a three-sequence. Then it considers a card that constructs a couple or triplet.

The strategy of Helper4 is derived from a player with more than 15 years of experience in playing Sichuan Mahjong. The Helper4 firstly gets rid of one of the 3 sets with the least tiles, and it does not pong or gong tiles with the set. And we design an algorithm to compute the value of each tile and it gets rid of the one with the least value: the tile in triplets values 100 and the tile in couples values 40; the tile in 3-sequence values 80 and the tile in 2-sequence values 80 - 2 $\times$ the number of possible remaining tile to form 3-sequence; and the 3-sequence without intermediate tile values 40 - 10 $\times$ the number of possible remaining tile to form 3-sequence. And the Q-value for the action of discarding the tile is the average value of the other tiles.