修士論文

Study on Imperfect Information Game Record Learning in the Game of Mahjong via Deep Convolutional Neural Networks

(深層畳み込みニューラルネットワークを用いた
麻雀における不完全情報ゲームのデータの学習)

2019年2月4日

指導教員　　川原 圭博　准教授

東京大学 大学院情報理工学系研究科
電子情報学専攻

48-176449　高 世祺

Abstract

With the development of deep learning, artificial intelligence on games has also achieved a great breakthrough. AI level on the game of Go, which is recognized as the most complex human intellectual perfect information game, already reached an overwhelmingly high level that no human beings can even approach. On the other hand, imperfect information games are usually more complex than this since each player only knows one part of information from the whole. There have been many studies on imperfect information ones, but they only achieved parts of success.

In this thesis, we focus on imperfect information game record learning in the game of Mahjong. We propose our new models with a three-dimensional data structure which is designed for containing information available on game table. We utilize deep convolutional neural networks for supervised learning on game records and adopt agreement rate on discard tile as the benchmark. We achieve an agreement rate of 69.48% while the state-of-art result among past studies is just 62.1%. Based on this model, we try to make two improvements. We make a design of a sequential model for longer memories among one subgame. Under the condition that half of the training data is filled with 0 paddings, we achieve an accuracy of 68.14% on test. We also manage to conclude one part of off-table information missing in previous models which is rank and kyoku, and get an agreement rate of 70.44% on test which is the best result so far.

Finally, we build an AI player and make it playing online for evaluation by just simply combining several neural networks together and without any human knowledge given. The program achieves a rating of around 1850 after 300 matches online while the average rating of an intermediate player is around 1600.

Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Background

Deep learning has made remarkable achievements in various industrial and academic fields such as Computer Vision (CV), Natural Language Processing (NLP) and Reinforcement Learning (RL). The most popular and successful network structures used in deep learning include Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM).

The researches on Artificial Intelligence (AI) players on perfect information games are already highly advanced and the strength of AI has reached the level that no human beings can reach even in the game of Go, which is recognized as one of the most complex perfect information games, by the AI AlphaGo series in the year of 2017 [11, 12].

Compared with perfect information which refers that each player has the same information, there is still

a long way to go for imperfect information games since they are more complex and players need to balance all possible outcomes while making a decision. It is still very difficult for AI to find the best strategy when faced with situations of imperfect information.

The usual solution for imperfect information game solving is to achieve a Nash equilibrium situation [26]. The Nash equilibrium is a proposed solution for multi-player non-cooperative games in which each player is rational enough and no one will gain any additional benefits by only changing his own strategy. There has been a breakthrough [8, 9] of approaching Nash equilibrium points in no-limit Texas Hold'em, a two-player unlimited form of poker by using subgame solving and Counterfactual Regret Minimization (CFR) [27]. However, for the game of Mahjong, it has more players and much larger exploring space and complexity, so we aim at other solutions.

1.2 Purpose

In this work, we choose mahjong as our research tool due to its large complexity and popularity. Japanese Mahjong has a strict frame of rules for competitions, together with many high-level past game records which are called 'haifu' for the network to study. Agreement rate, the rate of the network choosing the same strategy with the real data records, is a very important benchmark for estimating the situation of the game which is also called the evaluation function. There are many traditional AI methods for the game of Mahjong, but they are mainly built by artificially extracting features and designing function blocks. By contrast, deep learning is recognized as the next generation method for game AI for its ability of automatic feature extraction and learning. Although there have been related studies in full-connected network [2] and CNN [3], they still cannot exceed the performance of traditional methods' due to their own limitations in both data structure and network training.

We propose a new method by using deep convolutional neural networks for the task. We design a new data structure with three dimensions in order to contain the information available on the table, and make fine training. We focus on the benchmark of agreement rate, and try to achieve a best accuracy for game record learning.

We show that finally our model achieves an agreement rate accuracy of 70.44%, which is much better

than those of previous traditional [4, 10] and deep learning models [3], demonstrating the potential of deep learning models on imperfect information tasks.

1.3 Contributions

The contributions of this work are listed below.

1. A new data structure designed for containing the game information.

   Normally, people use a one-dimensional matrix to represent for game information. In this thesis, we manage to build a three-dimensional data structure and it does well on containing information on table. With this data structure, our result of prediction accuracy is much higher than past studies.

2. A deep learning method for imperfect information game record learning.

   As mentioned before, deep learning has conquered perfect information games but for imperfect information ones it still has a long way to go. This thesis successfully provides a new approach of using deep learning tools for imperfect information game record learning. It shows the perfect understanding ability of deep convolutional neural networks.

3. Utilizing sequential models on game record learning.

   For previous works, there have few studies on how to utilize past memories to help present choice making. We propose a GRU model for sequential model learning and achieve a relatively good result.

4. AI player building with large potential.

   We make an AI program for playing online. By just combining several trained neural networks and without any human knowledge about the game, it already reaches a level higher than an average intermediate level player.

1.4 Thesis Organization

This thesis is organized as follows.

In Chapter 2, basic rules and terms for the game of Mahjong are introduced. Chapter 3 provides an introduction about related works in several fields, including deep learning, game of Go and game of Mahjong. In Chapter 4 we share our new model together with the data structure and network structure design on tile information learning and show that our result gets much better than past works. In Chapter 5 and Chapter 6 we explore two ways for improvements, which are obtaining longer memories and containing more off-table information. We build an AI program in Chapter 7 and show its strength. Finally Chapter 8 summaries the thesis with contributions and future work.

Chapter 2

Basic Rules and Terms of Japanese Mahjong

2.1 Basic Rules of Japanese Mahjong

Mahjong is a tile-based imperfect information game which has been developed in China since the Qing dynasty and has been further developed throughout the world since the early 20th century. It was first brought into Japan in 1924. The game is commonly played by four players.

Japanese Mahjong, also known as Richi Mahjong, is a popular variation. Japanese Mahjong has 136 tiles with 34 types which are shown in Fig. 2.1 and four same tiles in each type. The 34 types are 1m (man)–9m, 1p (pin)–9p, 1s (sou)–9s, and seven honor tiles: East, South, West, North, Haku, Hatsu and Chun. The

tiles are mixed and then arranged into four walls that are each two stacks high and 17 tiles wide. 26 of the stacks are used for starting hands, while 7 are used for a dead wall and the remaining 35 stacks form the playing wall [22]. Each player will have 13 tiles in hand, and take turns drawing a tile from the wall to form a 14-tile pattern. After that, he needs to choose one tile in hand to discard. The basic winning tile combination is x(AAA) + y(ABC) + DD while $x + y = 4$, where AAA represents for a triplet (three same tiles, which is also called as set/mentsu), ABC represents for a sequence (three number tiles with continuing numbers, which is also called as run/shuntsu) and DD represents for a pair (two same tiles). The player can also make stealing when others discard specific tiles. The player can call a pon if he already has two same tiles in hand as the discarded one or he can call a chi if he can form a sequence with this tile.

Normally, four players start a game of score 25000 each, and one game of Mahjong usually has four or eight parts of subgames. The rank is decided by players' scores at last.

2.2 Special Terms of Japanese Mahjong

Compared with original mahjong games, Japanese one has several other new features, which are listed below.

2.2.1 Richi

Richi means declaring a ready hand and it is also a kind of yaku. A player can declare richi showing all others that he is waiting to win, just needing one more tile to form a legal hand. One can only declare richi when he does not have any stealing called. Once declaring richi, one's tiles in hand cannot be changed anymore unless he has a closed kan or plus kan opportunity which is rare though, but he will own big chance of gaining larger scores if winning, since uradoras will count at that time.

2.2.2 Yaku

Yakus are specific patterns of tiles or conditions that possess values. For Japanese Mahjong, in order for winning, at least one yaku is needed. Each yaku has a specific hand value called han, which devotes to

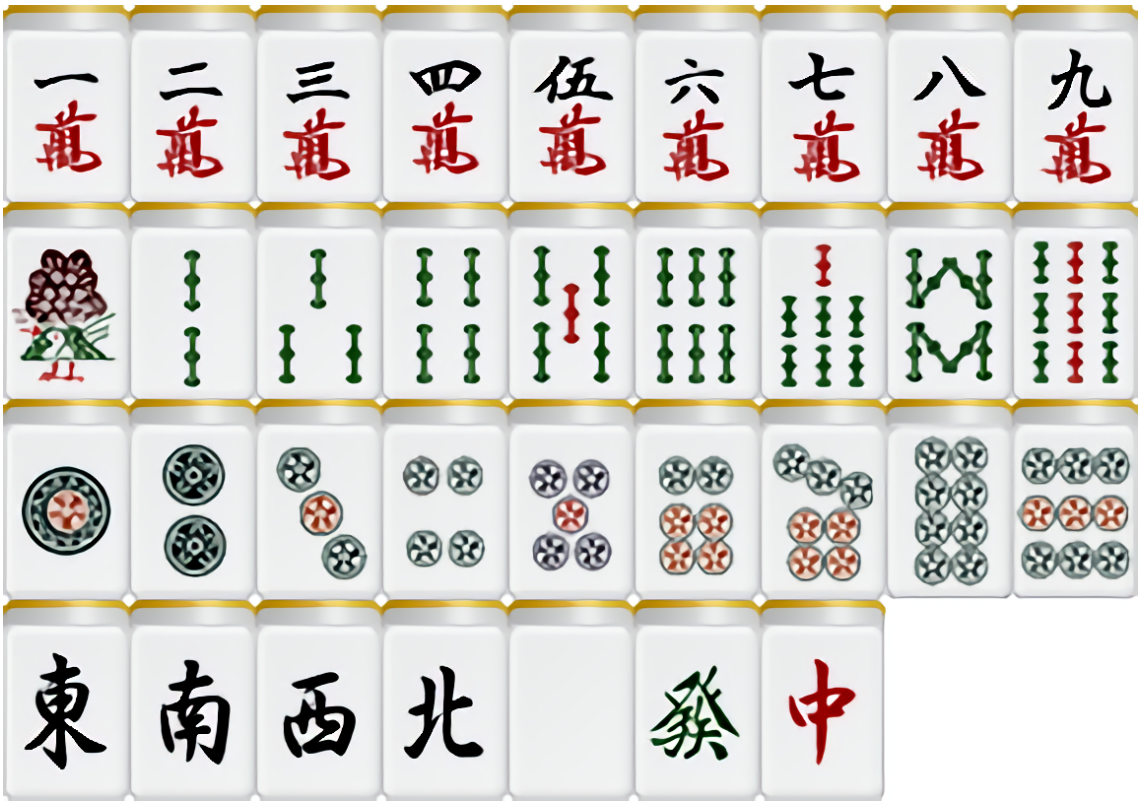

Fig. 2.1: Tile kinds in game of Mahjong

hands' values exponentially. Doras in hand cannot be counted as yakus, but will also devote to hand values once winning.

2.2.3 Dora

Hand pattern and the number of dora tiles in hand are two important factors for winning with a large score. Dora is a kind of bonus tile that enhances han value once winning. Each time a new subgame starts, there will be one dora tile known, since the dora indicator tile located atop the third stack from the last will be flipped over and revealed face up to all players. During the game, another dora indicator will be shown each time a kan is called, starting from the fourth stack from the last. It can have at most four Kans called among one subgame. Normally, the succeeding tile of dora indicator is regarded as a dora. However, this is

not always a linear relationship but divided into five smaller loops of 1m-9m, 1p-9p, 1s-9s, East-North and Haku-Chun. Among each loop, the succeeding tile of dora indicator becomes a dora, for example the next tile of 9m is not 1p but 1m, the next tile of North is not Haku but East. Aka five is another type of specific dora. Compared with dora indicators mentioned previously, aka five tiles are always a regular part on board, and each one devotes to one han value. Each game will have three red five tiles which are counted as aka five doras, one each for man type, pin type, and sou type respectively. Different from normal five number tiles, they are colored red, while having all other same functions of a normal one.

2.2.4 Uradora

There is an uradora indicator under each dora indicator among the same tile stack. Uradoras are revealed and counted only when one player wins and with a richi declaration beforehand, and no one can know what the uradora indicators will be before final revealing, hence it is a potential bonus with uncertainty only for richi-declared players.

2.2.5 Wind

During the game, among all 34 kinds of tiles, we have seven types of honor tiles: East, South, West, North, Haku, Hatsu and Chun. For each game, there are two kinds of wind tiles that are very important to all players: round wind and own wind. Round wind is set same to all players when one subgame starts, initialized as East and may change as the game continues. Own wind, on the other hand, is different from player to player in each subgame. Not like Haku, Hatsu and Chun that any triplet of them is already recognized as one yaku and one han value, for wind tiles only round wind and player's own wind devote to yaku calculation.

2.2.6 Winning

There are mainly two types of winning, winning from a wall or winning from a discard. Winning from a wall is also called self-drawn or tsumo, when the player gets his own tile picked up and then forming a legal hand. When facing a self-drawn, the other three players pay the winner some amount of score together.

When the player catches a tile discarded by others and completes a legal hand, it leads to a ron, and the original owner of the discarded tile needs to pay the owner by himself entirely. The dealer, who has tile East as his own wind, always pays more and gets more. Once the dealer wins, he will win a score 50% larger, no matter by self-drawn or others' discards. On the other hand, if another player wins by self-drawn but not the dealer, the dealer will need to pay the score one time more than the other two normal players.

CHAPTER 3

RELATED WORKS

3.1 Deep Learning

The term 'Deep Learning' (DL) was first introduced to Machine Learning (ML) in 1986 and later used for Artificial Neural Networks (ANN) in 2000 [5]. Deep learning methods are composed of multiple layers to learn features of data with multiple levels of abstraction [6]. Deep learning has gained numerous amounts of achievements in recent years. In this section, we take a brief look at some features mentioned in this article.

3.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is one of the most successful structures so far in deep learning. CNN is not only widely used in CV tasks such as object recognition, object detection and Generative Adversarial Networks (GAN), but also does a quite good job in NLP and game fields. The four basic features of CNN are local connections, shared weights, pooling, and using many layers. First parts of a CNN network are usually made of convolutional layers and pooling layers and latter parts are mainly full-connected ones. Convolutional layers detect local conjunctions from features and pooling layers merge similar features into one [6]. Besides pooling layers, there are still some other common techniques usually utilized together with convolutional layers.

3.1.2 Activation Function

Among a neural network, the output of neurons always needs to be processed by activation function before transmitting to the next layer. The concept is defined by Bengio. Y. [7] in 2016 as: An activation function is a function h: R $\rightarrow$ R that is differentiable almost everywhere.

As shown in Fig. 3.1, there are quite many different activation functions. Sigmoid is the most commonly used activation function, but it may face the problem of gradient vanishing within even fewer than five layers [24]. Besides, Sigmoid is not zero-centered. Tanh is similar to sigmoid, and their relationship is:

$$tanh(\mathrm{x}) = 2sigmoid(2\mathrm{x}) - 1$$

Tanh is improved from Sigmoid, and it solves the problem of non zero-centered. ReLU also becomes very popular these years, and the breakthrough in direct supervised learning of deep learning is caused by this. ReLU solves the problem of gradient vanishing, and becomes most commonly used among CNN. Leaky ReLU, ELU, Maxout, together with PReLU, are different variants of ReLU function. Compared with Sigmoid, besides the gradient vanishing problem, ReLU function still mainly has another two benefits, for it simplifies calculations among back propagation procedure and causes sparsity of the network by leading some parts of the neurons toward zero.

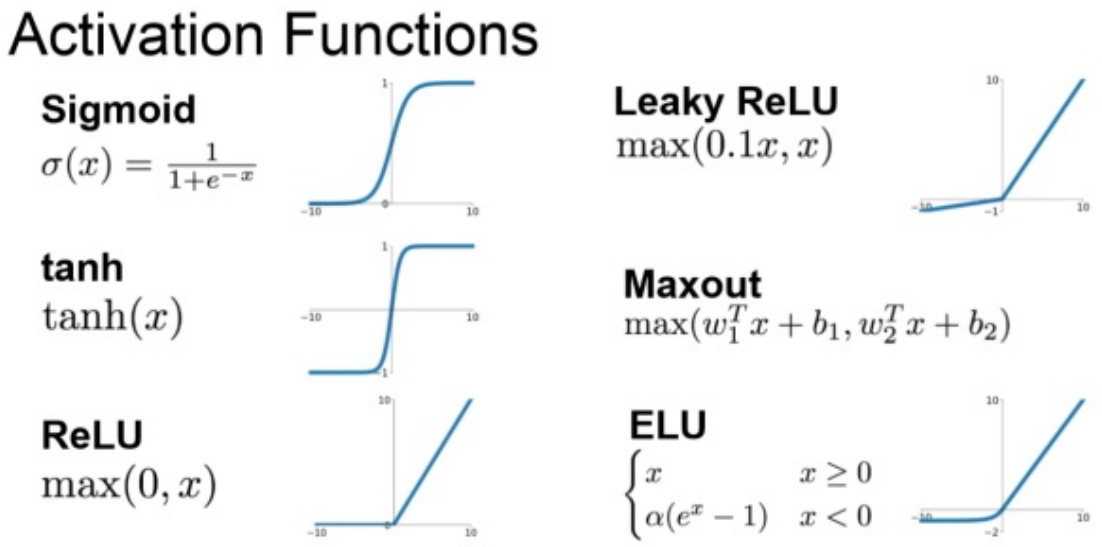

Fig. 3.1: Common activation functions [23]

3.1.3 Regularization

Overfitting is a common problem that can be met during training when the network considers sampling errors and goes towards high-variance, fitting well on training data but not well on test data, showing a weak generalization ability. Besides containing more data for training, L0, L1 and L2 Regularization are common ways for overfitting depression. There are also some other useful techniques, such as Noise, Model Ensemble, Batch Normalization, and Dropout.

3.2 Game of Go

The game of Go has been recognized as one of the most exciting challenges in the field of Artificial Intelligence, and also the most difficult one among perfect information games. The legal complexity of the game of Go is about 10^{170}, much more than the whole number of atoms in the universe (about the level of 10^{80}). It is impossible to list all the branches of the game of Go for a computer Al program. Although the great power of Monte Carlo Tree Search (MCTS) has been found and used for Go AI programs for decades, it

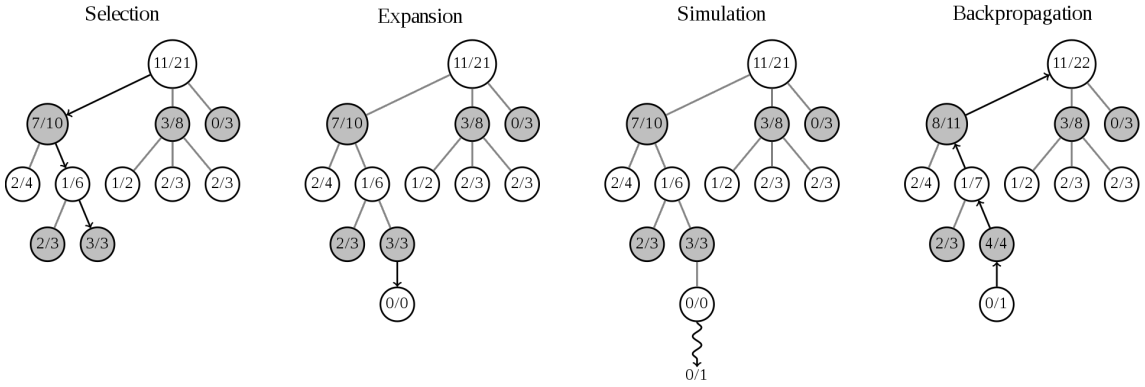

Fig. 3.2: Steps of Monte Carlo Tree Search [32]

is still very difficult for people to construct a proper evaluation function for positions or moves. Till the year of 2015, among over 6000 kinds of perfect information games in human society, only the program of Go could not own the ability to defeat professional players. However, the story changes when the Go AI program AlphaGo appears. Till now, go AI players have already reached a level far higher than human players, based on the idea of AlphaGo series. This section will make a brief introduction to the algorithms behind.

3.2.1 Monte Carlo Tree Search (MCTS) and Related Methods

Monte Carlo Tree Search (MCTS), as a very powerful algorithm for go and other perfect information games, made great contributions for go programs. Monte Carlo Method was firstly explored by Bruce Abramson [28] in 1987, and in 2006, Rmi Coulom [29] applied it to game tree search, and coined the present name. L. Kocsis and Cs. Szepesvri [30] developed the UCT algorithm, and S. Gelly et al. [31] implemented UCT in their program MoGo. MCTS is an indispensable algorithm for go AI players. It is a heuristic search algorithm for some kinds of decision process, and it has four steps in each playout: selection, expansion, simulation and backpropagation, which can be seen in Fig. 3.2.

Selection: Start from the root node down to one leaf node that can be expanded. The method of choosing that node varies according to the algorithm.

Expansion: Create one child node for this node.

Simulation: Make random play from this child node until reaching the final ending of the one game and get the win/lose result. This step is also called as playout or rollout.

Backpropagation: Use this new result to update information in the nodes way through to the root.

Monte Carlo Tree Search works well for the game of Go, for it is zero-sum, perfect information, determinism, sequential and discrete, which can be summarized as combinatorial games according to game theory [33].

Next, two important versions of AlphaGo series and their ideas behind will be introduced.

3.2.2 Mastering the Game of Go with Deep Neural Networks and Tree Search

The paper of the first version [11] was published by Google DeepMind in Nature in Jan., 2016. All games of perfect information can be solved by recursively computing the optimal value function in a search tree at about b^d sequences of moves where b represents for the game's breadth (number of legal moves per position) and d represents for the game's depth (game length). Compared with chess ($b \approx 35, d \approx 80$), the game of Go is much more complex ($b \approx 250, d \approx 150$) [13]. Based on Monte Carlo Tree Search framework, deep convolutional networks are adopted for reducing this huge complexity. The policy network is used for reducing game ' s breadth while the value network is used for reducing game's depth.

The policy network takes a representation of the board position s as input, calculates it through the neural network and outputs a probability distribution $p(a|s)$ over legal moves a. The value network similarly takes the same input, going through networks, but outputting a scalar v_θ(s') which predicts the expected outcome (win/lose) of different move choices.

To be in detail, a fast rollout policy network p_π and supervised learning (SL) policy network p σ are trained to make move predictions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network but improved during self-playing. Finally, a value network p_θ is trained for predicting the expected outcome.

Supervised Learning (SL) Policy Network

The SL policy network uses convolutional layers with weights σ and rectifier nonlinearities, with a final softmax layer outputting a probability distribution over all legal moves a. The input s representing board

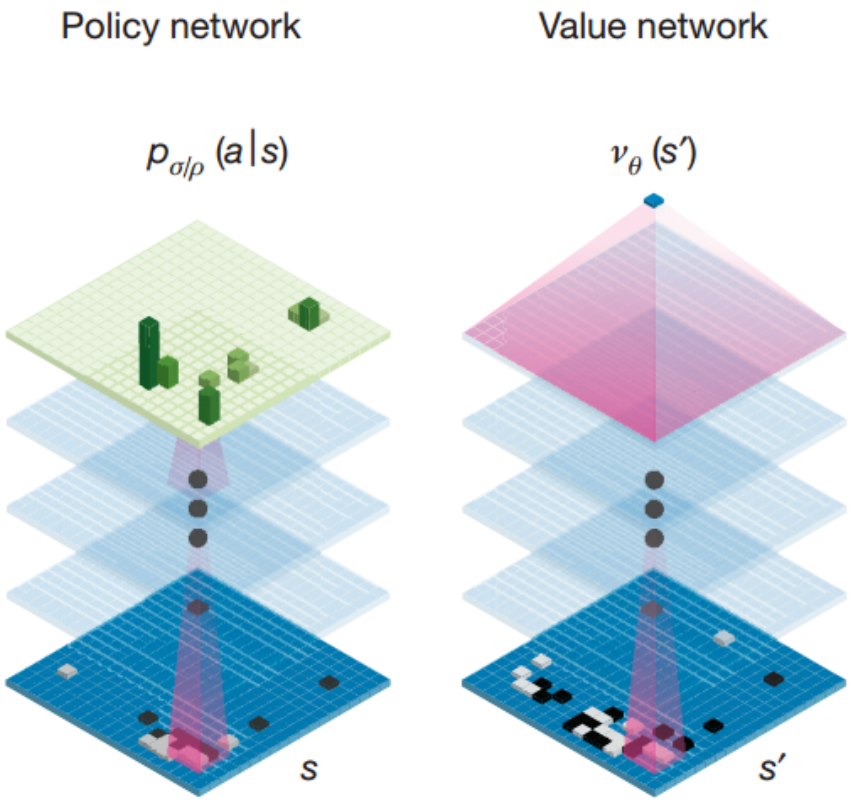

Fig. 3.3: Schematic representation of the neural network architecture used in AlphaGo [11]

state is designed in Fig. 3.5.

The first 3 planes are designed for containing the raw board information, showing black/white/empty space in the board, for instance for the black stone information in the corresponding 19x19 layer, 1 is placed in black stone positions while 0 is filled in rest spaces. The whole input has 48 planes, and only the first three ones show real board information, others are manually designed according to game rules. Finally, a 13-layer policy network is trained, using 30 million data from the KGS Go Server, and achieving a much higher prediction accuracy of 57.0%, while the state-of-art result from other groups is just 44.4%. With only SL policy network, the program already reaches an amateur 3 dan level.

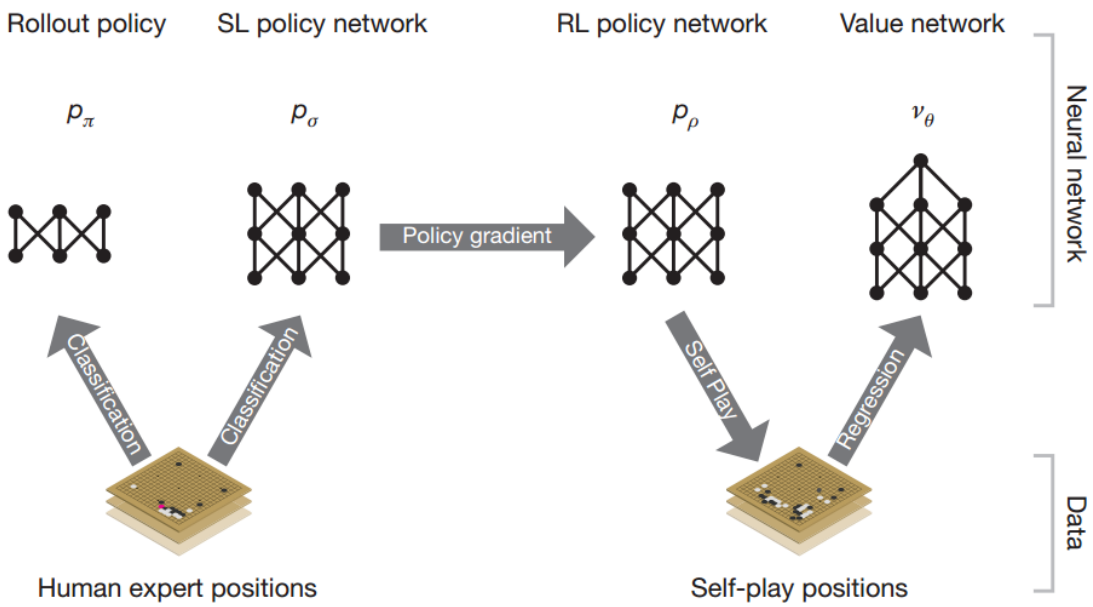

Fig. 3.4: Neural network training pipeline and architecture [11]

Rollout Policy

SL Policy network makes a good job on making move predictions, however, for better accuracy, much time is needed for training. Thus a faster but less accurate rollout policy $p_\pi(a|s)$ is trained, using traditional local pattern matching and logistic regression method, achieving an accuracy of 24.2%, but just needing 2μs for making one choice, more than one thousand times faster than SL policy network which needs 3ms for the same task. It reaches a good bias between accuracy and speed, and can be used for helping estimating value functions with fast simulation.

Value Network

A value network v_θ is trained to predict the winner of games played by reinforcement learned policy network against itself. This network has a similar architecture to the policy network, but with one more feature plane (last plane in Fig. 3.5) and with output changed to a single prediction (regression, with mean squared error loss).

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).

Fig. 3.5: Input features for neural networks [11]

Combining in Monte Carlo Tree Search

AlphaGo combines policy and value networks in MCTS algorithm. Each edge in search tree stores 3 elements, action value Q, visit count N and prior probability P. This is a UCT (Upper Confidence Bounds applied to trees) method with prior. Each time MCTS selects nodes with the highest score, and the state score is decided together by value network output and fast rollout policy output evaluated at leaf node and SL policy network output acting as a bonus of selection score. Here AlphaGo uses a 50-to-50 bias between the value network and rollout.

3.2.3 Mastering the Game of Go Without Human Knowledge

This second version [12] is later published by Google DeepMind in Nature in Oct., 2017. This AlphaGo Zero version becomes much stronger than previous versions and even the top professional players mainly due to several aspects:

Studying Without Human Data

First and foremost, AlphaGo Zero discards existing game records. Instead, it is solely trained by self-play reinforcement learning, starting from random play, without any human data.

Reducing Extracting Features

As shown in Fig. 3.5, the previous version uses 48 planes of input for training the convolutional network, most of which are artificially designed features for the game of Go. In contrast, AlphaGo Zero only uses raw board information with black and white stones from the board as input features, together with past 7 situations, 17 planes in all for input, as shown in Fig. 3.6.

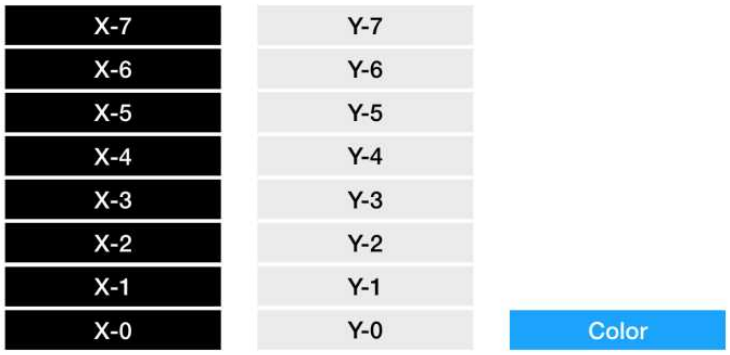

Fig. 3.6: A simple representation for input features of neural network [12]

Combining Policy and Value Networks

Previous version trained policy and value networks by different networks. In this version, instead of the previous two networks, two heads, the policy head and the value head are added after the sole network.

ResNet instead of ConvNet

Deep Residual Networks was firstly introduced in 2015 by He Kaiming Group, MSRA [14]. The main architecture can be seen Fig. 3.7.

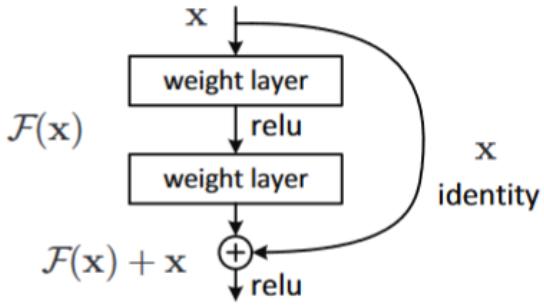

Fig. 3.7: One building block of residual learning [14]

Shortcut connections are the main features the residual networks, and these networks can be well-trained with much deeper layers, thus achieving much better results. AlphaGo Zero adopts 40-block residual network structure, not adding too many improvements compared with original ResNet.

3.3 Game of Mahjong

A one-on-one version of Texas Hold'em, one of the most popular poker variations, has been solved by using subgame solving to reach an approximate Nash equilibrium strategy [8, 9]. However, there are still no suitable strategies on multi-player poker solutions for the unsustainable computational cost. Game of Mahjong, theoretically, has even larger complexity than the game of Texas Hold'em. With existing techniques and skills, it is impossible to reach the Nash equilibrium for this game.

Traditional mahjong AI usually has two function blocks, the offense part and the defense part. For offense part, the tile efficiency is the main thing considered in order for faster winning and larger scores, ignoring any other players' actions. It is acted as a one-player mahjong game. For the defense part, how to avoid ron (which means defeated) by others is the task, aiming at discarding safe tiles but not attempting to get a winning. The training procedure is to achieve a suitable balance between offense part and defense part and decide output strategies in diverse situations [4, 10]. The state-of-art agreement rate accuracy for haifu data learning on test dataset before is 62.1% [4].

Deep learning has been hot these several years and is featured for its strong automatic feature extraction

ability which needs no artificial extraction, together with surprising learning capability. However how to design the data structure and how to build the network is still a difficult task. Tsukiji *et al.* (2015) [2] made a 2-layer full-connected network with 1653-dimension input, receiving a test accuracy rate of 43%. In Tsukiji's new article in 2017, he designed a 5 by 34 by 5 data structure like an image to contain the tile information as shown in Fig. 3.8, 5 planes for own hand tiles and discarded tiles, and the 34 by 5 structure for each plane. As shown in Fig. 3.9, it just manages to contain a small part of the information available, left a large amount of information missing. Finally, it achieves a test accuracy of 53.98% [3].

Fig. 3.8: Tile coding method of Tsukiji [3]

特徴量	次元数
自分の手牌	170
自分が捨てた牌	170
下家が捨てた牌	170
対面が捨てた牌	170
上家が捨てた牌	170
計	850

Fig. 3.9: Tile features of Tsukiji [3]

Chapter 4

Tile Information Learning

In this chapter, we show our new model with a data structure designed in three dimensions to include the tile information. We choose the agreement rate of discard tile strategy which is the most important part in this game for strategy learning as the judgement benchmark, and design a deep convolutional neural network for training. We show our result much better than previous state-of-art training results.

4.1 Data Structure Design

In this thesis, we propose a basic data structure named as 'plane', which is a matrix with two dimensions, 34 in height and 4 in width. A normal plane structure is shown in Fig. 4.1. The 34 rows representing 34 types of tiles, while the 4 columns representing the existing number of each tile type in given situations since the maximum number of each type of tile is 4. We believe data in such kinds of structures is easy for training. Compared with the one-hot structure in Fig. 3.8, we believe it has at least three merits:

1. Firstly, it can have more potential and operating space. The only thing we can learn from the previous structure is the number of tiles in a given part, however in this structure, we can also be able to know some other hidden information, such as aka five information and the relations between the same kind of tiles. Although we do not choose to code aka information in this way, it still has potential in the future.

2. Secondly, as introduced later, we do not just simply contain the information about the tiles shown on table. We also design other planes to represent some other information, such as players' richi information and wind tile information. For these kinds of irregular information, we need to code in a more different way, and our structure is obviously more suitable for coding.

3. Last but not least, it saves 20% of the memory space. For deep learning tasks, more data usually leads to better training. One-fourth of the training data amount increase under limited hardware resources will be a huge improvement.

For information on table, we divide them into different parts to represent them respectively, such as own hand tiles or players' discard tiles. For each divided part, we use one plane to represent. To be in detail, for instance in Fig. 4.1, we have three 3m tiles in this specific part, so we have three ones filled in the 3m row, from the left to the right. Other rows are all filled in the same way. The left elements are all filled with zeros. In this approach, tile information can be represented properly.

In this chapter, we propose the data structure of in all 56 planes to represent the information available on table. The structure is shown in Table 4.1.

For information of present situation, we adopt first 16 planes to store, including two planes representing for one's own hand tiles and aka five information, four planes for four players' discard tiles, four planes

	1	2	3	4
1m	0	0	0	0
2m	1	0	0	0
3m	1	1	1	0
4m	1	0	0	0
⋮				
9m				
1s	1	0	0	0
⋮				
9s	0	0	0	0
1p	0	0	0	0
⋮				
9p	1	1	1	0
東	0	0	0	0
⋮				
白	0	0	0	0
⋮				
中	1	1	1	0

Fig. 4.1: Plane structure for data

for four players' stealing tiles, one plane for dora indicators, three planes for other three players' richi information, one plane for round wind and one plane for own wind. The details are introduced below.

One player can have at most 14 closed tiles (which means not melded) in hand, therefore the first plane which represents for the player's own hand tiles will have at most 14 ones within one plane of 136 elements, while all other elements are all filled with zeros. We add one more plane to mark the aka five information in hand. As introduced in Section 2.2.3, one aka five dora in hand will increase one han value once winning which is quite important, usually will not be discarded. Players who have aka five doras in hand will usually

Table 4.1: Input features of tile information learning network

Feature	# of planes
Own hand tiles	1
Aka five mark	1
Discard tiles	4
Stealing tiles	4
Dora indicators	1
Richi players	3
Round wind	1
Own wind	1
Past 1 situation	13
Past 2 situation	9
Past 3 situation	9
Past 4 situation	9

try to build a legal hand with these tiles, so we believe that the aka five mark is very necessary. One thing needs to be noticed is that although we fill the first plane with the plane style mentioned previously, for aka five information, we change the filling method a little bit. This time we let the whole corresponding row represent the aka five tile, which means we four times enhances the number of the tile information. In this way, we aim not to destroy the feature that each type of tiles can appear at most four times on table.

For stealing information, one can call at most four melds in one game, with each meld containing three (pon & chi) or four (kan) tiles, hence each stealing plane will have at most 16 ones, with all other elements filled with zeros. The filling method is same as explained in Fig. 4.1.

For dora indicator plane, each subgame will have one indicator showing to all at the beginning, and at most four indicators during play, since once the fourth kan is declared which means the fifth dora indicator appears, it usually leads directly to the end of this round of game according to game rules. The dora indicators are usually different since the tile wall is formed randomly, but occasionally it may appear the

same in one subgame. The filling method is still the same as before.

Besides, Richi information of other players is also very important, for it is a clear declaration that some other players are already prepared to achieve a winning and having a relatively large chance of winning patterns with high scores, because they have a chance of hitting uradoras. The player should be especially careful not to discard dangerous tiles that the richi-declared players need to avoid ron by them. These richi planes contribute much on this task. Compared with models without richi information structure, the agreement rate of discard tile on test dataset raises over 1%. The richi plane here is coded in another way compared with previous layers. For previous ones, they are used to record tile numbers of each kind for one part of information. Obviously, this is not a suitable coding method for richi information representation. Therefore, here we adopt 0/1 planes to represent. A 0/1 plane is a plane filled with internal elements all zeros or all ones, or we can recognize it as a pure white/black channel in CV tasks. This works well in our situation.

For round wind and own wind information, in order to cause emphasis and maintain the principle of at most four tiles for each type of tile on the table, we mark the whole corresponding row with four ones to mark wind signs, the same coding approach of aka five information.

Past actions that players made also have great influence on present choice decision strategy. People can usually learn what other players need and not need from their past discard tiles. This is especially important for making a defense when another player declares richi, because all players' passed discard tiles after that declaration are all safe, and tiles which are around early discard tiles are usually safer than normal tiles. Although we already have planes of players' discard tiles already, it can only show what tiles players already discarded, but the discard orders cannot be known. The closer the discard tiles are, normally more information they possess. After experiments, we include the last four rounds' information including own hand tiles, four players' discard tiles and four players' stealing tiles. The number of dora indicators will increase by one during playing if a kan is called by one player, hence last round's indicator information is also needed in order to mark a last round's kan sign. For richi information, the round just after one's richi is the most threatening time for other players since it will add another one yaku and one han value if it wins during that round, so we also need last round's richi information to take care of that dangerous round. We include past dora indicators and other three players' richi information only in past 1 situation but not others,

this is why we have 13 planes for the past 1 situation but 9 planes each for others.

In all, we design a 56-plane data structure to represent information on table.

4.2 Neural Network Structure Design

In this section, we propose our filter design for the CNN network. The network structure is shown in Fig. 4.2. As we can see, there are three convolutional blocks in the network, each one including one convolutional layer (conv), one Batch Normalization layer (bn) and one Dropout layer (drop).

Since our input is 34 by 4 in plane structure with 56 planes as shown in Fig. 4.1 and Table 4.1, our plane structure is too narrow with only 4 in width but 34 in height, normal convolutional kernels such as 3x3 or 5x5 cannot be utilized in our data structure. Therefore we finally adopt three convolutional layers with 5 by 2 in filter size and 100 in filter height. We adopt this filter three times with three convolutional layers, and we do not use paddings that keeps the matrix of size no changed, which means the padding style is set to 'VALID'.

A Batch Normalization layer and a Dropout layer of drop rate 0.5 are added after each convolutional layer for overfitting depression. We select ReLU as each block's activation function. We add one full-connected layer of 300 neurons after the flatten layer. We have a final output layer at last. Since for tile discard strategy, the game has 34 types of tiles, so we can simplify this problem as a 34-class classification problem, and we use a softmax layer for the classification, with the loss of categorical crossentropy. Adam is chosen as the optimizer with an initial rate of 0.001 while batch size is set to 256.

4.3 Training Data and Experiment

Our models and networks are trained on NVIDIA Tesla K10 GPU with 32 GB memory size. We adopt the supervised machine learning method for prediction model and use game records collected from the 'Houou' table at the biggest Japanese online mahjong site 'Tenhou' in the year of 2015 as the training data. The 'Houou' table is only open for the top 0.1% mahjong players, so the game records can be considered as good quality. All game records from the 'Houou' table are open for research and can be downloaded from the official website. During each subgame, we just follow one player's game record, and make sampling

until the player's richi declaration since once richi declared, the player cannot determine one's discard tiles anymore, the discard tile is always the same as the drawn tile then. And in order for a better quality of the dataset, we will not learn the subgames that the player loses over 1500 scores.

4.4 Training Results

We sample through all game records among the year of 2015 and we obtain 2135331 different situations for training, with 10% among it divided as the validation dataset. We set dropout rate at 0.5 and batch size at 256. The final validation accuracy reaches 69.5%, which is shown in Fig. 4.3 and Fig. 4.4. For test data, we pick situations from the year of 2014 so the test dataset is totally different from training data. We picked 50000 valid data without relevance between, and achieve an agreement rate accuracy of 69.48%.

4.5 Interesting Findings

Pooling is recognized as a very efficient and useful down-sampling tool for convolutional neural network training especially in computer vision tasks. However during our experiment, we find that the pooling layer not only not contributing to the accuracy but also making demerits instead. Usually for computer vision tasks, image recognition has the property of movement invariance and rotation invariance, hence pooling usually does a great job for down-sampling, making fewer parameters and forming more robust neural networks. However, for our task, the data plane structure is very elaborate and concise, therefore the pooling down-sample will lose too much information, causing lower accuracy.

From the test cases, we also find that our proposed data structure perfectly understands the information of hand tiles and what are the dora tiles only from the indicators. For prediction output, it is a 34-class softmax classification problem, however the player will just be able to have at most 14 closed tiles in hand. Therefore, whether all predictions made out from the 34 classes are legal choices within the at most 14 types of tiles in hand at any situation is a meaningful question. Although designing another classification method is also an approach, for instance making predictions of which order is the tile that should be discarded in hand where the tiles are in sequential order. However, this does not make much sense and performs worth than our present model. For result, within the 50 000 test cases, we find the predictions made by our neural

network are all among the legal tile actions in hand, at the rate of 100%. Obviously, the network already learns the output range of legal predictions, which is an exciting finding. Besides, according to the game rules, dora tiles and dora indicators do not own a simple linear relationship but several loops as mentioned previously. However, our network perfectly learns that, and with the dora indicator plane instead of directly using a plane to contain dora information achieved from dora indicators, the agreement rate accuracy even raises about 1%, which we think is due to the maintenance for the principle of at most four tiles each kind on table.

4.6 Discussion

In this chapter, we proposed a new data model designed for the game of Mahjong, achieving the best performance so far on agreement rate accuracy after training on a CNN network, with totally no concepts about game rules in advance. We simplified the prediction tasks as multi-classification problems and achieved good performances.

However, this model is not perfect. There are still mainly two problems here:

1. Firstly, as mentioned previously, past actions made by players have great influence on present decision making. In this chapter, our model just includes the last four rounds' information. However, this is not enough. An ideal model should be able to know all actions made before and their sequences. So in Chapter 5, we will make a trial of adopting a sequential model for neural network training.

2. Besides, as an imperfect information learning task, in order to achieve better results, information should be contained as much as possible for learning. It is very intuitive that the more information we know, the better decisions we will be able to make. However, for present data model, there is still some parts of information missing. We still lack some information that is hard to be contained in, such as kyoku number, honba number, richi stick number and players' scores. This information is not one that we can get only from tiles, so we call it 'off-table information'. As shown later in Chapter 6, we try to find a good approach for containing this information.

Layer (type)	Output Shape	Param #
main_input (InputLayer)	(None, 56, 34, 4)	0
conv1 (Conv2D)	(None, 100, 30, 3)	56100
bn1 (BatchNormalization)	(None, 100, 30, 3)	12
drop1 (Dropout)	(None, 100, 30, 3)	0
conv2 (Conv2D)	(None, 100, 26, 2)	100100
bn2 (BatchNormalization)	(None, 100, 26, 2)	8
drop2 (Dropout)	(None, 100, 26, 2)	0
conv3 (Conv2D)	(None, 100, 22, 1)	100100
bn3 (BatchNormalization)	(None, 100, 22, 1)	4
drop3 (Dropout)	(None, 100, 22, 1)	0
flatten (Flatten)	(None, 2200)	0
fc1 (Dense)	(None, 300)	660300
bn4 (BatchNormalization)	(None, 300)	1200
drop4 (Dropout)	(None, 300)	0
softmax (Dense)	(None, 34)	10234

Total params: 928,058
Trainable params: 927,446
Non-trainable params: 612

Fig. 4.2: Tile information learning network structure

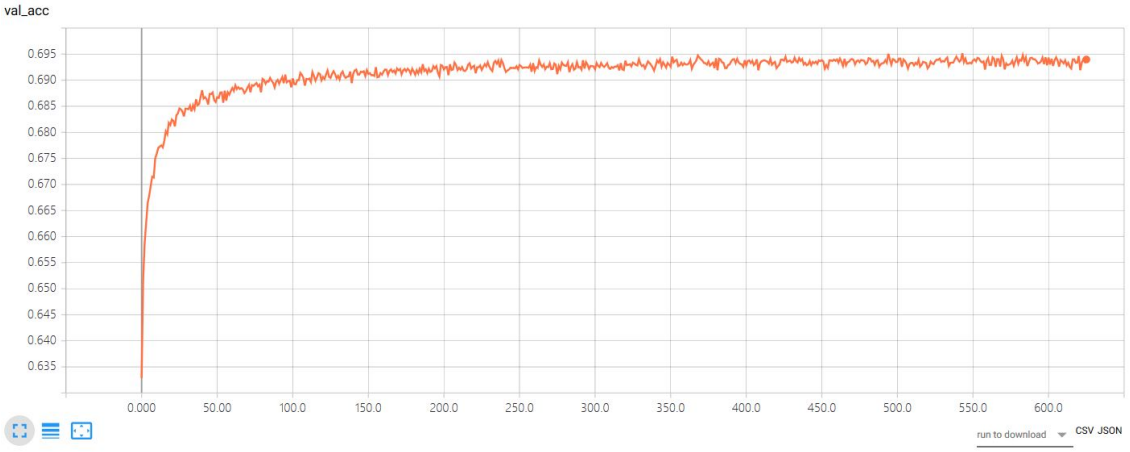

Fig. 4.3: Validation accuracy for tile information learning

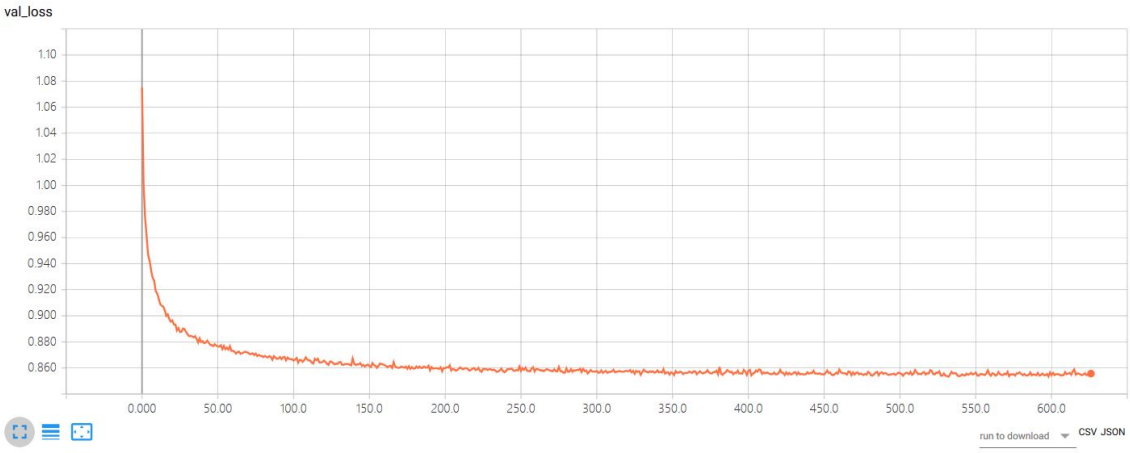

Fig. 4.4: Validation loss for tile information learning

Chapter 5

Past Information Learning

In the previous chapter, we introduced our new model for tile discard strategy learning in the game of Mahjong, and we achieve a best result of agreement rate so far. However, only by including the last four round's information may not be enough. In this chapter, we make a trial to adopt a sequential model for game record learning, which theoretically should be able to learn all past information from beginning to present.

5.1 Sequential Models

5.1.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) has done a very good job on time series processing tasks, especially in the field of Natural Language Processing (NLP). The idea behind RNNs is to use sequential information. Compared with traditional neural networks that assume all inputs and outputs are independent of each other, RNNs perform same tasks for every element of the issue, and the output of each element depends on previous outputs, so it seems to have a 'memory' function in it. A basic cell in RNN is shown in Fig. 5.2, consisted of a cell, an input gate, and an output gate.

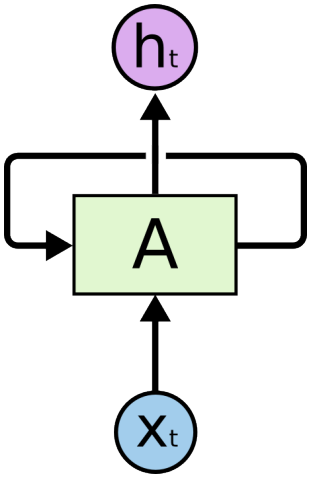

Fig. 5.1: A basic cell structure of RNN [35]

A chunk of neural network A gets an input x_t and output information of h_t. The information is allowed to transfer from step to step. If the loop is unrolled, it will be more clear, which is shown in Fig. 5.2. The network can be seen as a repetition of a single cell, which is a single time-step.

5.1.2 Long Short-Term Memory (LSTM)

RNN models usually meet the problem of gradient vanishing or gradient explosion, that once the sequence becomes long, the network will have a bad memory about early stages. LSTM is a variation of the RNN

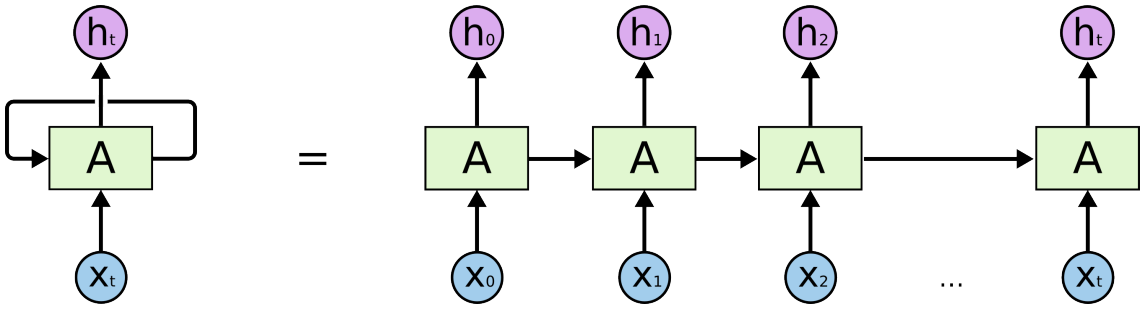

Fig. 5.2: Unrolled structure of RNN [35]

network and is designed for solving these problems. It was first proposed in 1997 [36]. LSTM is normally augmented by recurrent gates called 'forget' gates. LSTM prevents backpropagated errors from vanishing or exploding [37]. Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space, so LSTM networks can work well even required events are thousands of time-steps away from now. The unit structure is shown in Fig. 5.3.

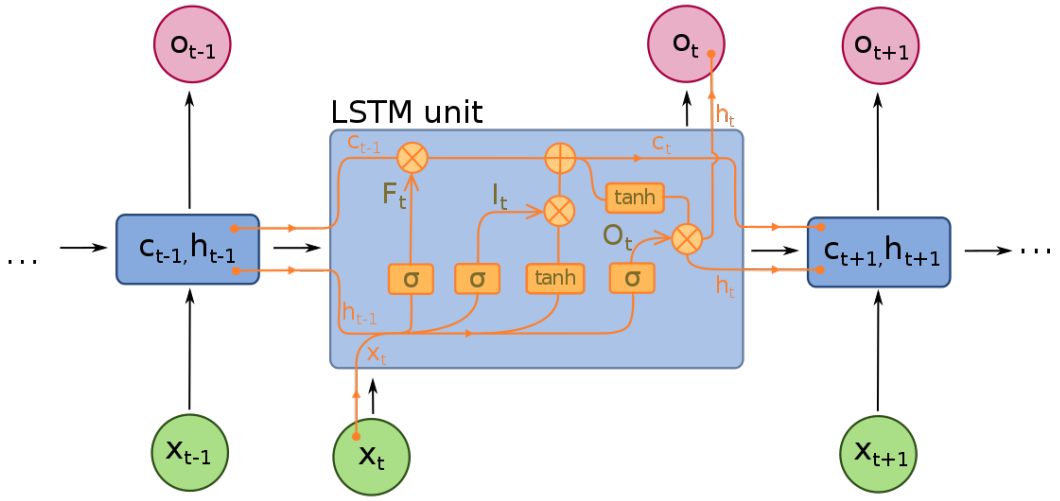

Fig. 5.3: Structure of a LSTM unit [38]

5.1.3 Gated Recurrent Unit (GRU)

Gated recurrent units (GRU) is a relatively new kind of variation of RNN and was first introduced in 2014 [39]. Compared with the LSTM network, they have fewer parameters, as they lack an output gate [40]. Compared with LSTM networks, GRU networks perform similarly, but is computational cheaper, so it is more widely used now. The unit structure is shown in Fig. 5.4.

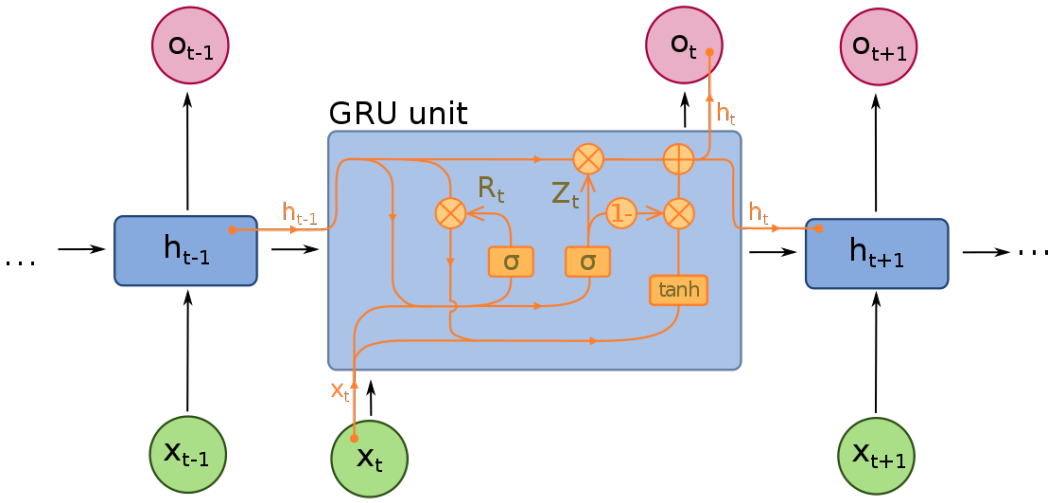

Fig. 5.4: Structure of a GRU unit [38]

5.2 Data Structure Design

In our last model introduced in Chapter 4, we concluded last four rounds' past information. In order for achieving longer memory, here we try to utilize sequential models for learning. The data structure is designed as shown in Table 5.1.

As we can see, we remove the memory function from the data structure, so we delete past rounds' information compared with the previous model, because the sequential model will contain past information by itself.

Here we adopt a GRU model for this task. Here we set each subgame as a sequence, because the information within one subgame is related to each other but information between subgames is irrelevant.

Table 5.1: Input features of sequential model

Feature	# of planes
Own hand tiles	1
Discarded tiles	4
Stealing tiles	4
Dora indicators	1
Richi players	3
Aka five sign	1
Round wind	1
Own wind	1

So each time we start learning a new subgame, we need another new sequence for learning. Usually, we adopt invariant time-step training for time series learning. Since the length of each time-step may not be the same, we need to decide a suitable length, cutting the time-steps that longer than the length and filling the blanks where the sequences are shorter than the decided length. Here we set the sequence length to the longest one among all, so no time-steps will be missed. After going through all game records. the largest round length in one subgame is 22, so for invariant time-step training, the time-step length is set to 22. For those which are shorter than 22 rounds, we adopt 0 paddings to fill, complementing them to the length of 22. Therefore, the input structure for one subgame becomes a four-dimension matrix of (22, 16, 34, 4). The size of each data structure in Table 5.1 is a three dimension of (16, 34, 4), so the input is a repetition of this data structure by 22 times.

As shown in Fig. 5.4, for each time-step t, it has an input gate x_t and an output gate o_t. x_t represents for the present situation which is same as before, however o_t is also affected by a hidden state h_{t-1} transmitted from last the time-step.

For sequential tasks, there may have multiple inputs and outputs. Normally we have four categories: one-to-one, one-to-many, many-to-one and many-to-many. For our task, this is a many-to-many problem, and the number of inputs and outputs are set the same. For each time-step which also means each round, we

collect its output. We manage to make discard tile predictions from these outputs after training. For output from each time-step, we make a prediction at that situation.

5.3 Neural Network Structure Design

Our network structure is shown in Table 5.5. We still use filters and convolutional layers to extract features. We increase the height up to 128 for training. We add a BatchNormalization layer and a Dropout layer of rate 0.5 after each convolutional layer, together with ReLU as the activation function. In order for sequential training, the inputs of the GRU layer should also be in sequences. Hence here we use a wrapper called TimeDistributed to wrap all convolutional layers in order to make training on sequences. We set return sequences for the GRU layer, so that it will output all hidden states among the sequence. For each output, we lead it to a softmax layer which has 34 classes for the final discard tile prediction. Adam is selected as the optimizer with an initial rate of 0.001. Batch size is set 256, the same as before.

5.4 Training Data and Experiment

Our models and networks are trained on NVIDIA Tesla K10 GPU with 32 GB memory size. We adopt the supervised machine learning method for prediction model and use game records collected from the 'Houou' table at the biggest Japanese online mahjong site 'Tenhou' in the year of 2015 as the training data. We only study the subgames that the player loses less than 1500 scores. In all, we obtain 193 696 sequences of subgames for learning. We fill 0 paddings for the sequences that are less than 22 time-steps long. In all, we have valid 2 135 331 situations for the whole year. This actually causes a problem, because 0 paddings fill about half of the whole sequences.

5.5 Training Results

We divide 10% of data as the validation dataset. During training, for validation accuracy, although it easily passes more than 80%, it has little meaning because about half of the training dataset is filled with 0 paddings. We focus on test dataset which actually makes sense. For test data picked in the year of

2014 which is totally different from training data, we picked 50000 valid data without relevance between, and achieve an agreement rate accuracy of 68.14%. Although it's better than other researchers' result, it performs a little bit worth than our previous model.

5.6 Discussion

In this chapter, we propose a data structure and network design for utilizing time series method to learn game records in a sequential way. We select GRU as our sequential model and choose the mode of invariant time-step for data conversion. Too much redundant data is an important factor that causes our performance not that perfect. Variant time-step is absolutely another way for improvement, but it is hard to implement, which will be a future work for this chapter.

Layer (type)	Output Shape	Param #
conv1 (TimeDistributed)	(None, 22, 128, 30, 3)	20608
bn1 (TimeDistributed)	(None, 22, 128, 30, 3)	12
drop1 (Dropout)	(None, 22, 128, 30, 3)	0
conv2 (TimeDistributed)	(None, 22, 128, 26, 2)	163968
bn2 (TimeDistributed)	(None, 22, 128, 26, 2)	8
drop2 (Dropout)	(None, 22, 128, 26, 2)	0
conv3 (TimeDistributed)	(None, 22, 128, 22, 1)	163968
bn3 (TimeDistributed)	(None, 22, 128, 22, 1)	4
drop3 (Dropout)	(None, 22, 128, 22, 1)	0
flatten (TimeDistributed)	(None, 22, 2816)	0
gru (GRU)	(None, 22, 256)	2360064
drop_gru (Dropout)	(None, 22, 256)	0
softmax (Dense)	(None, 22, 34)	8738

Total params: 2,717,370
Trainable params: 2,717,358
Non-trainable params: 12

Fig. 5.5: Sequential model network structure

Chapter 6

Off-table Information Learning

As mentioned in Section 4.6, there are mainly two problems needs improving. Besides the time series problem of past information mentioned in Chapter 5, another problem is that it misses some off-table information, such as kyoku number, honba number, richi stick number, and players' scores. In this chapter, we will try to make a way to contain this information.

6.1 Data Structure Design

Among all missing off-table information, not all can be managed to contain in our present model. Specific number values such as richi stick number and scores are very difficult to include. Although from first sight honba number seems to depict game process conditions from a macro way just like the kyoku number, it is actually more similar to the richi number since both of them affects scores. As the honba number increases when the subgame of each kyoku continues, the pattern value of hand tiles will increase a little bit proportional to honba number. In fact, there is a way to contain this information, by designing another tensor for representing it, and making concatenation between this tensor and the full-connected layer coming out from the main network route. However, we find that by this way it does not achieve an obvious improvement. So after experiments, we decide to add rank and kyoku information as shown in Table 6.1.

Table 6.1: Input features of off-table model

Feature	# of planes
Own hand tiles	1
Aka five mark	1
Discard tiles	4
Stealing tiles	4
Dora indicators	1
Richi players	3
Rank	4
Kyoku	8
Round wind	1
Own wind	1
Past 1 situation	13
Past 2 situation	9
Past 3 situation	9
Past 4 situation	9

We think rank and kyoku information is closely related to each other. Although for this game the final result of win or lose is not decided by only one subgame, but it has a limited length of subgames among one game. As the game goes toward the final, the difference between scores becomes more important. Even a tiny 100 score may cause different ranks in final. Therefore, as kyoku number increases, players' strategy may change. Besides, a player at top and a player at last may also make completely different decisions. Instead of containing players' scores which is hard for modeling, we use ranks instead, and achieve a quite good result. For the coding style of rank and kyoku planes, we adopt the same method as richi planes. We adopt 0/1 planes to represent each, filled with internal elements pure zeros or ones.

6.2 Neural Network Structure Design

Our network structure is shown in Fig. 6.1, which is quite similar with in Chapter 4. This time we delete full-connected layers in order for more direct training. We set the dropout rate of 0.5 and a batch size of 256. We adopt Adam as the optimizer with an initial rate of 0.001.

6.3 Training Data and Experiment

Our models and networks are trained on NVIDIA Tesla K10 GPU with 32 GB memory size. For training data, it is picked in the same way as introduced in Chapter 4.

6.4 Training Results

We sample through the whole year of 2015 in the same way as before. In all we obtain 2135331 different situations for training. We divide 10% among it as validation dataset. Our training result on validation dataset is shown in Fig. 6.2 and Fig. 6.3. Finally, we achieve an agreement rate of 70.67% on validation.

For test, on 50000 data picked from the year of 2014, we get a test accuracy of 70.34%, which is higher than our previous models.

Although the sequential model does not work very well in Chapter 5, the idea of pursuing longer memories is always correct. So upon this 68-plane model, we make another trial of adding more past

Layer (type)	Output Shape	Param #
main_input (InputLayer)	(None, 68, 34, 4)	0
conv1 (Conv2D)	(None, 100, 30, 3)	68100
bn1 (BatchNormalization)	(None, 100, 30, 3)	12
dr1 (Dropout)	(None, 100, 30, 3)	0
conv2 (Conv2D)	(None, 100, 26, 2)	100100
bn2 (BatchNormalization)	(None, 100, 26, 2)	8
dr2 (Dropout)	(None, 100, 26, 2)	0
conv3 (Conv2D)	(None, 100, 22, 1)	100100
bn3 (BatchNormalization)	(None, 100, 22, 1)	4
dr3 (Dropout)	(None, 100, 22, 1)	0
flatten (Flatten)	(None, 2200)	0
softmax (Dense)	(None, 34)	74834

Total params: 343,158
Trainable params: 343,146
Non-trainable params: 12

Fig. 6.1: Off-table model network structure

rounds' information. We add another two past round situations of past 5 and past 6, and get a data structure of 86 planes in all. We achieve an agreement rate of 70.44% on test data, which is a little bit higher than previous 68-plane one, and is also the highest agreement rate so far. So this is a bias between accuracy and computational cost.

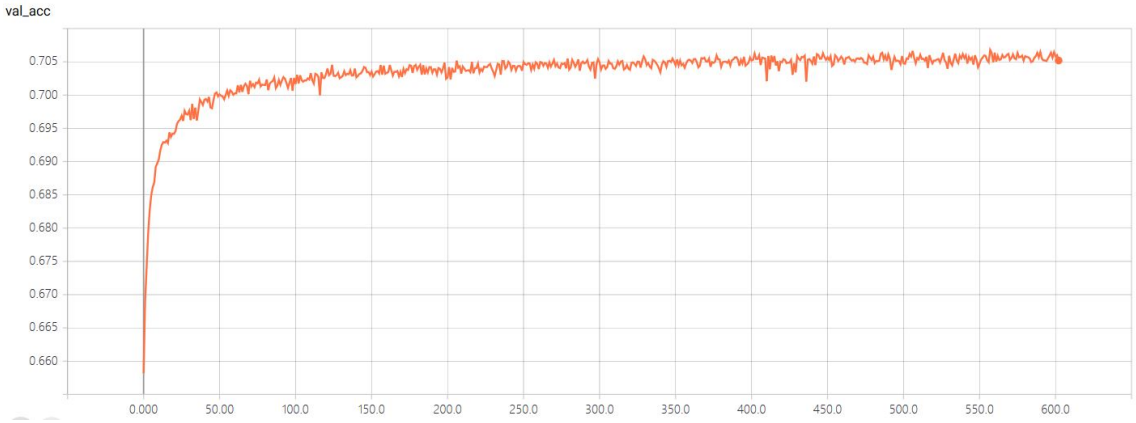

Fig. 6.2: Validation accuracy for off-table model

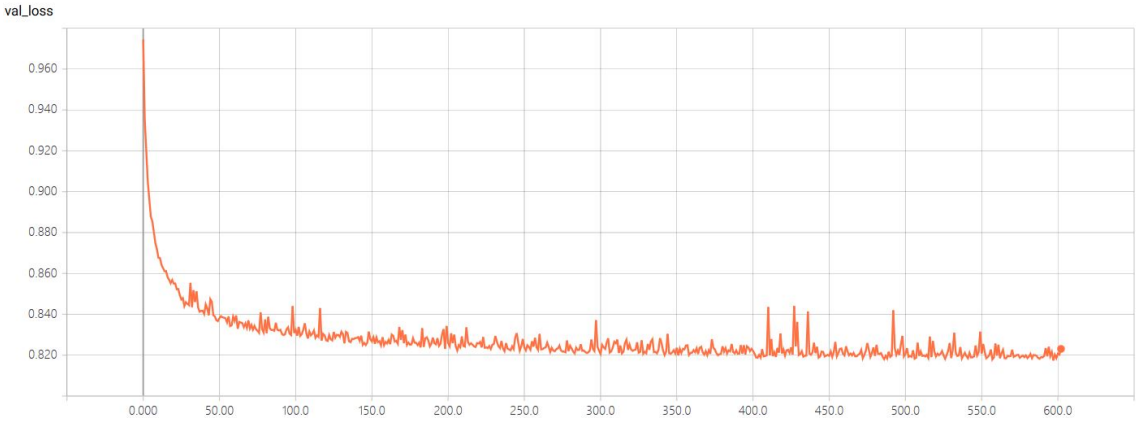

Fig. 6.3: Validation loss for off-table model

6.5 Discussion

In this chapter, we explore ways of containing missing information lost from the tile information learning model mentioned in Chapter 4. We ignore the off-table information that has specific numbers such as score and richi stick number, instead, we manage to contain rank and kyoku information successfully. And with this information learnt, we achieve the highest agreement rate on discard tile prediction so far.

6.6 Comparison Between Models

Finally, a comparison between models on discard tile prediction is made as shown in Table 6.2. Not only the first choice accuracy, rank1, is counted, but also the second and third choice, named as rank2 and rank3, are also compared.

Table 6.2: Agreement rate for tile discard

Approach	Rank1	Rank2	Rank3
Mizukami's [4]	62.1%	82.9%	90.9%
Tsukiji's [3]	53.98%		
Tile Information Model	69.48%	87.59%	93.98%
Past Information Model	68.14%		
Off-table Information Model with 68 Planes	70.34%	94.28%	96.94%
Off-table Information Model with 86 Planes	70.44%	94.37%	97.03%

It is very clear that all our models introduced before work better than past models on this task. Especially for the off-table information model with 86 planes, the rank3 accuracy reaches over 97%, which is a surprisingly high agreement rate.

Chapter 7

AI Player Build

In previous chapters, we designed data structures and models doing quite well on the benchmark of agreement rate on discard tile prediction. Although discard strategy is the most important part in this game, it is not the unique one for playing. We still can make other actions besides discard, such as pon, chi and kan which is called stealing actions and declarations of richi and ron. In this chapter, we will build a simple AI player and make a running online, playing against human players. The logic of our AI player is highly simplified, just containing several neural networks in charge of making decisions on tile discard, stealing, richi declaration, and totally without any human knowledge of this game given. We show that even by just simply combining these several networks together, the level of our AI player already exceeds an intermediate human player.

7.1 Mahjong Website of Tenhou

Tenhou [34] is one of the most popular online mahjong service platforms in Japan, and is also recognized as the most professional one among all. Therefore, we select this website for online program playing. For previous works which also made plays on Tenhou website [4, 10], they use libraries like OpenCV for extracting features from the computer screen in order to obtain information while playing online using Tenhou programs. However, this way is not sufficient and robust. In this chapter, we will use socket protocols to make a direct connection with the Tenhou server. Sockets and socket APIs are usually used to send messages across a network. The network can be either a logical, local network or one that is physically connected to an external network.

In the next section, we will make a brief introduction about protocols used between Tenhou server and how to build an AI program on Tenhou website.

7.2 Tenhou Protocols

7.2.1 Server Connection

For socket, in order to connect to a server, we need to know the host IP address and the port of it. The IP address of Tenhou server is '133.242.10.78', and the port is 10080. With IP address and port, we can use socket to construct a connection now. Besides, we need a user id in order for playing, which can be registered free from the website.

7.2.2 Game Type

We can send the game type we want for playing. The game type is encoded as an 8-bit integer. Each bit is described as below, from right to left:

0-th: 1 means playing online while 0 means playing with bots.

1-th: 1 means playing without aka five doras while 0 means including aka dora bonus

2-th: 1 means open tanyao is forbidden while 0 means open tanyao agreed.

3-th: 1 means East-South battles while 0 means with only subgames in East.

4-th: 1 means a three-player version of mahjong while 0 means playing with four players in all.
6-th: 1 means fast game (7s for decision making) while 0 means slow game (15s for decision making).
5-th & 7-th: Both are dan flags and they are combined to represent four kinds of game rooms from low to high level.

Although it seems that all combinations are legal, however normally we have four main game types, which are 1, 9, 137 and 193 respectively.

7.2.3 Tile Messages

In Tenhou server, all tiles are marked from 0 to 135. Each time a new subgame begins, the tile wall will be randomized into one sequence for players to pick from. The message of our own player's tile drawing starts with '<T' and follows with the picked tile number. The other three players are marked as '<U', '<V' and '<W' respectively. Of course, we cannot know what tiles they pick, but can only see the result that one player draws a tile from the wall. For tile discard information, the messages of our own player and other three players start with '<D', '<E', '<F' and '<G' respectively. In this time, which tiles are discards can be clearly learnt from the message.

7.2.4 Action Messages

If with a specific pattern or faced with some special situations, the server will send some suggestion message for reminding.

Winning Suggestions

't="16"' and 't="48"' are two suggestion messages players can get when facing the situation of being able to make a self-drawn. In order to announce tsumo, the player needs to send message '<N type="7"' where '<N' means doing special actions and type 7 represents for a winning declaration. Besides, t can also be equal to 8, 9, 12 and 13 when faces the situation of winning by others' discards. At this time, a message '<N type="6"' is needed to be sent.

Action Suggestions

Besides winning situations, the server also offers some suggestions about calling melds, which are listed below.

't="1"': The suggestion of being able to call a pon set. A message of '<N type="1"' is needed for reply.

't="3"': The suggestion of being able to call a kan set. A reply of '<N type="4"' calls for a closed kan while a reply of '<N type="5"' calls for an open kan.

't="4"': The suggestion of being able to call a chi set. A message of '<N type="3"' is needed for reply.

't="5"': The suggestion of being able to call either a pon set or a chi set. Either way can make a reply.

Besides stealing strategies, there is another suggestion message for richi declaration. The player will be able to declare a richi once there is 't="32"' in the message. In order for richi, a reply of '<REACH hai= / >' is needed.

7.3 Neural Network Structure Design

In last section, we briefly introduced some important points in socket connections with Tenhou server. In this section, we will make an AI program for real playing online. Since for a real play, besides tile discard strategy which is obviously the cornerstone of the game, we also have several other actions that can be made which are also indispensable factors, such as stealing strategy, richi declaration strategy and winning declaration strategy. The networks are quite similar to each other and the only difference is the last layer which makes the classification problem prediction. Since in previous chapters the model with 86 planes introduced in Section 6.4 behaves best on agreement rate of discard tile benchmark, we also adopt the main part of this network structure here. We train these strategies relatively with each neural network and make a very simple union of combining these networks together to finally form a complete AI program player.

7.3.1 Pon

One player can declare a pon when any other player discards a type of tile that the player already exists two closed ones in his own hand. The pon strategy decision can be recognized as a binary classification problem, while one represents for making the pon action and zero represents for not making that action. However,

during real training for stealing network, unbalanced dataset becomes a problem. Among all pon-able situations, only about 30% of the whole data makes a real pon, which means only 30% of positive labels exist among the dataset. Usually for normal training problems, we can use the f1 score or other techniques to deal with unbalanced data in order to achieve good performances in all categories, however it becomes different for this task to some extent. The final goal for our task is to do a good job collaborating with the discard tile strategy while playing, but not just pursuing a relatively good result in all categories. A small ratio of positive labels means in real situations people tend not to do that decision. From this perspective of view, different loss functions and judgment levels act like representing for different styles when playing, for instance normal training will lead to an average style while f1-score-focused training may lead to a more offensive playing style. In this chapter, we adopt the normal classification training judgment method at last.

7.3.2 Chi

Different from the pon decision, one player can only declare a chi with tiles discarded by the left-side player to form a legal sequence meld. Chi situations get a more severe ratio between positive and negative labels, among which only about 14% of data has positive labels. Besides, not like pon, chi has three dealing methods with one tile, for that called tile can be either the smallest, largest or the middle part of the chi meld formed. Therefore, we recognize chi as a 4-class classification problem, with zero denoting action canceling, one, two, and three denoting three types of chi actions with the called tile at the left, the middle and the right part of the meld called.

7.3.3 Kan

For kan-able situations, we only have too small dataset since compared with pon and chi situations, kan becomes much rarer, and the ratio of declaring a kan is too low among the training dataset. Therefore, here we take a simplified strategy that we will not declare any open kans and will only declare a closed kan when that kan is an isolated one which means that tile does not own any potential chances of involving in declaring a chi meld according to the present hand tile pattern. Therefore, for kan decisions, we do not make a network for training, but make judgements by just a very simple check.

7.4 Richi

Besides stealing networks, we also build a network trained for making richi decisions. Of course the strategy of making instant richi declaration once one can is a more simplified way, but it may cause problems because according to the game rules, once the player declares richi, he cannot decide the discard tile by himself, so it usually leads to dangerous situations. Besides, different hand patterns will lead to different scores, so sometimes waiting for several rounds to form a better winning pattern could also happen. The network structure is the same as the pon strategy network, since both are binary-classification problems.

7.5 Wining Declaration

In this chapter, we adopt a simplified strategy for winning declaration. Although the player can have a chance to decide whether to announce wining even he faces a winning situation, but here for our program, we simplify this strategy by announcing the winning once the player can.

7.6 Training Data and Experiment

Our models and networks are trained on NVIDIA Tesla K10 GPU with 32 GB memory size. We go through all the game records from 'Houou' table in the year of 2015 for training. Details will be introduced in the next section. We set the dropout rate of 0.5 and a batch size as 256 for training. Adam is adopted as the optimizer with an initial rate of 0.001.

7.7 Training Results

7.7.1 Pon

For the year of 2015, we obtain in all 212 845 valid situations for pon strategy training. Our validation accuracy arrives 88.39% during training as shown in Fig. 7.1 and Fig. 7.2. For the test dataset, we collect 30 000 data from the previous year of 2014. Test results are shown in Table 7.1. The rows represent for real labels of True or False, while the columns represent for the predictions made by the neural network.

We achieve a whole accuracy of 88.32% on test data. We get precision of 81.46% and recall of 78.21%, leading to an f1 score of 0.798.

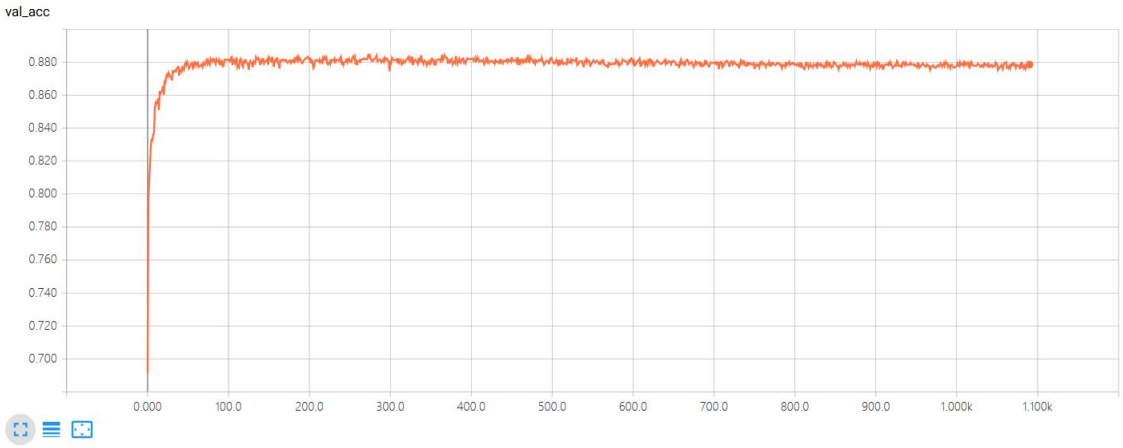

Fig. 7.1: Validation accuracy for pon network

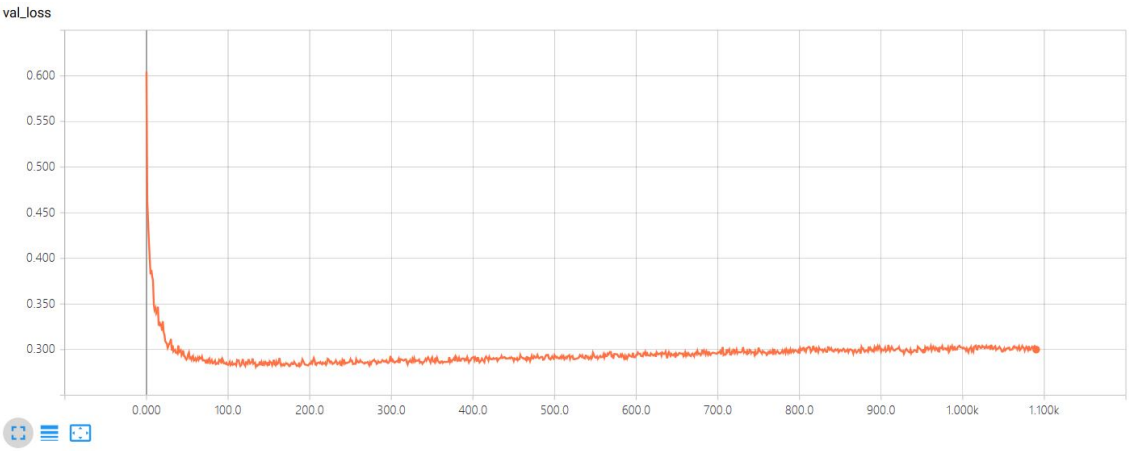

Fig. 7.2: Validation loss for pon network

Table 7.1: Test results on pon network

	True	False
True	6923	1576
False	1929	19572

7.7.2 Chi

For the year of 2015, we obtain in all 315 692 valid situations for chi strategy training. Our validation accuracy arrives 90.79% during training, as shown in Fig. 7.3 and Fig. 7.4. For test, we collect 40 000 data from the previous year of 2014. Test results are shown in Table 7.2. The rows represent for real labels, while the columns represent for the prediction ones. As a 4-class classification problem, instead of True/False predictions, we make a prediction of 0, 1, 2 and 3. Here 0 means chi canceling, while the other three numbers indicate three types of chi mentioned before. We achieve a whole accuracy of 90.62% on test data. We can also notice that, for real chi-called labels, to call which kind of chi reaches a very high accuracy of around 96.11%, which means once the player decides to call a chi set, he can usually make the chi action type right. If we simplify it as a whether-to-chi-or-not binary classification problem, the accuracy will raise by another 0.3%.

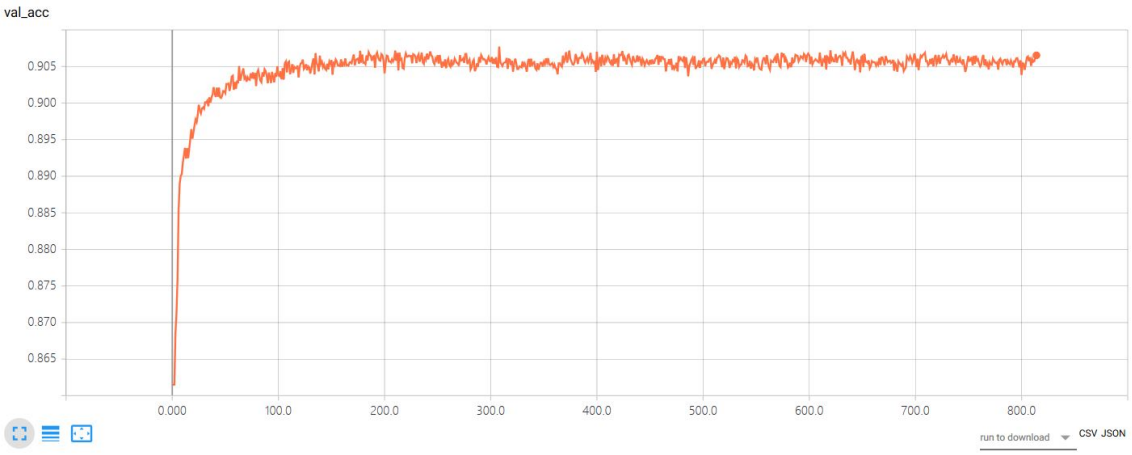

Fig. 7.3: Validation accuracy for chi network

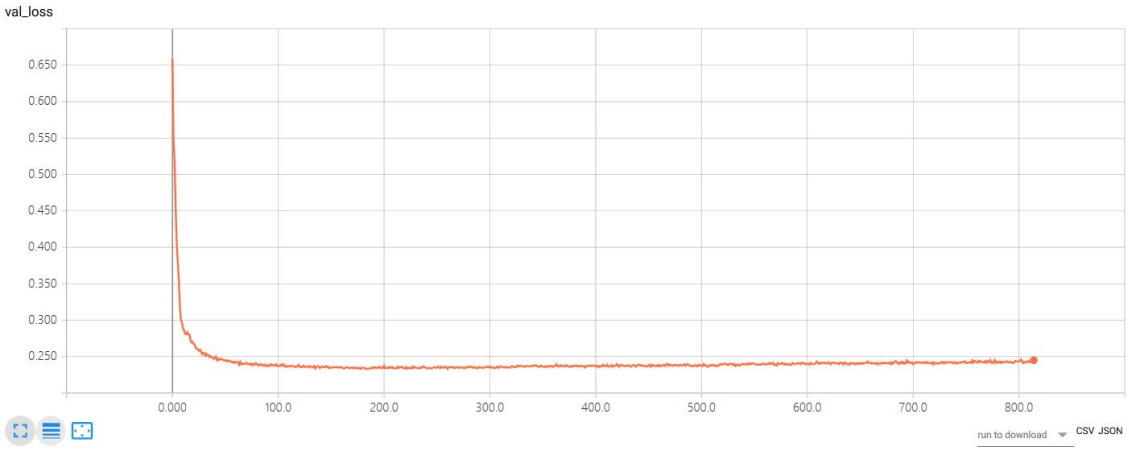

Fig. 7.4: Validation loss for chi network

Table 7.2: Test results on chi network

	0	1	2	3
0	33357	817	777	858
1	338	813	22	9
2	509	18	1289	29
3	335	7	32	790

7.7.3 Richi

For the year of 2015, we obtain in all 99 999 valid situations for richi strategy training. Our validation accuracy arrives 77.18% during training. For test, we collect 15 000 data from the previous year of 2014. Test results are shown in Table 7.3. The rows represent for real labels of True/False and the columns represent for the predictions same as in pon strategy. We achieve a whole accuracy of 75.85% on test data.

Table 7.3: Test results on richi network

	True	False
True	3017	1915
False	1558	8510

7.8 Evaluation on Tenhou

We make our program playing matches on Tenhou sever against human players. The rule is set with East-South Battle, with Aka dora and open tanyao. Here we choose to play East-South Battles instead of East Battles for East Battles are usually too short that fortune takes too much ratio. We double the length of the game in order to get a more precise performance judgment. After playing 300 matches, we reach a rating of around 1850, while an average intermediate player is around 1600, which means our program is much stronger than an intermediate player.

CHAPTER 8

CONCLUSIONS

8.1 Summary

As the title already stated, this work focuses on imperfect information game record learning in the game of Mahjong. New models using deep convolutional neural networks are proposed, with a newly designed three-dimensional matrix managing to contain information available on table. We made well training on deep convolutional neural networks. We adopted the agreement rate of discard tile as the benchmark and simplified this prediction task as a multi-classification problem. We did a very good job on tile information learning, reaching an agreement rate of 69.48% while state-of-art accuracy is just 62.1%. We managed to make improvements in two ways. We designed another sequential model in order for longer memories. Based on the GRU model, with half data filled with 0 paddings we achieved an agreement rate of 68.14%, which is also much higher than past works. We also made a try to include some off-table information that

is missing in our previous models. We successfully imported rank and kyoku information, and achieved an accuracy of 70.44%, which is the best result on this benchmark so far. Finally, we built an AI program for real playing on Tenhou website. We show that even by just simply combining several trained neural networks together without any human knowledge, our program reaches a rating of around 1850 after 300 matches, while the average rating for an intermediate player is just around 1600.

8.2 Future Work

The followings could be regarded as our future work.

1. Better dealing with other missing information.

   Although we managed to contain rank and kyoku information into our model as mentioned in Chapter 6, we still have other information such as richi stick number and honba number missing. We took trials including these ones, but no obvious improvements can be seen. Besides, as introduced, the rank information can be recognized as an approximation of the score information, so this is still not perfect enough. A small score difference and a large score difference may cause totally different actions actually.

2. Better dealing with sequential models.

   In Chapter 5, we adopted invariant time-steps for GRU training. With about half among all training data filled with 0 paddings, we achieve an agreement rate better than past studies but a little bit worth than our other models. The improvement way is trying to design a variant time-step model for training, which may be difficult but necessary for this task.

3. Better strategy design for the program.

   In Chapter 7, we constructed an AI player by just simply combining several trained networks together. Although it achieved a relatively good result, it is still not perfect. We adopted simplifications for kan and winning declaration strategies. Besides, the player may get no idea when facing some very rare situations, for these situations are too rare, the player is not well-trained for them.

4. Reinforcement learning method explore.

As stated in the title, this work uses the supervised learning method for game record study. Although the game records learnt are all of the good quality, it does not mean all their decisions are the best under their situations. The player can achieve a level around the players it learns, but will be hard to pass their levels by only supervised learning methods. In order for further improvements, an reinforcement learning method may be needed then.

Publications

Domestic General Conferences

[1] Gao, S., Okuya, F., Mizukami, N., Kawahara, Y. and Tsuruoka, Y., 2018. Improved Data Structure and Deep Convolutional Network Design for Haifu Data Learning in the Game of Mahjong. 情報処理学会第 80 回全国大会, 2, p.05.

[2] Gao, S., Okuya, F., Kawahara, Y. and Tsuruoka, Y., 2019. A behavior analysis of sequential and off-table information in the game of Mahjong via deep convolutional neural networks. 情報処理学会第 81 回全国大会, To Appear.

Domestic Workshops

[3] Gao, S., Okuya, F., Kawahara, Y. and Tsuruoka, Y., 2018. Supervised Learning of Imperfect Information Data in the Game of Mahjong via Deep Convolutional Neural Networks. ゲームプログラミングワークショップ 2018 論文集.

Acknowledgements

To mark an end for my master student life, I would like to make great thanks to Professor Kawahara, my advisor, together with Professor Tsuruoka and Dr. Okuya for their great supervisor and advice for my research and paper writing. I would also like to make thanks for Dr. Mizukami, without his past studies in this field, I might have no opportunity making such an interesting research like this.

I have been learning a lot during my two-and-half year life in AKG Lab. Not only for how to conduct a meaningful research, how to tell a good story and how to write a scientific paper, but also for how to live in Japan, how to make a job search and how to make a life plan. I would like to thank all the members of AKG Lab, especially Okuya, Ikeuchi, Weiwei, and Lee for all of their support, advice and patience to my scientific research, life in Japan and Japanese communications.

I would like to make thanks to my college mates in Zhejiang University with whom came to study at the University of Tokyo together. When we first arrived in Japan, we helped each other and shared our experiences both in life and in study. I appreciate that very much.

I am terribly grateful for my parents and grandparents for being my great supporters in all fields. Finally, all my thankfulness go to my lovely girlfriend, for bringing light and happiness to my life.

Thank you very much.

References

[1] Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural networks, 61, pp.85-117.

[2] 築地毅 and 柴原一友, 2015. ディープラーニング麻雀-オートエンコーダとドロップアウトの有効性. ゲームプログラミングワークショップ 2015 論文集, 2015, pp.136-142.

[3] 築地毅 and 柴原一友, 2017. CNN 麻雀-麻雀向け CNN 構成の有効性. ゲームプログラミングワークショップ 2017 論文集, 2017, pp.163-170.

[4] 水上直紀 and 鶴岡慶雅, 2015. 期待最終順位に基づくコンピュータ麻雀プレイヤの構築. ゲームプログラミングワークショップ 2015 論文集, 2015, pp.179-186.

[5] Juergen Schmidhuber. http://www.idsia.ch/ juergen/

[6] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. nature, 521(7553), p.436.

[7] Gulcehre, C., Moczulski, M., Denil, M. and Bengio, Y., 2016, June. Noisy activation functions. In International Conference on Machine Learning (pp. 3059-3068).

[8] Brown, N. and Sandholm, T., 2017. Safe and nested subgame solving for imperfect-information games. In Advances in Neural Information Processing Systems (pp. 689-699).

[9] Brown, N., Sandholm, T. and Amos, B., 2018. Depth-Limited Solving for Imperfect-Information Games. arXiv preprint arXiv:1805.08195.

[10] Mizukami. N. and Tsuruoka. Y.: Building a computer Mahjong player based on Monte Carlo simulation and opponent models, Computational Intelligence and Games (CIG), 2015 IEEE Conference on, IEEE, pp.275–283 (2015).

[11] Mizukami, N. and Tsuruoka, Y., 2015, August. Building a computer Mahjong player based on Monte Carlo simulation and opponent models. In Computational Intelligence and Games (CIG), 2015 IEEE Conference on (pp. 275-283). IEEE.

[12] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. Mastering the game of Go without human knowledge. Nature, 550(7676), p.354.

[13] Allis, L.V., 1994. Searching for solutions in games and artificial intelligence. Wageningen: Ponsen & Looijen.

[14] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[15] 北川竜平, 三輪誠 and 近山隆, 2007. 麻雀の牌譜からの打ち手評価関数の学習. ゲームプログラミングワークショップ 2007 論文集, 2007(12), pp.76-83.

[16] 西野順二 and 西野哲朗, 2011. 多人数不完全情報ゲームのモンテカルロ木探索における推定の効果. 研究報告バイオ情報学 (BIO), 2011(31), pp.1-4.

[17] 小松智希, 成澤和志 and 篠原歩, 2012. 役を構成するゲームに対する効率的な行動決定アルゴリズムの提案. 研究報告ゲーム情報学 (GI), 2012(8), pp.1-8.

[18] 我妻敦, 原田将旗, 森田一, 古宮嘉那子 and 小谷善行, 2014. SVR を用いた麻雀における捨て牌の危険度の推定. 研究報告ゲーム情報学 (GI), 2014(12), pp.1-3.

[19] 原田将旗, 古宮嘉那子 and 小谷善行, 2014. 麻雀における手牌と残り牌からの上がり探索による着手決定アルゴリズム CHE. 研究報告ゲーム情報学 (GI), 2014(13), pp.1-4.

[20] 佐藤諒, 西村夏夫 and 保木邦仁, 2014. 有効牌を数えて牌効率をあげる面前全ツッパ麻雀 AI の性能評価. 研究報告ゲーム情報学 (GI), 2014(11), pp.1-6.

[21] Sato, H., Shirakawa, T., Hagihara, A. and Maeda, K., 2017. An analysis of play style of advanced mahjong players toward the implementation of strong AI player. International Journal of Parallel, Emergent and Distributed Systems, 32(2), pp.195-205.

[22] Wiki. https://en.wikipedia.org/wiki/Japanese_Mahjong/

[23] Introduction to Exponential Linear Unit, https://medium.com/@krishnakalyan3/introduction-to-exponential-linear-unit-d3e2904b366c/

[24] Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256).

[25] Nair, V. and Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).

[26] Nash, J.F., 1950. Equilibrium points in n-person games. Proceedings of the national academy of sciences, 36(1), pp.48-49.

[27] Bowling, M., Risk, N.A., Bard, N., Billings, D., Burch, N., Davidson, J., Hawkin, J., Holte, R., Johanson, M., Kan, M. and Paradis, B., 2009, May. A demonstration of the Polaris poker system. In Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2 (pp. 1391-1392). International Foundation for Autonomous Agents and Multiagent Systems.

[28] Abramson, B., 2014. The expected-outcome model of two-player games. Morgan Kaufmann.

[29] Coulom, R., 2006, May. Efficient selectivity and backup operators in Monte-Carlo tree search. In International conference on computers and games (pp. 72-83). Springer, Berlin, Heidelberg.

[30] Kocsis, L. and Szepesvri, C., 2006, September. Bandit based monte-carlo planning. In European conference on machine learning (pp. 282-293). Springer, Berlin, Heidelberg.

[31] Gelly, S., Wang, Y., Teytaud, O., Patterns, M.U. and Tao, P., 2006. Modification of UCT with patterns in Monte-Carlo Go.

[32] Wiki. https://en.wikipedia.org/wiki/Monte_Carlo_tree_search/

[33] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S., 2012. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1), pp.1-43.

[34] Tenhou. https://tenhou.net/

[35] Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[36] Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

[37] Hochreiter, S., 1991. Untersuchungen zu dynamischen neuronalen Netzen. Diploma, Technische Universitt Mnchen, 91(1).

[38] Wiki. https://en.wikipedia.org/wiki/Recurrent_neural_network/

[39] Cho, K., Van Merrinboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

[40] Implementing a GRU/LSTM RNN with Python and Theano. http://www.wildml.com/