Improving Mahjong Agent by Predicting Types of Yaku

Long Honghai[1,a)] Tomoyuki Kaneko[1,2,b)]

Abstract: Over recent years, many great achievements have been made in the area of Artificial Intelligence. There are AI agents which have already exceeded the level of top human players in perfect information games such as playing go [1]. On the other hand, there is still some work to do with imperfect information games. In this paper, we have decided to choose Japanese Mahjong as our research object because of its popularity all over the world and appropriate difficulty. And in a previous research, an agent trained by a convolutional neural network finally reached a good rating in online mahjong site Tenhou [2]. In this paper, we tried to use an array of numbers to record the current states and tried to use Deep Neural Networks to fit the model. Our accuracy on validation data reached 66.81% which is a little bit lower than that in research [3]. We also tried to train a neural network to predict possible yaku. We showed that if we added predicted yaku information to the input of the neural networks, the accuracy of our prediction would increase.

Keywords: Japanese Mahjong, Convolutional Neural Networks, Deep Neural Networks

1. Introduction

Many strong agents have been trained by researchers in more and more types of games these years. Human experts have been defeated in many perfect information games, such as go and chess. But it is much more difficult to train an agent in imperfect information games.

In this paper, we choose Japanese Mahjong, an imperfect information game, as the object of our study because of its popularity and appropriate difficulty. We collected records, which is also called 'haifu', of top players from online mahjong site Tenhou. We trained our agent to discard the same tile as top players do as well as possible and the agent finally reached an accuracy of 64.88% with no yaku prediction information and 66.81% with predicted yaku prediction information.

The paper is organized as follows. Section 2 introduces the game of mahjong and then rules of Japanese Mahjong. Section 3 shows some related works and studies. In Section 4 , we will introduce the way we represent the information and the neural networks we used to train the agent. Section 5 shows some results of our research. Section 6 will summarize the paper and show some of our future works.

2. Basic Rules and Terms of Japanese Mahjong

Mahjong was developed in China from the Qing dynasty. It is a tile-based game and usually played by four players. Mahjong is based on draw-and-discard card games which were popular in 18th and 19 century China. In 1924, mahjong was first brought into Japan.

Japanese Mahjong is a standardized form of Mahjong in Japan and there are totally 136 tiles in the game. More specifically, there are 34 types of tiles and each type has 4 same tiles. The 34 types are 1m-9m (man), 1p-9p (pin), 1s-9s (sou), wind tiles (East, South, West, North), and dragon tiles (Haku, Hatsu, Chun). At the beginning of one game, the tiles are disordered and arranged into four walls. Each wall has two stacks high and 17 tiles wide. 26 of the stacks are used for starting hands and 7 of them are used for a dead wall [4]. Each player will have 13 tiles in his hand, and after drawing a tile from the wall, there will be 14 tiles. A player can call win if his hands can be expressed by the following combination: x (AAA)+y (ABC)+DD, x+y=4. Three same tiles compose an AAA, three number tiles with sequential numbers compose an ABC and two same tiles compose a DD. One can also declare a win by other players' discards. There are some other types of winning hand such as seven pairs, which means a player forms even pairs in his hands. In each round, one player needs to draw a tile from the wall and discard a tile from his hands (including the drawn tile). A player can also call a pon, chi, or kan from others' discards in certain situations. At the beginning of the game, each player will have a score of 25000, and the score will change by the result of each subgame. A whole Japanese Mahjong game usually has four or eight subgames. In addition to score changes, Japanese Mahjong also has some unique rules such as richi and dora, which will be explained in the later subsection.

2.1 Yaku and Yakuman

Yaku are specific combinations of tiles. A winning hand must have at least one yaku. Yakuman is a value for some hands which are very difficult to form. When scoring, each yaku has its own han value, and the han works as a doubler. A player who is new at this game usually forms a hand with no yaku, which will cause that he can not call win in this subgame. So it is very important and this is why we tried to add this information into the input.

[1] Interfaculty Initiative in Information Studies, the University of Tokyo
[2] JST, PRESTO
[a)] longhonghai@g.ecc.u-tokyo.ac.jp
[b)] kaneko@acm.org

2.2 Richi

A player can declare richi if the player needs only one tile to complete a legal hand and the player has not claimed other players' discards to make open melds. Richi is a kind of yaku and a player can win on an other players' discard or a tile drawing from the wall after declaring richi. The player can not change his hand except when forming closed quads. The player may win more scores in a subgame by declaring richi. If another player has declared richi, we have to be careful because we might pay large scores if we discarded a dangerous tile.

2.3 Dora

Dora is a bonus tile that adds han value to a hand. It may cause a big winning score in some subgames. At the beginning of each subgame, there will be a dora indicator which are shown to all players. Every time a player calls a kan, a new dora indicator will be added and there are at most five dora indicators. The next tile of the dora indicator is regarded as dora. More specifically, the next tile of 9man (pin or sou) is 1man (pin or sou). The order of wind tiles are: East, Sorth, West and North. The order of dragon tiles are: Haku, Hatsu, Chun.

Depending on the type of games, there can also be some akadoras which are three five tiles (one 5-man tile, one 5-pin tile, one 5-sou tile). Akadoras will be colored red in the game and they can be regarded same function as normal dora tiles.

There is an uradora indicator under each dora indicator and uradora indicators are only revealed and calculated when a player wins after calling richi. It can bring a big winning score in sometimes.

2.4 Wind Tiles and Dragon Tiles

As explained above, there are four types of wind tiles and three types of dragon tiles in each game.

Two type of wind tiles are very important for each player: prevailing wind and own wind. Prevailing wind is set at the beginning of a subgame and all four players have the same prevailing wind in a certain subgame. Prevailing wind is East at first and may change in the game. On the other hand, every player has different own wind in a certain subgame. The player who first draw a tile from the wall (we also call this player 'oya' or 'dealer') has East as his own wind. The next player's own wind is North, and so on. A player can use a triplet of prevailing wind tiles or own wind tiles as a yaku.

For dragon tiles, a player can use a triplet of them as a yaku in arbitrary subgames.

2.5 Winning

A player can call win from drawing a tile or an another player's discard. Winning from a wall is also called Tsumo. When a player declares win by tsumo, all other players will pay scores to him. On the other hand, if a player declares win by a discard from another player, only the player who discarded this tile will pay the winner scores. The dealer always pays more and wins more when scoring.

3. Related works

In research [3], three networks have been trained. The first one is discarding network. Discard is recognized as a 34-classification problem. A player has most 13 tiles in his hand and he can only choose one of them to discard. But after training as a 34-classification problem, the agent successfully learned to discard a tile that appears in its hand and it performed better than agent trained as a 14-classification problem. The second network is stealing network. Pon is recognized as a binary classification problem: 0 for cancel and 1 for calling a pon. Chi is recognized as a 4-classification problem: 0 for cancel and 1 to 3 represent 3 types of chi. The third one is richi network. The agent will declare richi once it can and the discard selection after declaring richi is the same as the normal discarding network.

In research [5], prediction of the final rank is recognized as a 4-classification problem. Then the prediction will be used to decide whether ori (give up winning and discard a safe card) should be chosen or not.

In research [6] an one-player mahjong agent is trained as first. This agent only gets a tile from the wall and discard a tile after that. It will choose how to act as a one-player mahjong agent only after it does not choose to ori. Whether it should choose ori or not depends on some other features. Current strategy is trained as a 2-classification problem.

A new data structure called plane has been designed and it has been used to train an agent with convolutional neural network in research [3], but information such as akadora and current rank have not been considered. The agent finally achieved a rating of 1822 after 76 battles.

One player mahjong agent has been extended to be used in 4-players mahjong games and reached a rating of 1507 in research [5].

The agent trained with Expected Final Rank (EFR) can play fast in the game and reached a rating of 1844 with 1508 battles in research [6].

4. Data Structure and Neural Networks

In paper [3], the author used a data structure named as 'plane' to represent the current game and then used a convolutional neural network to train the agent. In research [7], reinforcement learning and deep neural networks were combined and strong agents have been trained in playing Atari 2600 games. In this paper, we suppose to use simple data structures to store the information of current situation and then use deep neural networks to let the agent imitate top human players.

4.1 Data Structure

To store the information, we use an array of numbers to represent the features as shown in **Table 1**.

We firstly use an array of length 472 to save the information of the game. The first value of the array is our seat number. The following 34 values are our hands because there are totally 34 types of tiles in this game. Each value saves the number of a certain tile in our hand. For example, if we have a tile '1m' in our hands,

Table 1 Features Represented in Array

Feature Name	Length	Index in Array	Possible Value
Seat Number	1	[0,0]	0,1,2,3
Hands	34	[1,34]	0,1,2,3,4
Discards	136	[35,170]	0,1,2,3,4
Melds	256	[171,426]	0,1
Dora Indicators	34	[427,461]	0,1,2,3,4
Prevailing Wind	1	[462,462]	0,1,2,3,4
Own Wind	1	[463,463]	0,1,2,3,4
Richi Players Last Round	4	[464,467]	0,1
Richi Players	4	[468,471]	0,1
Prediction with Yaku	9	[472,480]	[0,10]
Pon or Chi Tiles	1	[472,472]	0,1,2,...,33
Chi&Pon (With Yaku)	9	[473,481]	[0,10]

we increase the value which represents '1m' by one. Besides our hands, we need to know the discards of each player. So we use 136 numbers (34 for each player) to save the discards and we ignore the order of the discards in this method. If a player discards a tile, the value of the corresponding position increases by one. The next 34 numbers store the information about dora indicators. The melded tiles are significant as well and we use 256 numbers to store this information. We used online mahjong site Tenhou to test our agent and it used a 16-bit integer to represented every meld type and corresponding tiles. So we used 64 numbers for every player (16 for each player) and believed that the agent could learn from it. Information about prevailing wind and own wind are also important so we use four numbers for prevailing wind, four numbers for own wind to save these two information. The value represents current wind and own wind is one and the values of other winds are zero. We then used four numbers for players who called richi last round and four numbers for players who have called richi. If a player called richi last round, the value of its corresponding position is one. We added the last round richi players' information like [3] because richi players win more in the first round after declaring richi and we need to be careful.

So we finally accomplish a 472-length array as an input of the neural networks. We will use the array to predict which tile we should discard and what yaku we can form if we suppose that we would win the subgame. When we train a chi network or a pon network, we would add the information about the lastest discarded tile and when we train a network with yaku information, we would add the prediction of yaku. More details will be shown later in this section. Table 1 also shows all features we used.

Also, we supposed another structure, which is more complex, to save the current information. In the occasion of the 472-length array, we just stored all discards but ignored the order of them. But it is important to know the order in some certain cases. For example, if a player declared richi and the next player discarded a tile '1m'. We can know that tile '1m' is a safe tile if the richi player does not declare a win. On the other hand, if a player discarded '1m' and the next player declared richi, '1m' is not safe but even very dangerous in this case. In order to save the sequence of discards, we suppose to use an array of length 178 and then use one-hot encoding. The first 14 numbers (start from index 0) will be tiles in our hands. Before we store them into the array, we have to sort them by their indexes in all tiles. Then we use totally 100 numbers in tatol (25 for each player) to save the sequence of discards. The numbers of discards are all -1 at first, and they will be changed into tiles the player discards one by one. For example, we use numbers from index 14 to 38 to save our own discards. It is initialized as [-1,-1,...,-1] and when we discard a tile 'x', it becomes [x,-1,-1...,-1] after that. We use a similar way to save the meld tiles. Each player can call meld at most 4 times and most 4 tiles each time. So we use 16 numbers for each player and totally 64 numbers for all four players. Wind and dora information will be added after one-hot encoding and then we will train our agent by convolutional neural networks.

In this paper, we tried our first method and showed that the accuracy increased after we added yaku to the features.

Table 2 Network Structure of Discard Network

Layer (type)	Output Shape	Param #
fc_0 (Dense)	(None, 300)	144600
fc_1 (Dense)	(None, 1000)	301000
fc_2 (Dense)	(None, 30000)	30030000
fc_3 (Dense)	(None, 300)	9000300
fc_34 (Dense)	(None, 34)	10234

4.2 Neural Networks

We use deep neural networks to train our agent. The layers of the discard network are different from that of chi, pon or yaku prediction networks. Compared to chi, pon and yaku prediction networks, the discard network is much larger. We trained all networks except yaku prediction network twice. The first time we did not add yaku predcition information into input and the second time we added it. The difference between networks with yaku and network without yaku is only the input. The following subsections will introduce them in details.

4.2.1 Discard Network

In this paper, we also regard the discard problem as a 34-class classification problem like [3]. We just have at most 14 different tiles in our hands, but our agent can choose a tile to discard which is exactly in its hand with very high probability after training. It also showed that our agent had learned much knowledge from the training. The networks of training data with yaku and

Table 3 Network Structure of Pon Network

Layer (type)	Output Shape	Param #
fc_0 (Dense)	(None, 8)	3864
fc_1 (Dense)	(None, 8)	72
fc_2 (Dense)	(None, 8)	72
fc_3 (Dense)	(None, 8)	72
fc_4 (Dense)	(None, 8)	72
fc_5 (Dense)	(None, 8)	72
fc_6 (Dense)	(None, 8)	72
fc_7 (Dense)	(None, 8)	72
fc_8 (Dense)	(None, 8)	72
fc_9 (Dense)	(None, 8)	72
fc_10 (Dense)	(None, 8)	72
fc_11 (Dense)	(None, 8)	72
fc_12 (Dense)	(None, 8)	72
fc_13 (Dense)	(None, 8)	72
fc_14 (Dense)	(None, 8)	72
fc_sigmoid (Dense)	(None, 1)	9

Table 4 Network Structure of Chi Network

Layer (type)	Output Shape	Param #
fc_0 (Dense)	(None, 512)	65664
fc_1 (Dense)	(None, 128)	4128
fc_2 (Dense)	(None, 32)	1056
fc_3 (Dense)	(None, 32)	1056
fc_4 (Dense)	(None, 32)	1056
fc_5 (Dense)	(None, 32)	1056
fc_6 (Dense)	(None, 32)	1056
fc_7 (Dense)	(None, 32)	1056
fc_8 (Dense)	(None, 32)	1056
fc_9 (Dense)	(None, 32)	1056
fc_10 (Dense)	(None, 32)	1056
fc_11 (Dense)	(None, 32)	1056
fc_softmax_4 (Dense)	(None, 4)	132

Table 5 Network Structure of Yaku Prediction Network

Layer (type)	Output Shape	Param #
fc_0 (Dense)	(None, 512)	242176
fc_1 (Dense)	(None, 512)	262656
fc_2 (Dense)	(None, 256)	131328
fc_3 (Dense)	(None, 256)	65792
fc_4 (Dense)	(None, 256)	65792
fc_5 (Dense)	(None, 64)	16448
fc_6 (Dense)	(None, 1)	65

without yaku have the same layers except the input layer. The network works better with a large batch size and we finally chose to use a batch size of 16,384. **Table 2** shows more details about this network. We added regularizers in the layer which had most nodes to reduce overfitting.

4.2.2 Naki Network

There are three types of naki in Japanese Mahjong. We ignore kan because of too little data. Compared with discard network, we used relatively small neural networks to train chi an pon because we considered it easier than discard problem and we have much less data of pon and chi than discard. The discard after we call a pon or chi is decided by discard network. Before using discard work, we need to remove the tiles we used to call chi or pon from our hands. The following networks are trained for deciding whether we should chi or pon in certain situation.

4.2.2.1 Pon Network

We regarded pon as a binary classification. A player can call pon only if he has two or more than the same tiles with the discarded tile. In about 70% of our training data, a player called a pon when he could. We used pon network only when we could call a pon, and just used this network to decide whether we should make this call or not. In the training data, we have to know the latest discarded tile, so we add the tile to the end of the 472-length array. The length of the input increase from 472 to 473. The value of the tile is an integer between 0 and 33, corresponding 34 different types of tiles. When the nerual network was big, pon network got overfitting easily. So we finally use layers with just 8 nodes per layer. **Table 3** shows the whole pon neural network.

4.2.2.2 Chi Network

We regarded chi problem as a four-class classification problem. A player can call chi only from a discard of his left player, who is prior in order. Before we used the network to predict whether we should chi or not, we firstly judge if the discard came from

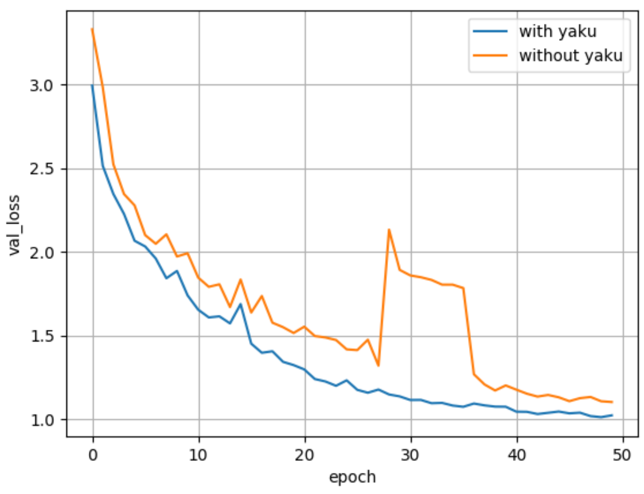

Fig. 1 Loss of Discard Network

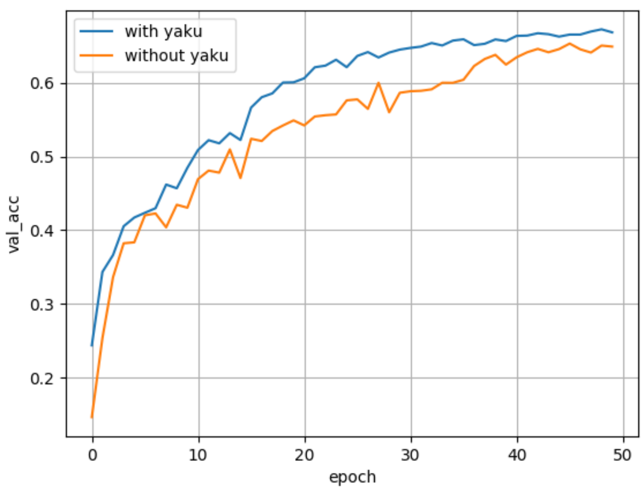

Fig. 2 Accuracy of Discard Network

our left player. And when we decided to declare chi, we had to declare the type of chi. So we defined zero as not calling chi, one, two and three as three types of chi. And we also have to know the latest discarded tile, so the length of the input is also 473, the same with pon network. If the agent chose a type of chi but actually we could not do this type of chi, we also canceled chi. **Table 4** shows more details about it.

4.2.3 Yaku Prediction Network

There are totally 55 yaku in Japanese Mahjong and 15 of them are yakuman. Yakuman are very hard to form and we have too little data so we don't consider it. We trained our agent to predict what yaku it could form except yakuman from the current hands and other information. For each yaku prediction, we regarded it as a binary classification problem and trained 40 models for each yaku. In this paper, each network has the same network structure as shown in **Table 5**.

4.2.4 Network with Yaku

When we tried to train a discard, pon or chi network with yaku information, we firstly used the 472-length input and the yaku prediction network to predict which yaku we could form in this subgame. We regarded the output value of the yaku prediction network as the probability of we could form this yaku and when we added it to the input array, we timed its value by 10 to emphasize it. From the training result, we finally chose nine types of yaku and added them at the end of the input array. All these nine types of yaku could be predicted at a relatively high accuracy. The total length of the input array became 481 in discard network with yaku prediction information and 482 in chi and pon networks with yaku prediction information. We did not change network layers in order to compare the training result between networks with yaku and without yaku.

5. Result

5.1 Discard Network

We used 1,400,000 record data for our training and 10% of it is the validation dataset. The discard network without yaku information finally reached an accuracy of 64.88% and the one with yaku reached an accuracy of 66.81%. As said above, we regarded the discard problem as a 34-class classification problem. The agent can choose a tile which is exactly in its hand at an accuracy above 99.96% after training. From **Fig. 1** and **Fig. 2**, we can easily find out that accuracy became higher after we added yaku prediction information.

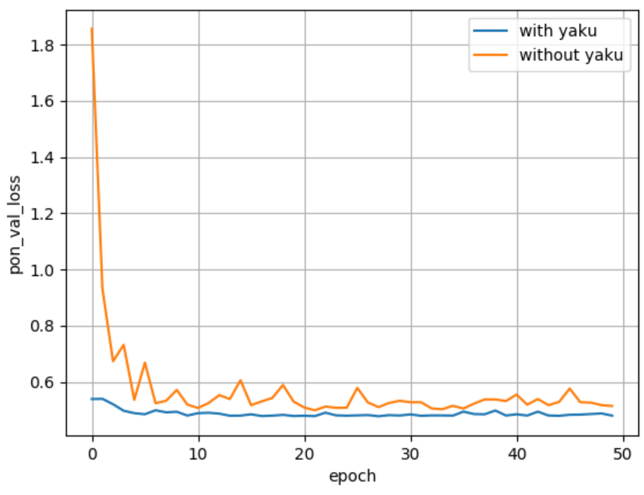

Fig. 3 Loss of Pon Network

5.2 Pon Network

We used about 160,000 data to train the pon network. The pon network without yaku information reached an accuracy of 76.25% and the one with yaku information reached an accuracy of 77.26%.The graph **Fig. 3** and **Fig. 4** show the loss and accuracy changes during the training. **Table 6** and **Table 7** show the result of pon network. From the figure, we can see that pon network became much more stable after we added yaku information.

5.3 Chi Network

There are 140,000 data for training a chi network. The chi network without yaku information reached an accuracy of

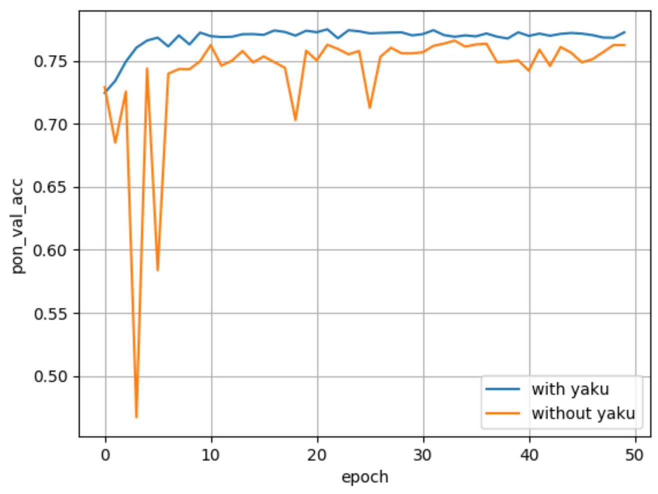

Fig. 4 Accuracy of Pon Network

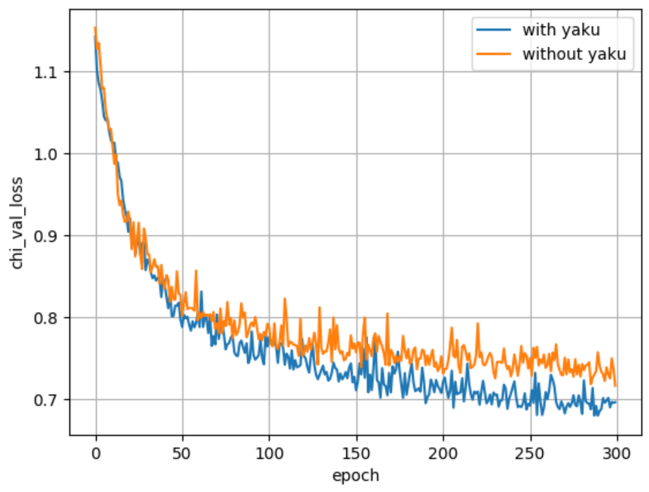

Fig. 5 Loss of Chi Network

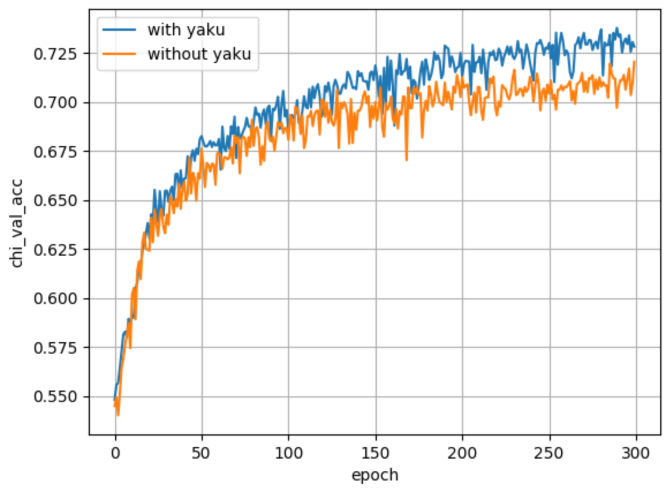

Fig. 6 Accuracy of Chi Network

Table 6 Pon Network Result without Yaku

Predicted \ Actual	True	False
True	11223	3700
False	528	1542

Table 7 Pon Network Result with Yaku

Predicted \ Actual	True	False
True	10189	2890
False	1562	2352

Table 8 Chi Network Result without Yaku

Predicted \ Actual	True	False
True	3979	1817
False	2346	6752

Table 9 Chi Network Result with Yaku

Predicted \ Actual	True	False
True	4332	2148
False	1901	6513

72.05% and the one with yaku information reached an accuracy of 72.82%. The graph **Fig. 5** and **Fig. 6** show the loss and accuracy changes during the training. **Table 8** and **Table 9** show the result of chi network.

5.4 Yaku Prediction Network

We trained models for all yaku except yakuman. Some of the yaku like chiniisou is also hard to form and we had little training data. According to the training result, we chose 9 yaku and added them into the input when we trained networks with yaku. The result of these 9 yaku is shown in **Table 10**. We used the same yaku prediction networks when we added yaku to discard network, chi network and pon network. The accuracy increased after we added yaku information. The accuracy might be influenced that we used all same network structures for all yaku prediction networks and it supposed to be improved in the future.

5.5 Battle on Tenhou

The way we save the current situation costs less memory than that in [3] and we can train more data. At the beginning of our research, we used just 800,000 training data and just reached 61.02%. After we added more data into the training set, the accuracy improved a lot. Although we have almost the same prediction accuracy with that in research [3], our agent have not got good result on tenhou. We added yaku and supposed to solve

Table 10 Yaku Prediction Result

Yaku	Accuracy	Precision	Recall	F1-score
Richi	0.7250	0.5812	0.8964	0.7052
Tanyao	0.7921	0.5447	0.9010	0.6790
Prevailing Wind (East)	0.9756	0.7611	0.6023	0.6724
Prevailing Wind (North)	0.9788	0.7524	0.5883	0.6603
Haku	0.9614	0.7039	0.8634	0.7756
Hatsu	0.9606	0.6993	0.8432	0.7645
Chuu	0.9599	0.8100	0.6412	0.7158
Honiisou	0.9589	0.5735	0.6762	0.6206
Dora	0.7798	0.6907	0.8329	0.7552

the problem that the agent would form a hand without yaku but it seemed that our agent also could not work well on avoiding it. The reason why our agent performed bad on Tenhou is now unknown.

6. Discussion and Future Works

In this paper, we propose to use an array of numbers to represent the current situation and use deep neural networks to train our agent. In order to reduce the probability that the agent forms a good hand or calls chi or pon without yaku, we try to add yaku information into the input array. We regard all operations in this game as a binary classification problem or multiclass classification problem. The result shows that the agent can learn knowledge from the way we save information and it performs better in imitation accuracy after adding yaku prediction information.

From the result of battles on Tenhou, we find that our agent still has the problem that it forms a good hand with no yaku. It can sometimes win a subgame but usually discards very dangerous tiles and loses many scores. There are still large improvement space for our agent in the future works.

We do not use the sequence of the discards in this paper and we would like to try the second method showed in Section 4. Besides, we have not considered the information about the current rank and the difference between scores. The current rank and scores are important because if a player wants to get a higher rating on Tenhou, he has to avoid remaining at the last rank as much as possible. We also suppose to use different neural network structures for different yaku and add more yaku information when training in our future works and this might bring large improvement.

References

[1] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al.: Mastering the game of Go with deep neural networks and tree search, *nature*, Vol. 529, No. 7587, p. 484 (2016).

[2] Tenhou: `https://tenhou.net/`. [Online; accessed 08-October-2019].

[3] Gao, S., Okuya, F., Kawahara, Y. and Tsuruoka, Y.: Supervised Learning of Imperfect Information Data in the Game of Mahjong via Deep Convolutional Neural Networks, *Information Processing Society of Japan* (2018).

[4] Wikipedia: Japanese Mahjong — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Japanese\%20Mahjong&oldid=917279150` (2019). [Online; accessed 08-October-2019].

[5] Mizukami, N., Nakahari, R., Ura, A., Miwa, M., Tsuruoka, Y., Chikayama, T. et al.: Adapting One-Player Mahjong Players to Four-Player Mahjong by Recognizing Folding Situations, *The 18th Game Programming Workshop 2013*, pp. 1–7 (2013).

[6] Mizukami, N., Tsuruoka, Y. et al.: Building Computer Mahjong Players Based on Expected Final Ranks, *The 20th Game Programming Workshop 2015*, Vol. 2015, pp. 179–186 (2015).

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al.: Human-level control through deep reinforcement learning, *Nature*, Vol. 518, No. 7540, p. 529 (2015).