Phoenix: An open-sourced, Reproducible and Interpretable Mahjong Agent

Chen Cui / cuichen@usc.edu
Bingling Huang / binglinh@usc.edu
Aiqi Liu / aiqiliu@usc.edu
Na Li / nli10945@usc.edu
Jiabao He / jiabaohe@usc.edu
Jun Lin / junlin@usc.edu
Yu Xing / xingy@usc.edu
Jingwen Sun / sunjingw@usc.edu

2021-03-13

Revision History

Date	Version	Description	Author
02/01/2021	1.0	Initial Version	Yu Xing
02/15/2021	1.1		Aiqi Liu
03/13/2021	2.0		Jun Lin Bingling Huang Na Li Yu Xing Jingwen Sun Aiqi Liu

Table of Contents

1. Introduction

Phoenix is an open-source Mahjong Agent which is used to play Riichi Mahjong (Japanese Mahjong) game. Phoenix would be based on Suphx which was developed by Microsoft Research Asia. “Phoenix " is a Mahjong term in Riichi Mahjong used to name people who get level 10(less than 20 people).

1.1 Goal of Our Project

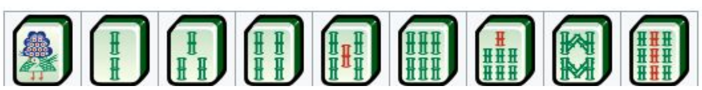
The goal of our project--Phoenix is to produce an open-source and interpretable Mahjong Agent which could be used to populate Riichi Mahjong and improve people’s level in Mahjong. Besides, if possible, we would try to promote the performance and level of Phoenix.

1.2 Overview of Mahjong

Riichi Mahjong is a board game with four players and is one of the most popular mahjong variants worldwide. It is usually played with 136 tiles. There are 34 different kinds of tiles, with four of each kind. there are three suits of
tiles, pin (circles), so (bamboo) and wan (characters), and unranked honor tiles [1] [2]:

- Pin

- So

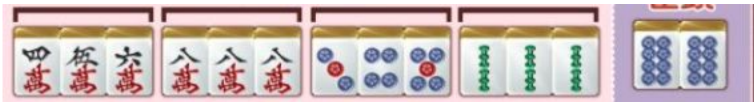


- Wan

- Wind tiles

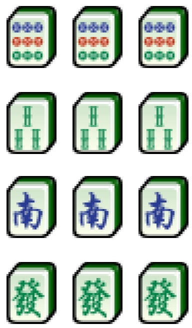
- Dragon tiles

At the beginning, every player has 13 private tiles. The other tiles are shuffled as the wall. In every round, every player would draw a tile from the wall and discard a tile. Players can make a meld (open group) by calling for another player's discard. Generally, a winning hand consists of four melds and a pair like below.



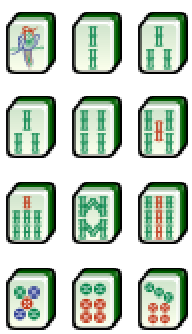
Melds in mahjong are groups of tiles within the player's hand, consisting of either a Pong (three identical tiles), a Kong (four identical tiles), a Chow (three Simple tiles all of the same suit, in numerical sequence), or Eyes (two identical tiles needed in a winning hand).

- **Pongs** are a set of three identical tiles.

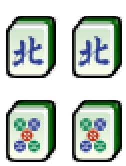


- **Kong** is a complete set of four identical tiles.

- **Chow** is a meld of three suited tiles in sequence.



- **Eyes** (also known as a pair) are two identical tiles which are an essential part of a legal winning hand.



Anyone who declares a winning hand wins the round. Different winning hands get different scores. The player who gets the highest accumulated scores in the end wins the game.

1.3 Related Research

1.3.1 Mahjong AI

Microsoft Research Asia has developed a Mahjong AI named Suphx [3]. Suphx is based on deep reinforcement learning, global reward prediction, oracle guiding, and run-time policy adaptation.

"Global reward prediction trains a predictor to predict the final reward of a game based on the information of the current and previous rounds.
Oracle guiding introduces an oracle agent that can see the perfect infor- mation including the private tiles of other players and the wall tiles.
As the complex playing rules of Mahjong lead to an irregular game tree and prevent the application of Monte-Carlo tree search techniques, we introduce parametric Monte-Carlo policy adaptation (pMCPA) to improve the run-time performance of our agent." [3]

Microsoft Research Asia evaluates Suphx on Tenhou, which is a web based mahjong platform in Japan with a complete ranking system and over 350,000 users. It shows that Suphx beats 99.9 percent of human players and reaches 10 dan.

1.3.2 Reinforcement Learning

In nature, the idea that learning from interacting with the environment is common like infant plays [1]. Such interactions may become a source of knowledge that can be used and applied to conduct competitive or exploratory tasks. And learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence. In the science field, some methods take advantage of this idea to solve some complex problems and reinforcement learning is a popular one among them. It focuses on goal-oriented learning from interaction than other approaches [4].

Reinforcement learning is different from supervised learning and unsupervised learning although they all belong to the machine learning field. Supervised learning is learning from a training set of examples with labels provided by a knowledgeable external supervisor [4] and unsupervised learning is learning a latent pattern from a bunch of unlabeled data.

Some challenges arose in reinforcement learning that are not in other types of learning methods. The first one is the trade-off between exploration and exploitation [4]. The agent has to exploit and explore in the learning process but they may lead to different influences on rewards. Exploitation means taking action as what has already been experienced. In contrast, exploration means taking actions that were not selected before. The challenge is that neither exploration or exploitation can be pursued exclusively without failing at the task [4]. Even though this problem has been intensively studied by mathematicians for a long time, it still remains unresolved yet [4]. Another challenge of reinforcement learning is that it explicitly considers the whole picture of the task for a goal-oriented agent interacting with an uncertain environment [4]. The dilemma is that it contrasts many approaches that consider dividing a big problem into several subproblems.

Overall speaking, reinforcement learning starts with interactive and goal-seeking learning agents that have explicit goals and have the ability to sense the environment. As agents choose actions, the environment is influenced and agents may sense the environment changes one step later.

Mahjong is a popular game especially among Asian areas. Reinforcement learning is an effective method to train players to master strategies. And how our team implements it to assist playing is introduced and further discussed in the part 7 Reinforcement Learning.

1.4 Overview

The EDD is divided into 6 sections with various subsections. The sections are:
1 Introduction
2 Glossary
3 Use cases
4 Design Overview
5 Data Design
6 Model Design
7 Reinforcement Learning
8 Results
9 Future Work

2. Glossary

Riichi: A player may declare ready if a player's hand needs only one tile to complete a legal hand (tenpai)
Chow: three consecutive simple tiles of the same suit
Pong: three identical tiles
Kong: four identical tiles
Yaku: Yaku are specific combinations of tiles or conditions that yield the value of hands. A winning hand consists of four melds requires at least one yaku.

3. Use Cases

3.1 Mahjong Beginners

A mahjong beginner would like to use Phoenix as a teacher. The Phoenix could either participate in a real game or demonstrate a classical game. The Phoenix would interpret why the move it chooses is the best move.

3.2 Mahjong challengers

A Mahjong challenger may like to test its mahjong level by challenging Phoenix with different levels written in different algorithms.

3.3 Mahjong players substitute

In a friend game, maybe there are less than 4 friends. In this situation, players could consider adding an agent as participants.

4. Design Overview

4.1 Introduction

In this part, we will introduce Suphx's architecture and in our Alpha stage the main goal is to reimplement Suphx. In our Beta stage we will introduce our advice to Suphx and additional features.

4.2 Suphx System Architecture

The Suphx system is consist of the following components:

- supervised models
- winning model
- global reward predictor
- oracle agent
- policy decision flow

4.2.1 Supervised Models

Supervised models build a foundation for reinforcement learning, so that reinforcement learning can be trained much faster. These supervised models are decomposed into 5 models.

- Discard model
  - Decide which tile to discard
- Chow model
  - Decide whether to take chow action
- Kong model
  - Decide whether to take kong action
- Pong model
  - Decide whether to take pong action
- Riichi model
  - Decide whether to take riichi action

4.2.2 Winning Model (Rule based)

The winning model of Suphx is based on the rule of mahjong. The system will check whether the current round is the last round of the game or not.

- If this is not the last round of the game, Suffix will declare and win the round;
- If this is the last round of the game,
  - If after declaring a winning hand the accumulated round score of the whole game is the lowest among the four players, do not declare;
  - Otherwise, declare and win the round

4.2.3 Global Reward Predictor (RNN based)

The reason why we need a Global Reward Predictor is that Mahjong is a multi-round game. And the scores gained in each round are mostly determined by the initial hand. For a professional player, he may choose to win more scores when he gets a good initial hand while choosing to lose small scores to keep his current rank. In this case, we can't judge if a player is professional just by the score in one round.
The Global Reward Predictor would be implemented in a RNN-like structure such as GRU, given scores of history rounds. The loss would be

$$\min \frac{1}{N} \sum_{i=1}^{N} \frac{1}{K_i} \sum_{j=1}^{K_i} (\Phi(x_i^1, \cdots, x_i^j) - R_i)^2,$$

We will choose Round score, dealer position, counters of repeat dealers, Riichi bets as input features.

4.2.4 Oracle Agent

The method we use to train an agent for imperfect-information is to train the agent with perfect information and gradually decay the "cheating information". Finally the agent would be trained only with imperfect information.

4.2.5 Policy Decision Models

We will train five separate decision models to aggregate our final policy decision. The five models are Discard Model, Riichi Model, Chow Model, Pong Model, Kong Model. Supervised learning would be applied with state as input and action as output. For example, in the training of discard model, the input is all the observable information and the output is the tile discarded in that state.

The training set would be expert room logs from Tenhou platform.

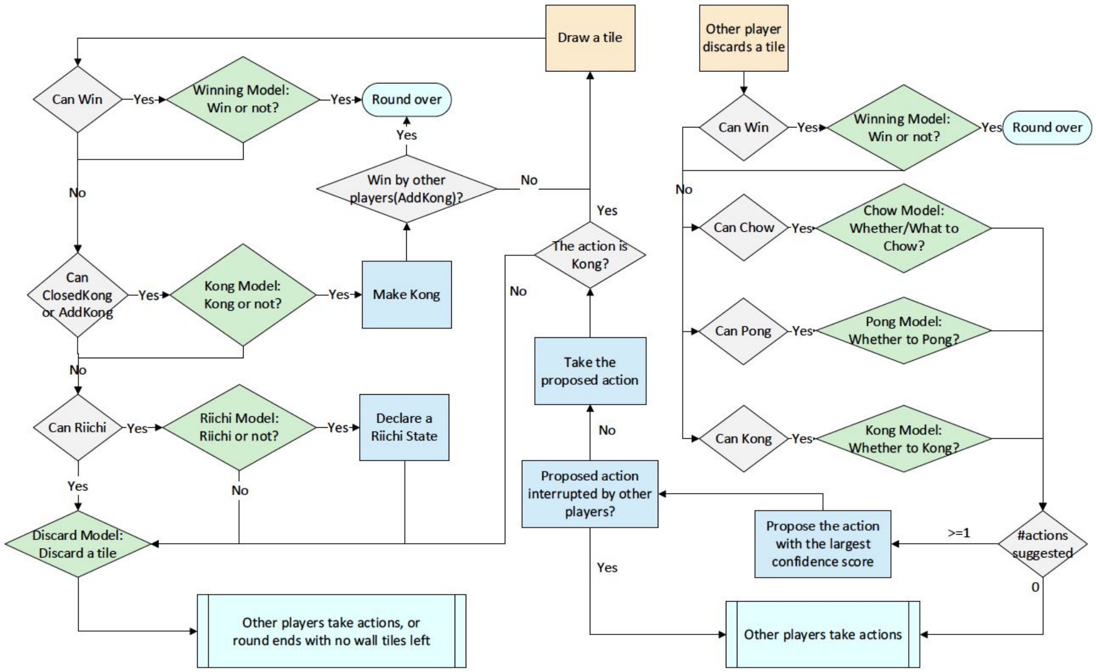


Fig. 1: Decision Flow

5 Data Design

5.1 Data Sources

Tenhou platform provided a well designed logging system, that will automatically log each game in the XML format (Extensible markup language). Each player's actions and table information are recorded as XML tags. In order to bootstrap the process of supervised learning training, we scrape the logs bundle from the tenhou platform server. Each file is bundled together by year from 2009 to current time. There are some earlier logs as well from 2005, but the highest ranking table became available only after 2009. Since we are looking for high quality games, the low ranking games are filtered and three person mahjong is also removed for the simplicities. Here is the simple statistic of the dataset. For some unknown reason the 2020 logs are missing in the tenhou platform. We will skip them for now.

Table Ⅰ: Size of Raw Data

Raw XML log from Tenhou											
2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2021
51k	40k	464k	85k	71k	48k	46k	44k	30k	36k	46k	1332

Now, we have all these raw logs and ready to preprocess and extract features from them.

5.2 Data Preprocessing

5.2.1 Data for Supervised Learning

In order to generate training data for supervised learning, we need to provide data from the perspective of each player that leads into their decisions. Since raw data are bundles of log files for each game with table information and player operations, we need to initialize a state at the beginning of the game and append changes to it.

Firstly, we will use the log parser to resolve data from logs, and extract features from it. We extracted features for the Discard Model, Riichi Model, Chow Model, Pong Model, and Kong Model. In this step, we generate a 2d vector for each state as the input of a supervised learning neural network. The detailed format will be shown as below.

The most challenging part is how to find a suitable format to save our file. At the beginning, we have tried to store all the data in Python dataframe. Although it is workable on small size csv files (such as 2021.csv), it becomes time consuming and huge memory consumption. we quickly turn to storing them in hdf5 format. In this way, we presistiting our data into the disk to relieve the memory burden. However, there are some drawbacks to storing data into the hdf5 dataset. We need to keep the data shape in the same shape in which it will consume more disk space. Same apply to npy files. After trying multiple types, finally we decided to use json as our output file, which also supports persisting data to disk, and could speed up the processing time by appending processed data at the end of file.

In this part, we also made an effort to remove three-player log, and fixed logs with duplicate nodes.

5.2.2 Data for Real-time Gaming

Features are needed as an input for trained models in real-time gaming. So generating exactly the same features as the training data from tenhou bot interfaces, which is an access to play riichi mahjong on tenhou.net server, is necessary.

5.3 Data Format

5.3.1 States Format

We have different data formats in different models. The following are the data formats for the Discard Model and Riichi_Chow_Pong _Kong Model.

Discard Model:
Each json represents a state, which includes the tile the player drawed, player's current hands, tiles discarded by four players, four players' open hands, as well as the tile the player decided to discard based on the previous features.

The example is shown here:

```
{ 'draw_tile': 8,
  'hands': [95, 110, 50, 117, 1, 108, 71, 127, 68, 51, 15, 122, 44, 8],
  'discarded_tiles_pool': [],
  'four_players_open_hands': [[], [], [], []],
  'discarded_tile': 8}
```

Riichi_Chow_Pong _Kong Model:
Each json represents a state, which includes all the information shown currently, and the decision that the player finally made based on the previous information.

The example is shown here:

```
{'player_id': 2,
 'dealer': 3,
 'repeat_dealer': 0,
 'riichi_bets': 0,
 'player_wind': 'S',
 'prevailing_wind': 'E',
 'player_tiles':
          {'closed_hand:': [18, 86, 50, 55, 84, 22, 57],
```

'open_hand': [125, 127, 124, 28, 21, 24],
'discarded_tiles': [123, 117, 74, 131, 133, 64, 7, 89, 33, 45, 128, 65, 119]},
'open_hands_detail':
[{'tiles': [125, 127, 124], 'meld_type': 'Pon', 'reacted_tile': 124},
{'tiles': [28, 21, 24], 'meld_type': 'Chi', 'reacted_tile': 28}]
{'tiles': [4, 5, 6, 7], 'meld_type': 'AnKan', 'reacted_tile': None}
{'tiles': [114, 112, 113, 115], 'meld_type': 'MinKan', 'reacted_tile': 114}
{'tiles': [67, 64, 66, 65], 'meld_type': 'KaKan', 'reacted_tile': 65}
'enemies_tiles':
[{'enemy_seat': 0,
'closed_hand:': [62, 83, 98, 63, 12, 79, 92, 54, 27, 97, 72, 61, 105],
'open_hand': [], 'discarded_tiles': [1, 120, 130, 129, 37, 118, 30, 29, 71, 46, 108, 32, 36]},
{'enemy_seat': 1,
'closed_hand:': [99, 78, 85, 103, 67, 58, 77, 90, 104, 60, 16, 13, 9],
'open_hand': [], 'discarded_tiles': [114, 73, 100, 38, 6, 28, 34, 116, 48, 26, 44, 113, 75]},
{'enemy_seat': 3, 'closed_hand:': [43, 87, 40, 88, 59, 19, 42, 17, 80, 94, 53, 109, 81],
'open_hand': [], 'discarded_tiles': [69, 121, 124, 31, 102, 70, 10, 122, 68, 106, 126, 111, 41, 110]}],
'dora': [47],
'scores': [24800, 27800, 22400, 25000],
'last_player_discarded_tile': 8,
'could_chi': 0,
'could_pon': 0,
'could_minkan': 0,
'is_FCH': 0,
'action': ['DRAW', 35]
}

5.3.2 Extracted Feature Format

General feature is a (x, 34) vector include information such as player tiles(12,34), enemies tiles(36,34), dora(5,34), current scores(4,34), and other basic board information(5,34) like dealer and wind. Manual crafted features will also be concatenated.

Each model has its own individual feature, for example, tile to Chi is an individual feature important to Chi feature generator.Thus the final feature will be individual feature concatenate to general feature.

6. Model Design

In mahjong there are few actions needed when playing. In the online setting, the actions are simplified. There are only five main actions players need to take, thus we designed these models below using supervised learning to gain the knowledge from high ranking human players.

6.1 Discard Model

Discard is one of the most common actions in the mahjong world and yet the most complicated model among others. There are many factors that will change the probability to win this game. Other players are able to make an action based on your discard tile, so it's important to not discard the tile that's needed for other players.

Adapted from the classic computer vision image CNN model, one of the most fundamental models of the deep learning field. We already know that the whole mahjong set is 136 and 4 same tiles for each kind of tile, so we have 0 to 34 columns as the unique type of tiles and 0 to 4 rows for each of the same kind of tiles. In the typical image processing, image has length, width and color channels and feeds into the CNN model. In our case, we have columns representing width and rows representing length. For channels, we set it to one in order to use the convolutional layer in the CNN. Our discard model takes the ideas from the well known ResNet architecture and in fact the rest of models are using the similar architecture as well. We first have our input features stack together to form a 16 x 34 x 1 matrix. In the current states, we only use 4 types of features and in the future, we will use some look ahead features to push the performance a little more. Three layers of convolutional layers with filter size 256 and kernel size 3x1 are applied to the input features. In the next step, we will introduce the residual block In ResNet, there are many deep convolutional layers, a classic example is ResNet 50, which used 50 blocks of residual blocks. The residual block is used to make a shortcut to each layer, so that the information can be passed from the first layer to the last layer with less effort and perversely. In deep neural networks, information propagated through each layer is difficult and slow. With the residual block, this is no longer a problem. In our setting, we only repeat the block 5 times for the sake of speed. Later, we will do hyperparameter tuning to find out the optimal repetition needed for this model. After the residual blocks, we apply one additional convolutional layer and flatten the output to 1 dimensional array. At this time, all we need to do is to apply a dense layer with softmax activation function to distribute the probabilities to 34 classes.

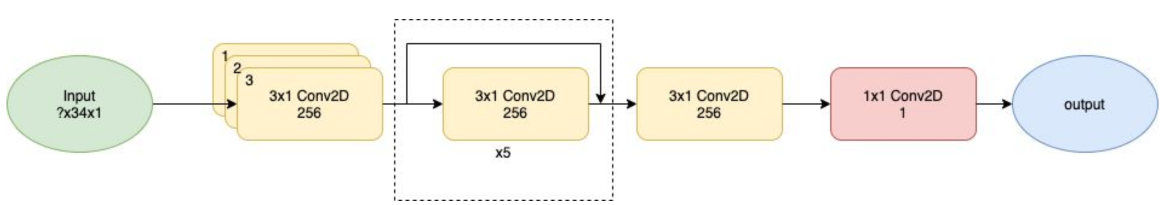


Fig. 2: Workflow of Discard Model

6.2 Riichi, Chow, Pong, and Kong Model (RCPK)

Riichi, Chow, Pong, and Kong model, RCPK model for short share the same network architectures due to the nature of binary actions. Users only need to decide whether or not to perform this action. Of course, the rule is more complex than just making a decision, but for now, we will keep it simple here. Each model has different kinds of features, but no matter what kind of model is, they are all in the D x34x1 form, where the D is the feature stack dimension depending on the model. Similar to the discard model, we used the 3 layer of convolutional layers and 5 repeated residual blocks. At this point, everything is identical to the discard model, but next instead of one layer convolution layer, three layers are used. Flatten apply to the output and go though the 1024 fully connected layer and another 256 fully connected layer. Finally, distribute the probabilities to binary classes.

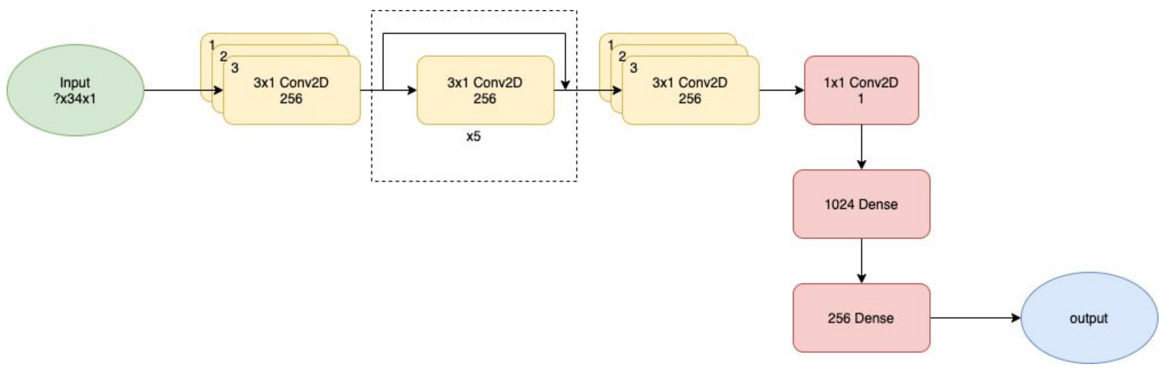


Fig. 3: Workflow of Riichi Model, Chow Model, Pong Model, and Kong Model

6.3 Rewards Predictor Model

Model structure is shown in the following figure.

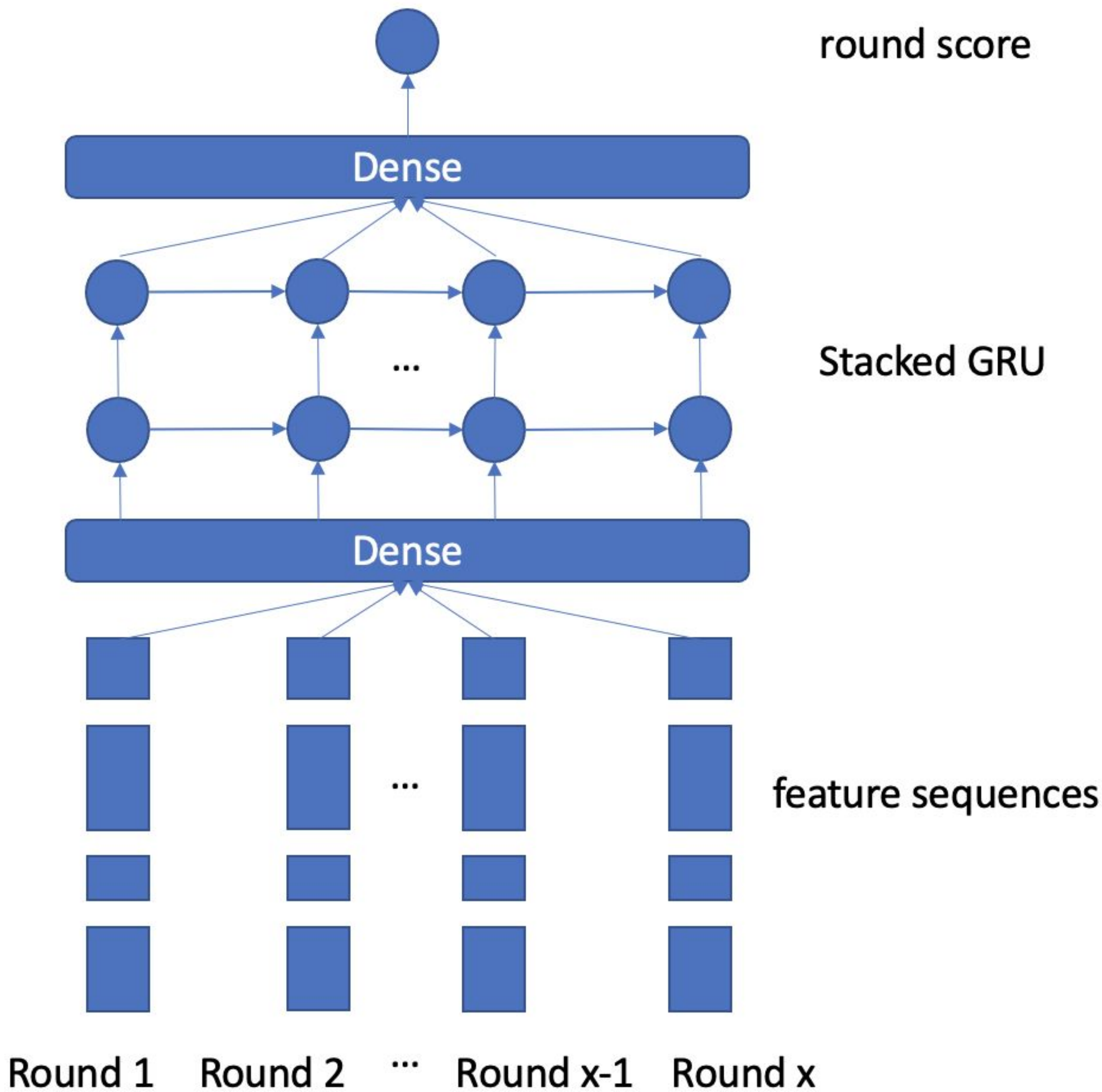


Fig. 4: Workflow of Rewards Predictor Model

The following features are used

- Dan (ranks of players)
- Dealer
- Repeat dealer number
- Riichi bets number
- Round initialization score and gains

The reason why we add dan is that the points gained for different rankings are influenced by players' dans.

<table>
<tr><th rowspan="2">位</th><th rowspan="2">初期pt</th><th colspan="4">東風戦</th><th colspan="4">東南戦</th><th rowspan="2">昇段pt</th></tr>
<tr><th>1位</th><th>2位</th><th>3位</th><th>4位</th><th>1位</th><th>2位</th><th>3位</th><th>4位</th></tr>
<tr><td>新人</td><td>0</td><td rowspan="9">一般
+20</td><td rowspan="9">一般
+10</td><td rowspan="18">+0</td><td>0</td><td rowspan="9">一般
+30</td><td rowspan="9">一般
+15</td><td rowspan="18">+0</td><td>0</td><td>20</td></tr>
<tr><td>9 級</td><td>0</td><td>0</td><td>0</td><td>20</td></tr>
<tr><td>8 級</td><td>0</td><td>0</td><td>0</td><td>20</td></tr>
<tr><td>7 級</td><td>0</td><td>0</td><td>0</td><td>20</td></tr>
<tr><td>6 級</td><td>0</td><td>0</td><td>0</td><td>40</td></tr>
<tr><td>5 級</td><td>0</td><td>0</td><td>0</td><td>60</td></tr>
<tr><td>4 級</td><td>0</td><td>0</td><td>0</td><td>80</td></tr>
<tr><td>3 級</td><td>0</td><td>0</td><td>0</td><td>100</td></tr>
<tr><td>2 級</td><td>0</td><td>-10</td><td>-15</td><td>100</td></tr>
<tr><td>1 級</td><td>0</td><td rowspan="4">上級
+40</td><td rowspan="4">上級
+10</td><td>-20</td><td rowspan="4">上級
+60</td><td rowspan="4">上級
+15</td><td>-30</td><td>100</td></tr>
<tr><td>初段</td><td>200</td><td>-30</td><td>-45</td><td>400▼</td></tr>
<tr><td>二段</td><td>400</td><td>-40</td><td>-60</td><td>800▼</td></tr>
<tr><td>三段</td><td>600</td><td>-50</td><td>-75</td><td>1200▼</td></tr>
<tr><td>四段</td><td>800</td><td rowspan="3">特上
+50</td><td rowspan="3">特上
+20</td><td>-60</td><td rowspan="3">特上
+75</td><td rowspan="3">特上
+30</td><td>-90</td><td>1600▼</td></tr>
<tr><td>五段</td><td>1000</td><td>-70</td><td>-105</td><td>2000▼</td></tr>
<tr><td>六段</td><td>1200</td><td>-80</td><td>-120</td><td>2400▼</td></tr>
<tr><td>七段</td><td>1400</td><td rowspan="3">鳳凰
+60</td><td rowspan="3">鳳凰
+30</td><td>-90</td><td rowspan="3">鳳凰
+90</td><td rowspan="3">鳳凰
+45</td><td>-135</td><td>2800▼</td></tr>
<tr><td>八段</td><td>1600</td><td>-100</td><td>-150</td><td>3200▼</td></tr>
<tr><td>九段</td><td>1800</td><td>-110</td><td>-165</td><td>3600▼</td></tr>
</table>

Fig. 5: Points Gained by Different Rankings and Dans

The Global Reward Predictor is trained by minimize the following mean square error:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\Phi\left(x_i^1,\ldots,x_i^j\right)-R_i\right)^2$$

Where N denotes the number of games in the training data, R_i denotes final points gained by four players. Φ is our global reward predictor. The purpose of it is to distribute final rewards to each round. As we mentioned, a small loss of score doesn't mean worthless in final ranking. In that case, we will use $\Phi\left(x_i^1,\ldots,x_i^j\right)-\Phi\left(x_i^1,\ldots,x_i^{j-1}\right)$ as the reward of round j.

7 Reinforcement Learning

7.1 RL Methods Introduction

Reinforcement learning is trying to learn mapping situations to actions while maximizing a reward function [4]. The learner is usually called agents. During the learning process, agents target at discovering a series of actions that yield the most reward by thousands of tryings.

There are many solution methods according to different situations and can be classified in different perspectives. First of all, the algorithms can be classified as model-free or model-based methods. The model stands for the simulation of the dynamics of the environment and the transition probability matrix is known. The agent knows how likely to transfer from current state to next state based on the current action. Some popular algorithms are like dynamic programming including policy iteration and value iteration. They use the model's transition probability matrix or distribution of the next state in order to select optimal action. And the model needs to be provided by people but rarely a learned outcome. That may lead to the model-based algorithm becoming impractical as the state and action space largely. On the other hand, model-free algorithms can be more practical and it does not require space to store all the combinations of states and actions. They rely on real samples getting from the environment and may sample from experience memory. Monte Carlo control, Q-learning, SARSA and gradient policy are model-free algorithms.

Another perspective to look at reinforcement learning algorithms is on-policy or off-policy. An on-policy agent learns the value based on its current action derived from the current policy, whereas its off-policy counterpart learns it based on the action obtained from another policy. For example, Q-learning is an off-policy algorithm and SARSA is an on-policy algorithm.

Considering the mahjong game, the policy gradient algorithm was selected for optimization. Action-value methods select optimal actions based on their estimated action values [4]. However, in our case, the game environment is very complex and action and state space are very memory consuming. It is not wise to use action-value methods to deal with this problem. On the other hand, learning a parameterized policy can select actions without consulting a value function. At the same time, a value function may still be used to learn the policy parameter, but is not necessary for action selection [4].

In policy gradient methods, the policy as $\pi(a|s,\ \theta)$ can be parameterized in different ways. If the action space is discrete and not too large, a common kind of parameterization is to form parameterized numerical preferences for each state-action pair and the actions with the highest preferences in each state are given the highest probabilities of being selected [4]. The advantage is that action preferences themselves can be parameterized by using some advantages tools like neural networks.

7.2 RL System in Mahjong

Mahjong game is a very popular game all around the world and it is attracting so many people with a very range of ages for many reasons. This game needs to plan strategies and predict what will happen in the next few steps. Challenges for playing this game are imperfect information and randomness of future situations. Each player only knows thirteen coaches in his hand and abandoned coaches by all players. Others information is unknown and that influences decision making.

In recent years, mahjong has been successfully challenged by artificial intelligence, especially deep reinforcement learning has made breakthroughs on comprehending gameplay. The game has some characteristics which means reinforcement learning is a good option to solve it. The first one is the huge state space. In response to it, deep reinforcement learning can take advantage of neural network to predict and output actions. And the exploration process can be regulated dynamically and intelligently. Another characteristic is imperfect information. Many hidden information in this game can increase the difficulty of training. However, deep reinforcement learning can use those hidden information very well as guidance, because it can control how much information is input. The third challenge is Mahjong's complex reward mechanism. There is a gap between each game and the final results after a set of games. Thus, in the system design, the challenges above need to be considered and the work is shown in the below figure.

We can see in the figure below that a distributed reinforcement learning is applied. There are two main parts which are 'self-play workers' and 'parameter server'. Self-play workers take charge of simulating the game environment and conducting games and the parameter server takes care of parameters training. In 'self-play workers', for each round, the simulators will infer the inference engines by telling its current state. And in return, the inference engines give specific instruction on taking which action to simulators. All the state and action information will be saved in the experience replay buffer. In our system, the experiences will be saved separately according to the specific model involved. At the same time, the parameter server is working on training parameters and updating policy. Policy gradient algorithms are applied for training the policy and gradually update the policy with more rewards gotten. More details about implementation will be introduced in the next section.

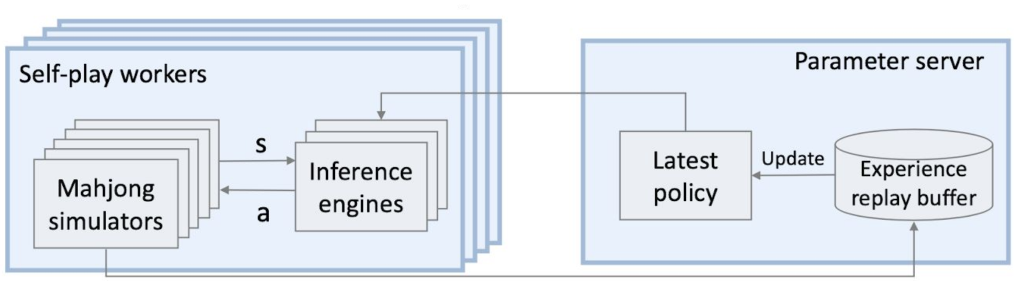


Fig. 6: Reinforcement Learning Structure in Mahjong Training

In order to solve those challenges mentioned above, the system needs several techniques. The first one is choosing policy-based algorithms rather than value-based algorithms to train. The Mahjong game usually has very huge state space and it may be hard for value-based algorithms to save all the state and action pairs. Thus, by taking advantage of deep learning and policy-based algorithms, this problem can be solved. Neural network can help to save a lot of memory space by only saving a series of parameters. Also, policy-based algorithms only need to save a probability matrix in memory which depends on the action space. And in the Mahjong game, action space is discrete and not that big.

Secondly, the imperfection information is solved by 'prior coach' technique which means that it uses hidden information in policy gradient optimization to guide the training direction of the model [5]. Thus, in the self-play phase, 'perfect' information can lower the difficulty of learning the path to win the game. And gradually the input information gets back to normal and it forces the model to learn and understand the visible information more deeply to make decisions. [5]

Lastly, a global reward prediction system is designed to respond to Mahjong's complex reward mechanism. In one game, there are usually more than 8 rounds and each player gets a round score. However, a player's game reward is not the sum of round scores which means the round score cannot reflect if this round's action is good or not. This has two main reasons. On the one hand, some rounds in one game share the same rewards. On other hand, the playing strategy may vary for different situations in one game. For example, a player may become more conservative in the last one or two rounds when he has a big lead to other players. And in contrast, he may become more aggressive when he needs to catch opportunities to earn more points. That means a negative round score may not necessarily mean a poor policy [3].

7.3 RL Implementation

In this section, details of deep reinforcement learning implementation will be discussed. As we have introduced before, the system structure has two parts: self-play works and parameter buffer. However, it cannot show how reinforcement learning works and implements. Thus, it is introduced in the aspects of algorithm and implementation.

Seen from the figure below, it shows how the reinforcement learning works in the whole system and how it connects with other parts of the system. The core applied algorithm is a reinforced policy based algorithm. The environment is simulated in an online game platform. For each game, our agents are arranged in private bots and matched with other players ramply, shown in the figure below. In the game, the tiles and how to deal with them can be converted to state vectors and actions. At the same time, the rewards can be calculated by a global reward predictor [3]. In order to train the policy, all information including states, action, parameters and rewards is saved in replay buffers separately based on what model it has used.

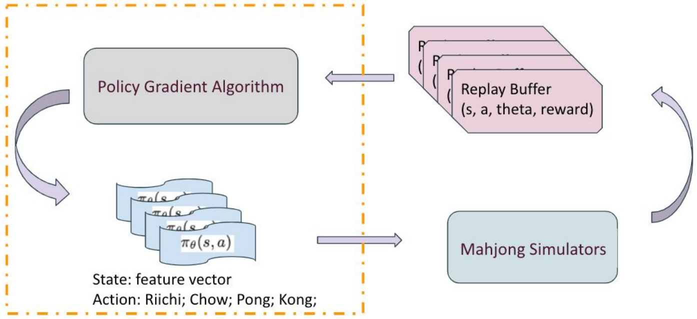


Fig. 7: Reinforcement Learning Work Flowchart

After the game information is saved in the replay buffer, it can be referred to by other parts when needed. As for the reinforcement learning algorithm, it needs to know current state, action, reward and current parameters as input and updates policy by training the neural network's parameters separately for five models which are riichi, chow, peng, kong and discard. By updating the new neural network, the system can select action with the biggest probability at current state. And the updated policy and action will be returned to the Mahjong environment. That is how it works in the system.

Fig. 8: Online Training Environment

The reinforcement learning algorithm applied is one of policy gradient methods which parameterized policy that selects actions without consulting a value function [4]. The policy parameters are the targets for learning by optimizing cost function $J(\theta)$. The notation θ is policy's parameter vector. Therefore the policy can be written as $\pi(\alpha|s,\ \theta) = Pr\{At = \alpha \mid St = s,\ \theta t = \theta\}$ for the probability that action α is taken at time t given that the environment is in state s at time t with parameter θ. To seek maximize performance, gradient ascent in $J(\theta)$ to update policy's parameter:
$\theta_{t+1} = \theta_t + \alpha\, gradJ(\theta_t)$ (1). And the objective function is shown as below [3].

$$\nabla_\theta J(\pi_\theta) = \underset{s,a\sim\pi_{\theta'}}{\mathrm{E}}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}\nabla_\theta \log \pi_\theta(a|s)\, A^{\pi_\theta}(s,a)\right] + \alpha\nabla_\theta H(\pi_\theta)$$

And the pseudo code is shown in the figure below.

```
function REINFORCE
    Initialise θ arbitrarily
    for each episode {s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T} ~ π_θ do
        for t = 1 to T − 1 do
            θ ← θ + α∇_θ log π_θ(s_t, a_t)v_t
        end for
    end for
    return θ
end function
```

Fig. 9: Pseudo Code for Policy Parameter Updating

This method is classical REINFORCE algorithm [4] whose update at time t only involved current action A_t. By summing over a certain amount of samples, policy gradient theorem can guarantee the objective function $J(\theta)$ are still proportional to the expectation of the sample gradient.

Also the algorithm can involve a baseline and include a comparison of the action value to an arbitrary baseline $b(s)$. The baseline can be any function as long as it does not vary with action [4]. The optimization equation is valid since the subtracted quantity is zero [4]. The policy gradient theorem with baseline can be used to update policy to generate a new version of REINFORCE that has a general baseline.

Another problem is reward approximation. In the common cases, the current reward is calculated by the bellman equation. However in the Mahjong game, players receive round scores at the end of each round and receive game rewards after eight to twelve rounds. The problem is that neither round scores nor game rewards are an effective sign of good performance [3]. Thus, a global reward to each round of the game is predicted by a global reward predictor Φ. In our system, the global reward predictor Φ is a recurrent neural network and is trained by minimizing the following mean square error [3]:

$$\min \frac{1}{N} \sum_{i=1}^{N} \frac{1}{K_i} \sum_{j=1}^{K_i} (\Phi(x_i^1, \cdots, x_i^j) - R_i)^2,$$

Where N is the number of games in the training data, R_i is the final game reward of the i-th game, K_i is the number of rounds in the i-th game. x_i^k is the feature vector of the k-th round in the i-th game, includes the score of the round and the current overall round score. When Φ is fully trained, not only the game's total score can be known, but also each round's score can be predicted. The algorithm flow is shown as the below figure.

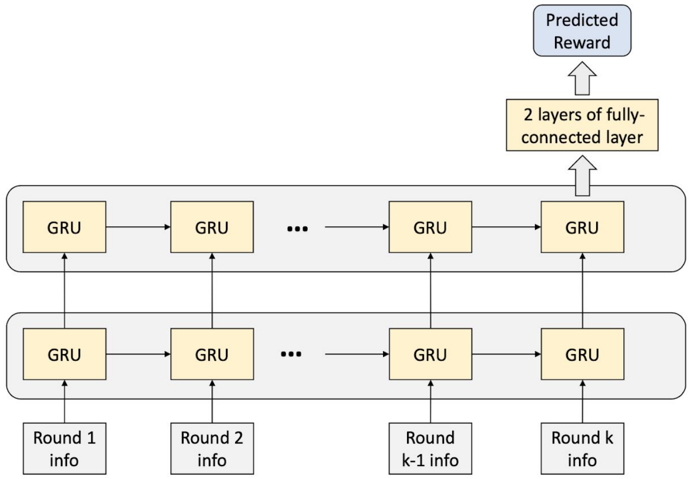


Fig. 10: Global Reward Predictor Work Flow

8 Results

8.1 Connection with Platforms

8.1.1 Google Cloud Platform

8.1.1.1 Preprocessing using Dataflow

Processing such a huge dataset locally is infeasible. Since Google Cloud Platform(GCP) offers $300 free credits for all new users, we have decided to use GCP to do preprocessing and training. In order to work on these dataset more efficiently, we move all our dataset from local storage to the google cloud storage bucket, this will significantly speed up the disk I/O and lower the network latency for other GCP services.

Utilizing the preprocessing logic from the section 5.2, we employ an apache beam pipeline to speedup and scale for processing our dataset. Apache beam is a unified programming model to define and execute data processing pipelines, including extract, transform, and load(ETL), batch and stream (continuous) processing. Beam can easily do distributed processing on the fly and the GCP dataflow services use apache beam as the main backbone.

In the preprocessing stage, the input of the pipeline is directly sourced from the google storage bucket. Logic in the pipeline is defined by one unit, meaning that every operation logic is done by one unit. Only one row of data is processed at a time. This is for the purpose of multithreading and multi workers distribute processing. Since each row in a csv file is a game log, we read the csv files line by line. In the next step of the pipeline, we convert the raw text to csv row data and only the log column is extracted. The extracted column consists of only one log in the raw xml format. We need to parse the xml format to an easy to manage format. This is done in the section 5.2. After getting the result from 5.2 in dictionary form, we transform it to the shape of (D,34,1), the first dimension is determined by different features in different models. Filtering out some of the invalid data points, we split it into train and validation datasets for later training. These datasets are saved into the TFRecord files in the bucket, we will use them to load our dataset into tensorflow training. At the point, we have concluded the process of preprocessing, all the relevant files are generated and ready to begin the training process.

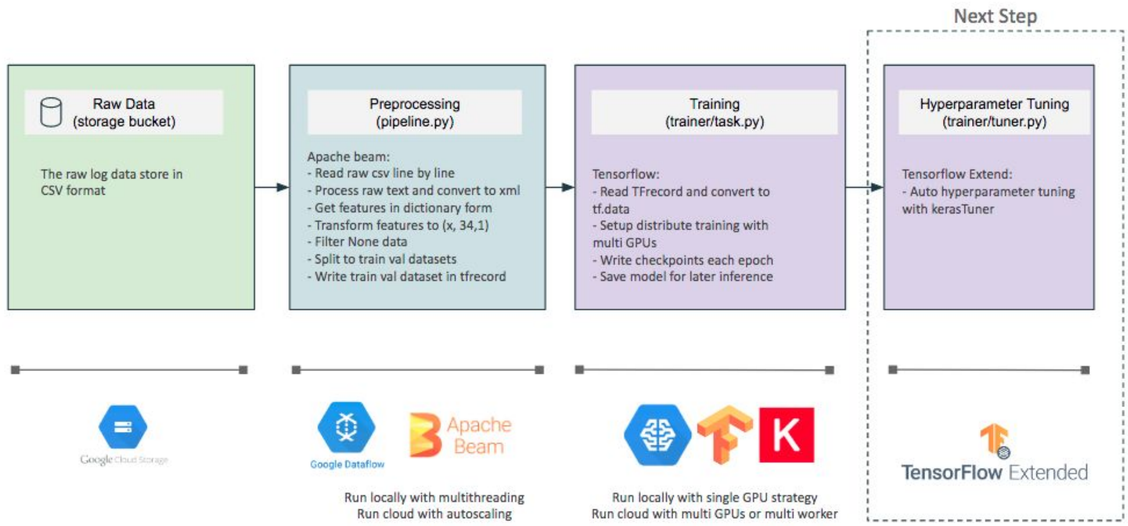


Fig. 11: Workflow of Preprocessing and Training on GCP

8.1.1.2 Distribute training using mirrored strategy

Training on a single GPU is slow and impractical. In order to expedite the process, we utilized the MirroredStrategy provided by the tensorflow api. MirroredStrategy is a synchronous training strategy that works in a single machine with multiple GPUs. In GCP, we are allowed to attach some amount of GPUs to our master server, so we can distribute training with multiple GPUs. The idea of MirroredStrategy is to make a copy of the model variable to all the processors, then it will combine the gradients all together and apply to all copies of processors. Since keras is the higher level API for tensorflow, we can easily use it for our code. All we need to do is to add the appropriate scope to our model and it will do the rest for us. In our experiment, we used a master server with n1-highmem-8 which has 8 vCPU and about 7 GB memory per vCPU. We attached 4 Nvidia T4 GPUs to our master server for our training task. With the help of multi GPUs, we are able to quickly train our RCPK model in a few hours. These models have smaller datasets compared to discard models due to the constraints to call these actions and discard is performed more often. On the other side, the Discard model is much slower compared to RCPK models and more difficult to get great performance out of the box. In the later phase, we will possibly use Multiworker mirrored strategy to do the hyperparameter tuning which allows us to scale up horizontally by adding more machines with multiple GPUs.

8.1.2 Tenhou Platform

Few days before the mid term, we are able to connect everything in tenhou bot with our supervised models. Now, our agent will perform chow, pong, kang, riichi, and discard action on the tenhou platform with human players. Our agent collects features from known information in the table as the game progresses and transforms them into the single feature row similar to our training dataset. This feature is used by models to make predictions. In discard, the prediction will give us the distributions of probabilities of each class. Argmax applied to find out which one has the highest probability and this number will be mapped to the unique tiles. We tell the tenhou client the tile we want to discard and a websocket message sends to the server and updates accordingly. The other models have the similar mechanism , the difference is that we need to first manual check the condition for valid action. In other words, some rules need to be checked before we ask the model to make an action. Models only make predictions when they meet all the requirements. Below is an example run against 3 human players. phx527 is our agent bot. Our current supervised models can achieve 2nd place. In this game, our agent plays defensively. Agent don't throw out any existing meld in hands and cautiously discard the tile that can't be used against self.

```
2021-03-16 14:20:26 DEBUG: id=draw
2021-03-16 14:20:26 DEBUG: Step: 13
2021-03-16 14:20:26 DEBUG: Remaining tiles: 12
2021-03-16 14:20:26 DEBUG: Hand: 3344067m123789p + 4p
2021-03-16 14:20:26 DEBUG: Send: <D p="26"/>
2021-03-16 14:20:26 INFO: Discard: 7m
2021-03-16 14:20:28 DEBUG: Get: <FURITEN show="0" /> <D26/> <U/>
2021-03-16 14:20:28 DEBUG: Send: <Z />
2021-03-16 14:20:29 DEBUG: Get: <e121/> <V/>
2021-03-16 14:20:31 DEBUG: Get: <f82/>
2021-03-16 14:20:35 DEBUG: Get: <N who="3" m="45351" />
2021-03-16 14:20:35 INFO: Meld: Type: chi, Tiles: 123s [72, 77, 82] by 3
2021-03-16 14:20:36 DEBUG: Get: <G3/> <T43 t="32"/>
2021-03-16 14:20:37 INFO: Drawn tile: 2p
2021-03-16 14:20:37 DEBUG: id=draw
2021-03-16 14:20:37 DEBUG: Step: 14
2021-03-16 14:20:37 DEBUG: Remaining tiles: 9
2021-03-16 14:20:37 DEBUG: Hand: 334406m1234789p + 2p
2021-03-16 14:20:37 DEBUG: Send: <D p="43"/>
2021-03-16 14:20:37 INFO: Discard: 2p
2021-03-16 14:20:39 DEBUG: Get: <D43/> <U/>
2021-03-16 14:20:41 DEBUG: Get: <E61/>
2021-03-16 14:20:43 DEBUG: Get: <PROF lobby="0" type="1" add="13.0,0,1,0,0,0,8,2,0,1,0"/> <AGARI ba="0,0" hai="36,41,45,56,61,67,129
2021-03-16 14:20:48 DEBUG: Send: <NEXTREADY />
2021-03-16 14:20:48 INFO: Log: http://tenhou.net/0/?log=2021031703gm-0001-0000-12b0e05d&tw=2
2021-03-16 14:20:48 INFO: Final results: [NoName (61,000) 71.0, phx527 (33,000) 13.0, TAKU (14,000) -26.0, jilojilo (-8,000) -58.0]
2021-03-16 14:20:48 DEBUG: Send: <BYE />
2021-03-16 14:20:48 INFO: End of the game
```

Fig. 12: Example run with 3 human players[1]

8.2 Training Results

In this section, we will talk about the training results we have for various models. In our current stage of development, only supervised models finished the training, hence, only those models have training data.

8.2.1 Supervised Model

As the foundation of our mahjong system, we want these supervised models to have some comparable level of skills to human players, so we can continue to improve using the reinforcement learning techniques later.

8.2.1.1 Discard Model

Since the number of unique tiles in mahjong is 34, our discard model is modeled as a 34 classes classification problem. Below is the training categorical accuracy of each class in every batch. As you can see here, we are achieving consistent results with about 60%

---

[1] Replay log - http://tenhou.net/0/?log=2021031703gm-0001-0000-12b0e05d&tw=2

accuracy on each class. compared to the performance from Suphx[5], we still have some work to do, but we are very close to what they have for this model.

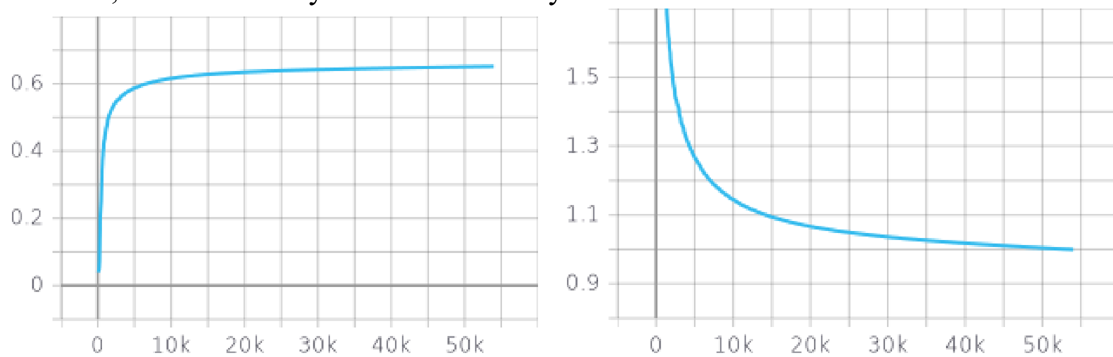


Fig. 13: Categorical Accuracy(left) and Loss(right) of Discard model

8.2.1.2 RCPK Models

RCPK models share similarities in terms of structures and tasks. They are all doing binary classification. For the sake of simplicity, some of the actions are rule based. Since they are binary, we are expecting much higher results from them. In the following graphs, these models indeed achieve some promising results. Although the loss looks strange, the problem might be the learning rate is too large at the beginning. These models still require some extensive training in order to reach their optimal performance.

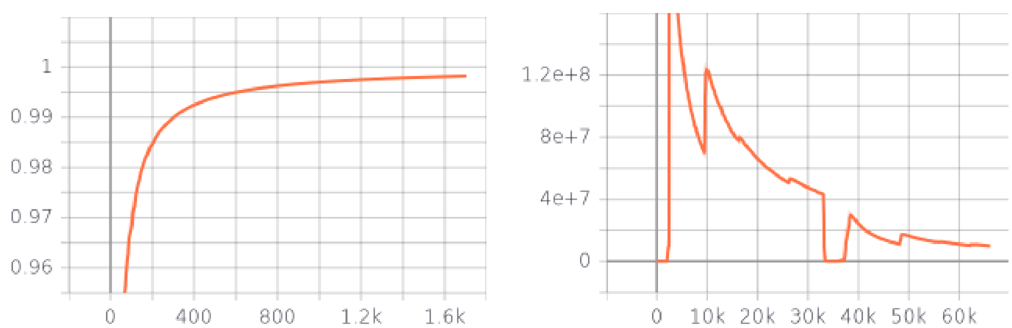


Fig. 14: Binary Accuracy(left) and Loss(right) of Kan model

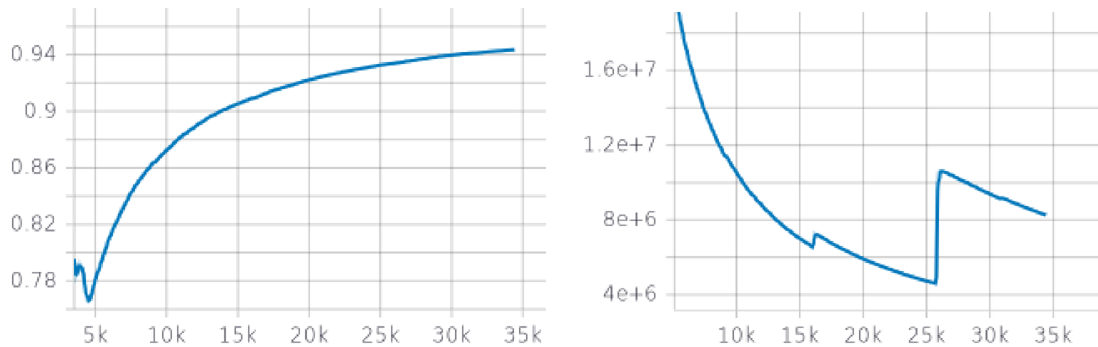


Fig. 15: Binary Accuracy(left) and Loss(right) of Riichi model

9 Future Work

We will continue to work on the following sections:

- Targeting higher ranking in tenhou platform
- Reinforcement learning boosting
- Explaintble and interpretable AI agent

References

[1]"Japanese Mahjong", *En.wikipedia.org*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Japanese_Mahjong. [Accessed: 16- Mar- 2021].

[2]"Mahjong", *En.wikipedia.org*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Mahjong. [Accessed: 16- Mar- 2021].

[3]J. Li et al., "Suphx: Mastering Mahjong with Deep Reinforcement Learning", *arXiv.org*, 2021. [Online]. Available: https://arxiv.org/abs/2003.13590. [Accessed: 16- Mar- 2021].

[4]R. Sutton and A. Barto, *Reinforcement learning: an introduction*, 2nd ed. .

[5]M. Zhang, "Meet Microsoft Suphx: The World's Strongest Mahjong AI", *Medium*, 2021. [Online]. Available: https://medium.com/syncedreview/meet-microsoft-suphx-the-worlds-strongest-mahjong-ai-a0b0a63eb871. [Accessed: 16- Mar- 2021].