Phoenix: An Open-Source, Reproducible and Interpretable Mahjong Agent

Jun Lin
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
junlin@usc.edu

Yu Xing
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
xingy@usc.edu

Jingwen Sun
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
sunjingw@usc.edu

Bingling Huang
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
binglinh@usc.edu

Aiqi Liu
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
aiqiliu@usc.edu

Na Li
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
nli10945@usc.edu

Jiabao He
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
jiabaohe@usc.edu

Chen Cui
Computer Science Department, Viterbi
University of Southern California
Los Angeles, USA
cuichen@usc.edu

***Abstract*—Mahjong is the multi-round tile based multiplayer game with imperfect-information and it has more complex score rules than Go. AI has already achieved great success in the field of perfect-information games such as Go. We would like to design an AI which could succeed in playing imperfect-information game–mahjong. We will start our research based on the rules of Riichi mahjong since it is one of the most popular mahjong variants worldwide. Also, we would use Tenhou (a web-based mahjong platform in Japan with a complete ranking system) to evaluate our AI.**

***Index Terms*—Supervised Learning, Reinforcement Learning, Mahjong**

I. Introduction

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. All 34 kinds could be divided into tiles suits of pin (circles), so (bamboo), wan (characters) and unranked honor tiles[1] [2].

At the beginning of a typical game, each player has a randomized initialized hand consist of 13 private hands. The other tiles are shuffled as the wall (piles). In each round, every player draws a tile from the wall and then discard a tile from his hand. Besides, a player can make a meld (open group) by calling another player's discard for melds. A typical meld could be three or four same tiles or three continuous tiles. Once a player has collected four melds and a pair, he could declared his winning in this round and end this round. Informally, we could describe a winning hand as x * (AAA) + y * (ABC) + DD while x + y = 4, where AAA and ABC represents melds and DD represents the pair.

The goal of our project Phoenix is to produce an open-source and interpretable Mahjong Agent which could be used to populate Riichi Mahjong as well as improving people's level in Mahjong.

II. Related Work

A. Mahjong AI

Early Mahjong AI works are mainly focus on using statistic models such as Markov Chain [3] or Monte Carlo Simulation [4]. Recently more and more deep models are utilized in modeling the decision model in Mahjong. Starting from combining deep models with rule-based models to completely deep learning based [2].

Microsoft Research Asia has recently developed a Mahjong AI named Suphx which beat 99.9 percent human players in Tenhou platform [1]. The biggest improvement comparing to previous models are that Microsoft brings Reinforcement Learning, oracle guiding and global reward predictor into modeling.Global reward predictor 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 information 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-

[1]Japanese Mahjong, En.wikipedia.org
[2]Mahjong, En.Wikipedia.org

Carlo tree search techniques, we introduce parametric Monte-Carlo policy adaptation (pMCPA) to improve the run-time performance of our agent. 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 has beaten most of human players and reaches the toppest 10 dan.

B. Reinforcement Learning

The idea of learning from interacting with the environment can be traced to infant plays [1]. Such interactions may become a source of knowledge that can be used and applied to conduct competitive or exploratory tasks.Learning from interactions is a fundamental idea underlying nearly all theories of learning and intelligence. Reinforcement Learning, which is one of them, focuses on goal-oriented learning from interactions than other approaches [5].

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 [5]. 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 [5]. Even though this problem has been intensively studied by mathematicians for a long time, it still remains unresolved yet [5]. 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 [5]. The dilemma is that it contrasts many approaches that consider dividing a big problem into several sub-problems.

Overall speaking, reinforcement learning starts with interactive and goal-seeking learning agents that have explicit goals and have the ability to perceive the environment. After agents taking actions, the environment is influenced and agents may keep perceiving the environment.

C. Bakuuchi

Bakuuchi is another player who performs Monte Carlo simulation [3].The probability of Bakuuchi has higher accuracy, the equation. Evaluate each simulation and the simulation strategy learned from the game record at the end of the hand. In an early study [3], they reported that the point dependence on policy was inappropriate and only reached an intermediate level. It is worth noting that Bakuuchi has reached an advanced level. As far as we know, no tree has been found to find better decisions.

D. NAGA

Mahjong is a difficult game because you can't see your opponent's hand and don't know which card you will draw next. Mahjong also requires a long-term strategy, and it is difficult to predict the state of the next round. Traditional Mahjong AI deals with these problems based on the expected final ranking and Monte Carlo method, while NAGA solves these problems through deep learning[5].

Naga uses confidence to learn each CNN[5], and if you are not confident in the distribution of predicted behavior, you can refer to the correct answer by penalizing it. Therefore, the phase where the confidence is low is that NAGA can not have confidence in the tile selection, that is, it is a lost phase.

Naga enters the table information and calculates the slope that can expand the selection range[5], so that the produced features can be visualized. Conversely, you can also visualize features that will no longer make a selection by calculating the slope of the selection attenuation. In the deepest layer of the neural network, it seems that judgments are made with complex granularity.

III. Dataset

A. Data Sources

Thanks to the well designed logging system provided by Tenhou platform, each game is logged in the XML format (Extensible markup language). 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 since 2009. Although earlier logs from 2005 are ready, but the highest ranking game logs are available only after 2009. Since we are targeting for high quality games, the low ranking games and three-player game logs are filtered. Here are statistics of our datasets. For some unknown reason the logs in 2020 are missing in the Tenhou platform. We will skip them for now.

TABLE I: Size of Raw Data

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

B. Data Preprocessing

1) Data for Supervised Learning: In order to generate training data for supervised learning, we need to take a snapshot of the table before each actions. Since raw data is a log file which means it increases incrementally. We have to simulate the changes according to the log and take snapshots to our table.

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 takes a lot of space. Then we tried to store them in hdf5 format, since in this way, we store our data in disk, and will save space for memory. However, to store data into an hdf5 dataset, we need to keep the data shape the same, which takes more space. Similarly as npy5 file. So 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 persisting the data to json after completing one log.

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

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.

C. Data Format

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.

Attributes	Value Type or Options
draw_tile	[136tiles]
hands	[136tiles]
discarded_tiles_pool	[136tiles]
four_players_open_hands	[136tiles for player 0,1,2, 3]
discarded_tile	[136tiles]

Fig. 1: Data Format for Discard Model

Attributes	Value Type or Options
player_id	0/1/2/3
dealer	0/1/2/3
repeat_dealer	int
riichi_bets	int
player_wind	S/N/E/W
prevailing_wind	S/N/E/W
player_tiles	closed hand: [136tiles]
	open hand: [136tiles]
	discarded tiles: [136tiles]
open_hands_detail	tiles: [136tiles]
	meld_type: Chi/Pon/AnKan/MinKan/KaKan
	reacted_tile: [136tiles]
enemies_tiles	enemy_seat: 0/1/2/3
	closed_hand: [136tiles]
	open_hand: [136tiles]
	discarded_tiles: [136tiles]
dora	[136tiles]
scores	[scores for player seat 0, 1, 2, 3]
last_player_discarded_tile	[136tiles]
could_chi	0 for false, 1 for true
could_pon	0 for false, 1 for true
could_minkan	0 for false, 1 for true
is_FCH	0 for false, 1 for true
action	[136tiles]

Fig. 2: Data Format for Riichi/Chow/Pong/Kong Model

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.

IV. Methodologies

A. Supervised Learning

Supervised learning is considering one of the most well-known technique used in machine learning field. In supervised learning, each example is a pair of input array with the desire output or so called the ground truth label. Some of the well known examples are Convolutions Neural Network(CNN) and Recurrence Neural Network(RNN), etc. We adapted the idea from CNN model in our supervised model extensively and designed the following models using CNN to learn the knowledge from high ranking human players.

1) Discard Models: 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. In the discard model, we first have our input features stack vertically together to form the D x 34 x 1 matrix. In the current states, we only use four types of features and in the future, we will use some look ahead features to push the performance further more. We start by three layers of convolutional layers with filter size 256 and kernel size 3x1. In the next step, we utilized the residual block. In ResNet, there are many deep convolutional layers, a classic example is ResNet50, 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 Fig 3, 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 the one dimensional array. At this point, all we need to do is to apply a dense layer with softmax activation function to distribute the probabilities to 34 classes. In this model, we employ the categorical accuracy as our loss function

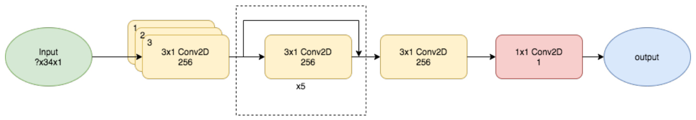


Fig. 3: Discard Model Architecture

2) Riichi, Chow, Pong, and Kong model: RCPK model for short share the same network architectures due to the nature of binary actions. Model only need to decide whether or not to perform this action. In gaming playing environment, the rule is more complex than just making a decision, but for now, we will keep it simple here. We will implement some rule based checks to check whether the agent are able to perform this action with available information on table. Each model in these four has different kinds of features, but structurally they are all in the Dx34x1 form, where the D is the feature stack dimension depending on the model. Similar to the discard model, we used the three layer of convolutional layers and five repeated residual blocks. In Fig 4 here, we observed that everything so far is identical to the discard model, but for next, instead of one layer convolution layer, three layers are used. This is used to propagate more information though the layers. Flatten applied directly to the output and go though the one layer of 1024 nodes fully connected layer and another layer of 256 nodes fully connected layer. Finally, distribute the probabilities to binary classes.

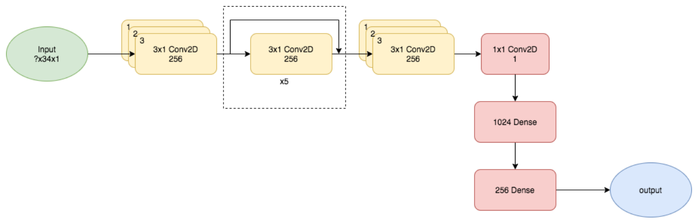


Fig. 4: Chow,Pong, Kong, Riichi Model Architecture

B. *Reinforcement Learning in Mahjong*

The complexity of mahjong is two fold. The first complexity is that we take different strategies under different circumstances. Most of the time the following factors are considered. Firstly, the level of your enemies. If in a low level room, you could defend less and become more aggressive. Secondly, our current rank in this game. If we are currently rank 1, we tend to be conservative to avoid dealing in. Otherwise, we tend to play aggressively to gain more scores. Thirdly, the quality of initial hand. With low-quality initial hand, we will have more difficulty reaching tempai. In this process, we are prone to dealing in leading worse status.

The second complexity is that how we invade or defend. Mahjong is an imperfect information game and the randomness bring more challenge for players to make a plan to invade or defend. A common strategy is to determine which tile is safe based on others' discarded tiles. There are many mahjong research about the correlation between safe tile and others' discarded tiles.

Here we mainly take the methods from Suphx. To solve the challenge of imperfect information, oracle guiding is applied. Firstly a strong agent is trained by providing all information, both visible and invisible. This agent will perform really well because he could see others' private tiles and take the best strategy. Afterwards we gradually decay the invisible information in the training. We will end up with an agent with visible information only. We could say that the visible information are now utilized appropriately in their knowledge structure.

As for the challenge of strategies, we take the same global reward predictor as Suphx. The advantages of global reward predictor is two fold. Firstly, it could help us distribute the final rewards into each round in a game so that we could get more precise and instant reward after a round. Secondly, it should be mentioned that losing scores doesn't always mean negative signals. To keep our advantageous state, if we start with a low-quality hand, we should decidedly defend to avoid lose too many scores. Low quality initial hand means high risk and low gains if blindly try to win. The global reward predictor will impose positive signals to "reasonable losing" and big penalty to "obtrusive losing".

An asynchronous and distributed learning method is applied. The RL pipeline could be divided into client side and server side. The client side integrates our agents into Tenhou platform and collect state, actions and rewards into replay buffer. The parameter servers keep training latest deep models using recent logs in replay buffer using policy gradient. The client would fetch latest parameters from server side after a certain period.

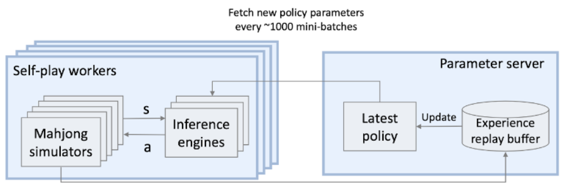


Fig. 5: Reinforcement Learning Structure in Mahjong Training

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.

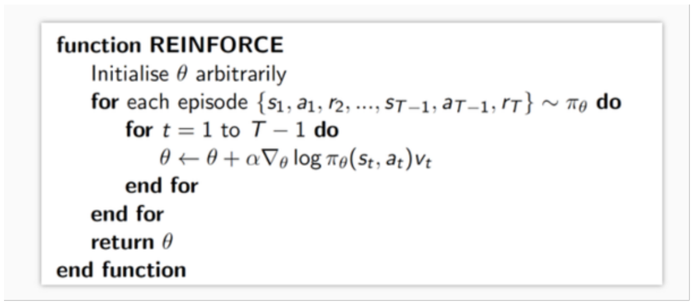
The reinforcement learning algorithm applied is one of policy gradient methods which parameterized policy that selects actions without consulting a value function [5]. To seek maximize performance, gradient ascent in J()to update policy's parameter: $t+1 = t + \alpha(\theta)$ (1). And the objective function is shown as below [1].

$$\Delta_\theta J(\pi_t heta) = \underset{s,a\sim\pi_{\theta'}}{E}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}\Delta_\theta log\pi_\theta(a|s)A^{\pi_\theta}(s,a)s\right] + \alpha\Delta_\theta H(\pi_\theta) \quad (1)$$

And the pseudo code is shown in the figure below. This method is classical REINFORCE algorithm [4] whose update at time t only involved current action At. By summing over a certain amount of samples, policy gradient theorem can guarantee the objective function J() 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).

Fig. 6: Online Training Environment



```
function REINFORCE
    Initialise $\theta$ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
        for $t = 1$ to $T - 1$ do
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return $\theta$
end function
```

Fig. 7: Pseudo Code for Policy Parameter Updating

The baseline can be any function as long as it does not vary with action [5]. The optimization equation is valid since the subtracted quantity is zero [5]. 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 [1]. Thus, a global reward to each round of the game is predicted by a global reward predictorΦ. In our system, the global reward predictor $\Phi$is a recurrent neural network and is trained by minimizing the following mean square error [1]:

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

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 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.

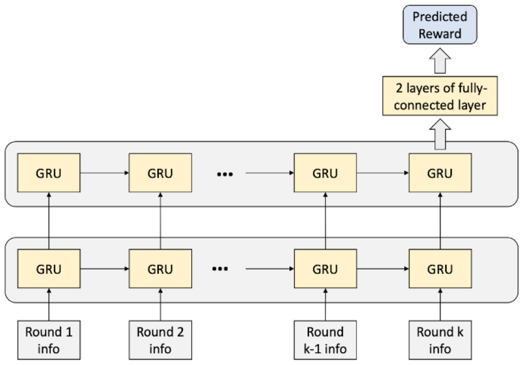


Fig. 8: Global Reward Predictor Work Flow

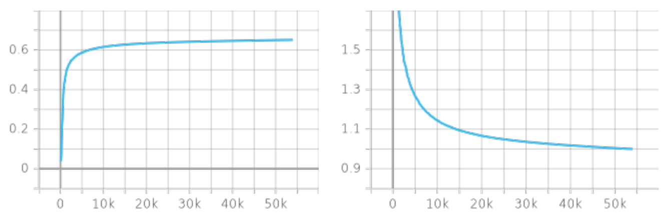
V. Experiments

A. Google Cloud Platform

1) Preprocessing using Dataflow: Processing such a huge dataset locally is infeasible and impractical. 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.

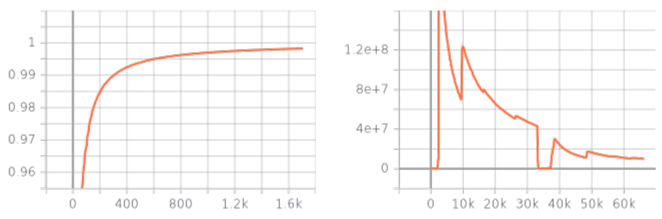
Utilizing the preprocessing logic from the III-B, we employ an Apache Beam pipeline to speedup and scale for processing our dataset. Apache beam is a unified programming model to define and execute data processing pipelines, including extract, transform, and load(ETL), batch and stream (continuous) processing. 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 III-B. After getting the result from III-B 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



(a) Discard Categorical Accuracy (b) Discard Losses

Fig. 9: Discard model results

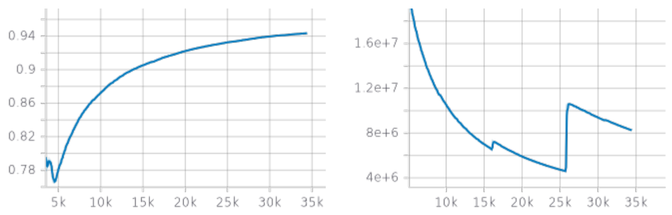


(a) Kong Binary Accuracy (b) Kong Losses

Fig. 10: Kong model results

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 file are generated and ready to begin the training process.

2) Distribute training using Mirrored Strategy: Training on a single GPU is slow and impractical in bigdata era. With the grow of cloud services providers like Amazon web servises(AWS) and Google Cloud Platform (GCP), distribute training is not longer the headache for researchers. Especially, with the help of higher level machine learning frameworks like Keras and Tensourflow, researchers can deploy multi-GPUs or even multi worker with multi-GPUs with ease. In our setting, we expedite the process by utilized the MirroredStrategy with 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 a server. This is some fast way to train with multiple GPUs without any overhead. The main idea of MirroredStrategy is to make a copy of the model variable to all the processors. During the training, it will combine the gradients all together and apply changes to all copies of processors. There are some other kinds of strategies. We currently adopte MirroredStrategy due to the simiple usage and the more stable compare to others. Since most of our training code is written in Keras and Keras is the higher level API for tensorflow, we can easily configure the strategy on our existing code without much modification. 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 machine which has 8 virtual CPU (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 few hours. These models have smaller datasets compared to discard models due to the nature of these actions. On the other hand, train 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 hyper-parameter tuning which allows us to scale up horizontally by adding more machines with multiple GPUs instead of single machine.



(a) Riichi Binary Accuracy (b) Riichi Losses

Fig. 11: Riichi model results

B. Training results with Supervised models

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. We currently have some potential issue with the Pong and Chow models. For this reason, they are not included in this part and will be supplied in the next iteration. In Fig 9, discard model achieved with about 60% categorical accuracy for this task. Although the result seem to be low, but in the example run, the performance of discard model is notable against human player. Compare to the previous work from Suphex [1], we are achieving some comparable results without any tuning. For the Kong and Riichi models in Fig 10, 11, we slightly outperform the baseline set by the Suphex [1].

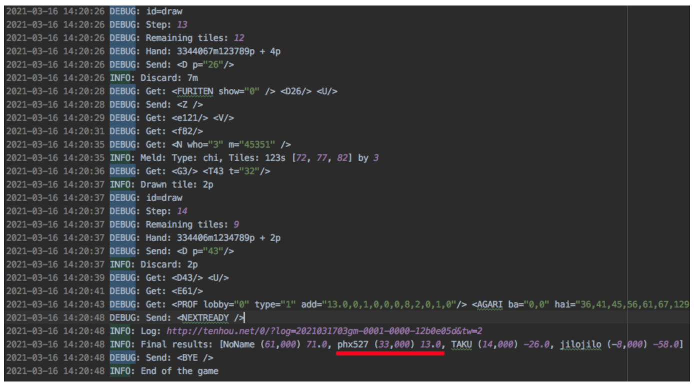

```
2021-03-16 14:20:26 DEBUG: id=draw
2021-03-16 14:20:26 DEBUG: Step: 13
2021-03-16 14:20:26 DEBUG: Remaining tiles: 12
2021-03-16 14:20:26 DEBUG: Hand: 3344067m123789p + 4p
2021-03-16 14:20:26 DEBUG: Send: <D p="26"/>
2021-03-16 14:20:26 INFO: Discard: 7m
2021-03-16 14:20:28 DEBUG: Get: <FURITEN show="0" /> <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 aginst three human players

C. Tenhou Platform Integration

One important aspect of our project is the ability to play mahjong in real-time with human players. In the other hand, the reinforcement learning needs an environment for agents to play around as well. In our current setup, agents are allow to play with builtin computer player, with other three human players, or form an table privately and play. We utilized the the builtin computer players in our debugging phase. Once everything work in the environment with builtin computer players, We release our agent to the public available table in tenhou to evaluate the performance against human player. In the reinforcement learning phase, we will deploy multiple agents with docker container and form many tables in tenhou for agent to do the self-play. We have the infrastructure for all three type of setup. The first two works as expect. During the game our agent collect features from the known information in the table as the game progresses and transforms them into the single feature row in the format of the training dataset. This feature is used by models to make predictions. In discard model, the prediction will give us the distributions of probabilities of each class. Argmax applied to find out which one has the highest probability. The results are mapped to the corresponding unique tiles. We send a web-socket message from tenhou client to server and update the states 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. Fig.12 shown as an example run[3] against three human players. The red underline phx527 is our agent bot. Our current supervised models achieved 2nd place in this game. Agent plays relatively defensive against human players. It doesn't throw out any existing meld in hands and cautiously discard the tile that can't be used against self.

Acknowledgment

We would like to express our sincere gratitude to our professor Dr. Michael Zyda for his valuable advice and guidance on our project. We also want to thank teach assistants team for their hard working on making everything happen.

References

[1] Ege Eren and A. Nihat Berker. Metastable reverse-phase droplets within ordered phases: Renormalization-group calculation of field and temperature dependence of limiting size. *Physical Review E*, 101(4), Apr 2020.

[2] Shiqi Gao, Fuminori Okuya, Yoshihiro Kawahara, and Yoshimasa Tsuruoka. Building a computer mahjong player via deep convolutional neural networks, 2019.

[3] Moyuru Kurita and Kunihito Hoki. Method for constructing artificial intelligence player with abstraction to markov decision processes in multiplayer game of mahjong, 2019.

[4] N. Mizukami and Y. Tsuruoka. Building a computer mahjong player based on monte carlo simulation and opponent models. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 275–283, 2015.

[3]http://tenhou.net/0/?log=2021031703gm-0001-0000-12b0e05dtw=2

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.