Towards a Competitive 3-Player Mahjong AI using Deep Reinforcement Learning

Xiangyu Zhao  
Department of Computer Science and Technology  
University of Cambridge  
Cambridge, United Kingdom  
xz398@cam.ac.uk

Sean B. Holden  
Department of Computer Science and Technology  
University of Cambridge  
Cambridge, United Kingdom  
sbh11@cl.cam.ac.uk

Abstract—Mahjong is a multi-player imperfect-information game with challenging features for AI research. Sanma, being a 3-player variant of Japanese Riichi Mahjong, possesses unique characteristics and a more aggressive playing style than the 4-player game. It is thus challenging and of research interest in its own right, but has not been explored. We present Meowjong, the first AI for Sanma using deep reinforcement learning (RL). We define a 2-dimensional data structure for encoding the observable information in a game. We pre-train 5 convolutional neural networks (CNNs) for Sanma’s 5 actions—discard, Pon, Kan, Kita and Riichi, and enhance the major (discard) action’s observable information in a game round. We then pre-train 5 deep CNNs, each corresponding to one action in Sanma, by achieving test accuracies comparable with AIs for 4-player Mahjong through supervised learning, and gaining a significant further enhancement from reinforcement learning.

Index Terms—Mahjong, deep learning, reinforcement learning, convolutional neural networks, policy gradient methods

I. INTRODUCTION

Mahjong is a popular tile-based, multi-round, multi-player, imperfect-information game developed in China in the late 19th century. It has hundreds of millions of players worldwide. It is a game of skill, strategy and calculation, and involves a degree of chance. It is challenging for AI because:

• It is played by more than 2 players and has significant hidden information;
• It has complex playing and scoring rules;
• It has a huge number of winning hands in various patterns, allowing flexible in-game strategy adaptation.

While there are many variants of Mahjong, we focus on Sanma, a 3-player variant of Japanese Riichi Mahjong. Sanma differs fundamentally from 4-player Mahjong because most of the tiles in one suit (Manzu) are removed, and one of the actions (Chii) is replaced with another (Kita). Consequently, both the number of states and the amount of hidden information are reduced, and hands tend to develop faster. As a result, players tend to play more aggressively, and valuable winning hands are more frequent. Since it is commonly understood by expert players that 4-player Mahjong should be played defensively, an AI trained to learn optimal strategy for the 4-player game is not guaranteed to acquire the optimal strategy for Sanma. Therefore, though Sanma is seemingly simpler than 4-player Mahjong, it still preserves all the challenging characteristics of Mahjong, and the difference in strategy makes it a worthwhile target for research.

In this paper, we describe Meowjong, the first Sanma AI based on deep RL. We propose a data structure for encoding the observable information in a game round. We then pre-train 5 deep CNNs, each corresponding to one action in Sanma, and improve the discard model using the Monte Carlo policy gradient method for self-play RL.

II. RELATED WORKS

As far as we are aware, there has been no published attempt to develop an AI player for Sanma. Various machine learning approaches have addressed 4-player Mahjong, including Baku-uchi by Mizukami et al. [1], Kurita et al. [2], Gao et al. [3] and Suphx by Li et al. [4]. While the performance of these systems is impressive, we emphasize that, as we are unaware of comparable work addressing Sanma, no such comparisons with human players are available in this context.

III. FEATURES AND DATA STRUCTURE DESIGN

Sanma has 108 tiles of 27 different kinds, with four identical tiles in each kind. Actions of Sanma include discard, Pon, Kan, Kita and Riichi, with discard being the major action. A description of the rules and terminology of Sanma can be found in the full version of this paper [5]. Unlike board games such as chess and Go, the layout of a Mahjong board is not standardized, and the observable information must be encoded to be digested by CNNs. As there are 34 tiles, we use a 34 × 366 array to represent a state. Although 2m–8m are not included in Sanma, leaving only 27 of the 34 tiles to be used, we include all the tiles and leave the excluded tiles blank, to add transferability of Meowjong to 4-player Mahjong. The mapping between the tiles and their row indices in the array encoding is shown in TABLE I.

To simulate the game environment and maximize Meowjong’s performance, all observable information should be included. Thus, we use 366 columns to represent 22 features, including...
as listed in TABLE II. We do not include the number of remaining tiles, as it can be calculated from the number of discarded tiles. For triplets, quads and Riichi status, we include not only the triplet/quad tiles and the Riichi status, but also the turn numbers of the Pon/Kan/Riichi calls. Although the spaces for the fourth player are not needed for Sanma, they are created for the sake of transferability of Meowjong to 4-player Mahjong.

The features can be divided into two categories:

- **Tile features** involve sets or sequences of tiles, such as private tiles, triplets/quads, and so on. They can be encoded by setting the corresponding row indices to 1 and leaving the rest to 0, as, for example, shown in Fig. 1.
- **Numerical features** such as player scores can be binary-encoded into multiple columns, each being either all zeros or all ones.

### IV. Method

We parameterize Meowjong’s policy on an action basis: we use models, called action models, to parameterize Meowjong’s policy for each action (discard, Pon, Kan, Kita, and Riichi). After experimenting with various choices for the model structures, including the number of hidden layers and the number of filters per layer, we adopted a CNN with 4 convolutional layers followed by a fully-connected layer. The first 3 layers have 64 filters, and the last has 32 filters. All filters share the same size, which is a hyperparameter to be tuned individually for each action model. The fully-connected layer has 256 hidden nodes. A batch normalization layer and a dropout layer with dropout rate 0.5 are added after each convolutional and fully-connected layer to prevent over-fitting. Action models share a similar structure, differing in the filter sizes and the output dimensions (34 for the discard network, and 2 for the rest). ReLU is used for the activation function of all layers except the outputs, for which softmax is used. The input and output dimensions of the models are shown in TABLE III. Since the decision-making problems in Sanma can be converted to multi-class classification problems, we define the loss for our CNNs to be the categorical cross-entropy

\[
L(w) = - \sum_{i=1}^{m} \sum_{k=1}^{K} y^{(i)}_k \log \hat{y}^{(i)}_k
\]

where \( m \) is the number of examples, and \( K \) is the number of classes. Although pooling is recognized as an efficient down-sampling tool for CNNs in computer vision tasks, in Meowjong’s case the data structure is not an image, but an encoding of discrete feature data, and we would expect the use of pooling to lose too much information, leading to lower accuracy. Therefore, pooling is not used in Meowjong’s CNN structure. No padding is used in any convolutional layer.

The discard problem can be interpreted as a 34-class classification problem, since there are 34 kinds of tile in total, or a 14-class classification problem, since each player can have at most 14 tiles in their private hand. As recommended by Li et al., we use models, called action models, to parameterize Meowjong’s policy for each action (discard, Pon, Kan, Kita, and Riichi). After experimenting with various choices for the model structures, including the number of hidden layers and the number of filters per layer, we adopted a CNN with 4 convolutional layers followed by a fully-connected layer. The first 3 layers have 64 filters, and the last has 32 filters. All filters share the same size, which is a hyperparameter to be tuned individually for each action model. The fully-connected layer has 256 hidden nodes. A batch normalization layer and a dropout layer with dropout rate 0.5 are added after each convolutional and fully-connected layer to prevent over-fitting. Action models share a similar structure, differing in the filter sizes and the output dimensions (34 for the discard network, and 2 for the rest). ReLU is used for the activation function of all layers except the outputs, for which softmax is used. The input and output dimensions of the models are shown in TABLE III. Since the decision-making problems in Sanma can be converted to multi-class classification problems, we define the loss for our CNNs to be the categorical cross-entropy

\[
L(w) = - \sum_{i=1}^{m} \sum_{k=1}^{K} y^{(i)}_k \log \hat{y}^{(i)}_k
\]

where \( m \) is the number of examples, and \( K \) is the number of classes. Although pooling is recognized as an efficient down-sampling tool for CNNs in computer vision tasks, in Meowjong’s case the data structure is not an image, but an encoding of discrete feature data, and we would expect the use of pooling to lose too much information, leading to lower accuracy. Therefore, pooling is not used in Meowjong’s CNN structure. No padding is used in any convolutional layer.

The discard problem can be interpreted as a 34-class classification problem, since there are 34 kinds of tile in total, or a 14-class classification problem, since each player can have at most 14 tiles in their private hand. As recommended by Li et al., we use models, called action models, to parameterize Meowjong’s policy for each action (discard, Pon, Kan, Kita, and Riichi). After experimenting with various choices for the model structures, including the number of hidden layers and the number of filters per layer, we adopted a CNN with 4 convolutional layers followed by a fully-connected layer. The first 3 layers have 64 filters, and the last has 32 filters. All filters share the same size, which is a hyperparameter to be tuned individually for each action model. The fully-connected layer has 256 hidden nodes. A batch normalization layer and a dropout layer with dropout rate 0.5 are added after each convolutional and fully-connected layer to prevent over-fitting. Action models share a similar structure, differing in the filter sizes and the output dimensions (34 for the discard network, and 2 for the rest). ReLU is used for the activation function of all layers except the outputs, for which softmax is used. The input and output dimensions of the models are shown in TABLE III. Since the decision-making problems in Sanma can be converted to multi-class classification problems, we define the loss for our CNNs to be the categorical cross-entropy

\[
L(w) = - \sum_{i=1}^{m} \sum_{k=1}^{K} y^{(i)}_k \log \hat{y}^{(i)}_k
\]

where \( m \) is the number of examples, and \( K \) is the number of classes. Although pooling is recognized as an efficient down-sampling tool for CNNs in computer vision tasks, in Meowjong’s case the data structure is not an image, but an encoding of discrete feature data, and we would expect the use of pooling to lose too much information, leading to lower accuracy. Therefore, pooling is not used in Meowjong’s CNN structure. No padding is used in any convolutional layer.
TABLE V
SIZES OF THE DATASETS

<table>
<thead>
<tr>
<th>Action</th>
<th>Dataset Size</th>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discard</td>
<td>797,285</td>
<td>88,588</td>
<td>147,444</td>
<td>3,814</td>
</tr>
<tr>
<td>Pon</td>
<td>151,348</td>
<td>16,817</td>
<td>16,887</td>
<td>3,548</td>
</tr>
<tr>
<td>Kan</td>
<td>34,319</td>
<td>3,814</td>
<td>3,548</td>
<td>797,285</td>
</tr>
<tr>
<td>Kitag</td>
<td>136,924</td>
<td>15,214</td>
<td>15,498</td>
<td>3,548</td>
</tr>
<tr>
<td>Riichi</td>
<td>109,804</td>
<td>12,201</td>
<td>11,944</td>
<td>3,548</td>
</tr>
</tbody>
</table>

TABLE VI
TEST ACCURACIES FOR THE ACTION MODELS IN SUPERVISED LEARNING

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meowjong</td>
<td>85.81%</td>
</tr>
<tr>
<td>Gao et al. [3]</td>
<td>68.8%</td>
</tr>
<tr>
<td>Suphx [4]</td>
<td>62.63%</td>
</tr>
</tbody>
</table>

Gao et al. [3] and Gao et al. [3], we adopt the 34-class classification interpretation. The only potential risk of the 34-class method is of illegal discards, where the discard model outputs a tile that does not exist in the player’s private hand. However, in practice all discard choices made by our discard model are legal. For all the rest of the actions, a player can either declare or skip that action in appropriate situations, so all those actions are binary classification problems and we set up a 2-dimensional output for each of their action models.

Hyperparameter tuning for the pre-training of the CNNs focuses on the filter size for each action model. As there is no guarantee that the same filter size works for every action, the filter sizes are tuned for each model. Grid search is used for hyperparameter tuning: for each action, all candidate filter sizes $(x, y)$ ranging from $2 \leq x, y \leq 5$ are tried, making 16 candidates in total. The best-performing filter sizes are shown in TABLE IV.

For RL training of Meowjong, we adopt the Monte Carlo policy gradient method to update the weights $w$ of the discard model by

$$w' = w + \eta \gamma T G_t \nabla_w \log \pi(A_t|S_t, w)$$  \hspace{1cm} (2)$$

where $S_t$ denotes the game state at step $t = 0, 1, \cdots, T$, $A_t$ denotes the action taken at step $t$, and $G_t$ denotes the agent’s cumulative reward, with the value at the terminal state $G_T$ being the round score/penalty. A detailed description of our RL algorithm can be found in [5]. After hyperparameter tuning, we adopt $\eta = 10^{-3}$ and $\gamma = 0.99$ as the optimum hyperparameter setting.

V. EXPERIMENTS

A. Supervised Learning

We sampled 50,000 rounds of Sanma game records from 2019 from the “Houou” table on Tenhou [6] for training our models. 10% of those data were divided, with stratification, to form the validation datasets of the actions. The “Houou” table is only open for the top 0.1% of the ranked players, so its game records can be considered to be of high quality. To test the generalizability of our models, we sampled another 5,000 rounds from 2020 to form the test dataset. We use data from 2020, rather than 2019, as this potentially represents a harder generalization problem. The sizes of the datasets are shown in TABLE V. Our models are implemented using TensorFlow, and after hyperparameter tuning, Adam with learning rate $10^{-3}$ is used as the models’ optimizer. The discard models are trained in mini-batches of size 64 for 200 epochs, and the rest of the models are trained in mini-batches of size 32 for 500 epochs. Each model is trained on 4 NVIDIA P100 GPUs with 64GB memory in total, and takes from 10 hours (Kan model) to 30 hours (discard model).

TABLE VI shows the test accuracies of Meowjong’s models, along with those achieved by previous works. Note that these test accuracies are not directly comparable due to the different training/validation/test data sources and structures, but they can serve as an approximate reference. The test accuracies of Meowjong are very satisfactory, achieving Gao et al.’s level [3] at the most important action—discard. Although there is still a gap between Meowjong and Suphx, it is worth noting that Suphx used very large, 102/104-layer residual CNN structures, with training datasets of 4M–15M examples, and cost much more to train [4], whereas Meowjong adopted a much simpler CNN structure, used a much smaller training dataset, and was much cheaper and faster to train.

B. Reinforcement Learning

The discard model was improved through self-play RL as described for 400 episodes. Evaluations against 2 baseline agents (described in Section VI-A) were carried out and recorded after every 10 episodes, each playing 500 rounds as East, South, and West. The curves of the win (1st place) and loss (3rd place) rates are in Fig. 2, showing an improvement on all winds. The curves also suggest that the RL agent is likely to have learned an aggressive style, and finessed its skill through self-play, increasing its win rate while lowering its loss rate.

VI. AGENT EVALUATION

A. Agents

We trained the following agents for evaluation:

- SL agent: the supervised learning agent with all 5 models trained using supervised learning;
- RL agent: the reinforcement learning agent with the Pon, Kan, Kita and Riichi models inherited from the SL agent, but its discard model enhanced through RL.

We also built an agent that takes actions randomly to serve as a baseline. This agent is believed to have a similar power to the bots on the major online platforms.
B. Results Against Baseline

We simulated 5,000 rounds of Sanma amongst 3 baseline agents, and 5,000 rounds for each of the SL and RL agents against 2 baseline agents, in each wind position. Their results, in terms of 1st/2nd/3rd-place and draw rates, are in TABLE VII. The results show that both Meowjong’s trained agents can outperform a baseline agent, and the RL agent’s 1st place rates are also much larger than both SL agents. The RL agent also has higher 3rd-place rates, which suggests that it has learned an aggressive style. This is also confirmed by the box plots of all scores shown in Fig. 3. We also devised a significance test to formally assess the differences between agents, which can be found in [5].

C. Comparison Between SL and RL Agents

In this series of evaluations, we simulated 5,000 rounds amongst 3 SL agents, and 5,000 rounds in each wind position between 2 SL agents and RL vs. 2SL. The results are in TABLE VIII. It is clear that an RL agent can outperform an SL agent. The 3rd place rates at the East position are due to the rule that the East player pays twice as much as the other player when a non-East player wins by self-draw. The difference can be confirmed by the box plots of the scores in the simulation shown in Fig. 4. Both SL agents seem to perform better at the South position.

VII. Conclusions

In this paper, we design a data structure to encode the observable states in Sanma, build a CNN structure that solves Sanma’s decision-making problem, and train several agents using supervised learning and RL. All our action models achieve test accuracies comparable with AIs for 4-player Mahjong through supervised learning, and gain a significant further enhancement from RL. Being the first AI for Sanma, we can claim that Meowjong stands as state-of-the-art in this game, and possesses potential to be successful in future evaluations against human players.

Future work might include taking multi-round ranking information into account, letting Meowjong try to maximize the full-game results in addition to the single-round performance. An expert player adapts to different strategies flexibly based on their score and running at the beginning of each round along with the progress of the entire game. For example, a player may play defensively in the later rounds when they have a lead, and may lose deliberately to the lowest-placed player in the last round to secure 1st place. If Meowjong could learn such a flexible full-game strategy, this would surely improve its multi-round performance. Our data structure for encoding the state already supports such an upgrade, and sequential neural network structures such as GRUs and LSTMs might have potential for tackling this task. We would also like to include a recent advance in RL for Mahjong, using variational latent oracle guiding [7], and seek to reproduce their results as well as leveraging Meowjong’s performance.

REFERENCES