Mastering the Game of Mahjong with Deep Reinforcement Learning

Tiancheng Fang, Hang Zeng, Ziyin Zhang, Yutian Liu
Department of Computer Science and Engineering
Shanghai Jiao Tong University
`{fangtiancheng, nidhogg, daenerystargaryen, stau7001}@sjtu.edu.cn`

Abstract

Reinforcement learning has achieved remarkable feats in playing games, but has yet to conquer Mahjong, a popular tile-based board game with imperfect information and complex rules. In this work, we implement a Mahjong game agent under Bayesian learning framework utilizing the technique of oracle guiding during training. We use Deep Q-learning as base algorithm, and apply improvements including double Q-learning, dueling architecture, and parametric noise. We also implement a Mahjong game frontend for human tests, and interface our agent with the online Mahjong platform Mahjong Soul. Experiments show that our agent achieves an average human-level performance in playing Mahjong.

1 Introduction

Games have long served as touchstones for reinforcement learning and more general artificial intelligence algorithms. In the past decade, deep reinforcement learning has achieved remarkable performance in many once human-dominated games, including Atari games [15, 20, 22, 16, 7, 12], chess [1, 18], Go [17, 19, 18], shogi [18], poker [3, 4, 5], Minecraft [8] and even multi-player esports games that are far more complex than board games, as in Dota [2] and StarCraft II [21].

However, Mahjong, a usually four-player tile-based game that is especially popular in east Asia, still remains a challenging ground for reinforcement learning. Mahjong is a multi-round game with imperfect information, where in each round the players draw and discard tiles alternatingly, competing with each other towards the first completion of a winning hand. The Japanese variant of Mahjong (also known as Riichi Mahjong, which is played with 136 tiles) starts with 13 private tiles for each player and 70 tiles in the live wall, with the state complexity as large as 10^{145} [9]. Apart from imperfect information and high complexity, Mahjong poses extra challenges for artificial intelligence algorithms with its complex rules, where the regular order of plays can be interrupted by certain actions such as Pon and Kan, rendering some widely adopted methods including Monte Carlo tree search ineffective.

In 2020, Super Phoenix (Suphx) [14], an RL agent for Mahjong trained with the techniques of global reward prediction, oracle guiding and run-time policy adaptation demonstrated human-level performance in Mahjong for the first time. And [9] generalized the oracle guiding technique introduced in Suphx into a Bayesian learning framework, named as Variational Latent Oracle Guiding (VLOG). In this project, we combine the VLOG framework and recent advancements in reinforcement learning algorithms, train an agent for Mahjong, implement a game frontend, and provide an interface for plugging our agent into online Mahjong platforms.

The rest of this work is organized as follows. In Section 2, we briefly review the settings of reinforcement learning, the common algorithms, their application in games, and give an introduction to Japanese Mahjong. In Section 3 and 4 respectively we give the two theoretical bases of our Mahjong agent - variational auto-encoder and variational oracle guiding. Then, we present the experimental settings and empirical resutls of our Mahjong agent in Section 5, and draw conclusions in Section 6.

2 Background

2.1 Reinforcement Learning

Generally, reinforcement learning (RL) environments are modeled as Markov Decision Processes (MDPs), defined by a tuple $< \mathcal{S}, \mathcal{A}, T, R, \gamma >$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively, and T specifies the state transition probabilities. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor. For applications with imperfect information such as Mahjong, MDP is generalized to Partially Observable MDP (POMDP), defined by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{O}, \Omega, T, R, \gamma >$, where $\mathcal{O}$ is the observation space, and Ω gives the function mapping from underlying $\mathcal{S}$ to $\mathcal{O}$.

During the training stage, RL agent interacts with the environment by repeatedly taking sensory input of the environment state, executing an action following a policy, and receiving a reward from the environment. In this process, the agent adjusts its policy to maximize the expected cumulative reward $Q^{\pi}(s, a) = E_{\pi}[\sum_{n=0}^{\infty} \gamma^n r_{t+n} | s_t = s, a_t = a]$, or equivalently expressed as the state value function $v^{\pi}(s) = E_{\pi(a|s)}[q^{\pi}(s, a)]$. A policy can be expressed either as a straightforward probability distribution of actions, which is the foundation of policy-based RL algorithms, or as a model of state values or Q-values, from which the actions are chosen greedily or ϵ-greedily. The later approach, referred to as value-based RL, is inherently suited to applications with discrete action space and widely adopted in AI game agents.

In value-based RL, the basic idea is to maintain a table or an approximation function of state values or Q-values, and update it as the agent interacts with the environment and gains experience. A naive method is Monte Carlo sampling, which samples an entire trajectory until a terminal state is reached and computes the actual cumulative rewards before updating a state's value. Temporal difference (TD) update improves sample efficiency by estimating the Q-value of an action with only the immediate reward and the current estimation of the next state:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]. \tag{1}$$

This algorithm is known as Sarsa. It is on policy, in that it chooses the next action and estimates its value using the same policy. A further improvement on Sarsa is off-policy Q-learning, which chooses actions following a behavior policy and evaluates it using a target policy, and by decoupling the two policies allows for utilization of expert data. For example, if target policy is greedy and behavior policy remains ϵ-greedy, the Q values are than updated by

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]. \tag{2}$$

And in the last decade, as deep learning rose to its heyday with the advancement in computation devices, reinforcement learning also started going deep. The seminal work of Deep Q-Network (DQN) [15] first combined Q-learning algorithm with deep neural networks, achieving human-level performance in Atari 2600 games and marking the beginning of end-to-end reinforcement learning. Since then, many improvements have been proposed over DQN. Double DQN (DDQN) [20] decouples the selection and evaluation of actions to avoid the over-estimation prevalent in DQN. Dueling DQN [22] factors the network into two branches, one for estimating state values and one for state-dependent action advantage function, so as to generalize learning across actions. NoisyNet [7] replaces the classical ϵ-greedy exploration with parametric noise that is added directly to the network's weights, producing deterministic results within an episode between weight update and enabling consistent exploration. Prioritized Experience Replay [16] gives data with larger TD errors higher probabilities to be sampled, and Distributional DQN [12] express Q-functions in a more granular level to differentiate state-action pairs with the same expected return. And later studies have shown that all these improvements can be complementarily combined to achieve higher performance and data efficiency [10].

Combined with deep neural networks, RL has achieved remarkable feats in playing games. AlphaGo [17] was trained by both supervised learning and a combination of reinforcement learning with Monte Carto tree search (MCTS), and became the first AI agent to beat a human professional player in Go. AlphaGo Zero [19] was trained solely through RL without any domain knowledge, and defeated

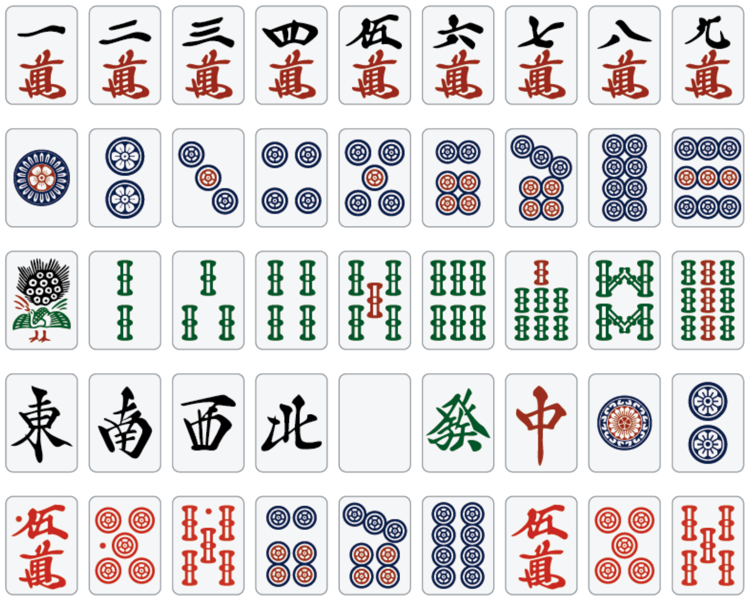


Figure 1: Tiles of Japanese Mahjong (Riichi Mahjong).

AlphaGo 100-0. Alpha Zero [18] stepped further toward a general game-playing system by dropping alpha-beta pruning and resorting back to Monte Carlo tree search, and applied the same general-purpose RL algorithm and network structure to Go, chess, and shogi without any domain-specific knowledge, defeating world champions in all three fields. Since then, the research community has started to pay more attention to the application of RL in more complex games, especially those with multiple players or imperfect information, including poker [4, 5], Minecraft [8], Dota [2], StarCraft II [21], and Mahjong [14].

2.2 Japanese Mahjong

Japanese Mahjong, also known as Riichi Mahjong, is one of the most popular Mahjong subvariants around the world. Like other Mahjong games, Japanese Mahjong is played by the participants alternatingly drawing and discarding tiles in an attempt to form a winning hand. It is played with 136 tiles (Figure 1), and starts with 13 private tiles in each player's hand, 70 in the live wall and 14 in the dead wall. The tiles consist of three suits (Dots, Bamboo, and Characters) numbered respectively from one to nine, and unranked honor tiles including four wind tiles (East, South, West, North) and three dragon tiles (White, Green, Red). Each type of tile has four identical copies.

As in most Mahjong games, a winning hand of Japanese Mahjong is formed by groups, or melds, i.e. valid collections of three or four tiles, further divided into triplets and sequences. A group can be made by Chii (make an open sequential group using a tile discarded by the previous player), Pon (make an open identical group using a tile discarded by any other player), or Kan (make a meld from 4 identical tiles in the same suit or 4 identical honor tiles, either by calling on another's discarded tile or by having the four tiles in one's hand). Additionally, Japanese Mahjong features yaku, riichi and dora. Yaku is a series of tile patterns that a player is required to have at least one in his hand to win and also the main factor in determining round score. Riichi is a special type of yaku, where a player declares ready if he has not claimed any other player's discards and needs only one more tile to complete a legal hand. And dora are bonus tiles, determined by dora indicators on the dead wall, that raise extra scores when present in a winning hand.

One major challenge in building a Mahjong AI agent is that when making a meld with Pon or Kan, the regular playing order can be interrupted. And since the agent does not have information about other players' private tiles, these interruptions are hard to predict, rendering it impossible to build a Monte Carlo search tree as the agents for Go and chess do. Suphx [14] tackles this challenge by predicting possible winning hands with depth first search (DFS) instead of directly performing MCTS to estimate state values. During training, these predictions are done ignoring opponents' behaviors and under the guidance of an oracle who has access to information about the opponents' privates tiles and tiles in the wall. This oracle guiding only serves as an approach to speed up training, and is gradually dropped during the training process. VLOG [9] generalizes this idea into a learning framework under Bayesian inference, which we will elaborate in detail in the following sections.

3 Variational Auto-Encoder

Variational Bayes is a theoretically well-established and practically powerful algorithm for modeling data distribution with latent factors. Consider a dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ where each observed sample $\mathbf{x}^{(i)}$ is generated by some random process following two steps: (1) a corresponding value $\mathbf{z}^{(i)}$ for the latent variable is drawn from some prior distribution $p(\mathbf{z})$; (2) a value for $\mathbf{x}$ is generated from the conditional distribution $p(\mathbf{x}|\mathbf{z})$. The meaning of $\mathbf{z}$ and $\mathbf{x}$ varies across applications, for example a player's private tiles and his discarded tile in Mahjong, or an action and its Q value under the general settings of reinforcement learning.

To model the distribution of $\mathbf{x}$, one approach is expectation-maximization (EM) algorithm, where a lower bound for the log likelihood of observed data is constructed and maximized in each step:

$$\begin{aligned}
\log p(\mathbf{x};\theta) &= \log \int p(\mathbf{x},\mathbf{z};\theta) d\mathbf{z} \\
&= \log \int q(\mathbf{z}) \frac{p(\mathbf{x},\mathbf{z};\theta)}{q(\mathbf{z})} d\mathbf{z} \\
&\geq \int q(\mathbf{z}) \log \frac{p(\mathbf{x},\mathbf{z};\theta)}{q(\mathbf{z})} d\mathbf{z} \\
&= \int q(\mathbf{z}) \left[\log p(\mathbf{x};\theta) - \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x};\theta)} \right] d\mathbf{z} \\
&= \log p(\mathbf{x};\theta) - D_{KL}\left(q(\mathbf{z}) || p(\mathbf{z}|\mathbf{x};\theta)\right), \qquad (3)
\end{aligned}$$

where $q(\mathbf{z})$ is an arbitrary probability distribution (i.e. a function that is non-negative and integrates to 1 over the entire domain), and the inequality in the third line of the equation follows from Jensen's inequality. As the Kullback-Leibler (KL) divergence is non-negative, Equation (3) provides a lower bound for the likelihood of observed data, which is referred to as variational lower bound. In E-step, $q(\mathbf{z})$ is set to the empirical posterior $p(\mathbf{z}|\mathbf{x};\theta)$, fixing the KL divergence to 0; in M-step a new lower bound is constructed.

However, when computing the posterior probablity using Bayes' rule $p(\mathbf{z}|\mathbf{x};\theta) = p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z};\theta)/p(\mathbf{x};\theta)$, the integral of the marginal likelihood $p(\mathbf{x};\theta) = \int p(\mathbf{z};\theta)p(\mathbf{x}|\mathbf{z};\theta)d\mathbf{z}$ is often intractable in many applications, especially in Mahjong where the state complexity could be as high as 10^{145}, rendering EM algorithm impractical. To accommodate such high complexity, variational auto-encoder (VAE) approximates the posterior with a distribution $q(\mathbf{z};\phi)$ parameterized by ϕ (the variational parameters) instead of solving for it analytically. Moreover, to better infer $p(\mathbf{x})$, the distribution is usually dependent on $\mathbf{x}$, i.e. $q(\mathbf{z}|\mathbf{x};\phi)$.

Also, by expanding the KL divergence $D_{KL}\left(q(\mathbf{z}|\mathbf{x};\phi)||p(\mathbf{z}|\mathbf{x};\theta)\right)$ it can be proven that Equation (3) is equivalent to

$$E_{z\sim q_\phi}[\log p(\mathbf{x}|\mathbf{z};\theta)] - D_{KL}(q(\mathbf{z}|\mathbf{x};\phi)||p(\mathbf{z};\theta)), \qquad (4)$$

which is denoted as the negative loss function $-L(\theta,\phi;\mathbf{x})$.

For simplicity, we assume that both the encoder and the latent prior are Gaussian

$$q(\mathbf{z}|\mathbf{x};\phi) = \mathcal{N}(\mathbf{z}|\mu(\mathbf{x};\phi), \mathbf{\Sigma}(\mathbf{x};\phi)), \qquad (5)$$

$$p(\mathbf{z};\theta) = \mathcal{N}(\mathbf{z}|\mu_{\mathbf{z}}, \mathbf{\Sigma}_{\mathbf{z}}), \qquad (6)$$

which makes the second term in Equation (4) analytically tractable. As for the first term, $p(\mathbf{x}|\mathbf{z};\theta)$ is also a Gaussian, whose mean and variance are produced by the decoder. It than simplifies to l_2 loss $||\mathbf{x} - f(\mathbf{z})||^2$, i.e. the reconstruction error of the observed data. Correspondingly, the second term can be viewed as a regularization which pulls $q(\mathbf{z}|\mathbf{x};\phi)$ to $p(\mathbf{z};\theta)$.

However, theoretically VAE also faces the problem of intractability, as the expectation operation in Equation (4) is virtually impossible to perform when the decoder is a deep neural network. One

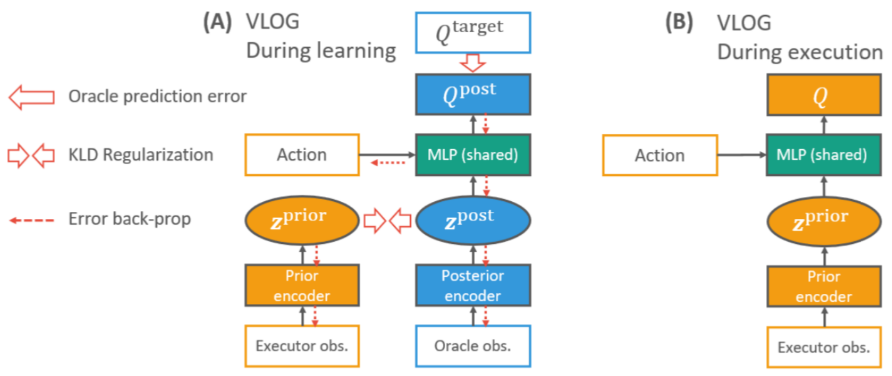


Figure 2: The architecture of variational oracle guiding framework.

naive idea to tackle this issue is to sample a $\mathbf{z}$ from q_ϕ to substitute the expectation, but that would cut off the gradient information related to the reconstruction loss from passing back to the encoder. For this, [13] proposed a reparameterization trick, where $\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\mu(\mathbf{x};\phi), \mathbf{\Sigma}(\mathbf{x};\phi))$ is expressed as $\mathbf{z} = \mu(\mathbf{x};\phi) + \sqrt{\mathbf{\Sigma}(\mathbf{x};\phi)} \cdot \epsilon, \epsilon \sim \mathcal{N}(\epsilon|0,1)$, and only the random noise ϵ is actually sampled in each training step, allowing the gradients to propagate back through $\mu(\mathbf{x};\phi)$ and $\mathbf{\Sigma}(\mathbf{x};\phi)$.

Since VAE proved to be a powerful tool for probability inference, improvements have been proposed for it. β-VAE [11] is one of the most prominent, which multiplies the second term in (4) by a hyperparameter β to control the latent information capacity. [9] further rids the model of this extra hyperparameter by introducing an auxiliary loss term

$$L_\beta = (D_{KL}^{tar} - D_{KL}(q(\mathbf{z}|\mathbf{x};\phi)||p(\mathbf{z};\theta))) \log \beta, \tag{7}$$

where D_{KL}^{tar} is a task-agnostic target KL divergence.

4 Variational Oracle Guiding

As Mahjong is a game with imperfect information, it is much harder to train a strong agent for Mahjong than chess or Go. Hence we borrow the idea of oracle guiding from [14, 9], and give the agent access to privileged information, namely other players' hand tiles, during training to learn a better model for predicting Q-values when being tested, i.e. using only the agent's private tiles and public information such as discarded tiles.

We encode the game state in the same way as [9], where the executor observation $\mathbf{x}$ has the shape of 93×34, and the oracle observation $\hat{\mathbf{x}} \in \mathbb{R}^{111\times 34}$. The first dimension contains binary information about a given type of tile, including the number of the tile the agent has in hand, whether it has been called or discarded by any player, whether it is dora or dora indicator, and whether it can be used for Chii, Pon or Kan. The 18 extra channels in oracle observation corresponds to information on the current tile with respect to other players' private tiles. And the second dimension corresponds to the 34 different types of tiles.

During training, the oracle observation is leveraged by the VAE framework as specified in Section 3, where the prior distribution of latent variable $\mathbf{z}$ is given by another encoder (“prior" encoder) taking the oracle observation as input:

$$p(\mathbf{z};\theta) = \mathcal{N}(\mathbf{z}|\mu(\hat{\mathbf{x}};\theta'), \mathbf{\Sigma}(\hat{\mathbf{x}};\theta')), \tag{8}$$

where θ' is the parameters of the posterior encoder (Figure 2). The encoders are one-dimensional convolution layers, as is common practice in Mahjong AI [14], and both the executor observation and oracle observation are encoded into 1088-dimension latent features.

We then apply a general-purpose DDQN algorithm with dueling architecture and parametric noise on the latent features to train an RL agent. As we mentioned in Section 2.1, DDQN decouples the selection and evaluation of actions by maintaining two copies of network parameters. To illustrate how this works, first consider the TD target in DQN (Equation (2)), and represent it explicitly as a function of the target network parameters θ':

$$y = r + \gamma \max_{a'} Q(s', a'; \theta'), \tag{9}$$

which is equivalent to

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta'); \theta'). \tag{10}$$

In DDQN, however, the next action to be evaluated is chosen by the other (online) network:

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta'). \tag{11}$$

In this way, we can prevent the model from overestimating the value of an action in a given state and falling into a vicious cycle.

And to better generalize the model across actions, we factor the last layer of our network into two branches to compute Q value:

$$Q(s, a; \alpha, \beta) = V(s; \beta) + \left(A(s, a; \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'; \alpha) \right), \tag{12}$$

where V, A stand for value branch, action advantage branch and β, α are their respective parameters. The advantage function is decentralized over the action space $\mathcal{A}$ to make sure that V and A can be identified uniquely.

On top of double Q-learning and dueling architecture, we also discard the exploration mode used by DQN, and replace it with parametric noise. Specifically, instead of drawing a random number beforehand to decide whether to generate a random action during training, we add noise to the parameters of the last layers, execute the forward pass through network to obtain the Q-values as usual, and choose an action greedily based on these (noisy) Q-values.

Consider either branch in (12). Before injecting parametric noise, the last linear layer is

$$\mathbf{q} = \mathbf{w}\mathbf{z} + \mathbf{b}, \tag{13}$$

where $\mathbf{z}$ is the output of the previous network layer. After injecting Gaussian noise, it is expressed as

$$\mathbf{q} = (\boldsymbol{\mu}^w + \boldsymbol{\sigma}^w \circ \epsilon^w)\mathbf{z} + \boldsymbol{\mu}^b + \boldsymbol{\sigma}^b \circ \epsilon^b, \tag{14}$$

where ϵ are random noise. A major advantage of parametric noise is that the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are learnable, allowing the model to automatically adjust exploration rate.

A visualization of dueling network architecture and parametric noise is shown in Figure 3, and the complete details of our network structure are presented in Appendix 7.1.

5 Experiments

We train our Mahjong agent with learning rate $lr = 1 \times 10^{-4}$, batch size $bs = 1024$, discount factor $\gamma = 1$, update interval $\tau = 1000$, and target KL divergence $D_{KL}^{tar} = 50$ for two million steps on human experts' playing data provided by [9]. The training was executed on an RTX 3090 and took about ten hours. The VLOG KL divergence and best action prediction accuracy during training are plotted in Figure 4. Note that these “best" actions refer to the actions chosen by professional human players. And as different players may have quite different playing styles, we do not aim to maximize the prediction accuracy here, but rather strive to learn a better estimation of Q-values across all the players.

For comparison, we also train three other agents. One is baseline, which is a general-purpose, task-agnostic RL agent. One is also trained with oracle guiding, but in the style of Suphx [14], where the oracle guiding is manually dropped from full to none over the training process. The other is trained by oracle policy distillation (OPD) [6], where a teacher is first trained using only oracle observation, and than the executor is trained to imitate the teacher's behaviours. All four models are trained with the same network architecture and RL hyperparameters.

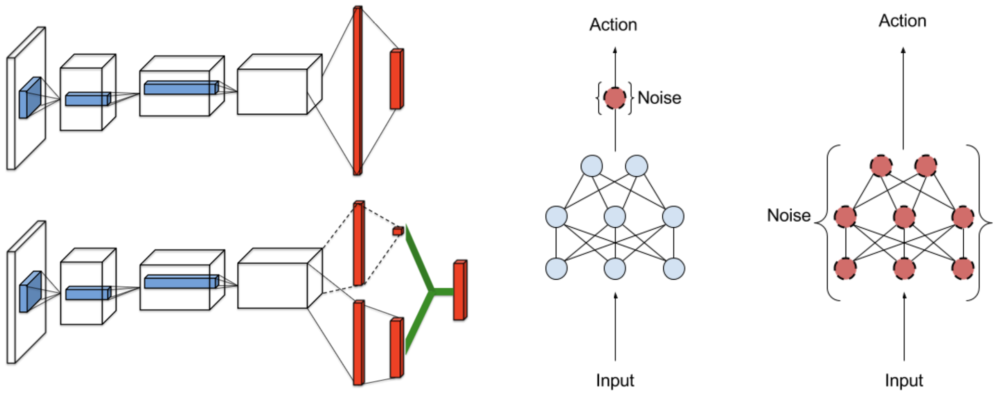


Figure 3: Comparison of (left) non-dueling and dueling network architecture, (right) non-parametric and parametric exploration noise.

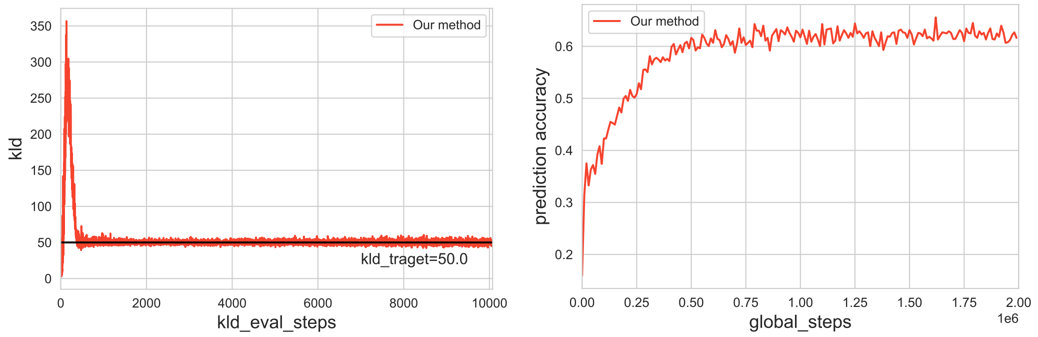


Figure 4: Training curves: (left) KL divergence between the oracle latent feature and executor latent feature; (right) prediction accuracy of the best action.

We test the four agents in 1000 matches, and the results are recorded in Table 1 and plotted in Figure 5. While baseline bests the other models in average payoff, VLOG agent has the highest top rate (ending a game with the highest score) and win rate (ending a game with a positive score), both by a considerable margin. As Mahjong is a highly stochastic game, the 4.1% top rate gap and 4.4% win rate gap between VLOG and Suphx, the former state-of-the-art Mahjong agent, indicate a significant leap in game skills. The payoff, on the other hand, is largely determined by luck, especially game opening, as the scoring system of Japanese Mahjong is quite complex with the different values of yaku and additional points of dora, explaining the near-zero average payoffs of all agents and their large standard deviations. Hence, we only consider the winning rates as metrics for evaluating the agents' Mahjong skills.

Table 1: Performance of VLOG, Suphx, OPD and baseline agents in 1000 matches.

	average payoff	top rate (%)	win rate (%)
VLOG	1.48±38.3	$\mathbf{37.1 \pm 48}$	$\mathbf{35.3 \pm 48}$
Suphx	-1.11±36.7	33.0±47	30.9±46
OPD	-2.72±36.6	29.0±45	27.1±44
baseline	$\mathbf{2.23 \pm 39.9}$	32.8±47	31.0±46

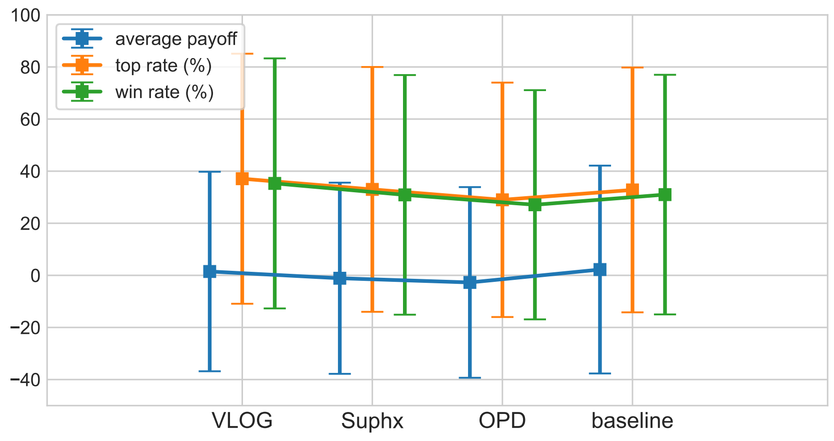


Figure 5: Performance of VLOG, Suphx, OPD and baseline agents in 1000 matches.

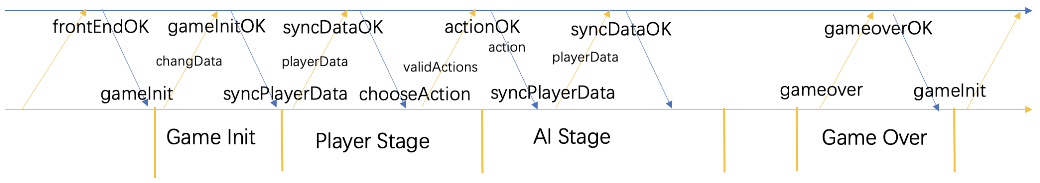


Figure 6: Control flow of our Mahjong game frontend.

We also note that the Suphx-styled model we use here is not a fully functional reproduction of the Suphx Mahjong AI proposed by [14], as the later use Mahjong-specific heuristic techniques other than oracle guiding, including global reward prediction and run-time policy adaptation. Rather, we only apply oracle guiding to the baseline in pursuit of a general-purpose game agent, explaining the fact that our Suphx model only performs a little better than baseline, which also suggests the significance of our variational learning framework.

Furthermore, we implement a Mahjong frontend for human tests (Figure 6), and integrate our Mahjong agent into Mahjong Soul [1], one of the most popular online Mahjong platform (Figure 7). Experiments show that our agent achieves an average human-level performance, and ranks around Adept level one on Mahjong Soul.[2] Figure 8 is a screenshot of our agent's earlier game records when still at Novice level, and video clips of our agent playing games both locally and on Mahjong Soul are presented in the supplemental material and at `https://jbox.sjtu.edu.cn/l/e1XQ7P`.

Although our VLOG agent bests Suphx agent trained with the same network architecture and RL algorithm, it does not perform that well in the real games. We believe that is a result of the high complexity of Mahjong games and the concomitant low training speed, which limited our network size and total number of training steps. Also, with limited time we only trained our agents with human expert data, and did not execute online self-play. We leave it to future works to explore these possibilities.

[1]`https://www.maj-soul.com/`

[2]The ranking system of Mahjong Soul is (from low to high) Novice-Adept-Expert-Master-Saint-Celestial. Each rank except Celestial has three levels.

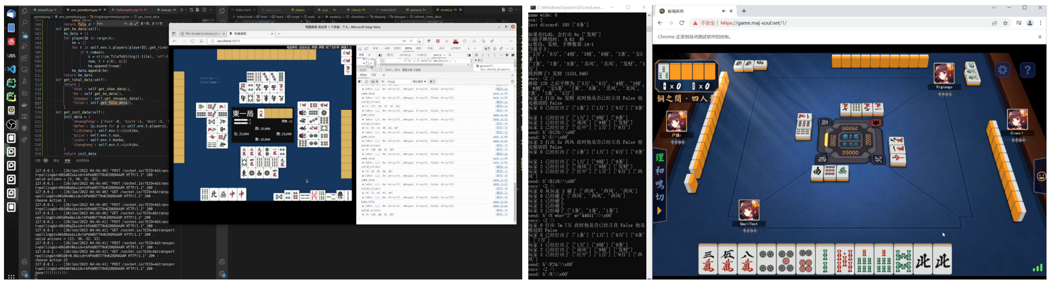
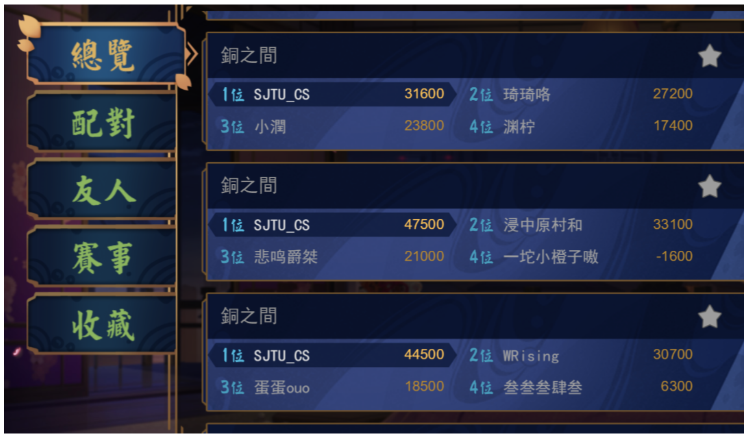


Figure 7: (left) Our Mahjong game in action. (right) Our Mahjong agent playing on Mahjong Soul.



銅之間
1位 SJTU_CS 31600 2位 琦琦咯 27200
3位 小潤 23800 4位 渊柠 17400

銅之間
1位 SJTU_CS 47500 2位 浸中原村和 33100
3位 悲鸣爵桀 21000 4位 一坨小橙子啵 -1600

銅之間
1位 SJTU_CS 44500 2位 WRising 30700
3位 蛋蛋ouo 18500 4位 叁叁叁肆叁 6300

Figure 8: A screenshot of several games by our agent on Mahjong Soul at Novice level.

6 Conclusion

In this work, we implement a DQN-based RL agent for the game Japanese Mahjong along with a frontend for playing with the agent either locally or online. The agent is trained with oracle guiding under variational Bayesian learning framework, and with improvements in RL algorithms including double Q-learning, dueling architecture, and parametric noise. Tests show that, when trained with the same number of steps, our agent performs better than both Suphx and OPD, the previous state-of-the-art Mahjong agents, and achieves a modest human-level performance.

References

[1] Stockfish: Strong open source chess engine. `https://stockfishchess.org/` accessed 22 June 2022.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[3] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Commun. ACM*, 60(11):81–88, 2017.

[4] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[5] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[6] Yuchen Fang, Kan Ren, Weiqing Liu, Dong Zhou, Weinan Zhang, Jiang Bian, Yong Yu, and Tie-Yan Liu. Universal trading for order execution with oracle policy distillation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 107–115. AAAI Press, 2021.

[7] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[8] William H. Guss, Cayden Codel, Katja Hofmann, Brandon Houghton, Noburu Kuno, Stephanie Milani, Sharada P. Mohanty, Diego Perez Liebana, Ruslan Salakhutdinov, Nicholay Topin, Manuela Veloso, and Phillip Wang. The minerl competition on sample efficient reinforcement learning using human priors. *CoRR*, abs/1904.10079, 2019.

[9] Dongqi Han, Tadashi Kozuno, Xufang Luo, Zhao-Yun Chen, Kenji Doya, Yuqing Yang, and Dongsheng Li. Variational oracle guiding for reinforcement learning. In *ICLR 2022*, April 2022.

[10] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3215–3222. AAAI Press, 2018.

[11] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[12] Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[13] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[14] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *CoRR*, abs/2003.13590, 2020.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[16] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[17] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot,

Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

[20] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2094–2100. AAAI Press, 2016.

[21] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019.

[22] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1995–2003. JMLR.org, 2016.

7 Appendix

7.1 Network Structure

Executor encoder:

```
Sequential(
    (0): Conv1d(93, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (1): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (2): ReLU()
    (3): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (4): ReLU()
    (5): Conv1d(64, 32, kernel_size=(3,), stride=(1,), padding=(1,))
    (6): ReLU()
    (7): Flatten(start_dim=1, end_dim=-1)
    (8): Linear(in_features=1088, out_features=1024, bias=True)
    (9): ReLU()
  )
```

Oracle encoder:

```
Sequential(
    (0): Conv1d(111, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (1): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (2): ReLU()
    (3): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (4): ReLU()
    (5): Conv1d(64, 32, kernel_size=(3,), stride=(1,), padding=(1,))
    (6): ReLU()
    (7): Flatten(start_dim=1, end_dim=-1)
    (8): Linear(in_features=1088, out_features=1024, bias=True)
    (9): ReLU()
)
```

h2logsig:

```
Sequential(
  (0): Linear(in_features=1024, out_features=512, bias=True)
  (1): MinusOneModule()
)
```

h2mu:

```
Linear(in_features=1024, out_features=512, bias=True)
```

s2q:

```
DiscreteActionQNetwork(
  (main_network): Sequential(
    (0): Linear(in_features=512, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): ReLU()
  )
  (value_layer): Linear(in_features=1024, out_features=1, bias=True)
  (advantage_layer): Linear(in_features=1024, out_features=47, bias=True)
)
```