VARIATIONAL ORACLE GUIDING FOR REINFORCEMENT LEARNING

Dongqi Han[*1], **Tadashi Kozuno**[2], **Xufang Luo**[3], **Zhaoyun Chen**[4],
Kenji Doya[1], **Yuqing Yang**[3], and **Dongsheng Li**[3]

[1]*Okinawa Institute of Science and Technology*
[2]*University of Alberta*
[3]*Microsoft Research Asia*
[4]*Institute of Artificial Intelligence, Hefei Comprehensive National Science Center*

ABSTRACT

How to make intelligent decisions is a central problem in machine learning and artificial intelligence. Despite recent successes of deep reinforcement learning (RL) in various decision making problems, an important but under-explored aspect is how to leverage oracle observation (the information that is invisible during online decision making, but is available during offline training) to facilitate learning. For example, human experts will look at the replay after a Poker game, in which they can check the opponents' hands to improve their estimation of the opponents' hands from the visible information during playing. In this work, we study such problems based on Bayesian theory and derive an objective to leverage oracle observation in RL using variational methods. Our key contribution is to propose a general learning framework referred to as variational latent oracle guiding (VLOG) for DRL. VLOG is featured with preferable properties such as its robust and promising performance and its versatility to incorporate with any value-based DRL algorithm. We empirically demonstrate the effectiveness of VLOG in online and offline RL domains with tasks ranging from video games to a challenging tile-based game Mahjong. Furthermore, we publish the Mahjong environment and an offline RL dataset as a benchmark to facilitate future research on oracle guiding[1].

1 INTRODUCTION

Deep reinforcement learning (DRL) has undergone rapid development in recent years (Sutton & Barto, 2018; Mnih et al., 2015; Vinyals et al., 2019). However, there is a common and important but under-explored aspect in RL: imagine that after playing a Poker game, a human player may look at the replay to check opponents' hands and analyze this information to improve his/her playing strategy (or *policy*) for the next time. We refer to the information like opponents' hands as *oracle observation*, defined as the information invisible to the agent during online task execution but available during offline training. By contrast, the information available during task execution is called *executor observation*. Such a scenario has been referred to as *oracle guiding* for RL (Li et al., 2020; Fang et al., 2021) (see Sec. 3 for a formal definition). Oracle guiding is common in real life. For example, when taking an examination (the oracle observation is the answers to similar questions, which are available only during preparing); and training a robot to perform some tasks on the Moon (when training the robot, we can provide it with the information of the territory, which are not available during execution). The type of oracle observation can be diverse, including hindsight information (Harutyunyan et al., 2019; Guez et al., 2020), human feedback (Knox & Stone, 2009; Loftin et al., 2016; MacGlashan et al., 2017), re-calibrated data with post-processing, and hidden states in a partially observed setting (Li et al., 2020).

While humans naturally perform oracle guiding when learning to make decisions, it remains challenging in RL. The difficulties include: (1) how to guarantee that learning with oracle observation

[*]Work done during an internship in Microsoft Research Asia. Email: dongqi.han@oist.jp
[1]https://github.com/Agony5757/mahjong

improves the main decision model using executor observation only, and (2) if introducing an auxiliary loss leveraging oracle observation, how to tradeoff between the main loss and auxiliary loss. While recent studies attempted to model oracle guiding in RL (Guez et al., 2020; Li et al., 2020; Fang et al., 2021), none of them addressed these difficulties (refer to the Related Work section for more details). In particular, all these proposed methods are heuristic: although empirical results showed performance gain with oracle guiding, it is not theoretically guaranteed that the usage of oracle observation improves execution performance.

In this paper, we propose a fundamentally new idea for oracle guiding based on Bayesian theory. Taking Poker as an example, we know that the learning of optimal strategy is tractable if knowing the global, true state of the environment (or simply *state*[2]), including all visible or invisible cards, the opponents' playing style, etc. (Azar et al., 2017; Jin et al., 2018; 2020). A key part of skill improvement is learning to estimate the probabilistic distribution of environmental state from executor observation. The common way to do this by human experts is to watch match replays where the oracle observation (e.g. opponents' hands) is available, and then use the oracle-estimated state to correct the executor-estimated state. We interpret this to Bayesian language: executor-estimated state as the prior distribution, and the oracle-estimated one as the posterior distribution. Thus, the training objective can be considered two-fold: learning to make decisions based on posterior estimation of state, and learning a prior distribution of state closer to the posterior one.

We formulate this idea by proposing a novel learning framework for general oracle guiding problems based on variational Bayes (VB) (Kingma & Welling, 2014), referred to as variational latent oracle guiding (VLOG). VLOG owns several preferable properties. First, VLOG is theoretically guaranteed to leverage oracle observation for improving the decision model using executor observation. Second, VLOG is a versatile DRL framework that can be integrated into any value-based RL algorithm and is agnostic to the type of oracle observation. Third, VLOG does not suffer from the necessity of tuning additional hyper-parameters. Finally, we empirically show that VLOG contributes to better performance in a variety of decision-making tasks in both online and offline RL domains. The tasks include a simple maze navigation, video games playing, and a particularly challenging tile-based game Mahjong in which humans heavily leverage oracle observation in learning (Li et al., 2020). We also contribute to the community by taking Mahjong as a benchmarking task for oracle guiding and publishing the RL environment and dataset to facilitate future research.

2 Related work

In the past few years, research interests have grown on DRL and imitating learning (Chen et al., 2020) with leveraging oracle or hindsight information. For DRL, Guez et al. (2020); Fang et al. (2021) considered hindsight observation (executor observation at future steps) as the oracle observation during training. Guez et al. (2020) used hindsight observation to facilitate learning a representation of current state. Another method (Fang et al., 2021) was used for stock trading: the authors trained a teacher (oracle) policy with hindsight information, and employed network distillation to make the student policy behaves more similar to the teacher policy. Both the methods (Guez et al., 2020; Fang et al., 2021) are heuristic and focused on making use of future observation for a better sequential modeling, while VLOG is theoretically guaranteed for any kind of oracle observation. For applications in imperfect-information games, Suphx (Li et al., 2020), a DRL-based AI for Mahjong, also introduced a method to leverage oracle observation (opponents' hand) for stronger performance. They concatenate oracle observation with executor observation as the input of policy network, where the oracle observation is timed by a scalar variable which is annealed from 1 to 0 during the training course. However, the method used in Li et al. (2020) is also heuristic and has only been tested in one task.

Variational Bayes (VB) is a well-established method and has been taken advantage of in RL. For example, control as probabilistic inference uses VB to connect the objective function of RL and variational lower bound of a probabilistic inference problem (Furmston & Barber, 2010; Weber et al., 2015; Levine, 2018). Our idea differs since it frames a value regression problem by the maximum-likelihood problem, and then, it applies VB to solve it (see Sec. 4). Also, the usage of VBian network models for DRL has captured attention from researchers recently. For example, Ha & Schmidhuber (2018) proposed to employ a VAE to reduce the high dimensionality of image observation; Igl et al.

[2]Generally, oracle observation does not necessarily contain all the information of environmental state.

(2018); Han et al. (2020); Lee et al. (2020) proposed variational RNNs as the state-transition models to encode the belief states of the agent; Yin et al. (2021) utilized a variational sequential generative model for predicting future observation and used the prediction error to infer intrinsic reward to encourage exploration; and Okada et al. (2020) demonstrated performance gain by using a deep Bayesian planning model in continuous control tasks. Our study differs from the mentioned works by focusing on oracle guiding, and VLOG does not involve learning a state transition model.

3 ORACLE GUIDING POMDP

Here we define the problem based on the Partially Observable Markov decision process (POMDP) (Sondik, 1978). An oracle guiding POMDP distinguishes from the original POMDP by having two types of observations: executor and oracle. The executor observation $\boldsymbol{x}$ is always available to the agent, on which the agent's decision making (execution) relies. The oracle observation $\hat{\boldsymbol{x}}$ is not accessible during execution, but can be obtained afterward. $\boldsymbol{x}$ is included in $\hat{\boldsymbol{x}}$ since the former is always available. Thus $\hat{\boldsymbol{x}}$ contains no less information of the underlying environment state than $\boldsymbol{x}$.

A formal definition of oracle guiding POMDP is a tuple $\langle \boldsymbol{S}, \boldsymbol{A}, \mathcal{P}_0, \mathcal{T}, \boldsymbol{X}, \hat{\boldsymbol{X}}, \mathcal{O}, \hat{\mathcal{O}}, \gamma \rangle$, where $\boldsymbol{S}$ and $\boldsymbol{A}$ are the state and action spaces, respectively. $\mathcal{P}_0$ specifies the initial state distribution such that $\mathcal{P}_0(\boldsymbol{s})$ is the probability of a state $\boldsymbol{s} \in \boldsymbol{S}$ being an initial state. $\mathcal{T}$ specifies the state transition probability such that $\mathcal{T}(\boldsymbol{s}', r|\boldsymbol{s}, a)$ is the probability of reaching to a new state $\boldsymbol{s}' \in \boldsymbol{S}$ with an immediate reward $r \in \mathbb{R}$ after taking an action $a \in \mathcal{A}$ at a state $\boldsymbol{s} \in \boldsymbol{S}$. $\boldsymbol{X}$ denotes the executor observation space and $\hat{\boldsymbol{X}}$ denotes the oracle observation space. $\mathcal{O}$ specifies the executor observation probability such that $\mathcal{O}(\boldsymbol{x}|\boldsymbol{s})$ is the probability of an executor observation $\boldsymbol{x} \in \boldsymbol{X}$ at a state $\boldsymbol{s} \in \boldsymbol{S}$. Similarly, $\hat{\mathcal{O}}$ specifies the oracle observation probability such that $\hat{\mathcal{O}}(\boldsymbol{x}|\boldsymbol{s})$ is the probability of an oracle observation $\hat{\boldsymbol{x}} \in \hat{\boldsymbol{X}}$ at a state $\boldsymbol{s} \in \boldsymbol{S}$. $\gamma \in [0, 1)$ is the discount factor. Value functions are the expected value of return from a particular state or state-action pair. For a policy π, its Q-function is defined by $q^\pi(\boldsymbol{s}, a) \coloneqq \mathbb{E}_\pi[\sum_{n=0}^{\infty} \gamma^n r_{t+n} | \boldsymbol{s}_t = \boldsymbol{s}, a_t = a]$, where $\mathbb{E}_\pi$ indicates the expectation when the policy π is followed. The state value function v^π is defined by $v^\pi(\boldsymbol{s}) \coloneqq \mathbb{E}_{\pi(a|\boldsymbol{s})}[q^\pi(\boldsymbol{s}, a)]$. The agent aims to maximize $\mathbb{E}_{\mathcal{P}_0(\boldsymbol{s})} v^\pi(\boldsymbol{s})$ with respect to π, and the value-functions play an important role to this end (Sutton & Barto, 2018).

4 VLOG: VARATIONAL LATENT ORACLE GUIDING

Let us introduce a latent vector $\boldsymbol{z}_t$, which is a probabilistic variable representing the environmental state $\boldsymbol{s}_t$. From a Bayesian perspective, we consider the prior distribution $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ as the agent's estimated probability density function (PDF) for $\boldsymbol{z}_t$ based on executor observation $\boldsymbol{x}_t$. Meanwhile, the posterior distribution PDF $q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)$ is modeled based on oracle observation $\hat{\boldsymbol{x}}_t$.

In RL, the most basic requirement is to make a good estimation of return by a value function approximator $v(\boldsymbol{x}_t) \coloneqq \int v(\boldsymbol{z}_t) p(\boldsymbol{z}_t|\boldsymbol{x}_t) d\boldsymbol{z}_t$ (we denote it by v, but it can also be a Q function by simply replacing $\boldsymbol{x}_t$ with $(\boldsymbol{x}_t, \boldsymbol{a}_t)$) based on available information, i.e. executor observation $\boldsymbol{x}_t$. The target of return, denoted by v_t^{tar}, can be estimated by any value learning algorithm such as TD(0) (Sutton & Barto, 2018), Peng's Q(λ) (Kozuno et al., 2021), etc. (generally, v^{tar} can always be given by Bellman equations. However, one can also use Monte-Carlo return as v^{tar} if available). In particular, we employed double Q-learning with dueling architecture (Wang et al., 2016) to compute v^{tar} due to its effectiveness and simplicity (Sec. 5 and B.1). We want to maximize the log-likelihood objective of **the estimation of return based on executor observation** $\boldsymbol{x}_t$ (i.e., for the executor model)

$$\begin{aligned}\mathcal{L} &\coloneqq \log \mathcal{P}\left(v(\boldsymbol{x}_t) = v_t^{tar} | \boldsymbol{x}_t\right) = \log \int_{\boldsymbol{z}_t} \mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar} | \boldsymbol{z}_t\right) p(\boldsymbol{z}_t|\boldsymbol{x}_t) d\boldsymbol{z}_t \\ &= \log \int_{\boldsymbol{z}_t} \frac{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)}{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)} \mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar} | \boldsymbol{z}_t\right) p(\boldsymbol{z}_t|\boldsymbol{x}_t) d\boldsymbol{z}_t.\end{aligned}$$

By Jensen's inequality, we have

$$\begin{aligned}\mathcal{L} &\geq \int_{\boldsymbol{z}_t} q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t) \log \frac{\mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar}|\boldsymbol{z}_t\right) p(\boldsymbol{z}_t|\boldsymbol{x}_t)}{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)} d\boldsymbol{z}_t \\ &= \int_{\boldsymbol{z}_t} \left[q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t) \log \mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar}|\boldsymbol{z}_t\right) - q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t) \log \frac{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)}{p(\boldsymbol{z}_t|\boldsymbol{x}_t)} \right] d\boldsymbol{z}_t \\ &= \underbrace{\mathbb{E}_{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)}\left[\log \mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar}|\boldsymbol{z}_t\right)\right]}_{\text{oracle prediction error}} - \underbrace{D_{KL}\left(q\left(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t\right) \| p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right)\right)}_{\text{regularization term}} := \mathcal{L}_{\text{VLOG}}. \end{aligned} \tag{1}$$

Thus we can maximize our initial objective $\mathcal{L}$ via maximizing $\mathcal{L}_{\text{VLOG}}$, which is also known as the variational lower bound (Kingma & Welling, 2014), but in our oracle-guiding scheme. Since $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ and $q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)$ represents the PDF of the latent vector obtained from executor observation and oracle observation, respectively, the meanings of the two terms in $\mathcal{L}_{\text{VLOG}}$ appear clear now: the first term, i.e., oracle prediction error, helps to improve value estimation from posterior latent state distribution ($\boldsymbol{z}_t$ computed with oracle observation); and the second term, i.e., regularization term, helps to shape the prior representation of $\boldsymbol{z}_t$ closer to the posterior one as latent oracle guiding. We would like to highlight that the VLOG objective is the lower bound of the objective for the prior executor model $v(\boldsymbol{x}_t)$ (the estimation of return using executor observation $\boldsymbol{x}_t$) with the usage of oracle observation $\hat{\boldsymbol{x}}_t$. This lower bound guarantees that the usage of oracle observation facilitates the learning of the executor model, which is our original motivation.

Remark 1. *One may use any shape of the approximate posterior q, depending on which different instance of VLOG is possible. Furthermore, one may directly use $v(\boldsymbol{x}_t)$ instead of $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$. These design choices allow users to incorporate any prior knowledge on oracle observation. For example, if one knows the range of a state-value at $\boldsymbol{x}_t$ is a closed interval $[l, u]$, the approximate posterior $q(v_t|\hat{\boldsymbol{x}}_t)$ can be restricted to a family of probability distributions supported on $[l, u]$.*

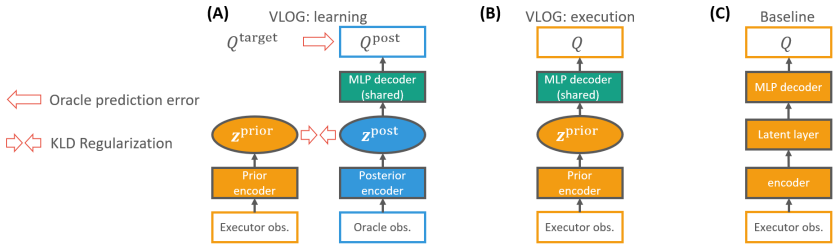

Figure 1: Implementation of VLOG for deep Q-learning during (A) learning and (B) execution. Executor observation $\boldsymbol{x}_t$ and oracle observation $\hat{\boldsymbol{x}}_t$ use two different encoders but share one decoder (MLP for estimating Q function). (C) Baseline model without using oracle observation. A rectangle means a deterministic layer/variable, and an ellipse means a stochastic one. Here the outputs are multi-head Q-values corresponding to discrete actions.

4.1 Implementation with neural networks

Inspired by the implementation of variational auto-encoder (VAE, Kingma & Welling (2014)), we propose the neural network architecture of VLOG (Fig. 1). The executor observation $\boldsymbol{x}_t$ and oracle observation $\hat{\boldsymbol{x}}_t$ are processed by two distinct encoder networks to compute the prior and posterior distribution of the latent vector, respectively. During training, both $\boldsymbol{x}_t$ and $\hat{\boldsymbol{x}}_t$ are available, and all the network parameters are updated by maximizing the VLOG objective in an end-to-end manner (Fig. 1A). During execution, the agent computes prior distribution $p(\boldsymbol{z}|\boldsymbol{x}_t)$ for decision making (Fig. 1B) without using oracle observation. $\boldsymbol{z}_t$ is computed by parameterized normal distribution

$$\begin{aligned} p(\boldsymbol{z}_t|\boldsymbol{x}_t) &= \mathcal{N}(\boldsymbol{\mu}_t^p, \exp\left(\log \boldsymbol{\sigma}_t^p\right)), \quad (\boldsymbol{\mu}_t^p, \log \boldsymbol{\sigma}_t^p) = \text{prior encoder}(\boldsymbol{x}_t), \\ q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t) &= \mathcal{N}(\boldsymbol{\mu}_t^q, \exp\left(\log \boldsymbol{\sigma}_t^q\right)), \quad (\boldsymbol{\mu}_t^q, \log \boldsymbol{\sigma}_t^q) = \text{posterior encoder}(\hat{\boldsymbol{x}}_t). \end{aligned}$$

For computing $\mathcal{P}(v(\boldsymbol{z}_t) = v_t^{tar}|\boldsymbol{z}_t)$ in Eq. 1, we simply assume it follows normal distribution, and estimate it with mean square error between $v(\boldsymbol{z}_t)$ and v_t^{tar} in practice. The reparameterization trick is used to perform end-to-end training as in VAE (Kingma & Welling, 2014). Then the output of the decoder (value function) can be obtained by $v(\boldsymbol{z}_t) = \text{decoder}(\boldsymbol{z}_t)$. Note that $\boldsymbol{z}_t$ is obtained using the posterior encoder during training, and using the prior encoder during execution (Fig. 1A, B).

4.2 TASK-AGNOSTIC LATENT BOTTLENECK CONTROL

To learn a better representation, we borrow the idea from β-VAE (Higgins et al., 2016) to multiply a coefficient β to the regularization term. Thus we have the loss function (negative lower bound) as

$$\mathcal{J}^{\beta}_{\text{VLOG}} = -\mathbb{E}_{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)}\left[\log \mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar}|\boldsymbol{z}_t\right)\right] + \beta D_{KL}\left(q\left(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t\right)\|p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right)\right). \tag{2}$$

The hyper-parameter β controls the capacity of the latent information bottleneck (Tishby & Zaslavsky, 2015; Alemi et al., 2017). We found the choice of β is important for the performance of VLOG in RL (see Appendix B.3). However, having extra hyper-parameters are not desired. Inspired by the method used in Burgess et al. (2017) for controlling the scale of KL divergence in β-VAE, we propose a task-agnostic method to automatically adjust β by setting a target of KL divergence D_{KL}^{tar}. In particular, we minimize the auxiliary loss function (β as the being optimized parameter)

$$\mathcal{J}_{\beta} = \left(\log_{10} D_{KL}^{tar} - \log_{10} D_{KL}\left(q\left(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t\right)\|p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right)\right)\right)\log(\beta). \tag{3}$$

The intuition here is to strengthen the regularization by increasing β when the divergence between prior and posterior is too large, and vice versa. This method is similar to that used in soft actor-critic for automatically adjusting the entropy coefficient (Haarnoja et al., 2019), while we used this for KL divergence coefficient. Importantly, we found a well-performing value $D_{KL}^{tar} = 50$ agnostic to other design choices. It worked well in a range of different tasks and networks (Sec. 5). Therefore, we do not need to tune β. We provide more discussion about this method in Appendix D.

5 EXPERIMENTS

How does VLOG perform in practice? We investigated the empirical performance of VLOG in three types of tasks using online or offline RL, from simple to difficult. In the following experiments, we used double DQN with dueling network architecture (van Hasselt et al., 2016; Wang et al., 2016) as the base RL algorithm, the model and loss functions for RL are defined in Appendix. B.1. As DRL is susceptible to the choice of hyper-parameters, introducing any new hyper-parameters might obscure the effect of oracle guiding. Double DQN and dueling architecture are preferable for the base algorithm since they require no additional hyper-parameters, in contrast to other DQN variants (Hessel et al., 2018), such as prioritized experience replay (Schaul et al., 2016), noisy network (Fortunato et al., 2018), categorical DQN (Bellemare et al., 2017), and distributed RL (Kapturowski et al., 2018). Importantly, we used the same hyper-parameter setting for all methods and environments as much as possible (see Appendix B.2).

5.1 MAZE

We first demonstrate how VLOG helps to shape the latent representation by leveraging oracle observation in learning. The testbed is a maze navigation task[3] (Fig. 2A) with 10×10 grids. The executor observation is the (x, y) position, where x, y are continuous values randomly sampled within each grid of the maze (thus the observations in two adjacent but wall-separated grids may be very close). At each step, the agent selects an action (going up, down, right, or left) and moves to another grid if not blocked by wall. We provided the VLOG agent with oracle observation (x_c, y_c, d_g) during training, where x_c, y_c are coordinates of the center of the current grid, and d_g is the (shortest) path distance to goal from the current grid. It is intuitive that although the raw observation is (x, y), d_g matters more in a maze navigation task. We empirically investigated how much such oracle observation used in learning of VLOG could help to shape the latent representation of $\boldsymbol{z}$ w.r.t. d_g rather than position. The encoder and decoder were both 2-layers multi-layer perceptrons (MLP) with width 256 and ReLU activation. The size of latent vector $\boldsymbol{z}_t$ for VLOG was 128 since we computed both $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ (Appendix C).

Experiments show that the baseline agent struggled to reach the goal (Fig. 2B), while VLOG agents stably solved the task after learning. To check how the usage of VLOG affected the learned latent representation, we visualize the latent representation of both VLOG and baseline models with principal component analysis (PCA, Pearson F.R.S. (1901)). In Fig. 2C, we map path distance to goal d_g to color and plot the scores of the first 2 PCs of $\boldsymbol{z}$ (computed using executor observation)

[3]https://github.com/MattChanTK/gym-maze

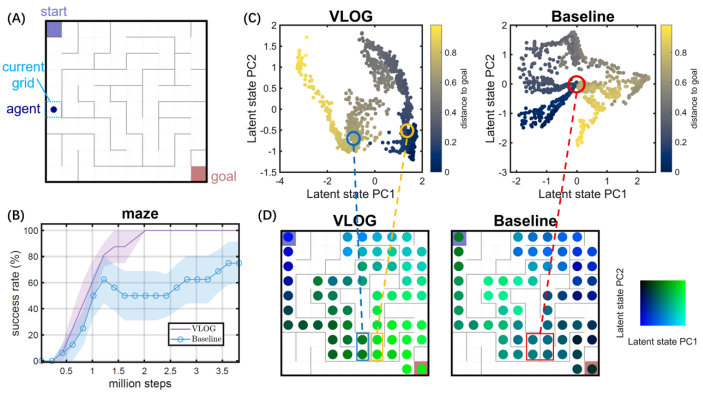

Figure 2: Learning latent representations by oracle guiding in a simple maze. (A) Illustration of the task. (B) Success rate of using VLOG and not using VLOG (baseline), each using 8 random seeds. (C) PCA of $\boldsymbol{z}$ in VLOG and the corresponding hidden state in baseline model ($\boldsymbol{z}$ collected during execution, not using oracle observation). Color indicates normalized path distance to goal. The latent state of VLOG, trained with oracle observation, showed a more sequential and interpretable representation w.r.t. path distance to goal. We circled the latent regions corresponding to some places in the maze that the $(x,\ y)$ positions are close but distances to goal distinguish. (D) The learned latent representation w.r.t position in the maze, where the green and black component of the color corresponds to the score of the first 2 PCs of the latent state, respectively, as shown by the colormap (similar colors indicate similar representations).

for VLOG and the corresponding latent state for baseline using successful trials ("latent layer" in Fig. 1C). The latent state of VLOG showed a relatively smoother and more interpretable representation of distance-to-goal comparing to that of baseline. We then plot the latent representations for different positions in the maze in Fig. 2D. The latent state of VLOG more clearly represented d_g, consistent with the result (Fig. 2C). In particular, we looked into a rectangle region (denoted by rectangles in Fig. 2D) inside which the left 2 grids and right 2 grids are segregated by a wall. We found the corresponding areas in the latent PC space and circled them in Fig. 2C. While these 4 grids are close in $(x,\ y)$ (executor observation), their distances-to-goal (oracle observation) are highly distinguished. By leveraging oracle guiding using VLOG, the agents can clearly differentiate the left 2 grids and the right 2 grids in latent space as shown in Fig. 2C, D, left (note that the latent state $\boldsymbol{z}$ here of VLOG was computed using executor observation only). By contrast, the latent representations of these grids were overlapped for the baseline model, which did not utilize the oracle observation (Fig. 2C, D, right). In sum, we demonstrated with a toy example that VLOG effectively helped the latent space to couple with oracle state useful for the task. The following sections will transfer to experiments on more complicated tasks and discuss how VLOG can improve practical performance.

5.2 Noisy MinAtar

To evaluate how VLOG scales with more high-dimensional state space, we tested it on a set of MinAtar videos games. MinAtar (Young & Tian, 2019) is a test platform for AI agents, which implements 5 miniaturized Atari 2600 games with discrete actions (Seaquest, Breakout, Space Invaders, Freeway and Asterix). MinAtar is inspired by Arcade Learning Environment (Bellemare et al., 2013) but simplifies the environments for efficiency. The observation is 10×10 pixels with multiple channels indicating different objects. In real-world, the observation usually contains some noise. Thus it is natural to consider the noisy observation as the partially-observable executor observation, and the original, non-noisy observation as the oracle one. Suppose that at each frame, each pixel may "break" randomly with an independent probability of 1/8 (Fig. 3A). The original observation at a broken pixel is erased and is replaced by a different value in all channels. We consider such noisy MinAtar environments with the noisy pixels as the executor observation and the original pixels as the oracle observation.

The network structure was the same as that for Maze, but the encoder was replaced by a CNN (Appendix C). We ran experiments on all the 5 environments of MinAtar with VLOG as well as

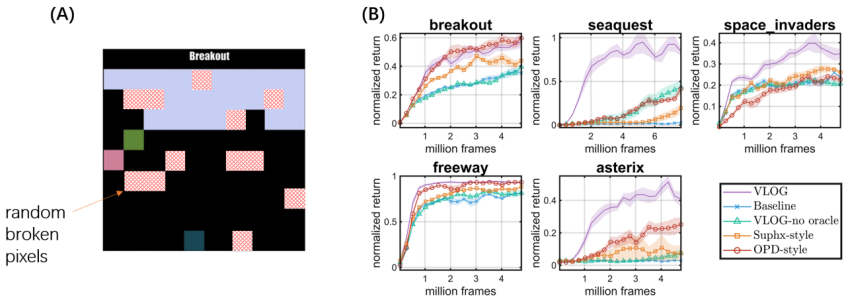

Figure 3: (A) Illustration of the MinAtar environments with randomly broken pixels. (B) Learning curves showing normalized average return (divided by the average return of trained oracle model) of VLOG and alternative methods each using 8 random seeds.

baseline, oracle and alternative oracle guiding methods (see Appendix A for details). **Baseline** model always uses executor observation as network input (Fig. 1C). **Oracle** is the same as baseline except that it always receives oracle observation (i.e., cheating, we ran the experiments of oracle for reference). **VLOG-no oracle** is an ablation of VLOG where we use the executor observation as the input to posterior encoder in VLOG (oracle observation is not used). **Suphx-style** oracle guiding is the oracle guiding method used in the Mahjong AI *Suphx* (Li et al., 2020), in which the executor observation and the dropout-ed oracle observation (with dropout probability $p^{dropout}$) are concatenated as the input to the network. With training proceeds, $p^{dropout}$ is gradually increased from 0 to 1, thus the trained network does not need oracle observation for input (Appendix A). **OPD-style** oracle guiding is the oracle guiding method used in oracle policy distillation (OPD) (Fang et al., 2021). OPD-style oracle guiding first trains a teacher model using oracle observation as input, and then trains the executor model using an auxiliary loss, which is the error between the executor and the teacher models' estimation for value function (Appendix A).

The results show that oracle performed usually the best, as expected. We normalized the performance of non-oracle models using oracle model as reference, for more clear comparison (Fig. 3B). Among all the oracle guiding methods (VLOG, OPD-style and Suphx-style), VLOG consistently performed the best. It is notable that VLOG and VLOG-no oracle performed surprisingly well in Seaquest. This can be explained by that Seaquest is a task with a local optimum (see Appendix E), while the stochasticity in hidden states of VLOG helped exploration in latent space to escape from the local optimum (a similar idea is Fortunato et al. (2018), but their noise was added to the network weights). Except in Seaquest, VLOG-no oracle did not show significant performance different with baseline, showing that the performance gain of VLOG in this task set mainly came from leveraging oracle observation for shaping latent distribution; and the usage of variational Bayesian model was, at least not harming the performance when there is no helpful oracle information.

5.3 Offline learning on Mahjong

Mahjong is a popular tile-based game with hundreds of millions of players worldwide (here we consider the Japanese variant). The game is like many other card games (but using tiles instead of cards), in which multiple (usually four) players draw and discard tiles (totally 136 tiles) alternatively to satisfy winning conditions. It is a highly challenging game characterized with (1) imperfect information in executor observation (a player cannot see opponents' private tiles and the remaining tiles to be drawn), (2) stochastic state transitions as in many card games, and (3) extremely high game complexity (i.e., the number of distinguished, legal game states). Complexity of Mahjong is much larger than 10^{166} (Appendix F.1). For reference, complexity of Go is $\sim 10^{172}$ (Silver et al., 2016) and complexity of no-limit Poker is $\sim 10^{162}$ (Johanson, 2013).

In Mahjong, it is hard to make optimal decisions based on executor observation because the outcomes heavily depend on invisible information, and complexity of invisible state space is as high as 10^{48} on average (Li et al., 2020). In response to this challenge, Li et al. (2020) introduced suphx-style oracle guiding and demonstrated a performance gain. Thus, we consider Mahjong as a promising test platform for oracle guiding methods. Since the number of possible states in Mahjong is extremely large. It is costly to explore with random actions in an online RL manner, and no pre-

vious work could train a strong Mahjong AI with purely online RL. Also, we would like to examine the effectiveness of VLOG in offline RL settings. For these reasons, we transferred to offline RL (Levine et al., 2020) for the Mahjong task using expert demonstrations.

We processed about 23M steps of human experts' plays from the online Mahjong game platform *Tenhou* (https://tenhou.net/mjlog.html) to a dataset for offline RL (data were augmented using the symmetry in Mahjong, see Appendix F). Also, we created a simulator of Mahjong as the testing environment. Though there are sophisticated ways to encodes the state and action space of Mahjong (Li et al., 2020), we attempts to make simplifications with reasonable amounts of approximations since our goal is to not create an strong Mahjong AI, but to use Mahjong as a platform to study oracle guiding problems. In our case, the action space is composed of 47 discrete actions covering all decisions in Mahjong. An executor observation is a matrix encoding public information and the current player's private hand; An oracle observation concatenates the executor observation with the information of the opponents' private hands. (see Appendix F). We used an 1-D CNN as the encoder as done in Mahjong AIs commonly (Li et al., 2020), and the size of $\boldsymbol{z}_t$ and the decoder network width was increased to 512 and 1024, respectively (Appendix C).

Note that although Mahjong is a 4-player game, using offline RL data to train an agent does not involve multi-agent RL (Zhang et al., 2021) because the offline dataset is fixed: the opponents have fixed policies and thus can be considered as parts of the environment. Our experiments focused on single-agent RL to avoid the complexity caused by considering multi-agent RL. We investigated two kinds of offline RL settings. The first is conservative Q-learning (CQL) (Kumar et al., 2020). Our CQL setting differed from the online RL setting in previous sections by adding auxiliary CQL loss (Kumar et al., 2020) to the Q-learning loss function (Appendix B.1.2). The other is behavior cloning (BC). Although VLOG was designed for value-based RL, we could straightforwardly incorporate VLOG with BC by letting the network predict action instead Q function. The learning was conducted by minimizing the cross entropy between the output and the target action (demonstration) as in a classification problem. Note that we did not test OPD-style oracle guiding in BC setting because it was equal to the baseline since we can directly use demonstration actions as oracle policy for distillation.

being evaluated model	CQL		BC	
vs. baseline model	avg. payoff	match win rate(%)	avg. payoff	match win rate(%)
Oracle (cheating)	**224** $\pm$ 13	53.6 $\pm$ 0.4	15 $\pm$ 12	50.5 $\pm$ 0.4
Suphx - style	59 $\pm$ 13	51.2 $\pm$ 0.4	37 $\pm$ 12	50.9 $\pm$ 0.4
OPD - style	-9 $\pm$ 13	50.0 $\pm$ 0.4	-	-
VLOG	**233** $\pm$ **13**	**55.7** $\pm$ 0.4	41 $\pm$ 12	51.4 $\pm$ 0.4
VLOG-no oracle	61 $\pm$ 13	52.0 $\pm$ 0.4	**67** $\pm$ **12**	**52.2** $\pm$ **0.4**

Table 1: Performance of each model after training vs. the trained baseline model on Mahjong, each using 4 random seeds. Each match was played by four independent players (agents) for a series of 8 games. A player wins a match if it has the highest points among all players when finishing 8 games. Each player played for itself and only consider to maximize its own payoff and ranking. Among the 4 players, two were being tested agents and the other two were the baseline models (no communication or team cooperation between agents). The baseline model was trained with the same offline RL process, but without oracle guiding (Fig. 1C)). To decrease the variance, we sum up the results of two players with the same model, and use it as the performance of the corresponding method. Average payoff is the mean of points change per game. Each type of match was repeated 12,500 times (100,000 games). Data are Mean $\pm$ S.E.M..

Because Mahjong is a zero-sum, four-player game, we tested the performance of trained models in two scenarios: playing with the (trained) baseline model (Table 1) and playing with each other (fight each other, Table 2). In the first scenario, four agents were playing the same matches on the game table, where two of them were being tested agents and the other two were the baseline models. Although each agent played for itself and there was no communication between players, we simply added up the payoff of the two being tested agents, and consider they won a match if one of them ranked top (so the match win rate will be 50% if equally strong as baseline), for statistics (Table 1).

Fight each other (CQL)	avg. payoff	match win rate(%)	game win rate(%)	deal-in rate(%)
Oracle (cheating)	**168** $\pm$ 11	**28.0** $\pm$ 0.4	18.45	**4.48**
Suphx - style	-75 $\pm$ 12	23.2 $\pm$ 0.4	18.32	8.84
OPD - style	-209 $\pm$ 12	20.1 $\pm$ 0.4	14.39	8.06
VLOG	116 $\pm$ 12	**28.7** $\pm$ 0.4	**19.36**	8.28
Fight each other (BC)	avg. payoff	match win rate(%)	game win rate(%)	deal-in rate(%)
Oracle (cheating)	-27 $\pm$ 11	23.9 $\pm$ 0.4	20.69	**7.96**
Suphx - style	-10 $\pm$ 11	24.6 $\pm$ 0.4	20.51	8.03
VLOG	26 $\pm$ 11	**25.6** $\pm$ 0.4	**20.82**	8.00
VLOG-no oracle	11 $\pm$ 11	**25.9** $\pm$ 0.4	20.76	8.15

Table 2: Trained models playing with each other on the same table. *Deal-in* means the player discards a tile and another player wins the game by picking up this tile to compose a winning hand. Dealing-in results a punishment to the player, thus it is better to be avoided. Each type of match was repeated for 12,500 times (i.e. 100,000 games). Data are Mean $\pm$ S.E.M..

For CQL, the results (Table 1 left and 2 upper) show that VLOG substantially outperformed the baseline and alternative methods (because Mahjong is a highly random game, 55.7% match win rate indicates a large skill gap). Interestingly, VLOG was even comparable to oracle. This can be explained by that VLOG also benefited from its Bayesian property, which is consistent with that VLOG-no oracle showed a significant performance gain over the baseline model (Table 1 left). Still, the oracle model learned to reduce deal-ins (i.e., player discards a tile and another player wins the game by picking up this tile to compose a winning hand) since it could explicitly see the opponents' private tiles, showing a much lower deal-in rate than other non-cheating models (Table 2 upper).

In BC setting, the agents did not learn a value function, but tried to predict human experts' actions. Therefore, the training procedure did not involve reasoning the relationship between playing outcome and oracle observation, but just imitating human behaviors. This can be seen from the results that oracle did not substantially outperform baseline in BC (Table 1 right and 2 lower). However, VLOG and VLOG-no oracle still showed performance gain, thanks to the stochastic modeling.

6 SUMMARY

We have proposed VLOG – a variational Bayesian learning framework for leveraging oracle observation to facilitate DRL especially in partially observable environments. VLOG is available for any RL problem in which there is oracle observation that may help the executor to make decisions.

We first introduced a latent vector $\boldsymbol{z}$ to represent the environmental state. The prior and posterior distribution of $\boldsymbol{z}$ is modeled using executor and oracle observation, respectively. Then, we derived a variational lower bound (Eq. 2) by maximizing which we can optimize the executor model using the oracle observation. We developed the corresponding methodology for DRL, which can be incorporated with most RL algorithms that need to estimate a value function.

If oracle observation contains more information to retrieval the true environmental state (or, it is the true environmental state), VLOG's oracle guiding in latent space helps to shape a latent representation in neural networks closer to the true one. We demonstrated this advantage of VLOG using the maze task. Then, we scaled VLOG up to solve image-based videos games, and compared it with alternative oracle-guiding methods. Though all oracle-guiding methods showed performance gain over the baseline model, VLOG performed consistently the best. Finally, we transferred to offline RL domain using a challenging tile-based game Mahjong in which an executor plays with hidden information and random state transitions, and observed VLOG achieved best overall performance.

We also conducted an ablation study of VLOG (VLOG-no oracle) in which posterior model did not receive oracle observation, but executor one. VLOG-no oracle demonstrated performance gain in the tasks that may benefit from the stochasticity; otherwise, it performs similar to the deterministic baseline. This clarified that the source of VLOG's promising performance is two-fold: oracle guiding and stochastic modeling. Finally, we publish the dataset of Mahjong for offline RL and the corresponding RL environment so as to facilitate future research on oracle guiding.

ACKNOWLEDGEMENT

This work was supported by Microsoft Research Asia. Kenji Doya was supported by Japan Society for the Promotion of Science KAKENHI Grant Numbers JP16K21738, JP16H06561 and JP16H06563, as well as by Okinawa Institute of Science and Technology.

REPRODUCIBILITY STATEMENT

The source code of VLOG can be found in Supplementary Material.

ETHICS STATEMENT

We declare no conflict of interests. We tried to use friendly colors to people with color recognition disabilities (Fig. 2C, D) and distinguishable markers for performance curves of different models (Fig. 2B, Fig. 3 and Fig6). Our Mahjong dataset was generated using downloadable, public game replay data from *Tenhou.net* with post-processing. The dataset contains no private information about players. Since VLOG is a general framework for leveraging the oracle information, we cannot foresee any direct application of VLOG to malicious purposes. However, any new RL algorithm might confer increased autonomy on an agent, and eventually lead to a completely autonomous agent, which can be used for malicious purposes, e.g., fully autonomous soldiers.

REFERENCES

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep Variational Information Bottleneck. In *International Conference on Learning Representations*, 2017.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning*, 2017.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning. In *International Conference on Machine Learning*, 2017.

Christopher P. Burgess, Irina Higgins, Arka Pal, Loïc Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE. 2017.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pp. 66–75. PMLR, 2020.

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 *AAAI Conference on Artificial Intelligence*, 2021.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration. In *International Conference on Learning Representations*, 2018.

Thomas Furmston and David Barber. Variational methods for reinforcement learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 241–248. JMLR Workshop and Conference Proceedings, 2010.

Arthur Guez, Fabio Viola, Theophane Weber, Lars Buesing, Steven Kapturowski, Doina Precup, David Silver, and Nicolas Heess. Value-driven Hindsight Modelling. In *Advances in Neural Information Processing Systems*, 2020.

David Ha and Jürgen Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1856–1865, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv*, abs/1812.05905, 2019.

Dongqi Han, Kenji Doya, and Jun Tani. Variational Recurrent Models for Solving Partially Observable Control Tasks. In *International Conference on Learning Representations*, 2020.

Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, and Remi Munos. Hindsight Credit Assignment. In *Advances in Neural Information Processing Systems*, 2019.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2018.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations*, 2016.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep Variational Reinforcement Learning for POMDPs. In *International Conference on Machine Learning*, 2018.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is Q-learning provably efficient? In *Advances in Neural Information Processing Systems*, 2018.

Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Provably efficient reinforcement learning with linear function approximation. In *Conference on Learning Theory*, 2020.

Michael Johanson. Measuring the size of large no-limit poker games. *arXiv preprint arXiv:1302.7008*, 2013.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*, 2018.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

W. Bradley Knox and Peter Stone. Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. In *International Conference on Knowledge Capture*, 2009.

Tadashi Kozuno, Yunhao Tang, Mark Rowland, Remi Munos, Steven Kapturowski, Will Dabney, Michal Valko, and David Abel. Revisiting Peng's Q(λ) for Modern Reinforcement Learning. In *International Conference on Machine Learning*, 2021.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2020.

Alex Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33, 2020.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

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. *arXiv*, abs/2003.13590, 2020.

Robert Loftin, Bei Peng, James MacGlashan, Michael L. Littman, Matthew E. Taylor, Jeff Huang, and David L. Roberts. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous agents and multi-agent systems*, 30(1):30–59, 2016.

James MacGlashan, Mark K. Ho, Robert Loftin, Bei Peng, Guan Wang, David L. Roberts, Matthew E. Taylor, and Michael L Littman. Interactive Learning from Policy-Dependent Human Feedback. In *International Conference on Machine Learning*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. 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. *Nature*, 518(7540):529, 2015.

Masashi Okada, Norio Kosaka, and Tadahiro Taniguchi. PlaNet of the Bayesians: Reconsidering and Improving Deep Planning Network by Incorporating Bayesian Inference. In *International Conference on Intelligent Robots and Systems*, 2020.

Karl Pearson F.R.S. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. In *International Conference on Learning Representations*, 2016.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Edward J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Oper. Res.*, 26(2):282–304, 1978.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop*. IEEE, 2015.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *AAAI conference on artificial intelligence*, 2016.

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. *Nature*, 575(7782):350–354, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *International conference on machine learning*, 2016.

Theophane Weber, Nicolas Heess, Ali Eslami, John Schulman, David Wingate, and David Silver. Reinforced variational inference. In *Advances in Neural Information Processing Systems (NIPS) Workshops*, 2015.

Haiyan Yin, Jianda Chen, Sinno Jialin Pan, and Sebastian Tschiatschek. Sequential generative exploration model for partially observable reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10700–10708, 2021.

Kenny Young and Tian Tian. MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.

A IMPLEMENTATION OF ALTERNATIVE ORACLE GUIDING METHODS

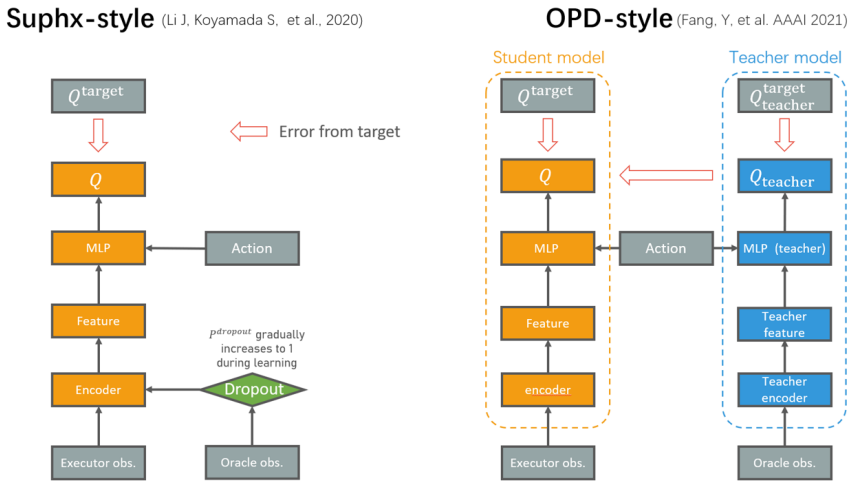

Figure 4: Diagram of our implementation for Suphx-style and OPD-style oracle guiding. For OPD-style oracle guiding, the teacher model is trained and then fixed in prior to training the student model (executor model).

In the original works of Suphx (Li et al., 2020) and OPD (Fang et al., 2021), the authors were performing oracle guiding to different RL objectives to ours. For comparison, we re-implemented Suphx-style and OPD-style oracle guiding in our settings using similar network structures (see Fig. 4) and used the same hyper-parameters wherever possible. However, the key methodology of Suphx-style and OPD-style oracle guiding remains the same as those in their original works.

As shown in Fig. 4 left, suppose executor observation is $\boldsymbol{x}_t$ and oracle observation is $\hat{\boldsymbol{x}}_t$, Suphx-style model receives input

$$\boldsymbol{x}_t^{\text{Suphx}} = \text{concatenate}(\boldsymbol{x}_t, \delta_t \odot \hat{\boldsymbol{x}}_t),$$

where δ_t is the dropout matrix whose elements are Bernoulli variables with $P(\delta_t(i,j) = 1) = 1 - p^{dropout}$, and $\odot$ denotes element-wise multiplication. During training, $p^{dropout}$ was linearly increased from 0 to 1 in the first $2/3$ training epochs, and kept to be 1 in the remaining $1/3$ epochs. The other parts of training were the same as the baseline model.

For OPD-style oracle guiding, we directly used the trained oracle model as the teacher model (Fig. 4, right). The loss function (in Q-learning) was

$$\mathcal{J}_{\text{OPD}} = \text{MSE}\left(Q_t^{tar}, Q(\boldsymbol{x}_t, a_t)\right) + \mu\text{MSE}\left(Q_{\text{teacher}}(\hat{\boldsymbol{x}}_t), Q(\boldsymbol{x}_t, a_t)\right),$$

where the first term $\text{MSE}\left(Q_t^{tar}, Q(\boldsymbol{x}_t, a_t)\right)$ was the same as the loss function for the baseline model. And the second term is the distillation loss with a coefficient μ (note that the teacher model is fixed during training). We selected the best-performing μ for each task using grid search (see Table 3).

B RL ALGORITHMS AND HYPER-PARAMETERS

B.1 RL ALGORITHMS

B.1.1 DUELING DOUBLE DQN FOR MAZE AND MINATAR TASKS

As we discussed in Sec. 5, we used double DQN with dueling network architecture (van Hasselt et al., 2016; Wang et al., 2016) as the base RL algorithm, because it work relatively well (Hessel et al., 2018) without introducing additional hyper-parameter.

The Dueling architecture of DQN (Wang et al., 2016) is defined as follows (see Appendix C for hidden layer size):

```
class DuelingQNetwork(nn.Module):
    def __init__(self, input_size, action_num, hidden_layers):
        super(DuelingQNetwork, self).__init__()
        self.input_size = input_size
        self.action_num = action_num
        self.hidden_layers = hidden_layers
        self.network_modules = nn.ModuleList()
        last_layer_size = input_size
        for layer_size in hidden_layers:
            self.network_modules.append(nn.Linear(last_layer_size, layer_size))
            self.network_modules.append(nn.ReLU())
            last_layer_size = layer_size
        self.value_layer = nn.Linear(last_layer_size, 1)
        self.advantage_layer = nn.Linear(last_layer_size, action_num)
        self.main_network = nn.Sequential(*self.network_modules)
    def forward(self, x):
        h = self.main_network(x)
        v = self.value_layer(h).repeat_interleave(self.action_num, dim=-1)
        q0 = self.advantage_layer(h)
        a = q0 - torch.mean(q0, dim=-1, keepdim=True).repeat_interleave(
            self.action_num, dim=-1)
        q = v + a
        return q
```

Double deep Q-learning (van Hasselt et al., 2016) was used to compute Q^{target} in Fig. 1 A (one can use any other algorithm to compute Q^{target} without changing other parts). In particular, as in Wang et al. (2016), we have

$$Q_t^{\text{target}} = r_t + \gamma Q(\boldsymbol{z}_t, \arg\max_{a'} Q(\boldsymbol{z}_{t+1}, a'; \theta); \theta^-),$$

where r_t is the reward at step t, γ is the discount factor (Table 3), θ denotes the parameters of the Q network (MLP decoder) for computing the Q-function (Appendix C). Note that $\boldsymbol{z}$ is given by the posterior decoder with oracle observation $\boldsymbol{x}$ as input, since the oracle prediction error term $\mathbb{E}_{q(\boldsymbol{z}_t|\hat{\boldsymbol{x}}_t)}\left[\log \mathcal{P}\left(v(\boldsymbol{z}_t) = v_t^{tar}|\boldsymbol{z}_t\right)\right]$ in Eq. 1 is the expectation over the posterior distribution $q(\boldsymbol{z}|\hat{\boldsymbol{x}})$. Following deep RL normals (Mnih et al., 2015; Wang et al., 2016; van Hasselt et al., 2016), we used a target Q network with the same structure as the original Q network, of which the parameters are denoted as θ^- (Table 3). Every 1,000 steps, the target Q network copies the parameters from the original Q network (Table 3). Then the first term of the VLOG loss function (Eq. 2) is simply given by the mean square error between Q^{target} and the output of Q network (MLP decoder) for the Maze and MinAtar tasks.

B.1.2 Dueling Double DQN with conservative Q learning for Mahjong

In Mahjong, as we transfer to offline RL domain (Sec. 5.3), directly using off-policy RL algorithm usually results to very unsatisfying performance (Levine et al., 2020).

Therefore, we complement the loss function of VLOG (Eq. 2) with an auxiliary conservative Q-learning (CQL) loss (Kumar et al., 2020),

$$\mathcal{J}_{\text{CQL}} = \alpha \mathbb{E}_{\hat{\boldsymbol{x}}, a \sim \mathcal{D}} \left[\log \sum_{a' \in \boldsymbol{A}} \exp(Q(\boldsymbol{z}(\hat{\boldsymbol{x}}), a', \theta)) - [Q(\boldsymbol{z}(\hat{\boldsymbol{x}}), a, \theta)] \right],$$

where $\mathcal{D}$ is the offline dataset we used for Mahjong and $\alpha = 1$. The combined loss function used for Mahjong (CQL) is $\mathcal{J}_{\text{VLOG}}^{\beta} + \mathcal{J}_{\text{CQL}}$.

B.2 Hyper-parameter selection

We summarize the hyper-parameters in Table 3.

Explanation	Symbol	Value	Comment
Common			
Discount factor	γ	1 (Mahjong) 0.995 (others)	
Learning rate		0.0001	For all loss functions
Batch size		1024 (Mahjong) 128 (others)	
coefficient of ϵ-greedy	ϵ	0.1 (learning) 0 (evaluation)	For online RL
target network update interval	τ	1000	
environment steps per gradient step		4	For online RL
Optimizer		Adam	For all loss functions
VLOG			
KL divergence target value	D_{KL}^{tar}	50	Fixed in all experiments
Initial value of β		0.00001	
OPD - style			
coefficient of policy distillation loss	μ	0.01 (Mahjong) 10 (others)	By grid search

Table 3: Hyper-parameters used in this paper. For OPD-style, we did grid search for the additional hyper-parameter μ to select the best-performing value among [0.001, 0.01, 0.1, 1, 10].

B.3 Sensitivity analysis for β

As we mentioned in Sec. 4.2, the coefficient β is important to the learning of VLOG agents. If we fix the value of β throughout training, a too large or too small β will result in worse performance. The corresponding results are shown in Fig. 6.

C Network structures

For simplicity, we use the same *hidden_layer_size* for all the fully connected layers, where the *hidden_layer_size* is 256 for maze and MinAtar, and 1024 for Mahjong.

C.1 Encoder

Since we targeted at various tasks, we used different encoder network for each type of environment. The prior and posterior encoder has the same structure except different sizes of input features/channels.

For the maze task, the encoder was a 2-layers MLP with ReLU activation. The output size is also equal to *hidden_layer_size*.

For the MinAtar tasks, we used 2-D CNN encoder defined as follows:

```
import torch.nn as nn
cnn_module_list = nn.ModuleList()
cnn_module_list.append(nn.Conv2d(n_channels, 16, 3, 1, 0))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Conv2d(16, 32, 3, 1, 0))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Conv2d(32, 128, 4, 2, 0))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Conv2d(128, 256, 2, 1, 0))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Flatten())
cnn_minatar = nn.Sequential(*cnn_module_list)
```

where *n_channels* is the number of channels of executor or oracle observation. The output size is equal to *hidden_layer_size*.

For Mahjong, because the second dimension of observation (tile ID) has local contextual relationship (Li et al., 2020), we used 1-D CNN (convolution along the tile ID dimension) as the encoders, defined as follows:

```
cnn_module_list = nn.ModuleList()
cnn_module_list.append(nn.Conv1d(n_channels, 64, 3, 1, 1))
cnn_module_list.append(nn.Conv1d(64, 64, 3, 1, 1))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Conv1d(64, 64, 3, 1, 1))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Conv1d(64, 32, 3, 1, 1))
cnn_module_list.append(nn.ReLU())
cnn_module_list.append(nn.Flatten())
cnn_mahjong = nn.Sequential(*cnn_module_list)
```

The output size is 1088, close to *hidden_layer_size*.

C.2 LATENT LAYER

For VLOG and VLOG (no-oracle), the size of $\boldsymbol{z}$ layer is half of *hidden_layer_size* because we need to estimate both the mean and the variance of $\boldsymbol{z}$. For all the other models (baseline, oracle, OPD-style, Suphx-style), the latent layer is one fully connected layer with size *hidden_layer_size* and ReLU activation.

C.3 DECODER

The decoders for all models were 2-layers MLPs with size *hidden_layer_size*. Except BC on Mahjong, the input of decoder was the output of the latent layer concatenated with the action, and we used the dueling Q-network structure (Wang et al., 2016) to output a scalar Q value. ReLU activation was used except for the output.

For BC on Mahjong, the input of decoder was the output of the latent layer. The outputs of decoder was logit of actions, and the actions could be obtained using softmax.

D TASK-AGNOSTIC LATENT BOTTLENECK CONTROL

As we discussed in Sec. 4.2, the coefficient β of the regularization term in the VLOG loss function (Eq. 2) is adaptively regularized by Eq. 3, provided with D_{KL}^{tar}, which is another hyper-parameter. While our experiments demonstrated the effectiveness of this approach, the follows discuss more thoughts behind this design choice.

In principle, replacing a hyper-parameter with another does not always make training easier. However, in practice (especially deep RL), the performance will be highly sensitive to some hyper-parameters (e.g., the entropy coefficient α in the original soft-actor critic algorithm (Haarnoja et al., 2018) need to be tuned in each robotic task. This is because the reward magnitude is different among tasks.). As a result, it will be beneficial to replace a sensitive hyper-parameter with another one that does not need fine tuning. For example, in the follow-up paper of soft-actor critic, the authors used adaptive entropy coefficient α by introducing another hyper-parameter, the entropy target (Haarnoja et al., 2019). They empirically found that it is good to set the entropy target equal to the negative of the degree of freedom of the agent, thus to avoid tuning α.

Our idea of replacing β with D_{KL}^{tar} is due to similar reasons. One obvious problem is that the magnitude of the "oracle prediction error" term (Eq. 2) relies on the reward magnitude of the task. Therefore β should also be adjusted to match with the magnitude of task reward. However, D_{KL}^{tar} is only relevant to the magnitude of prior $\boldsymbol{z}$ and posterior $\boldsymbol{z}$, which does not differ too much among tasks (usually at the order of 1). In practice, we found that $D_{KL}^{tar} = 50$ works well for all the tasks including Maze, MinAtar and Mahjong.

Another way of regularizing β is to employ a linear or exponential scheduler. For example, in Burgess et al. (2017), the authors used a linear scheduler for the target KL-divergence and got good

results. However, using a scheduler introducing more hyper-parameters (at least two: initial β and final β), which is against our intention to reduce the impact of hyper-parameters.

E SEAQUEST LOCAL OPTIMUM

In Seaquest, the agent drives a submarine diving into the sea to shoot at enemies and rescue divers to earn scores. However, the submarine has limited oxygen. It must surface to replenish the oxygen before running out for surviving, by doing which it temporarily cannot earn scores. A typical local optimum is to use the last remaining oxygen for diving instead of surfacing.

F MAHJONG

F.1 ESTIMATION OF GAME COMPLEXITY OF MAHJONG

We consider 4-players Japanese Mahjong[4]. Although there are minor variants of rules, the following estimation applies to general cases.

For easier computation, we make two major simplifications (i.e., we estimate a lower bound of the game complexity): (1) melding from discard[5] is not considered and (2) the information other than tiles, such as results of contextual games, points of players, is not considered.

There are 34 types of tiles, each with 4 duplicates (so totally 136 tiles). We further restrict our estimation to the last turn (i.e., the last tile is drawn). Among 136 tiles, 53 tiles are in someone's hand (the 4 players have 14, 13, 13, 13 tiles in hands, respectively). Permutation of tiles in one's hand does not make a different, while the permutation matters if not in one's hand. The number of distinguishable configurations of 136 tiles thus can be computed as $\frac{136!}{(13!)^3 \times 14! \times (4!)^{34}} \sim 10^{145}$

Meanwhile, for each discarded tile, it is important to know whether it was discarded immediately after being drawn or not. For 70 discarded tiles, the number of possibilities is simply $2^{70} \sim 10^{21}$.

Therefore, the lower bound of game complexity of Mahjong is estimated as

$$10^{145} \times 10^{21} \sim 10^{166}.$$

If considering the other being simplified information, the state space could be much larger. For example, let's consider the current points of each player. The most common rule is that each player starts with 25,000 points, with 100 points as the minimal unit (totally 1,000 units), and the game terminates if someone got negative points. So the number of possibilities can be converted to the answer of "how many ways to distribute 1000 candies to 4 kids", which is $\frac{(1000+1)^{(4-1)}}{(4-1)!} \sim 10^8$.

F.2 DETAILS OF OBSERVATION SPACE AND ACTION SPACE ENCODING

In our Mahjong environment[6], the action space is composed of 47 discrete actions (Table 5). Because not all actions are available at a certain step, we also provide the set of valid actions among all 47 actions at each step according to the rules, which can be used during playing and learning.

The oracle observation has the shape of 111×34 (executor observation is a part of oracle observation, with shape 93×34) (Fig. 5). The first dimension corresponds to 111 features (channels). The second dimension of observation (with size 34) corresponds to 34 Mahjong tiles (the order is Character 1-9, Dot 1-9, Bamboo 1-9, East, South, West, North, White, Green, Red). We used 1-D CNNs with convolution along the second dimension for the encoders. Suppose the current player is player 0, and other players are numbered 1, 2, 3 counter-clockwise. The value of any element in an observation is 1 or 0, explained in Table 4.

[4] https://en.wikipedia.org/wiki/Japanese_Mahjong

[5] A meld is a specific pattern of three or four tiles. A player can pick up a discarded tile from others to form a meld by displaying the meld to public, if certain condition is satisfied.

[6] The Mahjong environment we used in this papers is available on https://github.com/pymahjong/pymahjong for reproducibility. However, we recommend to use the newer version https://github.com/Agony5757/mahjong which is better-supported by the authors and much faster.

Rough meaning	Feature index	Meaning (t is the corresponding tile)
Hand of player 0	0	If player 0 has $\geq$1 t in hand
	1	If player 0 has $\geq$2 t in hand
	2	If player 0 has $\geq$3 t in hand
	3	If player 0 has 4 t in hand
	4	If player 0 has dicarded t in this game
	5	If player 0 has Red Dora t in hand
Callings of player 0 (melding from discarded tiles)	6	If player 0 has $\geq$1 t in callings
	7	If player 0 has $\geq$2 t in callings
	8	If player 0 has $\geq$3 t in callings
	9	If player 0 has 4 t in callings
	10	If player 0 has $\geq$1 t in callings and t is from other's discarded tiles
	11	If player 0 has Red Dora t in callings
Callings of player 1	12 to 17	Same as 5 to 11, but for player 1
Callings of player 2	18 to 23	Same as 5 to 11, but for player 2
Callings of player 3	24 to 29	Same as 5 to 11, but for player 3
Discarded tiles from player 0	30	If player 0 has discarded $\geq$1 t
	31	If player 0 has discarded $\geq$2 t
	32	If player 0 has discarded $\geq$3 t
	33	If player 0 has 4 t in calling
	34	If player 0's first dicarded t is Tegiri (if applicable)
	35	If player 0's second dicarded t is Tegiri (if applicable)
	36	If player 0's third dicarded t is Tegiri (if applicable)
	37	If player 0's fourth dicarded t is Tegiri (if applicable)
	38	If player 0 has discarded Red Dora t
	39	If player 0 has discarded t to annouce Riichi
Discarded tiles from player 1	40 to 49	Same as 30-39, but for player 1
Discarded tiles from player 2	50 to 59	Same as 30-39, but for player 2
Discarded tiles from player 3	60 to 69	Same as 30-39, but for player 3
Other public information	70	If t is Dora indicator ($\geq$ 1 repeats)
	71	If t is Dora indicator ($\geq$ 2 repeats)
	72	If t is Dora indicator ($\geq$ 3 repeats)
	73	If t is Dora indicator (4 repeats)
	74	If t is Dora ($\geq$ 1 repeats)
	75	If t is Dora ($\geq$ 2 repeats)
	76	If t is Dora ($\geq$ 3 repeats)
	77	If t is Dora (4 repeats)
	78	If t is wind of the table
	79	If t is wind of self
The tile of the latest action	80	If t is the tile corresponding to the latest action
Information for available actions	81	If at t is in player 0's hand
	82	If at t can be Chi, and is the smallest in the meld
	83	If at t can be Chi, and is the middle in the meld
	84	If at t can be Chi, and is the largest in the meld
	85	If at t can be Pong
	86	If at t can be An-Kan
	87	If at t can be Kan
	88	If at t can be Ka-Kan
	89	If Riichi is possible by discarding t
	90	If t is the latest discarded tile from others enabling RonAgari
	91	If t is the latest drawn tile enabling Tsumo
	92	If t is Kyuhai and is in player 0's hand
Hand of player 1 (only for oracle)	93 to 98	Same as 0 to 5, but for player 1
Hand of player 2 (only for oracle)	99 to 104	Same as 0 to 5, but for player 2
Hand of player 3 (only for oracle)	105 to 110	Same as 0 to 5, but for player 3

Table 4: Explanation of the 111 features of oracle observation encoding (the first 93 features are available to the executor) of our Mahjong environment. Player 0 is the current player who is making decision and player 1, 2, 3 are opponents counterclockwise. Tegiri ("discard from hand") means the tile is not discarded immediately after drawing it.

Action index	Explanation
0	Discard Character 1
1	Discard Character 2
2	Discard Character 3
3	Discard Character 4
4	Discard Character 5 (non-Red Dora with higher priority)
5	Discard Character 6
6	Discard Character 7
7	Discard Character 8
8	Discard Character 9
9	Discard Dot 1
10	Discard Dot 2
11	Discard Dot 3
12	Discard Dot 4
13	Discard Dot 5 (non-Red Dora with higher priority)
14	Discard Dot 6
15	Discard Dot 7
16	Discard Dot 8
17	Discard Dot 9
18	Discard Bamboo 1
19	Discard Bamboo 2
20	Discard Bamboo 3
21	Discard Bamboo 4
22	Discard Bamboo 5 (non-Red Dora with higher priority)
23	Discard Bamboo 6
24	Discard Bamboo 7
25	Discard Bamboo 8
26	Discard Bamboo 9
27	Discard East Wind
28	Discard South Wind
29	Discard West Wind
30	Discard North Wind
31	Discard White Dragon Tile (Haku)
32	Discard Green Dragon Tile (Hatsu)
33	Discard Red Dragon Tile (Chu)
34	Chi (the picked up tile is the smallest in the meld)
35	Chi (the picked up tile is the middle in the meld)
36	Chi (the picked up tile is the largest in the meld)
37	Pon
38	An-Kan
39	Kan
40	Ka-Kan
41	Riichi
42	Ron
43	Tsumo
44	Restart the game with Kyushukyuhai
45	Not to response (when Chi, Pon, Kan, Ron, etc. is possible)
46	Not to Riichi (When Riichi is possible)

Table 5: Action encoding of our Mahjong environment.

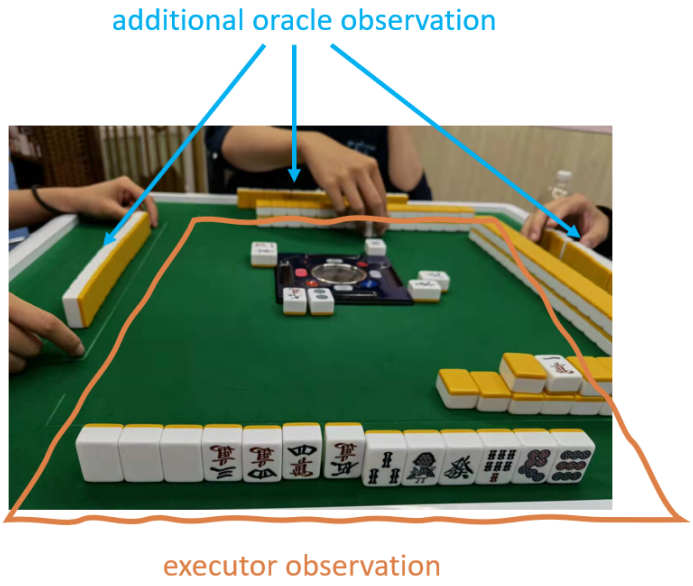

Figure 5: Picture of a Mahjong game. The executor observation includes the publicly visible information and the player's private hand tiles. The oracle observation includes the executor observation and additional information about the opponents' private tiles.

F.3 Data augmentation

In (Japanese) Mahjong, there are 3 suit sets of (characters, dots, bamboos) tiles, 4 wind tiles and 3 dragon tiles. The 3 suit sets are symmetric and thus exchangeable with each other[7]. So are the 4 wind tiles and the 3 dragon tiles. According to such symmetry, we augmented the offline RL dataset by randomly exchanging the 3 suit sets, the 4 wind tiles and the 3 dragon tiles, respectively.

[7]There is one exemption of winning hand pattern "all green". Since it is an extremely rare case, we simply ignore it.

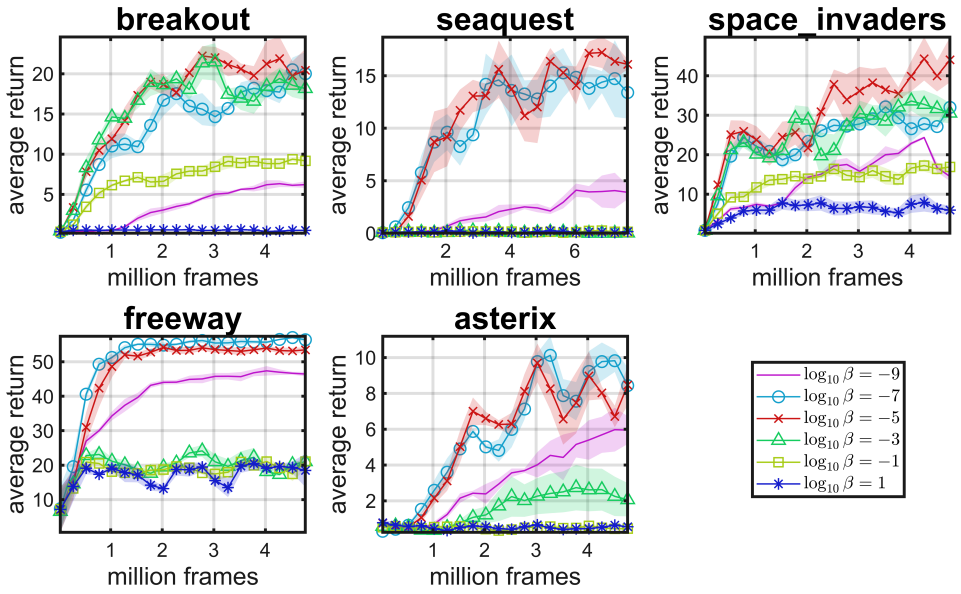

Figure 6: Sensitivity analysis of VLOG in noisy MinAtar environments w.r.t. the selection of β if β is fixed.