Greedy when Sure and Conservative when Uncertain about the Opponents

Haobo Fu [* 1] **Ye Tian** [* 1] **Hongxiang Yu** [2] **Weiming Liu** [1 3] **Shuang Wu** [1] **Jiechao Xiong** [1] **Ying Wen** [2] **Kai Li** [4 5] **Junliang Xing** [6] **Qiang Fu** [1] **Wei Yang** [1]

Abstract

We develop a new approach, named Greedy when Sure and Conservative when Uncertain (GSCU), to competing online against unknown and nonstationary opponents. GSCU improves in four aspects: 1) introduces a novel way of learning opponent policy embeddings offline; 2) trains offline a single best response (conditional additionally on our opponent policy embedding) instead of a finite set of separate best responses against any opponent; 3) computes online a posterior of the current opponent policy embedding, without making the discrete and ineffective decision which type the current opponent belongs to; and 4) selects online between a real-time *greedy* policy and a fixed *conservative* policy via an adversarial bandit algorithm, gaining a theoretically better regret than adhering to either. Experimental studies on popular benchmarks demonstrate GSCU's superiority over the state-of-the-art methods. The code is available online at `https://github.com/YeTianJHU/GSCU`.

1. Introduction

Developing agents that play competitively against other opponents is challenging in multiagent scenarios, especially when the opponents are nonstationary. One approach is computing a strong yet fixed policy, which has been the goal in many human-played games, such as Go (Silver et al., 2016), StarCraft II (Vinyals et al., 2019), poker (Moravčík et al., 2017; Brown & Sandholm, 2018; 2019), Mahjong (Fu et al., 2021), etc. In two-player zero-sum games, the target of such a policy is often a Nash Equilibrium (NE) policy, which is guaranteed to not lose in expectation to any opponent. Yet, playing a fixed policy may be too conservative if the opponent has some exploitable weakness, since the weakness can be otherwise modelled and exploited.

Alternatively, opponent modelling methods (Hernandez-Leal et al., 2017; Albrecht & Stone, 2018) condition an agent policy on not only its environmental observation but also predictions about relevant properties (such as policies and goals) of the opponents. Yet, making accurate such predictions is not easy (Hernandez-Leal et al., 2017; Albrecht & Stone, 2018), and training a policy dependent additionally on the predictions is often more difficult than solely on the observation. As a result, knowing when opponent modelling is effective is as important as opponent modelling itself.

In this paper, we investigate the problem of competing online against unknown opponents, which can be either stationary or nonstationary. We consider a competitive multiagent setting, where we control only one agent (the main agent) and all other agents are opponents. We contribute by answering two questions: (1) *What can be done offline to prepare the main agent for competing online against unknown opponents?* (2) *During online execution, what policy should the main agent use against the current opponents?*

- Answering *question* (1): We propose a novel training procedure based on Variational Autoencoder (VAE) (Kingma & Welling, 2014) for learning opponent policy embeddings offline. In comparison with previous policy embedding learning methods using VAEs, we decouple the learning of policy embedding from the representation learning of other information by conditioning the encoder solely on an opponent index. For the decoder, a sampled embedding together with an opponent observation produces the probability of an opponent action. The decoupling facilitates effective policy embedding learning in terms of both discrimination and generalization. Afterwards, a conditional Reinforcement Learning (RL) is invoked to train a single best response against potential opponents. Compared to a finite set of separate best responses targeting specific opponent policies, our single best response, which takes additionally our opponent policy embedding as input, is better at generalizing to infinite opponent policies.

[*]Equal contribution [1]Tencent AI Lab, Shenzhen, China [2]Shanghai Jiao Tong University, Shanghai, China [3]University of Science and Technology of China, Hefei, China [4]Institute of Automation, Chinese Academy of Sciences, Beijing, China [5]School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China [6]Tsinghua University, Beijing, China. Correspondence to: Haobo Fu <haobofu@tencent.com>.

Proceedings of the 39^{th} *International Conference on Machine Learning*, Baltimore, Maryland, USA, PMLR 162, 2022.

- Answering *question* (2): We offer a new perspective on competing online against unknown opponents, where we convert the problem to a two-armed adversarial bandit problem. One arm follows a fixed *conservative* policy, which hopefully has the best worst-case performance. The other arm exploits the current opponent by playing a real-time *greedy* policy, which is obtained by combing the online inferred opponent embedding with our offline trained best response. The real-time *greedy* policy generalizes to unknown opponents via the single best response and the continuous opponent policy embedding, without deciding which category the current opponent belongs to or conducting any form of change detection. Moreover, we benefit from selecting between the real-time *greedy* policy and the *conservative* policy using an adversarial bandit algorithm, in contrast to previous methods that either play a fixed policy or keep trying to exploit the current opponent.
- We further prove that the performance of the real-time *greedy* policy is lower bounded by a ground truth best response's performance minus two positive terms. One term is the performance gap between a ground truth best response and our approximate best response. The other term is a function of the KL divergence between the estimated and the ground truth opponent policy. In addition, we prove that selecting between the real-time *greedy* policy and the fixed *conservative* policy using an adversarial bandit algorithm is at least as good as, in terms of the regret (defined later), adhering to either.

The overall approach is named Greedy when Sure and Conservative when Uncertain (GSCU). We evaluate GSCU on a two-player zero-sum imperfect information game and a four-player general-sum imperfect information game. In comparison with prior methods, GSCU demonstrates more robust performance against a wide range of unknown and nonstationary opponents. In particular, GSCU performs the best in terms of the average and worst-case performance.

2. Notations and Preliminaries

2.1. Problem Definition and Assumptions

Stochastic games. We employ stochastic games (Shapley, 1953) to discuss the problem of competing online against unknown opponents. A stochastic game has n agents and a state space S. Each agent i has an action space A_i and an observation space O_i. At each time step t, an agent i receives an observation $\mathbf{o}_{i;t} \in O_i$ and executes an action $a_{i;t} \in A_i$ according to its policy $\pi_i : O_i \times A_i \to [0,1]$. Afterwards, the game outputs a reward $r_{i;t} \in \mathbb{R}$ for each agent i and proceeds to the next state according to some environmental transition function. Define an episode of the game as an instantiation of the game that starts at some state $\mathbf{s}_0 \in S$ and proceeds according to each agent policy π_i till a terminal state. We denote the main agent policy by π_1 and the policy of an opponent by $\pi_i, 2 \leq i \leq n$. The expected returns of the main agent when playing π_1 against $\{\pi_i\}_{i=2}^n$ is denoted by $u_1(\pi_1, \pi_{2::n})$, where $u_1(\pi_1, \pi_{2::n}) = \mathbb{E}[\sum_{t=0}^H \gamma^t r_{1;t}]$, with $\gamma \in [0, 1.0]$. Let $u_1^{\pi_1, \pi_{2::n}}(\mathbf{o}, a) = \mathbb{E}[\sum_{t=0}^H \gamma^t r_{1;t} | \mathbf{o}_{1;0} = \mathbf{o}, a_{1;0} = a]$, and the observation value is $u_1^{\pi_1, \pi_{2::n}}(\mathbf{o}) = \sum_a \pi_1(a|\mathbf{o}) u_1^{\pi_1, \pi_{2::n}}(\mathbf{o}, a)$. Assume $|u_1^{\pi_1, \pi_{2::n}}(\mathbf{o}, a)| < \infty$ and H is finite.

The objective. In our setting of competing online against unknown opponents for T episodes, the goal is to sequentially decide a policy $\pi_{1;j}$ for the main agent at each episode $j, 1 \leq j \leq T$, such that the regret R_T is minimized:

$$R_T = \max_{\pi_1 \in \Pi_1} \sum_{j=1}^{T} [u_1(\pi_1, \pi_{2::n;j}) - u_1(\pi_{1;j}, \pi_{2::n;j})]. \quad (1)$$

Note that we do not have control over opponent policies $\pi_{2::n;j}$ at each episode j. Intuitively, the regret R_T of a sequence of main agent policies $\{\pi_{1;j}\}_{j=1}^T$ measures the expected returns lost when compared with a best fixed main agent policy in its policy space Π_1 in hindsight.

Our assumptions. To enable effective opponent modelling, we assume full access to opponent history trajectories (sequence of observation-action pairs) in previous episodes but *not* the current episode. This is common in human-played games, where we can look back into replays that have full visibility of opponents. It is worth noting that during online test opponent policies are unknown to the main agent and allowed to change arbitrarily over time.

As our problem is competing online against unknown opponents, we assume a strong and fixed main agent policy π_1 is available offline. We further assume that for each opponent we have K different precomputed policies, which we denote by $\Pi^{Train} = \{\pi_i^{(k)} | 2 \leq i \leq n, 1 \leq k \leq K\}$. The policy π_1 can be obtained by running, e.g., regret minimization algorithms (Zinkevich et al., 2007; Fu et al., 2021) or competitive multiagent RL algorithms (Hernandez-Leal et al., 2019b; Zhang et al., 2021). The policy set Π^{Train} can be obtained by collecting intermediate versions of π_1 during training, if all agents are homogeneous. For reasons explained later, a more appropriate way of producing Π^{Train} may be running some multiagent RL algorithm that emphasizes policy diversity (Parker-Holder et al., 2020).

2.2. Variational Autoencoder

Considering some data set $\{\mathbf{x}^{(i)}\}_{i=1}^N$ with N i.i.d. samples, VAE (Kingma & Welling, 2014) applies an unobserved continuous latent variable $\mathbf{z}$ to each data point $\mathbf{x}$. The objective of VAE is to learn a model of the data generating process, by simultaneously training a probabilistic encoder $q_{\phi_e}(\mathbf{z}|\mathbf{x})$ and a probabilistic decoder $p_{\theta_d}(\mathbf{x}|\mathbf{z})$. The log likelihood of a data point $\mathbf{x}$ is $\log p(\mathbf{x}) = D_{KL}(q_{\phi_e}(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})) +$

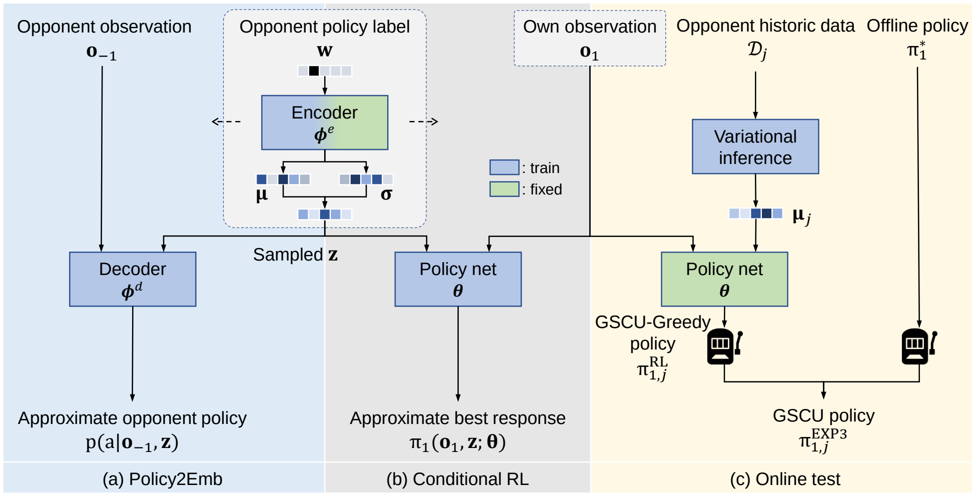

Figure 1. GSCU consists of two offline training components ((a) and (b)) and an online adaptation process ((c)).

$\mathcal{L}(\phi^e, \phi^d; \mathbf{x})$. $D_{KL}(q_{\phi^e}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ denotes the KL divergence of the approximate $q_{\phi^e}(\mathbf{z}|\mathbf{x})$ from the true posterior $p(\mathbf{z}|\mathbf{x})$. As the KL-divergence is non-negative, $\mathcal{L}(\phi^e, \phi^d; \mathbf{x})$ is called the Evidence Lower Bound (ELBO), which can be further written as: $\mathcal{L}(\phi^e, \phi^d; \mathbf{x}) = -D_{KL}(q_{\phi^e}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) + \mathbb{E}_{q_{\phi^e}(\mathbf{z}|\mathbf{x})}[\log p_{\phi^d}(\mathbf{x}|\mathbf{z})]$. The reparameterization trick (Kingma & Welling, 2014) can be applied to optimize the ELBO. β-VAE (Higgins et al., 2014) introduces a parameter $\beta \geq 0$ to trade-off between the reconstruction loss and the KL-divergence loss in the ELBO: $\mathcal{L}(\phi^e, \phi^d; \mathbf{x}) = -\beta D_{KL}(q_{\phi^e}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) + \mathbb{E}_{q_{\phi^e}(\mathbf{z}|\mathbf{x})}[\log p_{\phi^d}(\mathbf{x}|\mathbf{z})]$. The Conditional VAE (CVAE) (Walker et al., 2016) makes the probabilistic decoder dependent on some additional variable $\mathbf{c}$: $p_{\phi^d}(\mathbf{x}|\mathbf{z}, \mathbf{c})$.

2.3. Single Agent RL

Traditional RL has been widely studied in the single agent environment, where a policy is optimized to maximize the expected discounted returns starting from any state. Either value based methods, such as Deep Q Network (DQN) (Mnih et al., 2015), or policy gradient methods, such as Proximal Policy Optimization (PPO) (Schulman et al., 2017), can be employed.

When opponent policies are stationary, our multiagent environment is essentially a single agent environment from the perspective of the main agent. As a result, we can use single agent RL methods to approximate a best response $BR(\pi_{2:n})$ for the main agent against any $\pi_{2:n}$ in offline training, where $BR(\pi_{2:n}) = \arg\max_{\pi_1 \in \Pi_1} u_1(\pi_1, \pi_{2:n})$.

2.4. Multi-armed Bandit

The multi-armed bandit (Lattimore & Szepesvári, 2020) is concerned with how to optimally select among a fixed number of actions (arms) in an online manner. The optimality is determined by the cumulative rewards, where the reward of an action follows some unknown probabilistic distribution. In *stochastic bandits* where the reward distribution stays stationary, classical algorithms include Upper Confidence Bound (UCB) (Lai & Robbins, 1985; Auer et al., 2002a) and Thompson Sampling (Thompson, 1933). In *adversarial bandits* where no assumption is made about the reward distribution, a widely studied algorithm is EXP3 (Auer et al., 2002b). In a K-armed bandit, EXP3 chooses an action a at iteration j with probability $p_j(a) = (1-\eta)\frac{\exp(\eta\omega_j(a))}{\sum_{a'}\exp(\eta\omega_j(a'))} + \frac{\eta}{K}$, where $\omega_j(a)$ is a variable tracking the cumulative rewards of action a and η is a hyperparameter.

3. GSCU

We develop a new approach, i.e., GSCU for competing online against unknown opponents in this section. For the ease of discussion, we assume there is only one opponent, the index of which is denoted by -1. Extension to multiple opponents is straightforward, and we leave the case where the number of opponents can not be predetermined as future work. The general idea in GSCU is: exploiting the opponent when this is preferable and otherwise playing conservatively by following a fixed worst-case optimal policy π_1^*.

At the beginning of each episode j during online test, GSCU selects between two arms using EXP3. One arm follows the policy π_1^*. The other arm plays an approximate best response $\pi_1(\mathbf{o}, b(\pi_{-1;j}); \theta)$, which conditions on a belief $b(\pi_{-1;j})$ about the opponent policy $\pi_{-1;j}$ for the jth episode. The belief $b(\pi_{-1;j})$ is about an opponent policy embedding $\mathbf{z}_j$, whose posterior distribution $p(\mathbf{z}_j|\mathcal{D}_j)$ is updated online via Bayesian inference. The data set $\mathcal{D}_j$ refers to observation-action pairs of the opponent in previous episodes: $\{(\mathbf{o}_{-1}, a_{-1}) \in \tau_{-1;i}\}_{i=1}^{j-1}$, where $\mathcal{D}_1 = \emptyset$ and $\tau_{-1;i}$ denotes the opponent trajectory in the ith episode.

In offline training, GSCU employs a novel CVAE architecture to learn a probabilistic encoder and a probabilistic decoder for the opponent policy embedding $\mathbf{z}$. Afterwards, the parameters θ of the approximate best response $\pi_1(\mathbf{o}, \mathbf{z}; \theta)$ are trained by sampling $\mathbf{z}$ and using the corresponding opponent policy within a single agent RL procedure. An overview of GSCU is illustrated in Figure 1, and we elaborate each component of GSCU in the following.

3.1. Offline Policy Embedding Learning

We assume the existence of some latent embedding $\mathbf{z} \in Z$, which combined with an environmental observation $\mathbf{o}$ produces an action distribution: $O \times Z \times A \to [0, 1]$. We prefer the space Z to be regularized in the sense that a sample from Z generates a meaningful policy and that adjacent embeddings produce similar policies. To this end, we employ a CVAE architecture (illustrated in Figure 1(a)) to simultaneously learn a probabilistic encoder $q_{\phi_e}(\mathbf{z}|\mathbf{w})$ and a probabilistic decoder $p_{\phi_d}(a|\mathbf{o}, \mathbf{z})$.

Let the prior $p(\mathbf{z})$ be the standard multivariate Gaussian $N(\mathbf{z}; \mathbf{0}, \mathbf{I})$. Assume the variational approximate posterior $q_{\phi_e}(\mathbf{z}|\mathbf{w})$ be a multivariate Gaussian with a diagonal covariance $N(\mathbf{z}; \mu, \sigma^2\mathbf{I})$, where the mean μ and the standard deviation σ are outputs of a nonlinear function parameterized by ϕ^e with input $\mathbf{w}$. The K dimensional one-hot vector $\mathbf{w} = [w_1, w_2, .., w_K]$ is a policy label to an observation-action pair $(\mathbf{o}, a)$, such that $(\mathbf{o}, a, \mathbf{w})$ with $w_k = 1.0$ is produced by the kth opponent policy from Π^{Train}. The decoder $p_{\phi_d}(a|\mathbf{o}, \mathbf{z})$ outputs the probability of taking action a, conditional on an observation $\mathbf{o}$ and a sample from the posterior $q_{\phi_e}(\mathbf{z}|\mathbf{w})$. Similar to the Word2Vec method (Mikolov et al., 2013) where a word is input as a one-hot vector, our encoder $q_{\phi_e}(\mathbf{z}|\mathbf{w})$ takes the one-hot vector $\mathbf{w}$, which indexes a policy from Π^{Train}, as input. Accordingly, we name our policy embedding learning method as Policy2Emb.

Following the ELBO definition and using the reparameterization trick (Kingma & Welling, 2014), the training objective $L(\phi^e, \phi^d; (\mathbf{o}, a, \mathbf{w}))$ in Policy2Emb for a data point $(\mathbf{o}, a, \mathbf{w})$ can be estimated as:

$$\frac{\beta}{2}\sum_{i=1}^{J}[1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2] + \frac{1}{L}\sum_{l=1}^{L}\log p_{\phi_d}(a|\mathbf{o}, \mathbf{z}^{(l)}),$$

where $\mathbf{z}^{(l)} = \mu + \sigma \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim N(0, \mathbf{I})$. J is the embedding dimension, and L is the number of samples used.

Policy or skill embeddings have been extensively studied in previous literature. To name a few, Hausman et al. (2018) use variational inference for offline training of an embedding space of skills. Some methods (Grover et al., 2018; Rabinowitz et al., 2018; Xie et al., 2020; Raileanu et al., 2020; Papoudakis et al., 2021) learn only discriminative policy embeddings using some autoencoder architectures. Arnekvist et al. (2019) learn variational embeddings of optimal Q values for different tasks. A variational recurrent ladder encoder is developed towards applications in sports (Liu et al., 2020). The encoders of the VAEs in Papoudakis & Albrecht (2020) and Zintgraf et al. (2021a) depend on previous actions/states/rewards, which introduce unnecessary and interfering signals to encode a policy. Policy2Emb differs from previous policy embedding learning methods mainly in what our embedding represents (i.e., different opponent policies) and how it is trained: Policy2Emb employs a novel CVAE architecture to decouple the policy embedding learning from the representation learning of other unnecessary (for encoding a policy) information, which greatly improves both the discrimination and generalization of the learned policy embedding in the context of opponent modelling.

3.2. Offline Conditional RL

The purpose of our offline conditional RL in GSCU is to obtain a single best response against any opponent policy. Some previous methods (Johanson et al., 2007; Zheng et al., 2018) compute a separate best response against each opponent policy in Π^{Train}, which is not efficiently scalable with the size of Π^{Train} and may not generalize well outside Π^{Train}. In contrast, we leverage the policy embedding $\mathbf{z}$ learned by Policy2Emb to represent an opponent policy. We treat $\mathbf{z}$ as a new type of observation and train a single policy $\pi_1(\mathbf{o}, \mathbf{z}; \theta)$, where each assignment of $\mathbf{z}$ produces an approximate best response against the corresponding opponent.

We expect good generalization of $\pi_1(\mathbf{o}, \mathbf{z}; \theta)$ across the policy embedding space Z because of the regularization and discrimination training objectives in Policy2Emb. It is also worth noting that the training data for Policy2Emb is produced using opponent policies from Π^{Train}. Hence, the extent to which Π^{Train} is representative of the whole opponent policy space has a large influence on the generalization performance of $\pi_1(\mathbf{o}, \mathbf{z}; \theta)$ outside Π^{Train}. For this reason, we prefer Π^{Train} to be a diverse set of opponent policies, and we leave the generation of diverse opponent policies as future work.

In practice, we employ a single agent RL method to train $\pi_1(\mathbf{o}, \mathbf{z}; \theta)$. At each training episode, we randomly sample an opponent, represented by $\mathbf{w}$, from Π^{Train} and create the corresponding single agent adversary environment, where the input $\mathbf{z}$ is sampled according to the probabilistic encoder $p_{\phi_e}(\mathbf{z}|\mathbf{w})$. The pseudocode of the offline conditional RL in GSCU is given in Appendix A.

3.3. Online Bayesian Belief Update and Policy Selection

During online test, at the beginning of episode j, we compute a posterior distribution $p(\mathbf{z}_j|D_j)$ of the current opponent policy embedding $\mathbf{z}_j$ given the historic data set D_j. We fit an approximate distribution $q(\mathbf{z}_j)$ to the intractable posterior $p(\mathbf{z}_j|D_j)$ using variational inference (Kingma & Welling, 2014). Let $q(\mathbf{z}_j)$ takes the form of a multivariate Gaussian with a diagonal covariance $\mathcal{N}(\mathbf{z}; \mu_j, \sigma_j^2\mathbf{I})$, the KL divergence $D_{KL}(q(\mathbf{z}_j)||p(\mathbf{z}_j|D_j))$ can be minimized via maximizing the corresponding ELBO: $L(\mu_j, \sigma_j; D_j) = -D_{KL}(q(\mathbf{z}_j)||p(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z}_j)}[\log p(D_j|\mathbf{z}_j)]$. Treating instances $(\mathbf{o}_{-1}, a_{-1})$ in D_j as i.i.d samples and $p(\mathbf{o}_{-1}|\mathbf{z})$ as a constant, $L(\mu_j, \sigma_j; D_j)$ with constant terms eliminated is:

$$-D_{KL}(\mathcal{N}(\mathbf{z}; \mu_j, \sigma_j^2\mathbf{I})||p(\mathbf{z}))+ \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(\mathbf{z};\mu_j;\sigma_j^2\mathbf{I})}\Big[\sum_{(\mathbf{o}_{-1};a_{-1})\in D_j} \log p_{\theta^d}(a_{-1}|\mathbf{o}_{-1},\mathbf{z})\Big], \quad (2)$$

where the parameters θ^d are pretrained offline in Policy2Emb. In practice, we compute $q(\mathbf{z}_j)$, i.e., μ_j and σ_j every M^{VI} episodes, where the posterior from the last step serves as the prior $p(\mathbf{z})$ for the current step, using only newly collected data.

For the main agent, denote the regret of always using policy $\pi_{1;j} = \pi_1$ by $R_T(\pi_1)$ and the regret of always using policy $\pi_{1;j} = \pi_{1;j}^{\mathrm{RL}}$ by $R_T(\pi_1^{\mathrm{RL}})$, where $\pi_{1;j}^{\mathrm{RL}} = \pi_1(\mathbf{o}, \mu_j; \theta)$ at each episode j. Intuitively, $\pi_{1;j}^{\mathrm{RL}}$ is an approximate best response to the opponent policy $\pi_{-1;j}$, which is unknown to the main agent. More precisely, we can lower bound the expected returns of playing $\pi_{1;j}^{\mathrm{RL}}$ against $\pi_{-1;j}$ by proving:

$$\begin{aligned} u_1(\pi_{1;j}^{\mathrm{RL}}, \pi_{-1;j}) \geq\; & u_1(BR(\hat{\pi}_{-1;j}), \hat{\pi}_{-1;j}) \\ & - R^{RL}(\hat{\pi}_{-1;j}) - D(\pi_{-1;j}\|\hat{\pi}_{-1;j}), \end{aligned} \quad (3)$$

where $\hat{\pi}_{-1;j}$ with $\hat{\pi}_{-1;j}(a|\mathbf{o}) = p_{\theta^d}(a|\mathbf{o}, \mu_j)$ represents the inferred opponent policy. The term $u_1(BR(\hat{\pi}_{-1;j}), \hat{\pi}_{-1;j})$ is the best expected returns against $\hat{\pi}_{-1;j}$. The term $R^{RL}(\hat{\pi}_{-1;j})$ is the performance gap of playing $\pi_{1;j}^{\mathrm{RL}}$ instead of $BR(\hat{\pi}_{-1;j})$ against $\hat{\pi}_{-1;j}$: $R^{RL}(\hat{\pi}_{-1;j}) = u_1(BR(\hat{\pi}_{-1;j}), \hat{\pi}_{-1;j}) - u_1(\pi_{1;j}^{\mathrm{RL}}, \hat{\pi}_{-1;j})$. $R^{RL}(\hat{\pi}_{-1;j})$ is expected to be small, as $\pi_{1;j}^{\mathrm{RL}}$ is trained against $\hat{\pi}_{-1;j}$ via single agent RL offline. The term $D(\pi_{-1;j}\|\hat{\pi}_{-1;j}) = \frac{2}{1-\gamma}\sqrt{\mathbb{E}_{\mathbf{o}}[D_{KL}(\pi_{-1;j}(\mathbf{o})\|\hat{\pi}_{-1;j}(\mathbf{o}))]}$, which reflects the quality of the approximation to $\pi_{-1;j}$ using $\hat{\pi}_{-1;j}$.

Ideally, we expect that $\hat{\pi}_{-1;j}$ approximates $\pi_{-1;j}$ and $\pi_{1;j}^{\mathrm{RL}}$ approximates $BR(\pi_{-1;j})$. As a result, following $\pi_{1;j}^{\mathrm{RL}}$ may be much more profitable than π_1. However, in practice, the regret of always following $\pi_{1;j}^{\mathrm{RL}}$ can be high if $D(\pi_{-1;j}\|\hat{\pi}_{-1;j})$ is large, due to either that $\pi_{-1;j}$ is very different from offline opponent policies in Π^{Train} or that little data is available to make an effective online inference about $\pi_{-1;j}$. In order to improve upon always following either π_1 or $\pi_{1;j}^{\mathrm{RL}}$, GSCU employs the EXP3 (Auer et al., 2002b) algorithm to select between π_1 and $\pi_{1;j}^{\mathrm{RL}}$ during online test. Denoting the policy GSCU plays at each episode j by $\pi_{1;j}^{\mathrm{EXP3}}$, we can prove the following theorem.

Theorem 3.1. *When* $\eta = \min\left\{1, \sqrt{\frac{2\ln 2}{(e-1)T}}\right\}$*, the regret of playing* $\pi_{1;j}^{EXP3}$ *for* T *episodes is upper bounded:*

$$R_T(\pi_{1;j}^{EXP3}) \leq 3.1\sqrt{T} + \min\left\{R_T(\pi_1), R_T(\pi_{1;j}^{RL})\right\}.$$

Equation 3 and the above theorem are proven in Appendix B and C respectively. In the theorem, the first term $3.1\sqrt{T}$ is incurred by the bandit. The regret $R_T(\pi_1)$ may increase linearly with T, as π_1 is a fixed policy. Hence, $R_T(\pi_{1;j}^{\mathrm{EXP3}})$ is no worse than $R_T(\pi_1)$ and is much better if $\pi_{1;j}^{\mathrm{RL}}$ is effectively exploiting the opponent. The overall online Bayesian belief update and policy selection procedure in GSCU are summarized in Algorithm 1.

Algorithm 1 Online Bayesian Belief Update and Policy Selection in GSCU

Input: θ^d, θ, π_1, η, buffer $B = \emptyset$, M^{VI}
Initialize $\omega_1[i] = 1.0$, $\forall i = 1, 2$; $\mathbf{z}_1 = \mathbf{0}$
for episode $j = 1$ **to** T **do**
  if $j \mod M^{VI} == 0$ **then**
    Compute μ_j and σ_j via variational inference by optimizing Equation 2 using data in B
    $\mathbf{z}_j \leftarrow \mu_j$
    $B \leftarrow \emptyset$
  end if
  Set $p_j[i] = (1-\eta)\frac{\omega_j[i]}{\omega_j[1]+\omega_j[2]} + \frac{\eta}{2}, \forall i = 1, 2$
  $\pi_{1;j} \leftarrow \begin{cases} \pi_1, & \text{with probability } p_j[1], \\ \pi_1(\mathbf{o}, \mathbf{z}_j; \theta), & \text{otherwise} \end{cases}$
  Play $\pi_{1;j}$ against $\pi_{-1;j}$; Obtain a sampled returns $\hat{G}_{1;j}$ and an opponent trajectory $\tau_{-1;j}$
  $B \leftarrow B \cup \tau_{-1;j}$
  Update $\omega_{j+1}[i] \leftarrow \omega_j[i], \forall i = 1, 2$:
  if π_1 is selected **then**
    $\omega_{j+1}[1] \leftarrow \omega_j[1]\exp(\eta\frac{\hat{G}_{1;j}}{2p_j[1]})$
  else
    $\omega_{j+1}[2] \leftarrow \omega_j[2]\exp(\eta\frac{\hat{G}_{1;j}}{2p_j[2]})$
  end if
  $\mathbf{z}_{j+1} \leftarrow \mathbf{z}_j$
end for

4. Related Work

We review related methods that can be applied to our setting of competing online against unknown opponents. According to the way the main agent policy is determined at each episode during online execution, there are generally three categories of methods from the literature.

Playing a fixed policy. In two-player zero-sum games, the default goal of the fixed policy is a NE policy, which can be approximated via self-play training methods such as regret minimization algorithms (Zinkevich et al., 2007; Fu et al., 2021). In multiagent competitive environments, the fixed main agent policy is often obtained via a centralized training procedure (Hong et al., 2018; Foerster et al., 2018a; Lowe et al., 2017; Foerster et al., 2018b; Yang et al., 2018; Wen et al., 2019; Hernandez-Leal et al., 2019a).

Opponent modelling within an episode. The main agent conditions its policy on not only its own observation but also additional information about the opponent, which is either collected or inferred using previous interactions with the opponent within the current episode. He et al. (2016) propose a neural architecture that jointly learns a main agent policy and the opponent behavior, using additional hand crafted opponent features that summarize previous interactions with the opponent. Assuming the opponent has the same goal structure as the main agent, Raileanu et al. (2018) infer the opponent's goal using the main agent policy net and then condition the main agent policy additionally on the inferred goal. Recurrent VAEs (Papoudakis & Albrecht, 2020; Zintgraf et al., 2021a) are employed to encode a compact variational embedding of previous interactions with the opponent, on which the main agent policy conditions. Recently, Papoudakis et al. (2021) propose a similar approach using an autoencoder architecture for situations where the opponent's observation and action are not available during online execution.

Opponent modelling across episodes. Our approach GSCU belongs to this category, where data from previous episodes is analysed to help decide the main agent policy for the current episode. Assuming the opponent plays a fixed set of policies known to the main agent, Johanson et al. (2007) and Bard et al. (2013) create a mixture-of-expert counter-strategies in offline training against this set. During online test, the counter-strategy against the current opponent is selected using a bandit algorithm. One recent similar approach (DiGiovanni & Tewari, 2021) combines Thompson sampling with change detection against opponents that switch among several stationary policies.

Bayesian Policy Reuse (BPR) (Rosman et al., 2016) provides a framework for adapting to different tasks in a single agent environment. BPR+ (Hernandez-Leal et al., 2016) extends BPR to multiagent settings. Zheng et al. (2018) improve BPR+ with deep neural networks, a rectified belief model, and policy distillation. During the policy reuse stage, the main agent policy is selected based on the belief model, which is a categorical distribution over a discrete number of opponent policies. Once a new opponent policy is detected based on a moving average reward signal, a learning stage is started by training a DQN, which is initialized with a distilled neural network.

Meta RL (Al-Shedivat et al., 2018; Nagabandi et al., 2019; Kim et al., 2021; Wu et al., 2021; Zintgraf et al., 2021b) can be potentially applied to competing online against unknown opponents. These algorithms leverage data from training tasks to train a learning procedure that can quickly adapt to online test tasks. A general assumption in Meta RL is that the meta-training tasks and online test tasks are drawn from the same task distribution. Yet, in the domain of competing online against unknown opponents, it is difficult to enumerate all possible opponent policies offline, and even more difficult is simulating different types of nonstationarity of opponent policies.

Differing from prior methods in this category, GSCU makes no assumption about the online opponent policy. During online test, GSCU computes a posterior of the current opponent policy embedding, without deciding which type the current opponent belongs to or conducting any form of change detection. The real-time *greedy* policy $\pi_{1;j}^{RL}$ in GSCU generalizes to online unknown opponents via the single best response $\pi_1(\mathbf{o}, \mathbf{z};\ )$ and the decoder of the CVAE in Policy2Emb, both of which are trained offline using ^{train}. More importantly, GSCU selects between the real-time *greedy* policy $\pi_{1;j}^{RL}$ and the worst-case optimal *conservative* policy π_1 using EXP3, resulting in a theoretically better regret than adhering to either.

5. Experiments

The goal of the experimental study is to test the performance of different methods on competing online against unknown and nonstationary opponents. We also validate the effectiveness of each component in GSCU.

5.1. Experimental Setup

Competitive environments. We consider two competitive multiagent benchmarks: Kuhn poker (Kuhn, 2016) and grid-world Predator Prey (PP) (Mordatch & Abbeel, 2018). Kuhn poker is a two-player zero-sum imperfect-information simplified poker game. The PP environment studied in this paper is partially observable for the prey (the main agent) and fully observable for the three predators (the opponents). More details about the two benchmarks are given in Appendix D and E.

Comparing methods. GSCU is compared with:

- Playing the fixed policy π_1, which will be a NE policy in Kuhn poker and a robust policy trained offline using PPO for both sides in PP.
- DRON (He et al., 2016): A method that does opponent modelling via collecting opponent information within the current episode. The policy that conditions additionally on this information is trained offline.
- LIAM (Papoudakis et al., 2021): A method that does opponent modelling via summarizing an embedding using

previous interactions within the current episode. Both the embedding function and the policy that depends on the embedding are trained offline.

- Deep BPR+ (Zheng et al., 2018): A method that does opponent modelling across episodes. Deep BPR+ selects from a library of offline trained counter policies for the main agent using a belief model, which is a categorical distribution over identified opponents. A new counter policy is learned online if the current opponent is identified new according to a moving average reward signal.
- Tracking: Continuously updating the main agent policy online using PPO.
- GSCU-Greedy: Always playing the real-time *greedy* policy $\pi_1(\mathbf{o},\ ;\ )$, with being the inferred posterior mean of the current opponent policy embedding.

Training and test protocols. As mentioned in section 2.1, we assume the availability of π_1 and an opponent policy library ^{Train}. We introduce another opponent policy library ^{Test} for online test, where $^{Test} \setminus\ ^{Train} = ;$. DRON and LIAM train a single policy against ^{Train}. Deep BPR+ prepares a set of counter policies by training a separate policy using PPO against each opponent policy in ^{Train}. Afterwards, the performance model and the distilled policy network in Deep BPR+ are trained accordingly. For initializing the policy of Tracking, we train a single policy against ^{Train} using PPO. For GSCU, the training data for Policy2Emb is generated by playing π_1 against each opponent policy in ^{Train}. Afterwards, the approximate best response $\pi_1(\mathbf{o}, \mathbf{z};\ )$ is trained against ^{Train} via the conditional RL in GSCU.

For online test, we create four types of sequences of opponents: "seen", "unseen", "mix", and "adaptive". For the "seen" sequence, we randomly sample an opponent from ^{Train} every M episodes, during which the opponent policy stays stationary. We sample N^o times, which results in a total number of $M\ \ N^o$ episodes. The same procedure applies to the "unseen" and "mix" sequences, except that we sample opponent policies from ^{Test} and ^{Train} [^{Test} respectively. For the "adaptive" sequence, the opponent, initialized with the policy trained using PPO for both sides, continuously updates its own policy using PPO.

For results that require repeated runs, we average over five random seeds, with shaded areas (plots) and standard deviation error bars (histograms). More details about the settings of training and test, including the implementations and hyperparameters, are given in Appendix F and G.

5.2. Competing Online against Unknown Opponents

In this experiment, we study the average performance of different methods competing against the four types of sequences of opponents ("seen", "unseen", "mix", and "adaptive"). Note that both the opponent policies[1] and the way how the opponent changes its policy are unknown to all methods.

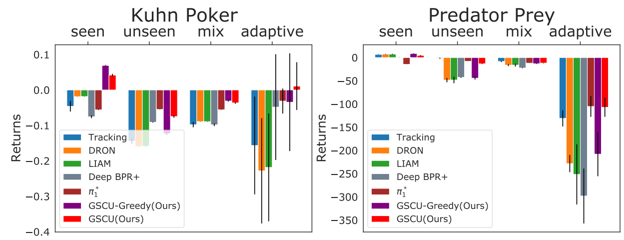

Figure 2. The average returns of different methods competing online against different types of sequences of opponents.

For the average performance on Kuhn poker shown in Figure 2, GSCU performs competitively across the four settings. In the "seen" setting, GSCU-Greedy performs the best, which suggests that the approximate best response is well trained and that the online Bayesian inference is effective in GSCU. In the "unseen" setting, π_1 performs the best as expected because ^{Test} is selected to be significantly different from ^{Train}. The reason why GSCU comes as the second best and performs better than GSCU-Greedy is that π_1 is one of the two arms in GSCU. The better performance of GSCU-Greedy compared to DRON and LIAM suggests better generalization of GSCU-Greedy. In the "mix" setting, GSCU-Greedy and GSCU are the top-two performing methods, which indicates that the online Bayesian inference of GSCU is able to identify "seen" opponents in a "mixed" sequence. This suggests better discrimination of GSCU-Greedy over other methods. In the "adaptive" setting where the opponent is doing online PPO and thus can potentially change to any policy, GSCU performs the best, which means GSCU is least exploitable among all methods. GSCU-Greedy performs better than other opponent modelling methods, which further demonstrates its better discrimination and generalization in competing online against unknown opponents.

For the average performance on PP shown in Figure 2, similar conclusions can be made to that on Kuhn poker. DRON and LIAM perform relatively better on PP than on Kuhn poker, and one reason may be that PP has significantly longer episodes than Kuhn poker. Another observation is that Tracking and π_1 perform competitively compared to other opponent modelling methods, which indicates that in some environment competing online against unknown opponents without opponent modelling, such as Tracking or playing π_1, may have satisfactory performance.

[1]The three opponents in PP share the same policy at each episode j: $\pi_{-1,j}$. Yet, the actions can be different because of the stochasticity of $\pi_{-1,j}$ and different observations. We learn a single embedding $\mathbf{z}$ using trajectories of the three opponents in PP.

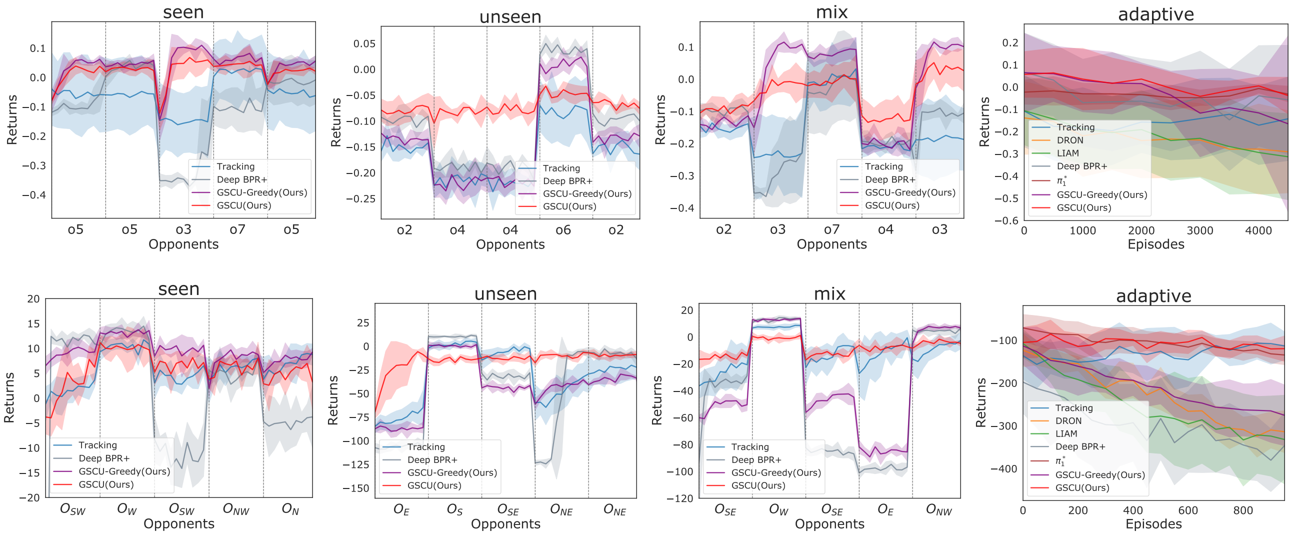

Figure 3. Online adaptation performance of different methods on Kuhn poker (top) and PP (bottom). For the "seen", "unseen", and "mix" sequences, we plot results only for methods that learn cross episodes.

In Appendix H, we summarize the average and worst-case performance of different methods across the four settings, where GSCU achieves the best performance. The results suggest that GSCU is robust and has low regret against a wide range of online unknown opponents. In addition, we perform a sensitivity analysis of GSCU to the change frequency M, where the performance of GSCU degrades slightly as M decreases dramatically.

We further illustrate the online adaptation process of each method in Figure 3. For each method, we save its checkpoint every several (100 in Kuhn poker and 20 in PP) episodes. We evaluate each checkpoint against its current opponent using extra (10000 in Kuhn poker and 100 in PP) episodes to obtain an average returns. On either Kuhn poker or PP, we observe that the performance of GSCU is more stable and better than other adaptive methods. In particular, GSCU is least exploitable (same as π_1) in the "adaptive" setting.

5.3. Analyzing the Learned Policy Embeddings

We use Kuhn poker to analyze the latent policy embedding space learned by Policy2Emb, as it has a well defined and structured policy space (Southey et al., 2009). More results about the learned policy embeddings in both Kuhn poker and PP are given in Appendix D and E respectively.

Figure 4 presents the visualisation of the learned policy embedding space in Kuhn poker on ^{Train} [^{Test} and the true policy space. Note that, for online test, Policy2Emb learns using only ^{Train}, the results of which are plotted in Appendix D. The learned policy embeddings in Figure 4 are well structured in the sense that it is almost a mirror image of the ground truth. This is largely because the policy embedding in Policy2Emb is produced by a probabilistic encoder that depends solely on an opponent index. In the mean time, the embedding, combined with an opponent observation, is trained to predict the opponent's action within the framework of CVAE. Yet, the policy embeddings, which encode information of both observation and action, have multiple clusters for the same opponent policy (e.g., Figure 4a in Papoudakis et al. (2021)).

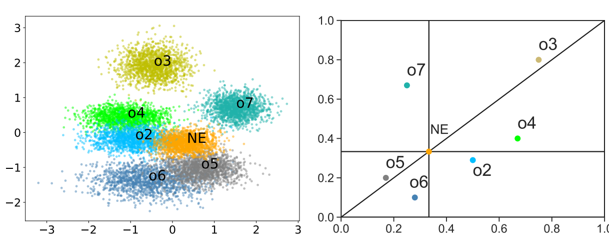

Figure 4. The policy embeddings learned by Policy2Emb in Kuhn poker on ^{Train} [^{Test} (left) and the true policy space (right).

5.4. Analyzing the Conditional RL

To evaluate the performance of the conditional RL in GSCU, we compare its learning process to those of PPO, DRON, and LIAM. Note that the purpose here is to train best responses against opponent policies in ^{Train}. PPO trains a single common approximate best response. DRON trains an approximate best response with additional opponent features as input, using both RL and supervision (regression to the policy parameters in Kuhn poker and classification of the opponent index in PP) signals. LIAM conditions its approximate best response on a recurrent encoder, which is trained via a reconstruction loss and the original RL loss. In contrast, the conditional RL in GSCU trains an approximate best response taking the opponent policy embedding **z** (learned by Policy2Emb) as additional input. For comparison, we also use PPO to train K separate best responses, with each targeting a separate opponent policy in ^{Train}.

We evaluate each method by saving a checkpoint every several (10000 in Kuhn poker and 500 in PP) training episodes and playing it against opponents from ^{Train} using enough episodes (10000 in Kuhn poker and 100 in PP) to obtain an average performance, the results of which are plotted in Figure 5. The conditional RL in GSCU performs the best on both benchmarks, which suggests that the opponent policy embedding learned by Policy2Emb facilitates the effective learning of a single approximate best response against different opponents. It is also worth noting that, with the consideration of sample efficiency, the average performance of K separate best responses has negligible advantage over the conditional RL in GSCU.

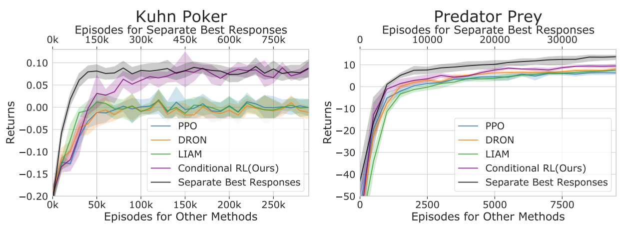

Figure 5. The offline RL training process of different methods.

5.5. Analyzing the Online Bayesian Inference

Based on the offline trained probabilistic decoder in Policy2Emb, GSCU makes online Bayesian inference of the opponent policy embedding using opponent historic trajectory data. The posterior mean of the online inferred opponent policy embedding is input to the offline trained approximate best response to obtain the real-time *greedy* policy in GSCU. In comparison, Deep BPR+ makes online inference by calculating the probability the current opponent policy is one of the opponent policies from ^{Train}. The categorical distribution over ^{Train} is obtained by combining a normalized KL-divergence score with a performance model. To deal with the situation where the current opponent policy is identified outside of ^{Train}, a novelty detection mechanism is introduced using a moving average reward signal.

For methods that make explicit inference about the online opponent policy, we hypothesize that inference in our opponent policy embedding space is more effective than calculating a discrete probability which category the current opponent policy belongs to. A straightforward reason is that the policy space is continuous, and it may be impractical to discretize the policy space into several categories. Hence, we examine the online inference performance of both GSCU and deep BPR+. For GSCU, we report the embedding error, which is the Euclidean distance between the estimated mean and the true mean of the opponent policy embedding. For Deep BPR+, we report the probability of the ground truth opponent. The corresponding results on Kuhn poker are plotted in Figure 6, which shows that the embedding error of GSCU decreases steadily on opponents from ^{Train} in both "seen" and "mix" sequences. Yet, Deep BPR+ sometimes fails to identify the right opponent in time.

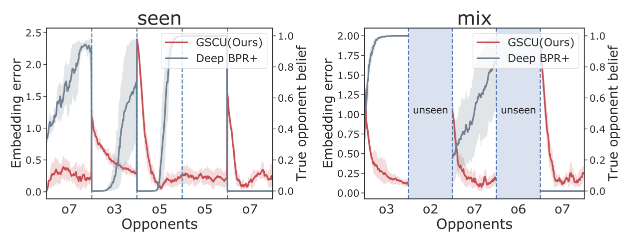

Figure 6. Online inference performance of GSCU and Deep BPR+.

6. Conclusion and Future Work

This paper develops a new approach (i.e., GSCU) for competing online against unknown opponents. GSCU selects between playing greedily and conservatively against the current opponents. The *conservative* policy is a fixed offline trained policy, which hopefully has the best worst-case performance. The real-time *greedy* policy is an offline trained approximate best response, conditioning additionally on an online inferred opponent policy embedding. We introduce a novel way (i.e., Policy2Emb) of learning opponent policy embeddings offline, which is of independent interest to policy representation learning. Our offline trained approximate best response generalizes to different opponents via the opponent policy embeddings learned by Policy2Emb. We prove a lower bound on the performance of the real-time *greedy* policy in GSCU. Moreover, we prove that the regret of selecting between playing greedily and conservatively using EXP3 is smaller than that of adhering to either.

Experimental results on both Kuhn poker and PP demonstrate the effectiveness of GSCU, compared to previous state-of-the-art methods. The learned policy embeddings by Policy2Emb are validated via both visualization and the conditional RL. We also show that the online variational inference of opponent policy embeddings in GSCU is more effective than inferring a categorical distribution over a discrete number of opponent policies.

One direction of future work is investigating the influence of policy diversity in ^{Train} on the online performance of GSCU, considering the performance gap between 'seen' and 'unseen' sequences. GSCU builds on the implicit assumption that online opponents are, to some extent, exploitable and predictable, so another interesting direction is testing GSCU against humans, who are generally known as being limited rational (Rubinstein, 1998) and partly inertial (Alós-Ferrer et al., 2016) in their decision making.

References

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *International Conference on Learning Representations*, 2018.

Albrecht, S. V. and Stone, P. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

Alós-Ferrer, C., Hügelschäfer, S., and Li, J. Inertia and decision making. *Frontiers in Psychology*, 7:169, 2016.

Arnekvist, I., Kragic, D., and Stork, J. A. Vpe: Variational policy embedding for transfer reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 36–42. IEEE, 2019.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002a.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

Bard, N., Johanson, M., Burch, N., and Bowling, M. Online implicit agent modelling. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 255–262, 2013.

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

Brown, N. and Sandholm, T. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

DiGiovanni, A. and Tewari, A. Thompson sampling for markov games with piecewise stationary opponent policies. In *Uncertainty in Artificial Intelligence*, pp. 738–748. PMLR, 2021.

Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 122–130, 2018a.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Fu, H., Li, C., Liu, X., Gao, J., Celikyilmaz, A., and Carin, L. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. *arXiv preprint arXiv:1903.10145*, 2019.

Fu, H., Liu, W., Wu, S., Wang, Y., Yang, T., Li, K., Xing, J., Li, B., Ma, B., Fu, Q., et al. Actor-critic policy optimization in a large-scale imperfect-information game. In *International Conference on Learning Representations*, 2021.

Grover, A., Al-Shedivat, M., Gupta, J., Burda, Y., and Edwards, H. Learning policy representations in multiagent systems. In *International Conference on Machine Learning*, pp. 1802–1811. PMLR, 2018.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

He, H., Boyd-Graber, J., Kwok, K., and Daumé III, H. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1804–1813. PMLR, 2016.

Hernandez-Leal, P., Taylor, M. E., Rosman, B., Sucar, L. E., and De Cote, E. M. Identifying and tracking switching, non-stationary opponents: A bayesian approach. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Hernandez-Leal, P., Kaisers, M., Baarslag, T., and de Cote, E. M. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

Hernandez-Leal, P., Kartal, B., and Taylor, M. E. Agent modeling as auxiliary task for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pp. 31–37, 2019a.

Hernandez-Leal, P., Kartal, B., and Taylor, M. E. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019b.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2014.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Hong, Z.-W., Su, S.-Y., Shann, T.-Y., Chang, Y.-H., and Lee, C.-Y. A deep policy inference q-network for multi-agent systems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1388–1396, 2018.

Johanson, M., Zinkevich, M., and Bowling, M. Computing robust counter-strategies. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pp. 721–728, 2007.

Kim, D. K., Liu, M., Riemer, M. D., Sun, C., Abdulhai, M., Habibi, G., Lopez-Cot, S., Tesauro, G., and How, J. A policy gradient algorithm for learning to learn in multiagent reinforcement learning. In *International Conference on Machine Learning*, pp. 5541–5550. PMLR, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Kuhn, H. W. A simplified two-person poker. In *Contributions to the Theory of Games (AM-24), Volume I*, pp. 97–104. Princeton University Press, 2016.

Lai, T. L. and Robbins, H. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1): 4–22, 1985.

Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V. F., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., Vylder, B. D., Saeta, B., Bradbury, J., Ding, D., Borgeaud, S., Lai, M., Schrittwieser, J., Anthony, T. W., Hughes, E., Danihelka, I., and Ryan-Davis, J. Openspiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

Lattimore, T. and Szepesvári, C. *Bandit algorithms*. Cambridge University Press, 2020.

Liu, G., Schulte, O., Poupart, P., Rudd, M., and Javan, M. Learning agent representations for ice hockey. *Advances in Neural Information Processing Systems*, 33, 2020.

Lowe, R., WU, Y., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems*, 30:6379–6390, 2017.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Mordatch, I. and Abbeel, P. Emergence of grounded compositional language in multi-agent populations. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019.

Papoudakis, G. and Albrecht, S. V. Variational autoencoders for opponent modeling in multi-agent systems. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2020.

Papoudakis, G., Christianos, F., and Albrecht, S. Agent modelling under partial observability for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Parker-Holder, J., Pacchiano, A., Choromanski, K. M., and Roberts, S. J. Effective diversity in population based reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Rabinowitz, N., Perbet, F., Song, F., Zhang, C., Eslami, S. A., and Botvinick, M. Machine theory of mind. In *International Conference on Machine Learning*, pp. 4218–4227. PMLR, 2018.

Raileanu, R., Denton, E., Szlam, A., and Fergus, R. Modeling others using oneself in multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4257–4266. PMLR, 2018.

Raileanu, R., Goldstein, M., Szlam, A., and Fergus, R. Fast adaptation to new environments via policy-dynamics value functions. In *International Conference on Machine Learning*, pp. 7920–7931. PMLR, 2020.

Rosman, B., Hawasly, M., and Ramamoorthy, S. Bayesian policy reuse. *Machine Learning*, 104(1):99–127, 2016.

Rubinstein, A. *Modeling bounded rationality*. MIT press, 1998.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897. PMLR, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shapley, L. S. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Southey, F., Hoehn, B., and Holte, R. C. Effective short-term opponent exploitation in simplified poker. *Machine Learning*, 74(2):159–189, 2009.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Walker, J., Doersch, C., Gupta, A., and Hebert, M. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pp. 835–851. Springer, 2016.

Wen, Y., Yang, Y., Luo, R., Wang, J., and Pan, W. Probabilistic recursive reasoning for multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2019.

Wu, Z., Li, K., Zhao, E., Xu, H., Zhang, M., Fu, H., An, B., and Xing, J. L2e: Learning to exploit your opponent. *arXiv preprint arXiv:2102.09381*, 2021.

Xie, A., Losey, D., Tolsma, R., Finn, C., and Sadigh, D. Learning latent representations to influence multi-agent interaction. In *Proceedings of the 4th Conference on Robot Learning*, 2020.

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. Mean field multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 5571–5580. PMLR, 2018.

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

Zheng, Y., Meng, Z., Hao, J., Zhang, Z., Yang, T., and Fan, C. A deep bayesian policy reuse approach against non-stationary agents. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 962–972, 2018.

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems*, 20:1729–1736, 2007.

Zintgraf, L., Devlin, S., Ciosek, K., Whiteson, S., and Hofmann, K. Deep interactive bayesian reinforcement learning via meta-learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1712–1714, 2021a.

Zintgraf, L., Schulze, S., Lu, C., Feng, L., Igl, M., Shiarlis, K., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021b.

A. The Pseudocode of the Conditional RL in GSCU

We illustrate the conditional RL procedure in GSCU using PPO in Algorithm 2. Note that any single agent RL method is applicable here.

Algorithm 2 Offline Conditional RL in GSCU

Input: ϕ^e, Π^{Train}, buffer $B = \emptyset$, training batch size
Initialize the parameters θ in $\pi_1(\mathbf{o}, \mathbf{z}; \theta)$
for episode $= 1, 2, \ldots$ **do**
  Sample an opponent policy $\pi_{-1}^{(k)}$ from Π^{Train} and set the corresponding one-hot vector $\mathbf{w}$
  for time step $t = 1, 2, \ldots, H$ **do**
    Sample an opponent policy embedding $\mathbf{z} \sim q_{\phi^e}(\mathbf{z}|\mathbf{w})$
    Sample the main agent action $a_{1,t} \sim \pi_1(\mathbf{o}, \mathbf{z}; \theta)$ and the opponent action $a_{-1,t} \sim \pi_{-1}^{(k)}$
    Step the environment and obtain a sample $(\mathbf{o}_{1,t}, \mathbf{z}, a_{1,t}, r_{1,t})$
    $B \leftarrow B \cup (\mathbf{o}_{1,t}, \mathbf{z}, a_{1,t}, r_{1,t})$
  end for
  if $|B| \geq$ training batch size, **then**
    Update θ using PPO on a batch of samples from B
    $B \leftarrow \emptyset$
  end if
end for

B. Proof of Equation 3

Proof. Equation 3 is the result of a known equation in single-agent RL:

$$v(\tilde{\pi}) = v(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_{\pi}(s, a), \tag{4}$$

where π and $\tilde{\pi}$ are two arbitrary policies; $v(\pi)$ and $v(\tilde{\pi})$ are the expected returns; $\rho_{\tilde{\pi}}(s)$ is the unnormalized discounted visiting frequencies; and A_{π} is the advantage function. A proof for this equation can be found in Schulman et al. (2015). In a multiagent environment defined in this paper, Equation 4 can be translated as

$$u_1(\tilde{\pi}_1, \pi_{-1}) = u_1(\pi_1, \pi_{-1}) + \sum_{\mathbf{o} \in O_1} \rho_{\tilde{\pi}_1, \pi_{-1}}(\mathbf{o}) \sum_{a \in A_1} \tilde{\pi}_1(a|o) A_{\pi_1}(\mathbf{o}, a), \tag{5}$$

where $\rho_{\tilde{\pi}_1, \pi_{-1}}(\mathbf{o})$ is the unnormalized discounted visiting frequencies according to the policy pair $(\tilde{\pi}_1, \pi_{-1})$, and $A_{\pi_1}(\mathbf{o}, a) = u_1^{\pi_1, \pi_{-1}}(\mathbf{o}, a) - \sum_a \pi_1(a|\mathbf{o}) u_1^{\pi_1, \pi_{-1}}(\mathbf{o}, a)$. From the perspective of the other player (Note that $u_1(\pi_1, \pi_{-1}) + u_{-1}(\pi_1, \pi_{-1}) = 0$), the equation can also be written as

$$u_1(\pi_1, \tilde{\pi}_{-1}) = u_1(\pi_1, \pi_{-1}) - \sum_{\mathbf{o} \in O_{-1}} \rho_{\pi_1, \tilde{\pi}_{-1}}(\mathbf{o}) \sum_{a \in A_{-1}} \tilde{\pi}_{-1}(a|\mathbf{o}) A_{\pi_{-1}}(\mathbf{o}, a), \tag{6}$$

where

$$\begin{aligned}
& \sum_{a \in A_{-1}} \tilde{\pi}_{-1}(a|\mathbf{o}) A_{\pi_{-1}}(\mathbf{o}, a) \\
= & \sum_{a \in A_{-1}} \tilde{\pi}_{-1}(a|\mathbf{o}) [u_{-1}^{\pi_1, \pi_{-1}}(\mathbf{o}, a) - \sum_{a \in A_{-1}} \pi_{-1}(a|\mathbf{o}) u_{-1}^{\pi_1, \pi_{-1}}(\mathbf{o}, a)] \\
= & \sum_{a \in A_{-1}} [\tilde{\pi}_{-1}(a|\mathbf{o}) - \pi_{-1}(a|\mathbf{o})] u_{-1}^{\pi_1, \pi_{-1}}(\mathbf{o}, a) \\
\leq & \sum_{a \in A_{-1}} |\tilde{\pi}_{-1}(a|\mathbf{o}) - \pi_{-1}(a|\mathbf{o})| \\
\leq & 2 \sqrt{D_{KL}(\tilde{\pi}_{-1}(a|\mathbf{o}) \| \pi_{-1}(a|\mathbf{o}))}.
\end{aligned} \tag{7}$$

The last inequality is because that $\frac{1}{4}\|\mathbf{p}-\mathbf{q}\|_1^2 \le D_{KL}(\mathbf{p}\|\mathbf{q})$ for any two probability vectors $\mathbf{p}$ and $\mathbf{q}$. So

$$\begin{aligned} u_1(\pi_1,\pi_{-1}) &\ge u_1(\pi_1,\hat{\pi}_{-1}) - 2\sum_{\mathbf{o}\in O_{-1}} \rho_{\pi_1;\pi_{-1}}(\mathbf{o})\sqrt{D_{KL}(\pi_{-1}(a|\mathbf{o})\|\hat{\pi}_{-1}(a|\mathbf{o}))} \\ &\ge u_1(\pi_1,\hat{\pi}_{-1}) - 2\frac{1}{1-\gamma}\mathbb{E}_{\mathbf{o}\sim\tilde{\rho}_{\pi_1;\pi_{-1}}}\sqrt{D_{KL}(\pi_{-1}(a|\mathbf{o})\|\hat{\pi}_{-1}(a|\mathbf{o}))} \\ &\ge u_1(\pi_1,\hat{\pi}_{-1}) - \frac{2}{1-\gamma}\sqrt{\mathbb{E}_{\mathbf{o}}D_{KL}(\pi_{-1}(a|\mathbf{o})\|\hat{\pi}_{-1}(a|\mathbf{o}))}. \end{aligned} \tag{8}$$

The last inequality holds according to Jensen's inequality. In the above equation, $\tilde{\rho}_{\pi_1;\pi_{-1}}$ is the normalized visiting distribution. Note that $\rho_{\pi_1;\pi_{-1}}(\mathbf{o}) \le \frac{1}{1-\gamma}\tilde{\rho}_{\pi_1;\pi_{-1}}(\mathbf{o})$. Therefore, for the left side in Equation 3, we have

$$\begin{aligned} u_1(\pi^{\mathrm{RL}}_{1;j},\pi_{-1;j}) &\ge u_1(\pi^{\mathrm{RL}}_{1;j},\hat{\pi}_{-1;j}) - \frac{2}{1-\gamma}\sqrt{\mathbb{E}_{\mathbf{o}}D_{KL}(\pi_{-1;j}(\mathbf{o})\|\hat{\pi}_{-1;j}(\mathbf{o}))} \\ &= u_1(BR(\hat{\pi}_{-1;j}),\hat{\pi}_{-1;j}) - \left(u_1(BR(\hat{\pi}_{-1;j}),\hat{\pi}_{-1;j}) - u_1(\pi^{\mathrm{RL}}_{1;j},\hat{\pi}_{-1;j})\right) - D(\pi_{-1;j}\|\hat{\pi}_{-1;j}) \\ &= u_1(BR(\hat{\pi}_{-1;j}),\hat{\pi}_{-1;j}) - R^{RL}(\hat{\pi}_{-1;j}) - D(\pi_{-1;j}\|\hat{\pi}_{-1;j}), \end{aligned} \tag{9}$$

which equals to the right side in Equation 3. ◻

C. Proof of Theorem 3.1

Proof. The regret of the bandit after T episodes is

$$R'_T = \max_{\pi\in\{\pi_1,\pi^{\mathrm{RL}}_{1;j}\}}\sum_{j=1}^{T}u_1(\pi,\pi_{-1;j}) - \sum_{j=1}^{T}u_1(\pi_{1;j},\pi_{-1;j}), \tag{10}$$

where $\pi_{1;j}$ is selected between π_1 and $\pi^{\mathrm{RL}}_{1;j}$ using EXP3. For a two-armed bandit , EXP3 with a learning rate $\eta \in (0,1]$ guarantees (Theorem 3.1 in Auer et al. (2002b)):

$$R'_T \le (e-1)\eta T + \frac{2\ln 2}{\eta}. \tag{11}$$

When $\eta = \min\left\{1,\sqrt{\frac{2\ln 2}{(e-1)T}}\right\}$, we have $R'_T \le 3.1\sqrt{T}$. Recall that the regret defined in Equation 1 is

$$R_T = \max_{\pi_1\in\Pi_1}\sum_{j=1}^{T}u_1(\pi_1,\pi_{-1;j}) - \sum_{j=1}^{T}u_1(\pi_{1;j},\pi_{-1;j}). \tag{12}$$

Therefore, we have

$$\begin{aligned} R_T(\pi^{\mathrm{EXP3}}_{1;j}) &= R'_T + \max_{\pi_1\in\Pi_1}\sum_{j=1}^{T}u_1(\pi_1,\pi_{-1;j}) - \max_{\pi\in\{\pi_1,\pi^{\mathrm{RL}}_{1;j}\}}\sum_{j=1}^{T}u_1(\pi,\pi_{-1;j}) \\ &= R'_T + \min\left\{R_T(\pi_1), R_T(\pi^{\mathrm{RL}}_{1;j})\right\}, \end{aligned} \tag{13}$$

where $R_T(\pi_1) = \max_{\pi_1\in\Pi_1}\sum_{j=1}^{T}u_1(\pi_1,\pi_{-1;j}) - \sum_{j=1}^{T}u_1(\pi_1,\pi_{-1;j})$ is the regret of using policy π_1 and $R_T(\pi^{\mathrm{RL}}_{1;j}) = \max_{\pi_1\in\Pi_1}\sum_{j=1}^{T}u_1(\pi_1,\pi_{-1;j}) - \sum_{j=1}^{T}u_1(\pi^{\mathrm{RL}}_{1;j},\pi_{-1;j})$ is the regret of using policy $\pi^{\mathrm{RL}}_{1;j}$. So,

$$R_T(\pi^{\mathrm{EXP3}}_{1;j}) \le 3.1\sqrt{T} + \min\left\{R_T(\pi_1), R_T(\pi^{\mathrm{RL}}_{1;j})\right\}. \tag{14}$$

◻

D. The Kuhn Poker Benchmark

We use the Kuhn poker benchmark from Lanctot et al. (2019). Kuhn poker is a simplified poker game with two players (P1 and P2) and three cards (Jack, Queen, and King). There are two actions (bet and pass) for each player. The value of a bet is 1.0. When two players have matched bets, the cards of both players are revealed, and the player with a higher card wins the pot. Our experiments focus on playing the main agent as P1, and hence P2 is the opponent. Following Southey et al. (2009), we eliminate dominated policies for P2, after which the policy of P2 can be parameterized using two parameters. The corresponding policy space of P2 is illustrated in Figure 7. The policy space of P2 can be divided into 6 regions, as shown in Figure 7. Within each region, there exists a single pure policy of P1 that is the best response to all the policies of P2 in that region. For points in the border lines, both best responses have the same returns.

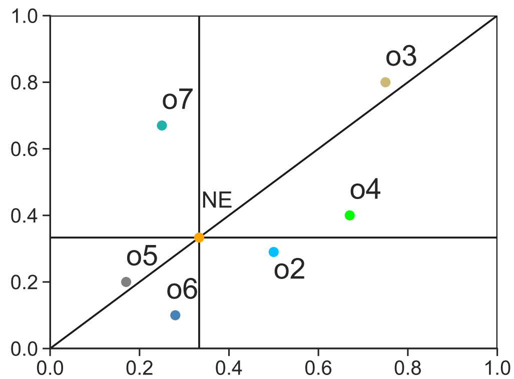

Figure 7. The policy space of player P2 in Kuhn poker. The only one NE policy $(\frac{1}{3}, \frac{1}{3})$ is denoted by 'NE'.

We sample a policy for each region according to the way in Southey et al. (2009), as illustrated in Figure 7. We set $^{Train} = \{o3, o5, o7\}$ and $^{Test} = \{o2, o4, o6\}$. The policy embeddings learned by Policy2Emb using ^{Train} are plotted in Figure 8.

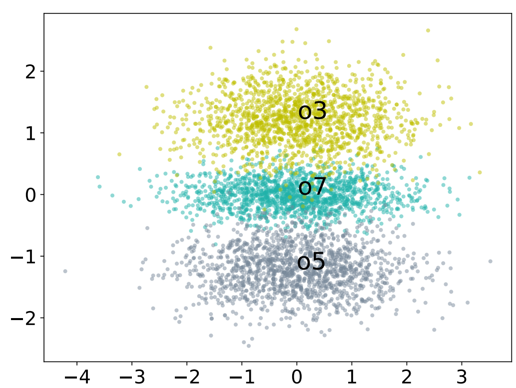

Figure 8. The policy embeddings learned by Policy2Emb in Kuhn poker using data generated by ^{Train}. The embeddings for 'o7' lie in the middle of those for 'o3' and 'o5', which corresponds to the layout in Figure 7.

E. The Predator Prey Benchmark

The PP environment consists of one prey (the main agent), three predators (the opponents), and two obstacles (grey circles), as illustrated in Figure 9. The prey is trying to keep a distance as far as possible from the predators while the predators are trying to catch the prey. Each agent has five actions: accelerating east, south, west, north, and no acceleration. The environment is partially observable from the perspective of the main agent: it can only observe predators and obstacles within its receptive field, which is 0.5. The predators yet have full visibility of the environment.

At each time step, the prey receives a reward that is proportional to the sum of distances from each predator. The prey receives a reward 10 whenever it is captured by one predator. The gird world is two dimensional, with each dimension within the range $[\ 1.0, 1.0]$. The prey is penalized for crossing the range. The maximal speed of the prey is 1.3, and the maximal speed of each predator is 1.0. Each episode of a PP game terminates after 50 time steps. For more information of the PP benchmark, readers are referred to Lowe et al. (2017) and Mordatch & Abbeel (2018).

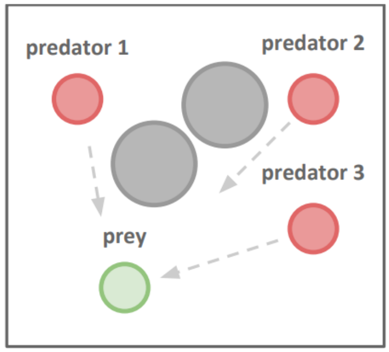

Figure 9. An illustration of the PP benchmark, extracted from Lowe et al. (2017).

We create eight different rule-based opponent policies[2], each of which acts greedily (i.e., minimizing its distance to the main agent) with probability 0.4 and otherwise moves according to its preferred direction. There are eight predefined directions, and this results in eight different opponent policies: east (O_E), south (O_S), west (O_W), north (O_N), southeast (O_{SE}), southwest (O_{SW}), northeast (O_{NE}), and northwest (O_{NW}). For a combined direction, e.g., southeast, the opponent O_{SE} moves south with probability 0.3 and east with probability 0.3. We set $\Pi^{Train} = \{O_N, O_{NW}, O_W, O_{SW}\}$ and $\Pi^{Test} = \{O_S, O_{SE}, O_E, O_{NE}\}$. The policy embeddings learned by Policy2Emb on $\Pi^{Train} \cup \Pi^{Test}$ are plotted in Figure 10. The policy embeddings learned by Policy2Emb on Π^{Train} are plotted in Figure 11.

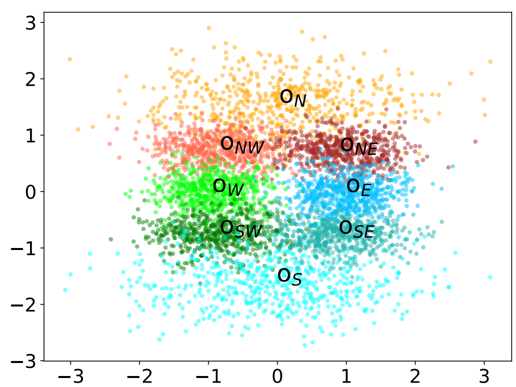

Figure 10. The policy embeddings learned by Policy2Emb in PP using data generated by $\Pi^{Train} \cup \Pi^{Test}$. The layout of different policy embeddings matches the real distribution of the eight directions.

F. Implementation Details for the Experiment on Kuhn Poker

F.1. Neural Architectures

PPO: The architecture for PPO has an actor network with three linear layers and a separate critic network with another three linear layers. All hidden layers consist of 128 nodes with ReLU activation.

LIAM: Different from the original architecture in Papoudakis et al. (2021) where a LSTM (Hochreiter & Schmidhuber, 1997) is applied to the encoder, we change the LSTM to linear layers in our implementation for the Kuhn poker experiment, as the observation in Kuhn poker includes information for all previous steps. The encoder in our implementation has two linear layers with ReLU activation. The decoder has one shared linear layer and two separate linear layers to reconstruct the observation and action of the opponent respectively. All hidden layers consist of 128 nodes. The embedding dimension is 8.

DRON: We use the DRON-concat architecture in He et al. (2016) for the experiment on Kuhn poker. A feature network is built to learn a 4-dim hidden representation using hand crafted features (i.e., the opponent's previous action). The hidden

[2] The three predators share a policy.

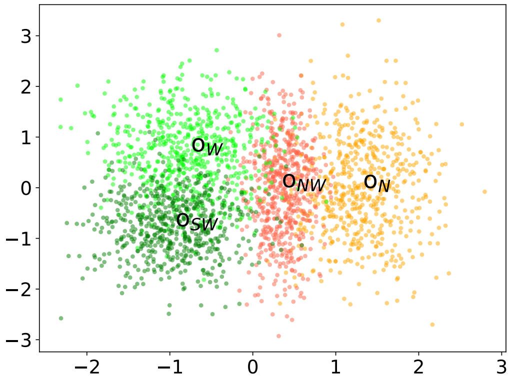

Figure 11. The policy embeddings learned by Policy2Emb in PP using data generated by ^{Train}. The layout of different policy embeddings is well structured in the sense that the embeddings for O_{NW} lie between those for O_N and O_W and that the embeddings for O_W lie between those for O_{NW} and O_{SW}.

representation is concatenated with the second layer of the actor and critic network, which is the same as those in PPO. In addition, an auxiliary task that predicts the opponent's 2 policy parameters is applied as extra supervision. The feature network contains 2 linear layers with ReLU activation and one linear layer with sigmoid activation for the auxiliary task. All hidden layers consist of 128 nodes.

Deep BPR+: We train three counter-strategies against the three opponent policies using PPO. As described in Zheng et al. (2018), a Distillation Policy Network (DPN) is built for fast learning in the online learning stage of Deep BPR+. In our case, we use two DPNs, one for the actor and the other for the critic. Each DPN consists of a trunk with two linear layers with ReLU activation and three separate parts following the trunk. Each part (a linear layer for the critic and a linear layer with Softmax activation for the actor) corresponds to each of the three opponent policies. All linear layers consist of 128 nodes.

The Conditional RL in GSCU: We use the same architecture as that of PPO, except that our opponent policy embedding **z** is used as additional input for both the actor and the critic networks.

Policy2Emb: The encoder q_{e} and decoder p_{d} have two and four linear layers respectively. All hidden layers consist of 128 nodes with ReLU activation function. The embedding dimension is 2.

F.2. Training Hyperparameters

We use the Adam optimizer (Kingma & Ba, 2014) for all experiments on Kuhn poker. The training parameters for each method are shown in Table 1.

In the training of Policy2Emb, in order to mitigate KL-vanishing, we apply the cyclical annealing schedule approach (Fu et al., 2019), which repeats the procedure of increasing β from 0 to a maximal value for multiple times.

Parameter	**Range**	**Best**
Shared		
Learning rate	{1e-4, 5e-4}	5e-4
Batch size	-	1000
Number of PPO update per batch	{5, 10}	5
PPO clip ratio	-	0.2
Training episodes	-	300000
Discount factor (γ)	-	0.99
Policy2Emb		
Learning rate	-	1e-3
The maximal value for (β)	{0.01, 0.1}	0.01
Number of annealing cycles	-	2

Table 1. The hyperparameters used for training on Kuhn poker.

F.3. Test Hyperparameters

For a single run of online test, we first randomly sample 10 sequences of opponents, where each sequence is of length 20 ($N^o = 20$). We test each method on the 10 sequences to obtain an average performance. During test on a single sequence of opponents, the opponent changes its policy every 1000 episodes ($M = 1000$). We repeat the above process 5 times with 5 different random seeds. The hyperparameters of each method for online test are shown in Table 2. DRON and LIAM have no hyperparameters for online test, as they do opponent modelling within an episode.

Parameter	**Range**	**Best**
GSCU		
Variational Inference (VI) frequency (M^{VI})	-	10
VI batch size	-	10 20
Number of VI update per batch	{10, 50}	50
Minimal VI standard deviation (σ)	-	0.5
VI learning rate	{0.005, 0.01}	0.005
EXP3 learning rate (η)	{0.2, 0.3}	0.3
Shared for Tracking, Deep BPR+, and the "adaptive" opponent		
Learning rate	{5e-5, 5e-4}	5e-5
PPO clip ratio	-	0.2
Batch size	-	100
Discount factor (γ)	-	0.99
Number of PPO update per batch	{5, 10}	10
Deep BPR+		
Moving average reward threshold	{-0.6, -0.4, -0.2}	-0.4
Moving average window size (episodes)	{200, 800, 1200}	800
DPN convergence reward threshold	{-0.1, 0.0, 0.1}	0.0

Table 2. The hyperparameters used for online test on Kuhn poker.

G. Implementation Details for the Experiment on PP

G.1. Neural Architectures

PPO: The architecture for PPO has an actor network with four linear layers and a separate critic network with another four linear layers. All hidden layers consist of 128 nodes with ReLU activation.

LIAM: We implement the original architecture of LIAM. The encoder contains a LSTM layer and two linear layers, with ReLU activation after the first layer. The encoder encodes the observation-action trajectory of sequence length 8 into an embedding of dimension 20. The decoder has two shared linear layers and two separate linear layers to reconstruct the observation and action of the opponent respectively. All hidden layers consist of 128 nodes.

DRON: We use the DRON-concat architecture in He et al. (2016) for the experiment on PP. A feature network is built to learn a 2-dim hidden representation using hand crafted features (i.e., the opponent's previous action frequency). The hidden representation is concatenated with the second layer of the actor and critic network, which is the same as those in PPO. In addition, an auxiliary task that predicts an opponent's identity is applied as extra supervision. The feature network contains 2 linear layers with ReLU activation and one linear layer with Softmax activation for the auxiliary task. All hidden layers consist of 128 nodes.

Deep BPR+: We train four counter-strategies against the four opponent policies using PPO. Similar to Kuhn poker, we use two DPNs, one for the actor and the other for the critic. Each DPN consists of a trunk with three linear layers with ReLU activation and four separate parts following the trunk. Each part (a linear layer for the critic and a linear layer with Softmax activation for the actor) corresponds to each of the four opponent policies. All linear layers consist of 128 nodes.

The Conditional RL in GSCU: We use a similar architecture as that of PPO except that we use a LSTM layer to encode previous observation-action trajectory information. The observation-action sequence length is 8. Our opponent policy embedding **z** is used as additional input for both the actor and the critic networks.

Policy2Emb: The encoder q_{e} and decoder p_{d} have two and four linear layers respectively. All hidden layers consist of 128 nodes with ReLU activation function. The embedding dimension is 2.

G.2. Training Hyperparameters

We use the Adam optimizer (Kingma & Ba, 2014) for all experiments on PP. The training parameters for each method are shown in Table 3.

Parameter	Range	Best
Shared		
Learning rate	{1e-4, 5e-4}	5e-4
Batch size	-	4000
Number of PPO update per batch	{5, 10}	10
PPO clip ratio	-	0.2
Training episodes	-	10000
Discount factor (γ)	-	0.99
Policy2Emb		
Learning rate	-	1e-3
The maximal value for (β)	{0.01, 0.1}	0.1
Number of annealing cycles	-	2

Table 3. The hyperparameters used for training on PP.

G.3. Test Hyperparameters

For a single run of online test, we first randomly sample 5 sequences of opponents, where each sequence is of length 20 ($N^{o} = 20$). We test each method on the 5 sequences to obtain an average performance. During test on a single sequence of opponents, the opponent changes its policy every 200 episodes ($M = 200$). We repeat the above process 5 times with 5 different random seeds. The hyperparameters of each method for online test are shown in Table 4. As on Kuhn poker, DRON and LIAM have no hyperparameters for online test.

Parameter	Range	Best
GSCU		
VI frequency (M^{VI})	-	1
VI batch size	-	50
Number of VI update per batch	{10, 50}	10
Minimal VI standard deviation (σ)	-	0.5
VI learning rate	{0.005, 0.01}	0.005
EXP3 learning rate (η)	{0.2, 0.3}	0.2
Shared for Tracking, Deep BPR+, and the "adaptive" opponent		
Learning rate	{5e-5, 1e-4}	5e-5
PPO clip ratio	-	0.2
Batch size	-	400
Discount factor (γ)	-	0.99
Number of PPO update per batch	{5, 10}	10
Deep BPR+		
Moving average reward threshold	{-500, -400, -300}	-500
Moving average window size (episodes)	{50, 100, 150}	100
DPN convergence reward threshold	{-30, -20, -10}	-10

Table 4. The hyperparameters used for online test on PP.

H. Additional Results on Competing Online against Unknown Opponents

In practice, online opponents can exhibit various dynamics, and our four settings “seen”, “unseen”, “mix”, and “adaptive” are designed to cover a wide range of such dynamics. Hence, it is worth comparing the average and worst-case performance of different methods over different opponent dynamics.

For the average performance of a method over different opponent dynamics, we simply average the returns under the four settings. For the worst-case performance of a method over different dynamics, we first compute its performance gap with the corresponding best performance under each setting. Afterwards, we report the worst gap across different dynamics as the worst-case performance of a method. Using the above definitions of average and worst-case performance, we reorganize the results presented in Figure 2 to obtain the new results in Figure 12. From the results in Figure 12, we can conclude that GSCU performs significantly better than prior methods in terms of the average and worst-case performance. Besides, we can observe that methods that learn across episodes generally have an advantage over methods that learn only within the current episode.

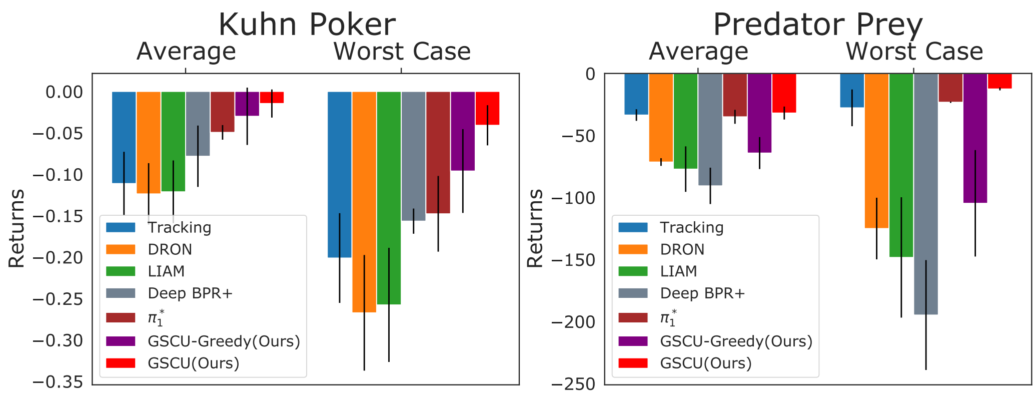

Figure 12. The average and worst-case returns of different methods across the 4 settings of online opponents: “seen”, “unseen”, “mix”, and “adaptive”.

The results shown in Figure 2 are obtained in situations where the change frequency M of opponent policies is set to 1000 episodes for Kuhn poker and 200 episodes for PP. Apart from the opponent policies, the change frequency M also has a large influence on the online test performance of a method, as it controls the amount of stationary data that can be used to infer about the current opponent. To this end, we conduct additional experiments to investigate the performance of GSCU under different settings of M on Kuhn poker, the results of which are plotted in Figure 13.

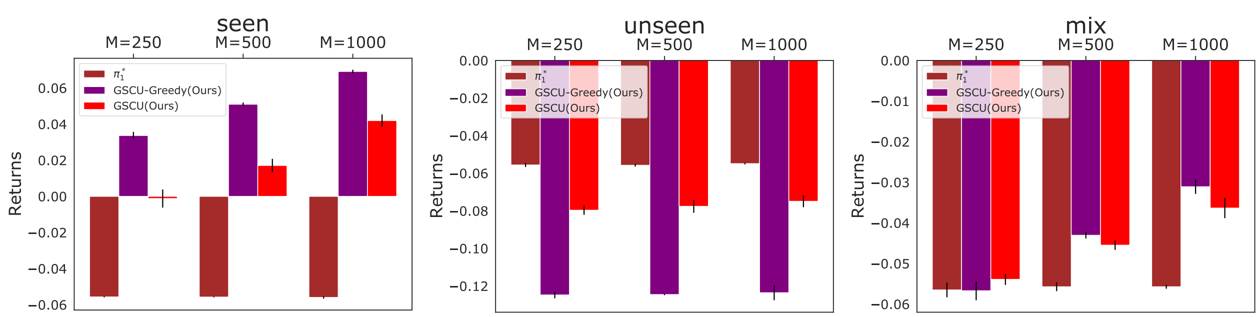

Figure 13. The influence of the opponent change frequency M on GSCU-Greedy and GSCU in Kuhn poker.

From Figure 13, we can observe a general tendency that the performance of either GSCU-Greedy or GSCU improves as M increases. The improvement is most significant in the “seen” setting, as GSCU-Greedy benefits most from more data of “seen” opponents. In the “unseen” setting, the change frequency M has nearly no influence on the performance of GSCU-Greedy. Yet, the performance of GSCU increases slightly with larger M, which may be due to the convergence of EXP3 to the arm of π_1 in GSCU. In the “mix” setting, similar results can be observed as the “seen” setting.