REGAETuning out of Rabbit Holes in Mixed-Motive, Contextual Bandit Problems

Ethan Pedersen
Brigham Young University
Provo, UT, United States
ethanp55@byu.edu

Jacob Crandall
Brigham Young University
Provo, UT, United States
crandall@cs.byu.edu

ABSTRACT

In mixed-motive settings, it can be challenging for agents to learn when (and how) to compete and cooperate. In this paper, we explore the impact of two mechanisms on the ability of deep reinforcement learning (RL) algorithms to learn (offline) to balance cooperation and competition in mixed-motive, contextual bandit problems. These mechanisms include: (1) an exploration policy motivated by the epsilon-first bandit strategy, called REGAETune, and (2) context awareness via features produced by Assumption-Alignment Tracking (AAT). To evaluate the effectiveness of these mechanisms, we use an adaptability metric that captures an algorithm's ability to balance cooperation and competition. Results across three mixed-motive domains indicate that REGAETune leads to greater variety in strategy selection and higher adaptability than does traditional ε-greedy exploration. Furthermore, AAT feature inputs appear to also lead to more adaptive behavior than raw inputs, though to a lesser degree. The primary contribution of this paper is that it provides important insights for designing deep RL algorithms for mixed-motive, contextual bandit problems.

KEYWORDS

Multi-agent Learning, Adaptive Agents, Mixed Motives

1 INTRODUCTION

Learning to effectively adapt to arbitrary (unknown) associates in mixed-motive (general-sum) domains remains an unsolved challenge. Research on learning in repeated games, which require agents to determine when to coordinate, compete, and cooperate, highlights this challenge. Early work by Sandholm and Crites [33] illuminated the difficulty of getting neural reinforcement learning (RL) algorithms to cooperate, as they tend to converge to local maxima. Crandall and Goodrich [12] showed how a specific exploration policy allows an agent to both learn to cooperate and compete in two-player repeated games using an adaptation of tabular Q-learning. However, slow learning rates limit the algorithms' ability to realistically adapt to arbitrary associates in real time. Furthermore, this policy is not easily encoded into deep RL algorithms. Wang et al. [42] showed that a deep RL agent is capable of learning to robustly cooperate in a repeated prisoner's dilemma, but doing so required extensive and specific hand-crafted training. These and many other example studies highlight the difficulties associated with learning to effectively balance cooperation and competition in mixed-motive scenarios played with arbitrary associates.

Proc.@ of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

In this paper, we consider how to design deep RL algorithms that can, after (offline) training, adapt to arbitrary associates in mixed-motive (multi-agent) contextual bandit problems. This setting provides an agent with a set of N expert algorithms. At each time step, the agent must select one of these experts. The selected expert then dictates the agent's behavior in that time step. The contextual bandit setting also supplies the agent with external (contextual) information about the environment, which it can use to help determine which expert to follow at any given time.

While one could attempt to solve this problem using ever more sophisticated neural architectures, we approach the problem with standard neural architectures. In so doing, we explore how two factors impact the ability of an RL agent to learn adaptive behavior in mixed-motive, contextual bandit scenarios. First, we explore methods for generating contextual information, or input features, that the agent can use to help it make decisions. Specifically, we consider how input features [23] derived using assumption alignment tracking (AAT) [9, 19] impact the ability of deep RL algorithms to learn effectively in mixed-motive, contextual bandit scenarios. Second, since the way that an algorithm is exposed to data samples in training can have a significant impact on performance (e.g., [31]), we compare how two offline training processes (using ε-greedy exploration explore REGAETune exploration, respectively) impact the ability of deep RL algorithms to learn to compete and cooperate. Using variations on these two factors (input features and training strategy), we construct six distinct deep RL algorithms for learning in mixed-motive contextual bandit problems.

We then compare and contrast the performance of these algorithms across three mixed-motive environments. Results show that REGAETune training consistently outperforms traditional ε-greedy exploration with respect to an adaptability score, which measures the ability of a contextual bandit algorithm to determine when and how to compete and cooperate. AAT feature inputs also appear to lead to more adaptive behavior overall than raw input features, though to a lesser degree. These results have implications for designing deep RL algorithms for mixed-motive, contextual bandit problems, as they give insights into how these algorithms can be designed to more consistently learn effective strategies.

2 CONTRIBUTION STATEMENT

Results presented in this paper confirm results from prior work: ε-greedy exploration often leads deep RL algorithms to learn myopic and non-adaptive strategies in mixed-motive contexts. Furthermore, results indicate that REGAETune exploration and, to a lesser degree, AAT feature inputs can produce deep RL contextual bandits that better adapt in real time in mixed-motive (general-sum) scenarios.

3 PROBLEM FORMULATION AND APPROACH

Our goal is to better learn how to create deep RL algorithms that can effectively adaptive in real time to arbitrary associates in mixed-motive scenarios. In this paper, we consider solving this problem using contextual bandit algorithms. We overview this approach and define metrics to evaluate the ability of algorithms to adapt to unknown associates in this section.

3.1 Contextual Bandits for Multi-agent Settings

We consider contextual bandit settings in which an autonomous agent has access to a set of *N generators* (also referred to as arms, behaviors, or experts). Each generator encodes an algorithm that defines agent behavior. At each time step, the agent selects one of its generators to dictate the action(s) taken by the agent in that time step. The contextual bandit setting also provides contextual information an agent can use to help determine which generator to select. We consider that this contextual information defines features of world state, including properties of the environment and of other agents in the environment. We chose to focus on this framework as selecting from generators (rather than raw actions) drastically reduces the strategy space, which helps simplify the learning process.

Most existing contextual bandit algorithms take one of two approaches [25]. The first approach encodes a separate bandit for each context (e.g., [1, 24, 26]). That is, for each unique context, a separate bandit is used to learn which generator the agent should use. While this approach allows the agent to select different generators in different contexts, it is intractable when the number of contexts is large. The second approach encodes a single bandit algorithm that selects generators without using the contextual information (e.g., [2, 5]). The contextual information is only used by the selected generator as it selects actions. While overcoming the problem of intractability encountered in the first approach, this second approach does not allow the agent to learn to select different generators in different contexts.

In this paper, we attempt to create contextual bandit algorithms that realize the best of both approaches. We encode a single bandit (a deep RL algorithm) that uses contextual information about the world to select which generator to follow in each time step. The contextual information is also used by the selected generator to select actions. This requires the bandit algorithm to be more sophisticated, as it complicates the evaluation of selection mechanisms.

In multi-agent (mixed-motive) scenarios, agents often do not have the luxury of repeatedly encountering the same scenarios. Rather, they are placed into a situation with arbitrary associates and must immediately know how to make good decisions to maximize payoffs (repeats are not assumed). Thus, in this paper, we consider that learning is done offline (prior to interacting with other agents).

3.2 Measuring Success: Adaptability

Bandit algorithms are most commonly evaluated using some form of regret (e.g., [7, 27]), which compares an algorithm's performance against how well it would have done had it always selected its best generator. However, in mixed-motive domains, minimizing naive measures of regret often do not equate with maximizing payoffs [4, 10, 11], and less naive measures of regret are difficult to obtain in complex environments. Furthermore, algorithms cab have low regret when paired with some associates, but high regret when paired with others in mixed-motive settings. Thus, a single measure of regret does not capture our objectives. Given our interest in observing the ability of an agent to adapt to the behavior of their associates in a way that maximize their own payoffs, we use an alternative metric, called the *adaptability score*, to measure algorithmic success in this paper.

The adaptability score is formed using two different measures of regret: *competitive regret* (R_{comp}) and *cooperative regret* (R_{coop}). Competitive regret measures how well an algorithm performs in scenarios when other agents are not inclined to cooperate. Formally,

$$R_{\text{comp}} = U_d^* - \bar{U}_d, \tag{1}$$

where U_d^* is the highest expected reward the agent can receive when paired with non-cooperative associates and $\bar{U}_d$ is the average reward the agent actually receives when interacting with these same agents.

A competitive score (denoted $\mathcal{S}_{\text{comp}}$) is derived from normalized competitive regret as follows:

$$\mathcal{S}_{\text{comp}} = 1 - \frac{R_{\text{comp}}}{U_d^* - U_{min}}, \tag{2}$$

where U_{min} is the lowest expected reward an agent could receive. A competitive score near one indicates that the algorithm performs effectively when placed in environments with non-cooperative associates, while a competitive score near zero indicates less effective performance in this regard. Note that both U_d^* and U_{min} are scenario-specific values that must be identified to obtain these measures.

Cooperative regret, which measures how well an algorithm performs when paired with other algorithms that are inclined to cooperate, is measured similarly. Formally, cooperative regret is given by:

$$R_{\text{coop}} = U_c^* - \bar{U}_c, \tag{3}$$

where U_c^* is the expected payoff when all agents cooperate and $\bar{U}_c$ is the agent's average reward when interacting with cooperative associates. Note that it is possible for $\bar{U}_c > U_c^*$ (i.e., $R_{\text{coop}} < 0$) if the agent successfully exploits cooperators. A cooperative score (denoted $\mathcal{S}_{\text{coop}}$) is derived from normalized cooperative regret as follows:

$$\mathcal{S}_{\text{coop}} = 1 - \frac{R_{\text{coop}}}{U_c^* - v_{\text{mm}}}, \tag{4}$$

where v_{mm} is the agent's maximin value in the scenario.

Ideally, an agent should have both high competitive and cooperative scores. To measure this, we compute the *adaptability score*:

$$\mathcal{S}_A = \min(\mathcal{S}_{\text{comp}}, \mathcal{S}_{\text{coop}}). \tag{5}$$

If an agent has both high cooperative and competitive scores, it will have a high adaptability score (near one). On the other hand, if it has either a low cooperative or competitive score, it will have a low adaptability score (nearer to zero).

The adaptability score punishes an algorithm if it does poorly in either competitive or cooperative scenarios, even if it does very well in the other scenarios. Justification for basing the adaptability score on the minimum (rather than, for example, the average) is to distinguish algorithms that adapt effectively to different situations

Table 1: Contextual bandit agents compared in our experiments.

Agent Name	ML Alg.	Training	Features
EG-Raw	DQN	EG	Raw
EG-AAT	DQN	EG	AAT
EG-RawAAT	DQN	EG	RawAAT
REGAE-Raw	DQN	REGAETune	Raw
REGAE-AAT	DQN	REGAETune	AAT
REGAE-RawAAT	DQN	REGAETune	RawAAT
(Baseline) AlegAATr	kNN	REGAETune	AAT

from those that do well only in a single scenario. Low performance in at least one area indicates a learning deficiency.

4 CONTEXTUAL BANDIT ALGORITHMS

In this paper, we compare and contrast six contextual bandits constructed using deep RL (Table 1). Each of these agents are encoded using the same deep Q-network (DQN). They vary from each other based on the training process they are subjected to and the input features (computed from the contextual information) that they use. The seventh algorithm in Table 1, AlegAATr [32], which does not use deep RL, is provided as a baseline comparison. In this section, we discuss the DQN, the two different training processes, and the three different feature inputs used by the agents.

4.1 Deep Q-Network

Each of the six RL agents we evaluate in this paper uses the simple DQN architecture discussed in SM-2 (code is also provided) to implement Q-learning [43]. Given our focus on studying how training processes and input features impact the ability of contextual bandit algorithms encoded using deep RL, this DQN model is simple and not intended to be overly sophisticated.

We acknowledge that more sophisticated networks do exist, and in fact we experimented with three of them in this work: Proximal Policy Optimization (PPO) [34], RDQN ("Really Deep Q-Network", a deeper variation of our simple DQN), and a unique version of a multi-agent deep Q-network (MADQN) [17]. We found that these architectures did not yield substantially different results from the simple DQN we use in the mixed-motive scenarios we considered in this paper. Some results for agents created with these architectures can be found in the supplementary material (SM-2.3-2.5 and 3).

4.2 Training Processes

In every scenario, we expose the agents to the same training scenarios (in the same order). However, two different exploration strategies were used during training: (1) traditional ε-greedy exploration (abbreviation: EG) and (2) a simple two-phase exploration process we call REGAETune. We discuss each of these exploration processes.

4.2.1 Training using ε-Greedy Exploration (EG). ε-greedy exploration remains a popular standard methodology for balancing exploration and exploitation in RL [13, 28, 39, 44]. During training, an agent explores a random action with probability ε; with probability $1-\varepsilon$, the agent instead exploits its current knowledge by choosing an action it believes to be superior. ε is typically decayed over time as the agent learns. In our experiments, at the end of each training epoch we used multiplicative decay with a common decay factor of 0.99: $\varepsilon_t = 0.99 \cdot \varepsilon_{t-1}$, where $\varepsilon_1 = 0.1$.

Despite its popularity, this exploration approach is known to sometimes converge to suboptimal policies (e.g., [21]). We liken this idea to *getting stuck in a rabbit hole*. ε-greedy exploration is especially prone to this failing in mixed-motive domains that contain two or more basins of attraction (e.g., cooperate or compete). As an agent learns using ε-greedy exploration, it often will fixate on one key strategy. As it fixates on a strategy, it repeatedly reinforces that behavior (in itself and others) in its learning, trapping itself in a *hole* (or local minimum). This tendency is highlighted further in our results.

Algorithm 1 REGAETune

1: **for** condition in training conditions **do**
2: randomly choose generators
3: **end for**
4: train an initial model on information gained in steps 1-3
5: **for** condition in training conditions **do**
6: greedily choose generators based on model trained in step 4
7: **end for**
8: train a final model on information gained in steps 5-7

4.2.2 REGAETune Training. Given frequent failings of ε-greedy exploration in mixed-motive scenarios, we are interested in finding a more robust exploration process for training deep RL algorithms as contextual bandits. Prior work has found centralized training, decentralized execution (see Amato [3] for a review) to be effective in cooperative domains. However, this exploration method is unlikely to be effective in helping agents to learn to compete in mixed-motive domains. Likewise, the *optimism in uncertainty* [8, 12, 38] exploration principle has been shown to be successful in tabular RL, but it is unclear how to encode this exploration policy into deep RL algorithms.

Motivation for an alternative exploration process for RL algorithms is provided by the epsilon-first exploration method (e.g., [30, 40]), a known method from the online traditional bandit (non-contextual) setting. In this method, a first phase is conducted in which the bandit only explores (i.e., $\varepsilon = 1$) for a certain number of rounds. Thereafter, the bandit never explores (i.e., $\varepsilon = 0$) and always acts greedily based on its current knowledge.

We encode this process as an offline training method for deep RL algorithms used in mixed-motive, contextual bandits settings. We call this training method REGAETune (*R*andom *E*xploration, *G*reedy *A*lgorithmic *E*nhancement *Tun*ing). The REGAETune method is summarized in Algorithm 1. In phase one training scenarios (steps 1-4), the agent only explores the environment by randomly selecting generators. It then uses the data gained from this exploration to learn an initial model. In phase two scenarios (steps 5-8), the agent greedily chooses actions based on the learned model from phase one. It then uses the new information to learn a final model. In other words, REGAETune completely separates the exploration and exploitation processes into separate iterations, allowing an agent to freely explore the various generators in different contexts before honing/tuning its beliefs.

4.3 Input Features

Since DeepMind showed that deep RL algorithms can be effective in learning in complex settings given only raw inputs [29], it is a common assumption that deep RL algorithms can learn to effectively extract important features (given sufficient data). Despite this, multi-agent learning algorithms still often fail to learn to cooperate in mixed-motive domains [16, 37]. We hypothesize that different forms of input features can provide clarity to deep RL algorithms that can help them get out of *rabbit holes* more easily.

To begin to explore this hypothesis, we compare and contrast three different kinds of feature inputs in this paper. We discuss each in turn.

4.3.1 Raw Features. Perhaps the most straightforward input features come from the contextual information itself. We refer to these inputs as *raw* features. For example, in a repeated prisoner's dilemma, contextual information could include the actions and rewards of both agents: $\langle a_1, a_2, r_1, r_2 \rangle$, where a_i and r_i are the actions and rewards of agent i from the previous round of play.

4.3.2 AAT Input Features. While raw features can be sufficient in some settings, feature engineering (i.e., crafting more intuitive features from preexisting raw features) can boost model performance (e.g., [15]). In this paper, we consider engineered features using Assumption-Alignment Tracking (AAT) [9, 19], a methodology developed in the robotics literature to perform proficiency self-assessment. However, AAT has also shown promise as a principled feature engineering approach [32]. AAT is based on the idea that a generator is created under certain assumptions about the environment. As such, the performance of the generator depends to a large degree on these assumptions being met. Thus, to predict the future effectiveness of a generator, the agent must be aware of the extent to which these assumptions are true. This can be accomplished by continually tracking the veracity of the assumptions upon which the generator relies, and then using these assessments to adjust performance predictions.

To track the veracity of the assumptions made in the construction of generator G, veracity assessments are calculated using alignment checkers, which operate on the contextual information (state) passed into the bandit algorithm. An alignment checker is a program that continually tracks the veracity of an assumption. Formally, let $\Phi = \{\phi_1, \ldots, \phi_M\}$ denote the set of M assumptions upon which a generator relies. Let $x_t^{\phi_j} \in [0, 1]$ be the assessment, made by an alignment checker of the veracity of assumption ϕ_j at time t. If the assumption is evaluated to be true, then $x_t^{\phi_j} = 1$; otherwise, $x_t^{\phi_j} = 0$. Real-valued assessments between 0 and 1 can reflect uncertainty in the truth of the assumptions. Given the individual assessments of assumption veracity, let $\mathbf{x}_t = \langle x_t^{\phi_1}, \ldots, x_t^{\phi_M} \rangle$ denote the veracity assessment vector at time t. AAT feature vectors used in our experiments are these veracity vectors ($\mathbf{x}_t$).

4.3.3 RawAAT Input Features. One concern that arises in the creation of AAT feature vectors is that identifying assumptions and creating checkers is a subjective process in which the designer might miss important information about the environment. Hence, AAT features may not be sufficiently comprehensive. In an attempt to overcome this problem, the third set of input features we consider combines raw and AAT features, which we call *RawAAT* features, thus allowing the algorithm to potentially learn information from both.

5 EXPERIMENTS

Our objective is to study how the training processes and input features described in the previous section impact the ability of deep RL algorithms to learn (offline) to balance cooperation and competition in mixed-motive, contextual bandit problems. This section describes the experiments we conducted to compare and contrast the deep RL algorithms derived from these training processes and input features.

5.1 Experimental Design

Our study has two factors: training process (EG and REGAETune) and input features (Raw, AAT, and RawAAT). This produces the first six agents listed in Table 1. A seventh algorithm, AlegAATr [32], which is not a deep RL algorithm, is included as a baseline to help evaluate the effectiveness of the first six algorithms. We chose AlegAATr as a baseline since (1) it has been shown to perform effectively in some mixed-motives scenarios and (2) it uses AAT input features. We trained AlegAATr with REGAETune, which essentially created another version of REGAE-AAT. However, since AlegAATr is not a deep RL algorithm, it allowed us to consider how effectively the deep RL algorithms learned to adapt in different domains.

Evaluations were performed in the three mixed-motive scenarios described in the next subsection. In each scenario, we compare and contrast the algorithms primarily with respect to the adaptability score ($\mathcal{S}_A$), along with the cooperative score ($\mathcal{S}_{\text{coop}}$) and the competitive score ($\mathcal{S}_{\text{comp}}$). To measure R_{coop} and R_{comp} in each setting, we paired the algorithms with both cooperative and non-cooperative agents.

We conducted 30 different evaluation trials in each condition. Additionally, because the result of training can vary for each algorithm due to an algorithm's internal learning mechanisms (e.g., random initialization of network weights or random explorations), we trained each algorithm multiple times. For the repeated prisoner's dilemma and grid stag hunt domains, we re-trained and re-tested each agent 30 times, for a total of $30 \times 30 = 900$ runs for each algorithm and test condition. Due to the more complex nature of the Junior High Game (resulting in much slower training times), which we define in the next subsection, we re-trained and re-tested each agent 10 times (for a total of $10 \times 30 = 300$ samples) in that domain.

5.2 Evaluation Scenarios

We evaluate how well our agents adapt to different conditions in three distinct evaluation scenarios: a repeated prisoner's dilemma, a stag hunt grid world, and the Junior High Game (JHG). For each domain, we overview the scenario (game), talk about the generators that are available to the agents, and discuss the training and evaluation conditions. Additional implementation details about these experiments are given in the supplementary material (SM-4-6).

5.2.1 Repeated Prisoner's Dilemma. We evaluate agents in a repeated prisoner's dilemma due to its popularity, simplicity (easy to

Table 2: Repeated prisoner's dilemma payoff matrix.

		Player 2	
		Cooperate	*Defect*
Player 1	*Cooperate*	(3, 3)	(−3, 5)
	Defect	(5, −3)	(−1, −1)

analyze), and the fact that deep RL algorithms often get trapped in competitive (defect) behavior in this domain.

Description. The repeated prisoner's dilemma is a two-player game played over multiple rounds. In each round, players can either cooperate or defect. The rewards are structured such that the Nash equilibrium in a single-shot game occurs when each player defects. This often traps the learning process of RL algorithms [42], though both would be better off if they both cooperate. In an infinitely repeated prisoner's dilemma, there are an infinite number of Nash equilibria [20]. The payoff matrix we used is shown in Table 2.

Generators. Bandits had access to generators that have previously been used to play repeated prisoner's dilemmas [32]. The generators either (1) always defect, (2) always cooperate, or (3) try to cooperate but defect as a form of punishment against partners that do not cooperate (i.e., similar to tit-for-tat).

Training and Evaluation Conditions. In training scenarios, agents were paired with three unique algorithms designed for playing repeated games: BBL (which employs fictitious play [18]), EEE [14], and S++ [11]. These algorithms choose from the same set of generators defined in the previous subsection. We only used these algorithms for training and not testing, as the goal in testing was to evaluate how well the algorithms balance purely cooperative and purely competetive environments. We trained for 10 epochs with round lengths of 20, 30, 40, and 50, respectively.

After training was complete, evaluations designed to compute competitive and cooperative scores were conducted as follows. To measure the competitive score, we paired agents with a partner that always defects. To measure the cooperative score, we used two conditions: (1) self-play (i.e., each agent was paired with a copy of itself) and (2) pairings with tit-for-tat players; we averaged results from these two scenarios to compute the cooperative scores. All evaluation scenarios had game lengths of 30 rounds.

5.2.2 Grid Stag Hunt. The Grid Stag Hunt, depicted in Figure 1, is a combination of the predator-prey game [6], which encodes a single prey that all hunters work together to hunt, and the stag hunt [36]. This game is, in some respects, a more complex scenario than the prisoner's dilemma in that (1) there are three agents in the Grid Stag Hunt (rather than just two) and (2) the decision to cooperate or compete is made over a sequence of moves. The game also has a different payoff structure than the prisoner's dilemma.

Description. Three hunters are placed in a grid environment together with two prey: a stag and a hare. Only one hunter is needed to capture the hare, while all three hunters are required to capture the stag. Hunters must be located immediately next to their prey of choice for capture to be possible. To avoid accidental captures, each agent specifies in each move its intended target. Agents can stay in the same position or move up, down, left, or right. Movements that would take an agent outside of the grid cause them to reappear on the other side (e.g., moving up from the top

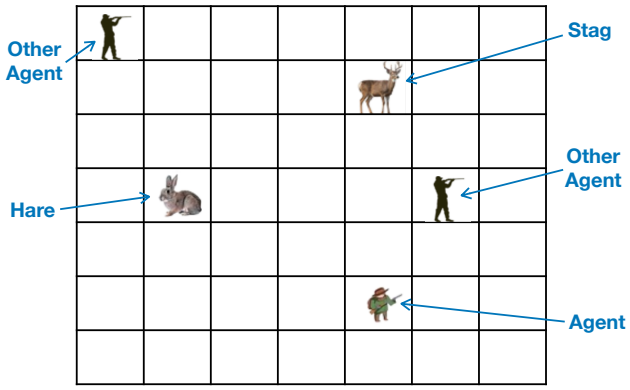


Figure 1: Depiction of the Grid Stag Hunt game.

row moves the agent to the same column in the bottom row). The stag and hare both move randomly. Each agent is initially placed in a random cell such that capture conditions are not immediately met. A captured stag yields 20 points to each hunter, whereas a hare only yields 10 points to the hunter that captured it. The encounter ends once a prey is captured. A hunter must evaluate their trust in the other two hunters in considering whether to pursue the stag.

Generators. Bandits had access to four generators, two that pursue the hare and two that pursue the stag: Greedy-Hare, Greedy Planner-Hare, Greedy Planner-Stag, and Team Aware. Greedy-Hare greedily chooses the position neighboring the hare that is closest to the agent and takes a step in that direction. Greedy Planner-Hare also chooses the position neighboring the hare that is closest to the agent, but uses the A^* path planning algorithm [22] to avoid collisions with other agents. Greedy Planner-Stag is identical to Greedy Planner-Hare, but pursues the stag instead of the hare. Team Aware pursues the stag using A^*, but considers the locations of the other hunters when choosing which position next to the stag to go to. The Greedy (with no planning) and Team Aware generators are defined in [6].

Training and Evaluation Conditions. We trained the algorithms in scenarios in which the other hunters used Team Aware, Greedy-Hare, and a bandit that randomly chose from the same set of generators as the agent (but slightly favoring the generator used in the last move). We trained for 10 epochs on 10×10 and 13×13 grids.

To test for adaptability after training, we exposed each agent to competitive and cooperative situations on 15×15 grids. To compute the competitive score, we paired each agent with two hunters that always greedily pursue the hare (using Greedy-Hare). For the cooperative score, we used three conditions: (1) self-play (i.e., the three hunters were copies of the agent being tested), (2) one hunter greedily pursued the stag and the other hunter was a copy of the agent being tested, and (3) the two other hunters greedily pursued the stag. The cooperative score was the average of the three conditions.

5.2.3 The Junior High Game. The final domain we use to evaluate agents in this paper is the Junior High Game (JHG) [35]. This game represents a more complex domain than the previous two domains, as it is played by n players (we used 10), and players

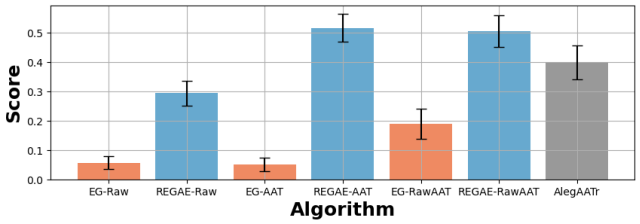
can simultaneously cooperate with some players while competing against others. This creates a vast and complex strategy space that is highly dynamic, often requiring players to adapt their strategy as the situation evolves.

Description. In the JHG, each player seeks to become popular. Over a sequence of rounds, players exchange a limited amount of tokens. Players can give tokens to other players, steal tokens from other players, and/or keep tokens for themselves. Giving tokens to other players positively boosts the popularity of the receiver, but does nothing for the giver. Stealing tokens from others negatively impacts the victim's popularity while boosting the thief's popularity. Keeping tokens helps the player maintain popularity while simultaneously serving as a form of defense by dampening the effect of theft by other players. A player's popularity also impacts the power of their actions. For example, tokens given by the most popular player are worth more than tokens given by the least popular player. Successful strategies depend substantially on strategies used by other players.

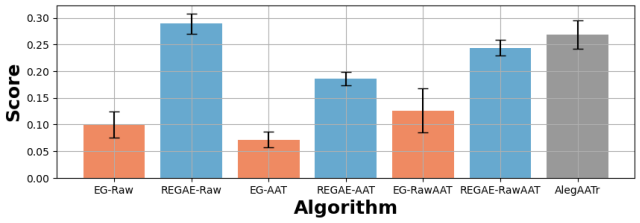
Generators. The agents were given a set of eight generators to choose from. Each of these generators use a parameterization of the CAB algorithm [35]. CABs have thirty distinct parameters that control behavior–changes to these parameters vastly alter behavior. Thus, we selected eight sets of parameters to create a variety of agent behaviors. The first six generators have parameters designed to encode the following high-level behaviors: (1) prey on players with lower popularities, (2) defend against coordinated theft, (3) seek to become friends with more popular players, (4) play a strategy optimized through genetic evolution, (5) *suck up* to more popular players by giving to them (potentially without reciprocation), and (6) always keep. The seventh and eighth generators use randomly selected parameters. We used the CAB agent code provided in [35]. More details about each generator and their parameter values are provided in the supplementary material (SM-6.1).

Training and Evaluation Conditions. Agents were trained in JHG games in which the other players were CABs with random parameterizations, CABs with parameters tuned via genetic evolution, agents that randomly allocate tokens, basic ε-greedy bandits that choose from the same set of generators as the agent being trained, and a random mixture of all of these societal conditions. We also randomly added coordinated assassin agents [35] to 10% of the simulations; these agents work by keeping all their tokens in the first round to identify who are fellow assassins, and then by stealing as much as possible from the least-popular, non-assassin player in each proceeding round. We trained each agent for five epochs in societies of 5, 10, and 15 players with games lengths of 20, 30, and 40 rounds. Initial popularity values were set in two ways: (1) all players have equal initial popularity and (2) all players have a random initial popularity.

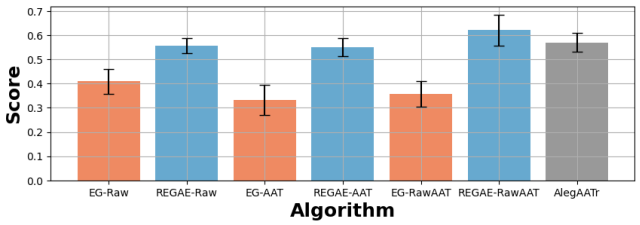
We used two evaluation conditions to compute competitive scores. Agents were placed in societies composed of players that (1) always keep all of their tokens and (2) always steal an equal amount of tokens from every other player. Competitive scores were computed as an average of these two conditions. To measure cooperation scores, we used an average of the following three conditions: (1) self-play (i.e., societies were composed of copies of the agent being tested), (2) half the other agents were copies of the agent being tested and the other agents played tit-for-tat, and (3) every player



(a) Prisoner's Dilemma



(b) Grid Stag Hunt



(c) JHG

Figure 2: Average adaptability scores in each domain. Error bars show the standard error of the mean.

but the agent being tested played tit-for-tat. In every evaluation scenario, we used society sizes of 10 players and game lengths of 30 rounds.

6 RESULTS

In this section, we present the results of our experiments, which are summarized in Figure 2 and Table 3. We first discuss the impact of the training process on agent performance. Next, we analyze how input features altered agent performance. Finally, we compare the algorithms to baseline conditions.

6.1 Impact of Training Process

Figure 2, which shows adaptability scores of each algorithm in each domain, shows a clear trend. Agents that use REGAETune training processes consistently have substantially higher adaptability scores than those that use EG training processes. For each type of input feature, the REGAETune agents substantially outperform their EG counterparts. An ANOVA shows that this trend is statistically significant ($F(1, 2100) = 122.162$, $p < 0.001$).

The competitive and cooperative scores in each domain (Table 3) give insights into how the training process impacted what the agents learned. In the repeated Prisoner's Dilemma, agents that used REGAETune generally had higher cooperative scores than

Table 3: Competitive scores ($\mathcal{S}_{\text{comp}}$), cooperative scores ($\mathcal{S}_{\text{coop}}$), and adaptability score ($\mathcal{S}_A$) for each algorithm in each domain.

	Prisoner's Dilemma			Grid Stag Hunt			JHG		
	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$
EG-Raw	0.967	0.059	0.057	0.474	0.575	0.099	0.791	0.424	0.409
EG-AAT	0.717	0.286	0.051	0.375	0.614	0.072	0.820	0.344	0.333
EG-RawAAT	0.720	0.391	0.189	0.431	0.659	0.126	0.841	0.420	0.358
REGAE-Raw	0.798	0.318	0.294	0.784	0.305	0.289	0.671	0.626	0.557
REGAE-AAT	0.726	0.640	0.517	0.546	0.245	0.186	0.607	0.628	0.550
REGAE-RawAAT	0.700	0.588	0.505	0.713	0.292	0.244	0.641	0.769	0.622
AlegAATr	0.543	0.710	0.399	0.357	0.446	0.269	0.636	0.743	0.570
Humans	–	–	–	0.095	0.510	0.049	–	–	–

those that used EG, while competitive scores were typically about the same (with the exception of perhaps EG-RAW, which had a higher competitive score). A somewhat similar trend is seen in the JHG domain: REGAETune agents had better cooperative scores, while EG agents had better competitive scores. However, the trend is reversed in Grid Stag Hunt, where EG agents had higher cooperative scores and REGAETune agents had higher competitive scores.

While these results seem to suggest that there is a simple tradeoff between REGAETune and EG training processes, a closer inspection reveals a stark and consistent difference. As mentioned, average adaptability scores, measured as the average minimum of competitive and cooperative scores over each trained model for each agent, tend to be substantially lower for EG agents than they are for REGAETune agents. This is because EG agents learned nearly static policies, wherein they almost exclusively select the same generator regardless of context. On the other hand, REGAETune agents learn to select different generators given different contexts, tending to be more likely to select generators that cooperate when associates are inclined to cooperate and generators that do not cooperate when associates are non-cooperative. As a result, their adaptability scores are much higher.

As an example illustrating this point, we consider results from the Grid Stag Hunt domain. Of the 90 trained EG models (30 for each input feature), 41 of the models *always* selected a generator that hunted hare in all evaluation conditions (regardless of associate), while 49 of the models *always* selected a generator that hunted stag. Thus, these models do not learn to adapt and seemingly converge to hunting stag or hare at random (perhaps depending on the initialization of network weights). On the other hand, REGAETune models select a wider variety of generators based on the context, and as such have higher adaptability scores. Entropy calculations over generators selected, presented in the supplementary material (SM-5.5), further show the static strategies learned by EG agents compared to REGAETune agents.

An analysis of the weights learned in the DQN illustrate stark differences caused by the EG and REGAETune training processes. Figure 3 depicts the penultimate layer of the weight vectors of the DQN for EG-AAT (left) and REGAE-AAT (right) in the repeated prisoner's dilemma, compressed to two dimensions via t-SNE [41]. Clearly, the learned weight vectors of EG-AAT are mapped to a more restricted space than those of REGAE-AAT, thus showing an indicator of EG-AAT's less adaptable behavior. Despite being exposed to the same training samples, EG-AAT maps those situations to a very small portion of the vector space. On the other hand, REGAE-AAT maps the situations to a larger portion of the vector space, which arguably means that it considers more possibilities for what the best strategy might be.

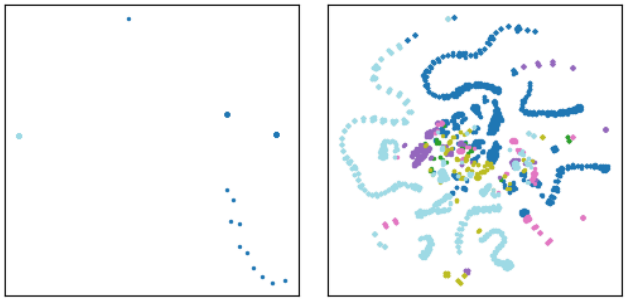


Figure 3: Learned state vectors of EG-AAT and REGAE-AAT, respectively, over every testing condition and epoch in the repeated prisoner's dilemma. Colors represent which generator was selected at the time of observing the state.

6.2 Impact of Input Features

Input features also have an impact on the adaptability scores obtained by these deep RL agents, though the differences between factor levels is less stark and clear. An ANOVA shows that the difference between input features is statistically significant ($F(2, 2100) = 4.153$, $p = 0.016$). A Tukey-Kramer multiple comparisons test shows that agents that used RawAAT input features were better than those that used only Raw input features ($p = 0.047$). RawAAT and AAT features were not statistically different ($p = 0.125$).

While using RawAAT features was better than only using Raw features, the overall difference was much smaller between different input features than it was for different training processes. Mean adaptability scores for input features were 0.284 (Raw), 0.285 (AAT), and 0.341 (RawAAT), which is just a 0.057 difference between the highest and lowest condition. On the other hand, mean adaptability scores for training process were 0.188 (EG) and 0.418 (REGAETune), showing a 0.230 difference. Additionally, we note that creating AAT input features does come at a cost, so one should consider whether that cost is worthwhile when designing deep RL agents.

In short, these results confirm that our deep RL agents can learn adaptive behavior from Raw inputs, but AAT feature inputs can also help somewhat.

6.3 Comparison to Baselines

While our results show that a REGAETune training process and (to a lesser degree) AAT input features appear to be beneficial to deep RL algorithms for learning in mixed-motive contextual bandit settings, it is interesting to further consider whether these deep RL algorithms learn successfully. To do this, we compare their performance to two baselines: AlegAATr [32] and people.

AlegAATr is an interesting comparison in that it uses AAT features, is a contextual bandit algorithm that uses the same generators as the other algorithms, and we trained it with REGAETune. However, it uses a different machine learning model (k-nearest neighbors as opposed to a DQN) to encode its policies. Thus, in comparing AlegAATr to REGAE-AAT, we can observe the impact of these two learning models. Figure 2 and Table 3 shows that, across the mixed-motive scenarios we consider in this paper, AlegAATr and REGAE-AAT are somewhat identical with respect to adaptability (mean adaptability scores across the three domains are 0.413 and 0.418, respectively).

These results suggest that the simple DQN model we used is not substantially better than using kNN when AAT features can be supplied (without AAT features, we predict that kNN's performance would drop substantially). Paired with results showing that our simple DQN model does not appear to be different than more advanced deep Q-network models (see the supplementary material), it appears that having a sophisticated architecture for learning with contextual bandits in mixed-motive scenarios may not be necessary. Other factors, such as training process, may be more important. Additional future work is required to better solidify this hypothesis.

We also compare our algorithms to the performance of people, albeit we only make comparisons in the Grid Stag Hunt domain. To do this, we conducted a small user study, wherein seven human participants played a series of 21 games (one per round) in the Grid Stag Hunt. This study placed humans in the same set of evaluation conditions (to obtain competitive and cooperative scores) as the DQN agents. In each round, three participants played with each other (i.e., self-play), two participants played with each other and a bot that greedily pursued the stag, one participant played with two bots that both greedily pursued the stag, and one participant played with two bots that both greedily pursued the hare. Participants were blinded as to which players they were paired with in each round and the order of testing conditions was randomly assigned. Overall, each participant faced the same distribution of conditions, though in different orders.

Average competitive, cooperative, and adaptability scores for human players in this domain are given in Table 3. These results show that human players performed worse than all other agents with respect to adaptability. Overall, their cooperative scores were reasonably good (lower than EG agents on average, but higher than REGAE agents), but they performed poorly in competitive scenarios. A brief, post-study survey found that the most common strategy used by human participants was to simply pursue the stag every round—a static strategy seemingly reflecting what some of the EG models learned to do. This suggests that the artificial agents were at least as successful as human players.

We do note possible confounding factors in comparing humans and artificial agents using this study. First, while we subjected human participants to the same set of evaluation conditions as the artificial agents, humans were not pretrained in this task like the artificial agents were. This means that learning effects could come into play, though we did not observe substantial variations in results from round 1 through round 21. Second, humans may have had difficulty reasoning about how the grid *wrapped around*, whereas this would not have been a confound for artificial agents. Finally, this was a small study (conducted in only a single domain) that involved only seven participants; a larger study could perhaps reveal more interesting insights. Despite these confounds, the results do seem to indicate that the agents learned reasonably good policies.

7 CONCLUSION

In this paper, we considered the design of deep RL algorithms in mixed-motive, contextual bandit settings. In these settings, agents must be able to quickly adapt their strategies to (previously unknown) associates. For example, agents must quickly identify when to cooperate versus compete. Our results show that, across the three domains considered, training deep RL algorithms (offline) in contextual bandit problems using the REGAETune exploration process resulted in substantially more adaptive agents (which better balance cooperation and competition) than training with ε-greedy exploration. We also studied how using input features with assumption alignment tracking (AAT) could help improve the adaptivity of these algorithms. Our experiments indicate that such input features can potentially be helpful, but that the impact is much less substantial compared to that of an agent's training process. As such, the improvement may not always justify the extra work required to create these input features. These results, paired with future investigations, can help us better understand how to create deep RL algorithms that learn more effectively in mixed-motive scenarios.

In this paper, true to the contextual bandit framework, the deep RL bandits select generators (experts) to use at each time step rather than raw actions. In some sense, generators reduce the complexity of the strategy space. This reduced complexity may be part of the reason that the random exploration phase of REGAETune works so well. As such, we caution against extrapolating these results beyond this context without further investigation. Future work can study how these mechanisms impact deep RL algorithms outside of the contextual bandit setting. This and other future work, combined with the results shown in this paper, can inform the design of deep RL algorithms capable of learning adaptive strategies in mixed-motive, multi-agent settings impacted by previously unknown associates.

8 SUPPLEMENTARY MATERIAL (SM)

Supplementary material, including appendix and code, can be found at https://github.com/jakecrandall/ALA2026.git.

REFERENCES

[1] Shipra Agrawal and Navin Goyal. 2013. Thompson sampling for contextual bandits with linear payoffs. In *International conference on machine learning*. PMLR, 127–135.

[2] Robin Allesiardo, Raphaël Féraud, and Djallel Bouneffouf. 2014. A neural networks committee for the contextual bandit problem. In *Neural Information Processing: 21st International Conference, ICONIP 2014, Kuching, Malaysia, November 3-6, 2014. Proceedings, Part I 21*. Springer, 374–381.

[3] C. Amato. 2024. An Introduction to Centralized Training for Decentralized Execution in Cooperative Multi-Agent Reinforcement Learning. *arXiv:2409.03052* (2024).

[4] R. Arora, O. Dekel, and A. Tewari. 2012. Online bandit learning against an adaptive adversary: from regret to policy regret. In *Proceedings of the 29th International Conference on Machine Learning* (Edinburgh, Scotland). 1503–1510.

[5] Santiago Balseiro, Negin Golrezaei, Mohammad Mahdian, Vahab Mirrokni, and Jon Schneider. 2019. Contextual bandits with cross-learning. *Advances in Neural Information Processing Systems* 32 (2019).

[6] S. Barrett, P. Stone, and S. Kraus. 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 567–574.

[7] Michael Bowling. 2004. Convergence and no-regret in multiagent learning. *Advances in neural information processing systems* 17 (2004).

[8] Ronen I. Brafman and Moshe Tennenholtz. March 2003. R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research* 3 (March 2003), 213–231.

[9] X. Cao, A. Gautam, T. Whiting, S. Smith, M. A. Goodrich, and J. W. Crandall. 2023. Robot Proficiency Self-Assessment Using Assumption-Alignment Tracking. *IEEE Transactions on Robotics* 39 (4) (August 2023), 3279–3298.

[10] N. Cesa-Bianchi, O. Dekel, and O. Shamir. 2013. Online learning with switching costs and other adaptive adversaries. In *Advances in Neural Information Processing Systems 26*. 1160–1168.

[11] Jacob W. Crandall. 2014. Towards minimizing disappointment in repeated games. *Journal of Artificial Intelligence Research* 49 (2014), 111–142.

[12] J. W. Crandall and M. A. Goodrich. 2005. Learning to Compete, Compromise, and Cooperate in Repeated General-Sum Games. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. Bonn, Germany.

[13] Chris Dann, Yishay Mansour, Mehryar Mohri, Ayush Sekhari, and Karthik Sridharan. 2022. Guarantees for epsilon-greedy reinforcement learning with function approximation. In *International conference on machine learning*. PMLR, 4666–4689.

[14] D. P. De Farias and N. Megiddo. 2006. Combining expert advice in reactive environments. *Journal of the ACM (JACM)* 53, 5 (2006), 762–799.

[15] Pedro Domingos. 2012. A few useful things to know about machine learning. *Commun. ACM* 55, 10 (2012), 78–87.

[16] Y. Du, U. Islam J. Z. Leibo, R. Willis, and P. Sunehag. 2023. A Review of Cooperation in Multi-agent Learning. *arXiv preprint arXiv:2312.05162* (2023).

[17] Maxim Egorov. 2016. Multi-agent deep reinforcement learning. *CS231n: convolutional neural networks for visual recognition* (2016), 1–8.

[18] D. Fudenberg and D. K. Levine. 1998. *The Theory of Learning in Games*. The MIT Press.

[19] A. Gautam, T. Whiting, X. Cao, M. A. Goodrich, and J. W. Crandall. 2022. A Method for Designing Autonomous Robots that Know Their Limits. In *International Conference on Robotics and Automation (ICRA)*. 121–127.

[20] Herbert Gintis. 2000. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior*. Princeton University Press.

[21] Aditya Gopalan and Gugan Thoppe. 2022. Demystifying Approximate Value-based RL with ϵ-greedy Exploration: A Differential Inclusion View. *arXiv preprint arXiv:2205.13617* (2022).

[22] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[23] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. 2014. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*. IEEE, 372–378.

[24] John Langford and Tong Zhang. 2007. The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in neural information processing systems* 20, 1 (2007), 96–1.

[25] T. Lattimore and C. Szepesvári. 2020. *Bandit algorithms*. Cambridge University Press.

[26] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. 661–670.

[27] Graham Loomes and Robert Sugden. 1982. Regret theory: An alternative theory of rational choice under uncertainty. *The economic journal* 92, 368 (1982), 805–824.

[28] Aakash Maroti. 2019. Rbed: Reward based epsilon decay. *arXiv preprint arXiv:1910.13701* (2019).

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

[30] Naoya Namiki, Kuratomo Oyo, and Tatsuji Takahashi. 2014. How do humans handle the dilemma of exploration and exploitation in sequential decision making?. In *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*. 113–117.

[31] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[32] Ethan Pedersen and Jacob W. Crandall. 2023. AlegAATr the Bandit. In *Proceedings of the* 26^{th} *European Conference on Artificial Intelligence*. 1867–1874.

[33] Tuomas W Sandholm and Robert H Crites. 1996. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems* 37, 1-2 (1996), 147–166.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[35] J. Skaggs, M. Richards, M. Morris, M. A. Goodrich, and J. W. Crandall. 2024. Fostering Collective Action in Complex Societies using Community-Based Agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Jeju, South Korea.

[36] B. Skyrms. 2003. *The Stag Hunt and the Evolution of Social Structure*. Cambridge Press.

[37] Martin Smit and Fernando P Santos. 2024. Learning Fair Cooperation in Mixed-Motive Games with Indirect Reciprocity. *arXiv preprint arXiv:2408.04549* (2024).

[38] Jeffrey R. Stimpson, Michael A. Goodrich, and Lawrence C. Walters. 2001. Satisficing and Learning Cooperation in the Prisoner's Dilemma. In *Proceedings of the* 17^{th} *International Joint Conference on Artificial Intelligence*. 535–544.

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

[40] Long Tran-Thanh, Archie Chapman, Enrique Munoz De Cote, Alex Rogers, and Nicholas R Jennings. 2010. Epsilon–first policies for budget–limited multi-armed bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 24. 1211–1216.

[41] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[42] Weixun Wang, Jianye Hao, Yixi Wang, and Matthew Taylor. 2018. Towards cooperation in sequential prisoner's dilemmas: a deep multiagent reinforcement learning approach. *arXiv preprint arXiv:1803.00162* (2018).

[43] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. (1989).

[44] Xin Xu, Lei Zuo, and Zhenhua Huang. 2014. Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information sciences* 261 (2014), 1–31.