Inference-Based Deterministic Messaging for Multi-Agent Communication

by

Varun Bhatt

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Communication is essential for coordination among humans and animals. Therefore, with the introduction of intelligent agents into the world, agent-to-agent and agent-to-human communication become necessary. Ideally, these agents should be trained in an incremental and decentralized manner. In this thesis, we first study learning in matrix-based signaling games to empirically show that, with certain payoff matrices, decentralized reinforcement learning methods can converge to a suboptimal policy. We then propose a modification to the messaging policy, in which the sender deterministically chooses the best message that helps the receiver to infer the sender's observation. Using this modification, we see, empirically, that the agents converge to the optimal policy in nearly all the runs. We then extend this method to function approximation settings, first applying it to larger matrix-based signaling games and then to a partially observable gridworld environment that requires cooperation between two agents. We show that, with appropriate approximation methods, the proposed sender modification can enhance existing decentralized training methods for more complex domains as well.

Preface

This thesis is an original work by Varun Bhatt done under the supervision of Prof. Michael Buro. The method presented in this thesis has been submitted and is under review as Bhatt, Varun and Buro, Michael, "Inference-based Deterministic Messaging For Multi-Agent Communication," at the 34th Conference on Neural Information Processing Systems (NeurIPS) 2020.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Michael Buro for providing invaluable guidance throughout my master's degree while giving me the freedom to pursue my own ideas. I am thankful to my peers for the stimulating discussions and suggestions that helped me in shaping this project. I would also like to extend my thanks to the University of Alberta Computing Science community, specifically Prof. James Wright and Prof. Rich Sutton, for their technical and non-technical advice. Finally, none of this would be possible without the continuous support from my parents and I am grateful for it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Humans rely extensively on communication to both learn quickly and to act efficiently in environments in which agents benefit from cooperation. As artificial intelligence (AI) applications become commonplace in the real world, intelligent agents therefore can benefit greatly from being able to communicate with humans and each other. For example, a group of self-driving cars can improve their driving performance by communicating with other cars about what they see and what they intend to do (Yang et al., 2004). As advances in other fields of AI have shown, a learned solution is often better than a manually designed one (He et al., 2015; Silver et al., 2018). Hence, training the agents to learn to communicate has the potential to lead to more efficient protocols than pre-defined ones.

One assumption that is commonly held when studying communication between agents is that messages do not directly affect the payoffs or the rewards that the agents obtain, which is also known as the "cheap-talk" assumption (Crawford and Sobel, 1982). While it does not accurately reflect all real-world scenarios, it is a reasonable assumption in many cases. For example, turn indicators and traffic lights do not affect driving directly because the driving outcome only depends on the drivers' actions, but not on the state of the lights.

The aim of this thesis is to study the performance of learning algorithms for synthetic cooperation tasks, such as one-step, two-agent cooperative signaling games (Gibbons, 1992), like the climbing game (Claus and Boutilier,

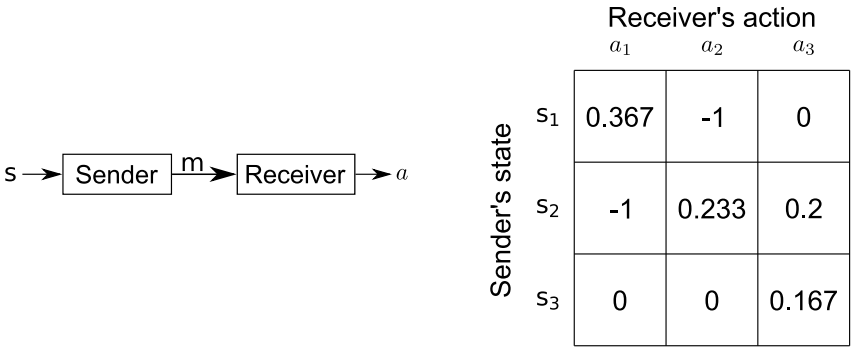

Figure 1.1: Signaling game with the normalized payoff matrix of the climbing game.

1998) depicted in Fig. 1.1 which is often used in the multi-agent reinforcement learning literature to study learning algorithms for simultaneous action games. In every round of such games, the sender receives a state s from the environment and then sends message m to the receiver which, based only on m, takes action a. In the problems studied here, both agents receive the same payoff $R(s, a)$, independent of m.

We also consider a more complex gridworld cooperation task, in which one agent receives private observations that are required by the other agent to take optimal actions. Since both agents receive the same rewards in both domains, the first agent is always motivated to correctly communicate its private observation.

We restrict our studies to decentralized training methods that learn from experience. In real-world settings, it is often necessary for agents to learn from experience since the exact model of the problem is unknown beforehand. In multi-agent cooperative problems, training can be centralized by controlling all agents through a single central controller. Centralized training makes it easier to assign credit among agents but comes at a cost of exponentially larger state and action spaces. On the other hand, decentralized training is more scalable and allows the agents to keep their training methods private while still allowing cooperation, but has the disadvantage of being harder than centralized training. Hence, we focus on finding good decentralized algorithms for cooperation and communication. In the algorithms we consider, each agent

maintains its own parameters and updates them only dependent on its private observations and rewards.

In what follows, we first discuss the relevant background and related work. We then show that learning to communicate through incremental decentralized training is a hard task even in simple signaling games. As a solution, we propose a method of communication based on the sender choosing the best message that would lead to the correct inference of its private observation. In the tabular case of signaling games, the sender exactly simulates the receiver's inference process, whereas, in the multi-step gridworld environment, an approximation is necessary. We then perform a more extensive set of experiments on signaling games and show that finding optimal policies incrementally through playing experience can be difficult for existing algorithms, but our method manages to reach an optimal policy in nearly all the runs. We also discuss how the payoff matrix affects the algorithms' ability to find the optimal policy. Finally, we present and discuss the results of our approximation-based method applied to a more complex gridworld environment.

Chapter 2

Background and Related Work

The problem considered in this thesis can be formulated as a multi-agent reinforcement learning problem and a variant of signaling games. This chapter provides a brief description of these formalisms and relevant work.

2.1 Signaling Games

A signaling game (Gibbons, 1992) is an incomplete information game played by two agents: a sender and a receiver. In each game round, the sender is assigned type s by the environment, where s is chosen from $\mathcal{S} = \{s_1, s_2, \ldots\}$ according to some known probability distribution $p(s)$. The sender then sends a message m to the receiver, which is chosen from $\mathcal{M} = \{m_1, m_2, \ldots\}$. The receiver only observes m and takes an action $a \in \mathcal{A} = \{a_1, a_2, \ldots\}$. The sender and the receiver receive payoffs $R_1(s, m, a)$ and $R_2(s, m, a)$ respectively. The problem chosen in this paper is a specific case of signaling game, in which, $R_1(s, m, a) = R_2(s, m, a) = R(s, a)$, i.e., both sender and receiver get the same payoff which does not depend on the message sent by the sender. The solution concept commonly used in signaling games is called perfect Bayesian equilibrium (Fudenberg and Tirole, 1991), which, with $\pi_1(s)$ and $\pi_2(m)$ being the sender's and receiver's respective strategies, satisfies the following conditions:

- Given message m, the receiver has a belief about the type of the sender: $p(s|m)$. The strategy of the receiver maximizes the expected payoff given

this belief. Mathematically,

$$\pi_2^*(m) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s' \in \mathcal{S}} p(s'|m) R_2(s, m, a)$$

- The sender's policy is a best response to π_2^*, i.e.,

$$\pi_1^*(s) = \underset{m \in \mathcal{M}}{\operatorname{argmax}} R_1(s, m, \pi_2^*(m))$$

- The belief held by the receiver is consistent with Bayes' rule and the sender's strategy, i.e., for all messages m that are sent by the receiver for some type s,

$$p(s|m) = \frac{\mathbb{1}_{\{m=\pi_1^*(s)\}} p(s)}{\sum_{s' \in \mathcal{S}} \mathbb{1}_{\{m=\pi_1^*(s')\}} p(s')}$$

In practice, the payoffs and the prior type distribution may not be known beforehand and only samples are observed, making it an incomplete information game and similar to the reinforcement learning setting. Hence, in this thesis, we focus on algorithms that learn policies incrementally rather than computing optimal policies directly.

2.2 Reinforcement Learning

Reinforcement learning (RL) (Sutton and Barto, 2018) is a learning paradigm that is well suited for learning incrementally through experience, and has been successfully applied to single-agent (Mnih et al., 2015) and adversarial two-player games (Silver et al., 2018; Vinyals et al., 2019).

An RL problem consists of an environment and an agent. The environment provides the agent with observations and rewards. The goal of the agent is to take actions such that the obtained rewards are maximized. Mathematically, it can be formalized as a Markov Decision Process (MDP): At each time step t, the agent receives state $s^{(t)} \in \mathcal{S}$ and takes action $a^{(t)} \in \mathcal{A}$. Based on the current state and the taken action, the agent receives reward $r^{(t+1)}$ and the next state $s^{(t+1)}$ based on distribution $p(r^{(t+1)}, s^{(t+1)}|s^{(t)}, a^{(t)})$. In a more general setting, the agents do not have access to the complete state and only

observe a partial view of the state $o^{(t)} = f(s^{(t)})$, creating a partially observable MDP (POMDP). The problems we consider in this thesis are episodic, which means that the MDP has a terminal state and the agents are guaranteed to reach it in a finite number of time steps (T), irrespective of the actions taken. In the control problem that we are interested in, the agents aim to maximize the discounted return, which is defined recursively as $G^{(t)} = R^{(t+1)} + \gamma G^{(t+1)}$, where γ is the discount factor and $G^{(T)} = 0$. The agent maintains a policy, $\pi(a|s)$, which it uses for action selection at each time step. The policy is improved incrementally in an online manner by balancing exploration to find good actions and exploitation to receive high rewards.

There are two main categories of algorithms used to train the agents. The first category of algorithms is value-based. The state-action values (called Q-values) for a policy π is defined as $Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G^{(t)}|s^{(t)} = s, a^{(t)} = a]$. Agents either maintain Q-values for each (state, action) pair (tabular setting) or estimate them using parameters. The values or the parameters are updated based on experience. Actions are selected using an exploration strategy such as ϵ-greedy, in which the action is chosen uniformly at random with probability ϵ and the action with the highest Q-value is chosen otherwise. The second category of algorithms is policy-based, in which the agents maintain a policy $\pi(a|s)$ for action selection which is often updated using the policy gradient theorem (Sutton, McAllester, et al., 1999).

2.3 Multi-Agent Reinforcement Learning

A multi-agent reinforcement learning (MARL) problem (Littman, 1994) consists of an environment and $N \geq 2$ agents, formalized using Markov games (Shapley, 1953): At each time step t, agents receive state $s^{(t)} \in \mathcal{S}$. Each agent i then takes action $a_i^{(t)} \in \mathcal{A}_i$ and receives reward $r_i^{(t+1)}$ and the next state $s^{(t+1)}$. The distributions of the reward and the next state obey the Markov property: $p(\mathbf{r^{(t+1)}}, s^{(t+1)}|s^{(\leq t)}, \mathbf{a^{(\leq t)}}) = p(\mathbf{r^{(t+1)}}, s^{(t+1)}|s^{(t)}, \mathbf{a^{(t)}})$. With partial observability or incomplete information, instead of the complete state, the agents only receive private observation $o_i^{(t)}$. In a pure cooperative setting, the

rewards $r_i^{(t)}$ agents receive are equal at every time step, converting the problem into a decentralized POMDP (Dec-POMDP).

Scenarios involving multiple learning agents can be very complex because of non-stationarity, huge policy spaces, and the need for effective exploration (Hernandez-Leal et al., 2019). One way to solve a MARL problem is to independently train each agent using a single-agent RL algorithm, treating the other agents as a part of the environment. However, due to non-stationarity, the convergence guarantees of the algorithms no longer exist (Bowling and Veloso, 2000). Additionally, when more than one equilibrium exists, selecting a Pareto-optimal equilibrium becomes a problem, in addition to actually converging to one (Claus and Boutilier, 1998; Fulda and Ventura, 2007). WoLF-PHC (Bowling and Veloso, 2002) attempts to solve this problem in adversarial games, while "hysteretic learners" (Matignon et al., 2007) and "lenient learners" (Panait et al., 2006) are examples of algorithms that work in cooperative games. A comprehensive survey of learning algorithms for multi-agent cooperative settings is given by Matignon et al. (2012), and more recently by Hernandez-Leal et al. (2019).

2.4 Multi-Agent Communication

In the context of MARL problems, communication is added in the form of messages that can be shared among agents. At each time step t, each agent i sends a message $m_i^{(t)} \in \mathcal{M}_i$ in addition to taking an action. Depending on the problem specification and the solution method, the sent message can be broadcast to all other agents or only to certain agents specified by the sender. Agents can use messages from other agents to take more informed actions.

Communication in multi-agent systems was initially studied using fixed protocols to share observations and experiences (Balch and Arkin, 1994; Tan, 1993). It was found that communication can speed up learning and leads to better performance in certain problems.

Recently, Foerster et al. (2016) proposed a method to learn to communicate using deep RL. Their first algorithm, RIAL, treats messages as additional

actions and trains using traditional deep RL algorithms, while the second algorithm, DIAL, propagates gradients through the communication channels during training. CommNet (Sukhbaatar et al., 2016) aggregates continuous messages from all agents and backpropagation through the communication channel is used for training the agents. However, backpropagating the gradient through the communication channel requires centralized training, which may not always be possible in real-world settings.

Jaques et al. (2018) and Eccles et al. (2019) focus on decentralized training. Jaques et al. (2018) use the idea of social influence to incentivize the sender to send messages that affect the actions of the receiver. Eccles et al. (2019) add additional losses to the agents in addition to the policy gradient loss. The sender optimizes an information maximization based loss while the receiver maximizes the usage of the received message.

Communication is also studied in the context of emergence of language (Evtimova et al., 2018; Lazaridou et al., 2017; Lowe et al., 2019; Wagner et al., 2003) in which agents are trained to communicate to solve a particular task with the goal of analyzing the resulting communication protocols. One of the commonly used problem setting is that of a referential game which is a special case of a signaling game in which the reward $R(s, a)$ is 1 if s and a match and 0 otherwise. In this thesis, we show that the problem becomes considerably harder when using an arbitrary payoff matrix and we propose methods to overcome this issue.

Chapter 3

Difficulties with Decentralized Learning

The signaling game described in Section 2.1 (Page 4), called Lewis signaling game (Lewis, 1969), is the simplest multi-agent communication problem. The agents have a fixed role as a receiver or a sender, with the sender only needing to send a message and the receiver only needing to take an action. There is only one step in the game which means that the agents do not need to use the history for action selection. Also, with small payoff matrices, the algorithms can use tabular policies and they can be studied in the context of communication without other confounding factors resulting from complex problems. Hence, we started experimenting with a simple signaling game which has 3 possible private states for the sender, 3 possible actions for the receiver, and 3 allowed messages.

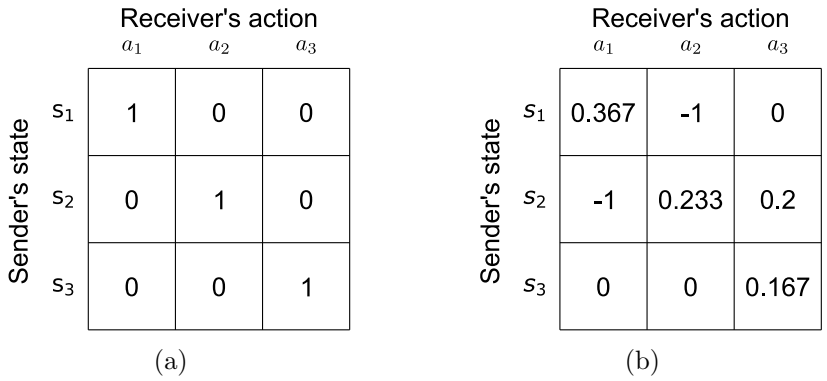

Figure 3.1: Payoff matrices used in the experiments. (a) Identity payoff matrix. (b) Climbing game.

The problem setting used in recent work studying communication is that of a referential game, in which the reward given to the agent is 1 if the receiver guesses the sender's private state and 0 otherwise, which corresponds to an identity payoff matrix. We allow the payoff to be arbitrary since, in real-world settings, some mistakes by the receiver can be costlier than others. An arbitrary payoff matrix also increases the complexity of the problem, exposing potential issues with algorithms that can are hidden when using an identity payoff matrix.

An optimal messaging policy, in this case, is to simply send a unique message for each state, which would then turn it into an easily solvable single agent problem for the receiver. In the experiments reported below, we use independent Q-Learning to train the agents and show that even in such simple settings, with certain payoff matrices, online decentralized learning is difficult.

3.1 Experimental Setup

We used two payoff matrices to illustrate the difficulties with decentralized training. In the first experiment, we used the identity payoff matrix (Figure 3.1(a)) to mirror the scenario commonly used in previous work. For the second experiment, we used the payoff matrix shown in Figure 3.1(b) (called the climbing game).

The training process consisted of 3 stages. First, the sender receives one of the three random private states from the environment and sends one of the three messages to the receiver based on the state. Then, the receiver takes one of the three actions based on the received message. Finally, the agents' policies are updated using the reward corresponding to the state and the action.

The sender maintains a Q-table with entries for each private state and each message. The receiver's Q-table contains entries for each message and each action. At each step, these values are independently updated using the Q-Learning update. We compared decentralized training with centralized training, in which a central agent has access to the observations of both agents and takes a joint action. This reduces to a single agent problem of finding the best

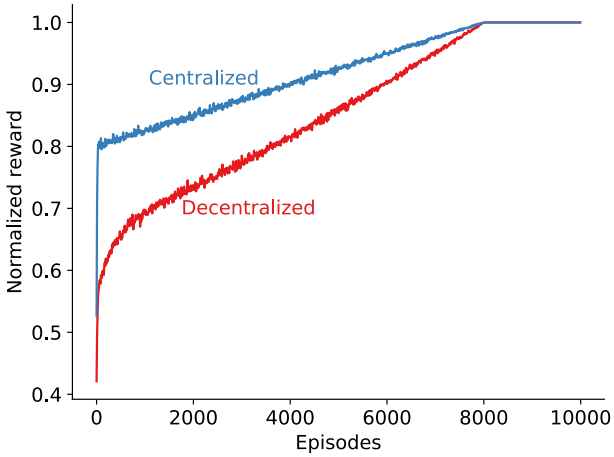

Figure 3.2: Normalized reward obtained during training, as a function of episodes, with identity payoff matrix. Both algorithms converged to the optimal policy.

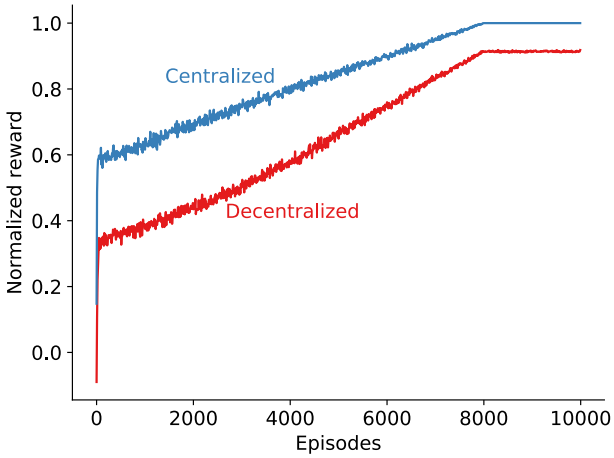

Figure 3.3: Normalized reward obtained during training, as a function of episodes, in the climbing game. Only centralized training found the optimal policy.

action for each state.

We trained the agents for 10,000 episodes with a step size of 0.1. We used ϵ-greedy exploration with an initial ϵ of 0.3, which was decayed to 0 over 8,000 episodes. Grid search was used to select the step size and exploration parameters.

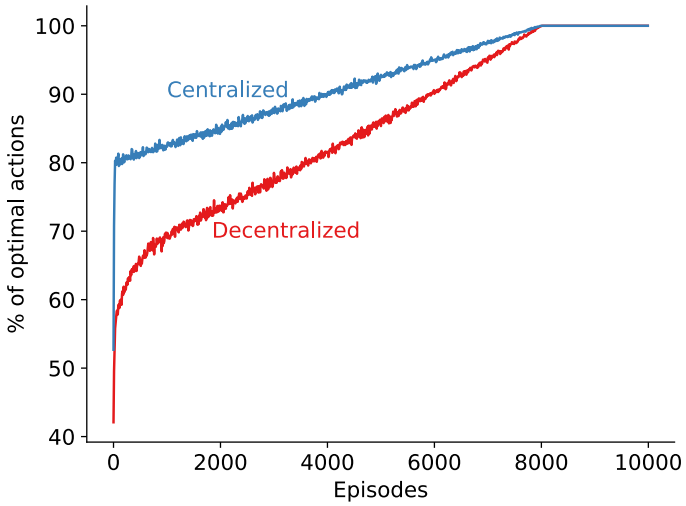

Figure 3.4: Percentage of runs that took the optimal actions, as a function of episodes, with identity payoff matrix.

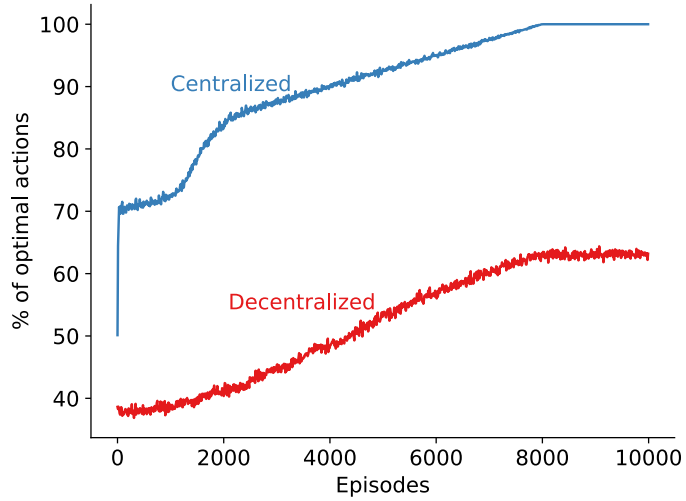

Figure 3.5: Percentage of runs that took the optimal actions, as a function of episodes, in the climbing game.

3.2 Results and Discussion

Figures 3.2 and 3.3 show the graph of reward obtained during training for centralized and decentralized training with the identity payoff matrix, and the climbing game, respectively. With the identity payoff matrix, agents find the optimal policy in both cases. But in the climbing game, only centralized training reaches an optimal policy.

The difference between the cases is clearer when looking at the percentage

of optimal actions taken. Figures 3.4 and 3.5 show this percentage during centralized and decentralized training with the identity payoff matrix and the climbing game respectively. Only 63% of the actions taken are optimal when using decentralized training on the climbing game.

Even with longer training (up to 1 million steps), decentralized Q-Learning is unable to find the optimal policy. One potential reason for this is that there are multiple sub-optimal Nash Equilibria for these games. For example, if the sender sends a single message for each state, then the best response for the receiver is to take the action that gives the highest average reward. When the receiver always takes a single action, the sender has no incentive to modify its messaging policy. We will investigate this further in Sections 5.3 (Page 23) and 5.4 (Page 26).

Chapter 4

Inference-Based Messaging Policy

In a multi-agent communication problem, agents could potentially require access to the private state of other agents to be able to find an optimal action. However, in a decentralized setting, the only information available about the private state of other agents is through received messages.

One way for the receiver to build beliefs about the private state of the sender is through Bayesian inference of private states given the message. Mathematically, given message m, prior state probability $p(s)$ for each state s, and sender messaging policy model $p(m|s)$, the posterior state probabilities are given by

$$p(s|m) = p(m|s) \cdot p(s) \ / \ \sum_{s'} p(m|s') \cdot p(s') \tag{4.1}$$

The receiver then uses the posterior belief for its action selection. For example, it can assume that the current state of the sender, $s^{(t)}$, equals $\text{argmax}_s \, p(s|m^{(t)})$ and act accordingly, or maximize the expected reward w.r.t. the posterior. There are two issues with this: During decentralized training, the receiver does not have access to the sender's messaging policy, and even if the receiver accurately models the sender's messaging policy, the posterior state probabilities would not be useful if the sender's messaging policy is not good.

In our method, we use the inference process to improve the sender's message. The sender calculates the posterior probabilities of its current state for each possible message that it can send. It then chooses the message that leads

to the highest posterior probability. Intuitively, the sender is assuming that the receiver is performing inference and chooses the message that is most likely to lead to correct inference. Mathematically, the chosen message $m^{(t)}$ is given by

$$m^{(t)} = \underset{m}{\text{argmax}}\, p(s^{(t)}|m) = \underset{m}{\text{argmax}} \left(p(m|s^{(t)}) \;/\; \sum_{s'} p(m|s')p(s') \right) \tag{4.2}$$

The $p(s^{(t)})$ term is not present in the numerator because it is constant. Henceforth, we will use the term ***unscaled messaging policy*** to refer to $p(m|s)$ and ***messaging policy*** to refer to the above inference-based policy.

The unscaled messaging policy $p(m|s)$ can be learned using any RL algorithm. For example, in our experiments, we use a value-based method, Q-Learning (Watkins and Dayan, 1992), for the matrix signaling games and an asynchronous off-policy policy gradient method, IMPALA (Espeholt et al., 2018), in the gridworld experiments. While the sender simulates inference, it does not require the receiver to infer the private state of the sender to work well. In fact, in our experiments, the receiver is trained using standard RL

Algorithm 1: Inference-Based Messaging (Signaling Game, Tabular Case)

Input: Step size α
Initialize $Q(s, m)$ arbitrarily
$\forall s \in \mathcal{S} : N(s) \leftarrow 0$
for $t \leftarrow 1, 2, \ldots$ **do**
  Receive state s from the environment
  $N(s) \leftarrow N(s) + 1$
  for $s' \in \mathcal{S}$ **do**
    $p(s') \leftarrow N(s') \;/\; \sum_{s''} N(s'')$
    $\pi(s') \leftarrow \text{argmax}_{m'} Q(s', m')$
  end
  $\forall m \in \mathcal{M} : p(s|m) \leftarrow \begin{cases} 1 & \sum_{s'} \mathbb{1}_{\{m=\pi(s')\}} p(s') = 0 \\ \frac{\mathbb{1}_{\{m=\pi(s)\}}}{\sum_{s'} \mathbb{1}_{\{m=\pi(s')\}} p(s')} & \text{otherwise} \end{cases}$
  $m \leftarrow \text{argmax}_{m'} p(s|m')$
  Send m to the receiver
  Observe r after the receiver acts
  $Q(s, m) \leftarrow Q(s, m) + \alpha(r - Q(s, m))$
end

Algorithm 2: Q-Learning Receiver

Input: Step size α, exploration rate ϵ
Initialize $Q(m, a)$ arbitrarily
for $t \leftarrow 1, 2, \ldots$ **do**
    Receive m from sender
    $a \leftarrow$ epsilon-greedy$(Q(m, \cdot), \epsilon)$
    Take action a and observe r
    $Q(m, a) \leftarrow Q(m, a) + \alpha(r - Q(m, a))$
end

algorithms: Q-Learning and REINFORCE (Williams, 1992) for matrix games, and IMPALA for gridworld. Algorithms 1 and 2 list the pseudocode of the described algorithms for the tabular cases.

4.1 Approximating Posterior Probabilities

In a small signaling game, the posterior state probability, and consequently, the message to send, can be calculated exactly by using Eq. 4.2. But in more complex environments, in which the state space is large or even infinite, it is necessary to approximate the probabilities. We use a simple empirical averaging method to approximate $p(s|m)$ while the agents are acting in the environment.

The numerator in Eq. 4.2 can be calculated using the unscaled messaging policy. The denominator can be written as $p(m) = \mathbb{E}_s[p(m|s)]$. This expectation can be approximated using the empirical mean of the unscaled message probabilities calculated during a rollout or by training a predictor to directly predict $p(m)$.

In practice, due to the small number of samples in a rollout batch, the variance in the mean estimate is high, reducing the quality of the messaging policy. We empirically found that it is better to use estimates from previous rollout batches to reduce the variance despite them being biased due to policy updates that have happened after those rollouts. Mathematically, we maintain an estimate $\hat{p}(m)$ that we update as an exponentially weighted moving average

of the empirical mean calculated during a rollout $\bar{p}(m)$ with weight μ:

$$\forall m \in \mathcal{M} : \hat{p}(m) \leftarrow \mu\hat{p}(m) + (1-\mu)\bar{p}(m) \tag{4.3}$$

Another way to estimate $p(m)$ is to train a predictor. As $p(m)$ depends on $p(m|s)$ and $p(s)$, using the policy parameters θ as the input to the predictor and the true mean as the training signal should allow the predictor to estimate $p(m)$ even when the policy is updated. Preliminary tests in the gridworld environment showed that this is a hard task, possibly due to the large number of policy parameters, the complex neural network dynamics, and not having access to the true mean. Empirically, the performance was lower when using a predictor compared to using a moving average.

Algorithm 3 implements these ideas for the sender in domains requiring approximation.

The parameters of the unscaled messaging probabilities can be updated using any RL algorithm. The updates need to account for the fact that we are updating the unscaled messaging policy (the target policy) while acting using the scaled messaging policy (the behavior policy).

We use IMPALA (Espeholt et al., 2018) in our gridworld experiments. The

Algorithm 3: Inference-Based Messaging (Gridworld, Approximation Case)

Input: Step size α, weight $0 < \mu < 1$, unscaled messaging policy $p(m|s;\theta)$
Initialize θ arbitrarily
$\forall m \in \mathcal{M} : \hat{p}(m) \leftarrow 1 \;/\; |\mathcal{M}|$
for *rollout* $k \leftarrow 1, 2, \ldots$ **do**
  $\forall m \in \mathcal{M} : \bar{p}(m) \leftarrow 0$
  for $t \leftarrow 1, 2, \ldots T$ **do**
    Receive state s from the environment
    $m \leftarrow \text{argmax}_{m'} \left(p(m'|s;\theta) \;/\; \hat{p}(m')\right)$
    $\bar{p}(m) \leftarrow \bar{p}(m) + p(m|s) \;/\; T$
    Send m to the receiver
  end
  $\forall m \in \mathcal{M} : \hat{p}(m) \leftarrow \mu\hat{p}(m) + (1-\mu)\bar{p}(m)$
  Update θ using any RL algorithm after off-policy correction
end

policy gradient in the original IMPALA update is given by

$$\rho^{(t)} \nabla_\theta \log p(m^{(t)}|s^{(t)};\theta)(r^{(t)} + \gamma v^{(t+1)} - V_\omega(s^{(t)})),$$

where $\rho^{(t)}$ is the importance sampling weight to account for off-policy asynchronous actors, $v^{(t+1)}$ is the V-trace target computed similar to the original paper, $r^{(t)}$ is the current reward, and $V_\omega(s^{(t)})$ is the value prediction given by the critic.

Since the behavior policy has a probability of 1 for the taken action and the target policy has a probability $p(m^{(t)}|s^{(t)})$, the importance weight is given by $\rho^{(t)} = p(m^{(t)}|s^{(t)}) \ / \ 1$. Hence, the policy gradient becomes

$$\nabla_\theta p(m^{(t)}|s^{(t)};\theta)(r^{(t)} + \gamma v^{(t+1)} - V_\omega(s^{(t)}))$$

To illustrate how this method of training the unscaled messaging policy and acting according to the scaled messaging probability affects the agent, we consider a problem with two messages m_1 and m_2 with the corresponding unscaled messaging policies $p(m_1|s)$ and $p(m_2|s)$. During training, the RL algorithm updates the unscaled messaging policy based on the received reward. For simplicity, we assume that $p(m_1|s^{(t)})$ is increased, and as a result $p(m_2|s^{(t)})$ is decreased, due to the update. This leads to $p(m_1)$ increasing and $p(m_2)$ decreasing. Consequently, the posterior probabilities $p(s|m_1)$ would decrease and $p(s|m_2)$ would increase $\forall s \neq s^{(t)}$. Thus, when an agent is more likely to choose a message in a particular state, it is automatically less likely to choose it in other states, leading to efficient use of the communication channel. Training the unscaled messaging policy using rewards ensures that the algorithm can override this implicit update if it improves the return.

Chapter 5

Experiments with Signaling Games

To show the effectiveness of our proposed methods we conducted three experiments using signaling games: First, we used the climbing game (Figure 1.1, Page 2) to explain the issue of convergence to sub-optimal policies. We then performed experiments on two sets of 1,000 payoff matrices of sizes 3×3 and 32×32, respectively, with each payoff generated uniformly at random between 0 and 1 and normalized to 1 by dividing all rewards by the maximum reward, to show that the observed convergence issues are not specific to the climbing game.

5.1 Experimental Setup

Each run of our experiments lasted for 1,000 episodes in 3×3 matrix games and 25,000 episodes in 32×32 matrix games. Using a Higher number of episodes gave qualitatively similar results and hence, we restricted it to the given numbers. In each episode, first, a uniform random state was generated and passed to the sender. The sender then computed its message and sent it to the receiver, which used the message to compute its action. Finally, the reward corresponding to the state and the action was sent to both agents and used to independently update their policies. The obtained rewards (normalized by the maximum possible reward) and the converged policies of the agents were recorded.

5.2 Algorithm Details

We compare our new method with a wide variety of algorithms. Our selection includes traditional single-agent RL algorithms and algorithms from the literature that are shown to be effective in cooperative games with simultaneous actions.

For each algorithm, we selected the hyper-parameters using a combination of grid search and manual tuning. We generated 100 random payoff matrices, and for each set of hyper-parameters and each payoff matrix, we repeated the experiment 1,000 times. The set of hyper-parameters with the highest percentage of runs that converged to the optimal policy was used in all other experiments. Hyper-parameter tuning was performed separately for 3×3 payoff matrices and 32×32 payoff matrices.

The algorithms we consider include:

• **Info-Q** – The information maximizing sender algorithm proposed in this thesis. The sender uses Algorithm 1 (Page 15) and the receiver uses Q-Learning (Algorithm 2, Page 16). All Q-values of the sender were initialized pessimistically (to -2) to prevent unwanted exploration by the sender. All Q-values of the receiver were initialized optimistically (to +2) to allow faster learning, but the same effect can be obtained by a higher step size. We used step size 0.1 and greedy action selection for the receiver. We found that ϵ-greedy exploration by the receiver was not required for the signaling games we chose in this thesis. The initial hyper-parameter guesses led to 100% convergence to optimal policy and hence, were not tuned further.

• **Info-Policy** – Similar to Info-Q, but the receiver uses a policy gradient method instead of Q-Learning. Updates to the receiver are performed using the REINFORCE update rule (Williams, 1992) with the value of the state as baseline. Q-values of the sender were initialized pessimistically (to -2). The step size for both the policy update and the value baseline update was set to 0.5, while the step size for Q-value updates for the sender was set to 0.05. The receiver's step size was tuned between $\{0.1, 0.5\}$ and the sender's step size was tuned between $\{0.5, 0.05\}$.

• **Fixed messages or fixed actions** – For each algorithm, we ran a version in which the policy of either the sender or the receiver was fixed to an optimal one and only the other agent is trained. With this, the problem was reduced to a single-agent problem.

• **Independent Q-Learning (IQL)** – Both the sender and the receiver are trained independently using Q-Learning. The sender maintains a Q-table with entries for each state and each message, while the receiver maintains entries for each message and each action. For 3×3 payoffs, ϵ-greedy, with initial ϵ of 0.3 and linear decay by 3.75×10^{-4} per episode, was used for action selection, and step size (α) 0.1 was used for the updates. For 32×32 payoffs, $\epsilon = 0.1, \alpha = 0.5$ were used, with ϵ decaying at a rate of 5×10^{-6} per episode. α and initial ϵ were tuned among $\{0.01, 0.05, 0.1, 0.5\}$ and $\{1, 0.5, 0.3, 0.1\}$ respectively.

• **Iterative Learning (IQ)** – The learning is iterative, with the sender updating its policy for a certain number of episodes (denoted by 'period') while the receiver is fixed, followed by the receiver updating its policy while the sender is fixed, and so on. During each period, the updates for either the sender or the receiver are exactly the same as in independent Q-Learning. An agent doesn't explore when its policy is fixed. When its policy is being trained, the action selection is ϵ-greedy, with ϵ set to 1 at the beginning of the period and linearly decayed by 0.125 after every episode. Step size 0.5 was used for the updates and periods spanned 10 episodes. For 32×32 payoffs, periods spanned 100 episodes, and ϵ decay rate was 0.0125. Step size, initial ϵ, and period were tuned among $\{0.01, 0.05, 0.1, 0.5\}$, $\{1, 0.5, 0.3, 0.1\}$, and $\{1, 10, 100\}$ respectively.

• **Model the sender (ModelS)** – The receiver maintains a Q-table for each state and each action. It also maintains a Q-table for each state and each message to model the sender. Based on the message sent by the sender and the model of the sender, the receiver calculates the posterior probabilities of each state ($p(s|m)$). The receiver assumes the true state to be the one with the highest probability, i.e., $\text{argmax}_s\, p(s|m)$, and the Q-table for states and actions is used to compute the best action as $\text{argmax}_a\, Q(s, a)$. ϵ-greedy, with initial ϵ of 1 and linear decay by 1.25×10^{-3} per episode was used for action

selection, and step size 0.05 was used for all the updates. The ϵ decay rate was lowered to 5×10^{-5}, while the step size was increased to 0.1 in the case of 32×32 payoffs. Step size and initial ϵ were tuned among $\{0.01, 0.05, 0.1, 0.5\}$ and $\{1, 0.5, 0.3, 0.1\}$ respectively.

• **Model the receiver (ModelR)** – The sender maintains a Q-table for each state and each action. It also maintains a Q-table for each message and each action to model the receiver. The sender calculates the payoffs that would be obtained if the receiver follows the modeled policy and selects the message that maximizes this payoff (with ϵ-greedy for exploration). This algorithm is similar to the one used by Sen et al. (2003), with the difference being that ModelR uses ϵ-greedy w.r.t. max of Q-values instead of Boltzmann action selection w.r.t. expected Q-values since we empirically found that ϵ-greedy performed better. An initial ϵ of 0.1 with linear decay of 1.25×10^{-4} per episode was used for action selection, and step size 0.5 was used for all updates. For 32×32 payoffs, $\epsilon = 1$, with a decay of 5×10^{-5} per episode was used. Step size and initial ϵ were tuned among $\{0.01, 0.05, 0.1, 0.5\}$ and $\{1, 0.5, 0.3, 0.1\}$ respectively.

• **Hysteretic-Q** – An adaptation of the algorithm given by Matignon et al. (2007) to our problems. The idea is to use a higher step size for positive updates and a lower step size for negative updates. ϵ-greedy was used for exploration instead of Boltzmann exploration since we empirically found that ϵ-greedy performed better for this problem. ϵ was initially set to 0.1 and linearly decayed by 1.25×10^{-4} per episode. $\alpha = 0.5$, $\beta = 0.05$ were used as the step sizes for positive and negative updates respectively. Problems with 32×32 payoffs used $\epsilon = 1$, which decayed by 5×10^{-5} per episode. α and initial ϵ were tuned among $\{0.01, 0.05, 0.1, 0.5\}$ and $\{1, 0.5, 0.3, 0.1\}$ respectively, while β was tuned among $\{0.1, 1, 10\}$ times α.

• **Lenience** – An adaptation of the algorithm presented in Panait et al. (2006) (specifically the reinforcement learning version given by Wei and Luke (2016)). In the initial time steps, Q-values are updated only if the update is positive, ignoring low rewards. As the number of episodes increases, Q-values are always updated. Step size 0.1 was used for updates. Boltzmann

action selection, with a maximum temperature of 5, a minimum temperature of 1.6×10^{-3}, and exponential temperature decay (δ) by 0.99 per episode was used. The action selection moderation factor (ω) and lenience moderation factor (θ) were set to 0.1 and 1, respectively. The minimum temperature was lowered to 0 and θ was increased to 10 in case of 32×32 payoffs. Step size, δ, maximum temperature, ω and θ were tuned among $\{0.01, 0.05, 0.1, 0.5\}$, $\{0.999, 0.995, 0.99\}$, $\{5, 50, 500, 5000\}$, $\{0.1, 1, 10\}$, $\{0.1, 1, 10\}$.

- **Comm-Bias** – Both the sender and the receiver are independently trained using REINFORCE. The sender additionally uses the positive signaling loss given by Eccles et al. (2019) during training. The auxiliary loss maximizes the mutual information between the sent message and the current state. We found that the receiver using the positive listening bias given by Eccles et al. (2019) didn't improve the performance in these games. Since our experiments are in a tabular setting, we calculated the loss exactly at each timestep instead of averaging over a batch. A step size of 0.5 was used for policy updates. Positive signaling loss was given a weight of 0.01. The weight for average message entropy term (λ) was set to 0.1 in the 3×3 case and 0.3 in the 32×32 case, while the entropy target was set to 0.5 in the 3×3 case and 0 in the 32×32 case. Step size, positive signaling loss coefficient, λ, entropy target were tuned among $\{0.1, 0.5\}$, $\{0.001, 0.01, 0.1\}$, $\{0.1, 0.3, 1.0\}$, $\{0.0, 0.5, 1.0, 1.5\}$.

5.3 Results for the Climbing Game

Experiments with the climbing game were performed using the payoff matrix given in Figure 1.1 (Page 2) to highlight the issues with the existing algorithms.

Figure 5.1 (a), (b) shows the plot of the mean normalized reward, as a function of episodes. While some of the algorithms obtain rewards close to that obtained by Info-Q, the difference is magnified in Figure 5.2 (a), (b) which shows the percentage of runs that took the optimal actions, as a function of episodes. One standard error of the mean is shaded in all the plots, but is less than the line width in most cases.

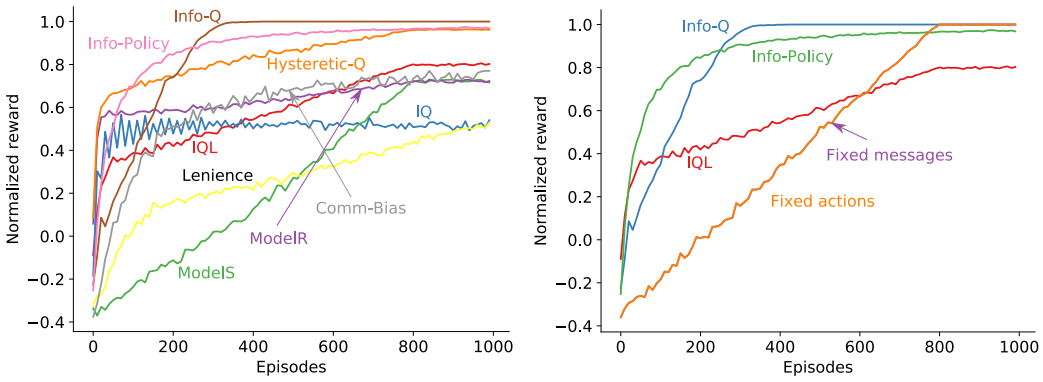

Figure 5.1: (a) Normalized reward obtained during training, as a function of episodes. The standard error across the runs is less than the line width. (b) Comparison of some algorithms with the fixed messages/actions baselines. The plots for baselines overlap since their peformance was only constrained by exploration.

For obtaining more insight into the potential issues of the baseline algorithms, we counted (state, action) pairs for each run. We iterated through the states, and for each state, we calculated the corresponding message sent by the sender and the action taken by the receiver. Figure 5.3 shows the matrix with the counts for each (state, action)-pair in case of Independent Q-Learning (IQL, on the left) and Info-Q (on the right). It can be observed that in state s_2 (middle row), the receiver takes inferior action a_3 (last column) with Q-Learning, whereas Info-Q takes optimal action a_2.

We also plotted Venn diagrams to visualize messaging policies. Figure 5.4 shows the Venn diagrams for independent Q-Learning and Info-Q. The region labeled s_i outside of any intersection corresponds to runs in which s_i was assigned a unique message. The intersection of regions s_i and s_j denote the runs in which s_i and s_j were assigned the same message. The intersection of all regions corresponds to runs in which all states were assigned the same message. Info-Q assigns a distinct message to each state, whereas there are some Q-Learning runs that assign the same message to multiple states.

The combination of (state, action) pair counts and the Venn diagrams suggests that Q-Learning is converging to a sub-optimal policy of choosing a_3

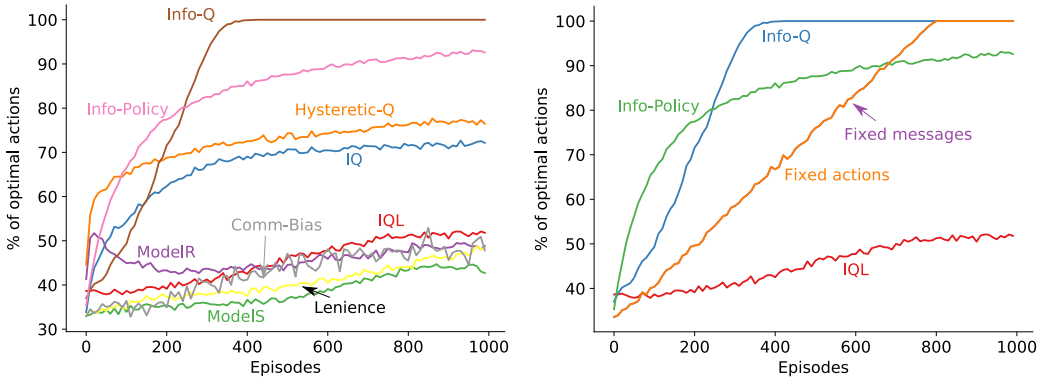

Figure 5.2: (a) Percentage of runs that took the optimal actions, as a function of episodes. The difference between the algorithms is magnified in this plot since the sub-optimal reward may be close in value to the optimal payoff. (b) Comparison of some algorithms with the fixed messages/actions baselines. The plots for baselines overlap.

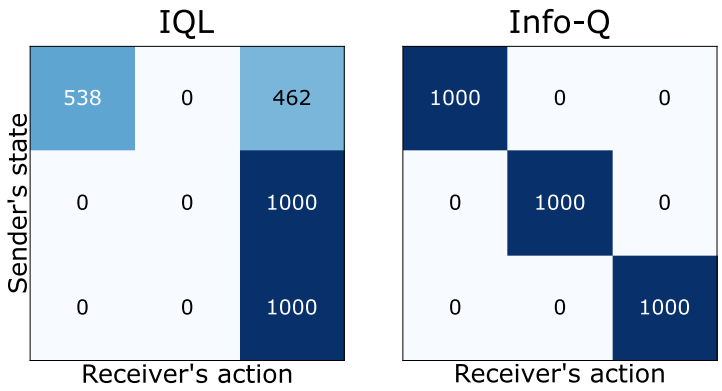

Figure 5.3: Counts of (state, action) pairs for IQL and Info-Q. With IQL, the receiver took a_3 in s_2, even though a_2 is the optimal action to take. With Info-Q, all runs converged to an optimal policy.

in s_2. We hypothesize that this is due to the closeness of the two payoffs and the fact that a_3 is a "safer" action to take since the penalty is not high if the message was incorrect. The messages corresponding to s_2 and s_3 are the same in many runs. A possible reason for this is that, since the receiver is more likely to take a_3, there is insufficient incentive to send distinct messages for s_2 and s_3. We believe that this vicious cycle leads to the agents converging to a sub-optimal policy. Info-Q overcomes this cycle by forcing the sender to fully utilize the communication channel irrespective of the incentive it receives

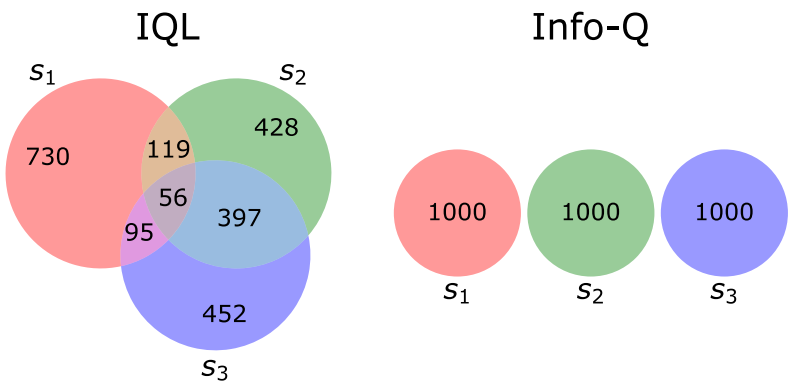

Figure 5.4: Venn diagram of messaging policy for IQL and Info-Q. Intersections in case of IQL imply that in some runs, multiple states were assigned the same message. In case of Info-Q, all states were assigned a distinct message.

through the rewards.

5.4 Random Payoff Signaling Game Results

To ensure that the issues were not specific to a single game, we conducted experiments on randomly generated payoff matrices of size 3×3 and 32×32. Figures 5.5 and 5.6 show the plots of the mean normalized reward, as a function of episodes, on 3×3 and 32×32 payoff matrices respectively. The mean, here, refers to all random payoff matrices and multiple runs for each matrix. One standard error of the mean is shaded, but less than the line width in most cases. Figures 5.7 and 5.8 show the boxplots of the percentage of runs that converge to an optimal policy for each payoff matrix of size 3×3 and 32×32 respectively. The boxplots clearly demonstrate the advantage of Info-Q compared to the other algorithms in terms of convergence to an optimal policy.

The version of our algorithm with the receiver being trained using policy gradient (Info-Policy) does not perform as well as Info-Q. This is due to the policy gradient method itself being slower to learn in this problem. Comm-Bias is also affected by it and performs worse than Info-Policy. IQ works fairly well in many problems since it is approximately equivalent to iterative best response. It fails in cases in which iterative best response converges to a sub-optimal policy. Since the agents explore more at the beginning of a

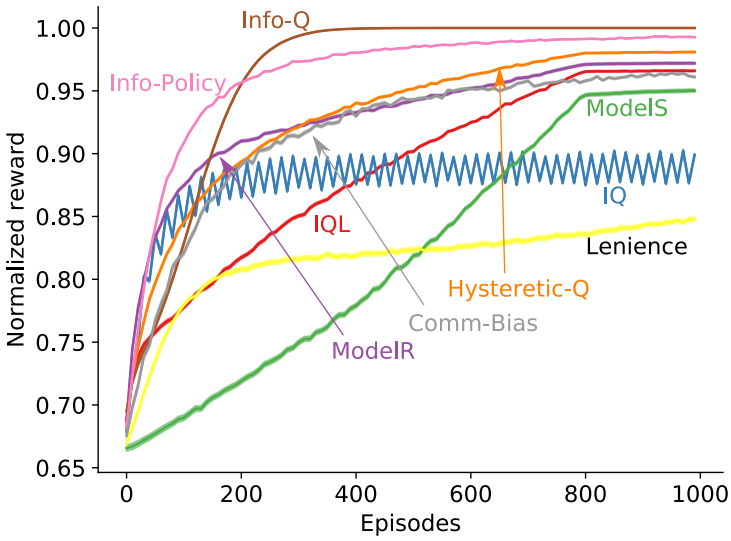

Figure 5.5: Mean normalized reward obtained during training, as a function of episodes, on random 3×3 payoff matrices.

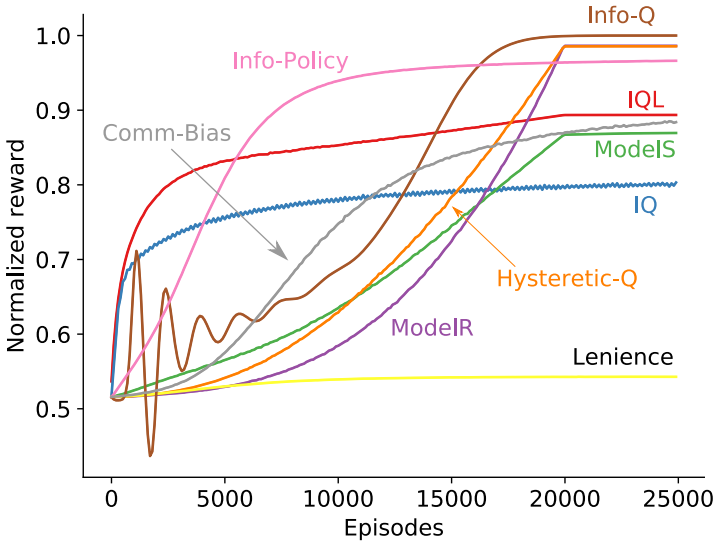

Figure 5.6: Mean normalized reward obtained during training, as a function of episodes, on random 32×32 payoff matrices.

period, the obtained reward is much lower. Hence, the reward curve for IQ does not truly reflect its test-time performance. Modeling the sender only works well if the receiver has access to the sender's state during training (but still not as well as Info-Q). Otherwise, the performance is poor as shown in the plots. Modeling the receiver suffered from the issue of sub-optimal policy of the receiver. The best response by the sender to a sub-optimal receiver policy

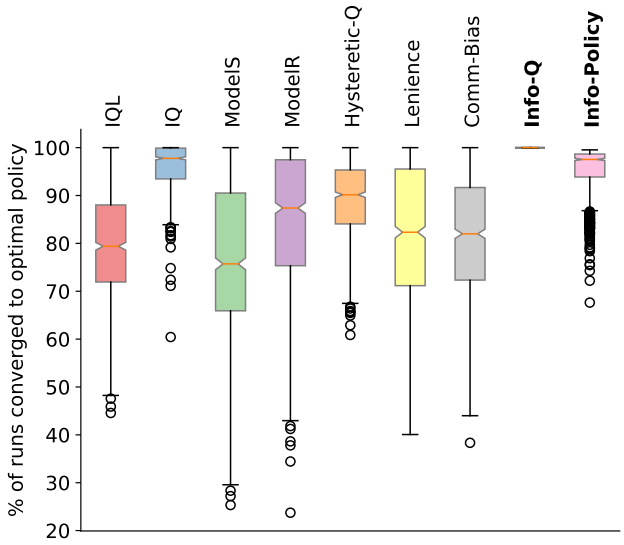

Figure 5.7: Boxplot of the percentage of runs that converged to an optimal policy for each 3×3 payoff matrix.

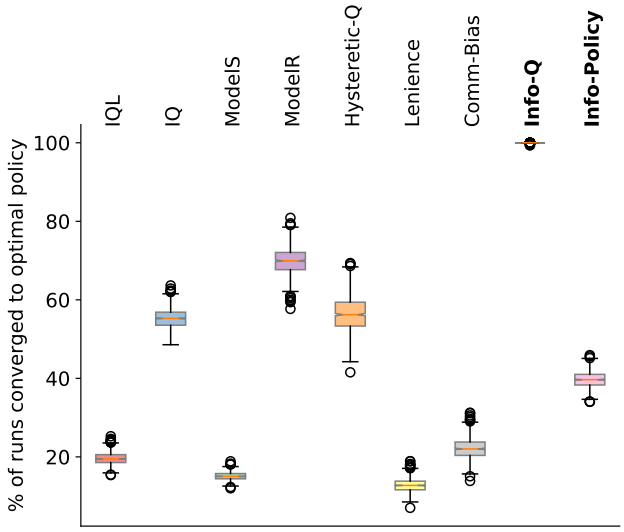

Figure 5.8: Boxplot of the percentage of runs that converged to an optimal policy for each 32×32 payoff matrix.

could be sub-optimal for the problem. The best response to this sender's policy in turn could be sub-optimal for the problem, leading to the agents not improving. Hysteretic-Q converged to an optimal policy in the case of the climbing game when the number of episodes was increased from 1e3 to 1e6. It could potentially converge to an optimal policy for random payoff matrices too if given enough episodes. But with a lower number of episodes, the percentage

of runs with optimal convergence is reduced to 89% for 3×3 payoff matrices and 56% for 32×32 payoff matrices.

5.5 Effect of the Payoff Matrix on Rewards and Optimality

Figure 5.9 shows examples of some of the randomly generated 3×3 payoff matrices that were hard for the algorithms in terms of convergence to an optimal policy. The three matrices shown in the figure have the lowest mean percentage of convergence to an optimal policy across all the algorithms. Info-Q converges to an optimal policy in all the runs with these payoff matrices. Earlier, we hypothesized that the low percentage of convergence to optimal policy is due to the closeness in the payoffs of optimal and sub-optimal actions in a state, combined with a sub-optimal action having higher payoffs in other states. This hypothesis is strengthened by these payoff matrices since they too have the same issues. Multiple pairs of actions in the top game, a_1 and a_3 in s_3 in the middle game, and a_1 and a_3 in s_2 in the bottom game have close payoffs. In each case, the sub-optimal action has a higher expected payoff across all the states.

Since the sub-optimal payoffs are close to the optimal payoffs in these matrices, the algorithms still receive a high reward even when they are choosing a sub-optimal action. In contrast, the payoff matrices shown in Figure 5.10 lead to the algorithms receiving the lowest mean reward. While the percentage of runs that converge to the optimal policy is higher for these matrices, sub-optimal actions give much lower reward compared to the optimal ones. The mean final reward over all algorithms and all runs was around 0.9 for these matrices.

While these examples are extremes, these problems are present even for other payoff matrices. We perform a simple experiment to test the effect of payoff matrix. We start with an 3×3 identity payoff matrix (with which all algorithms almost always reach the optimal policy) and modify certain payoffs by varying a variable x from 0 to 1:

Sender's state	Receiver's action a_1	a_2	a_3
s_1	0.170	1	0.999
s_2	0.889	0.012	0.888
s_3	0.391	0.332	0.336

Sender's state	Receiver's action a_1	a_2	a_3
s_1	0.655	0.685	0.755
s_2	0.856	1	0.117
s_3	0.502	0.149	0.503

Sender's state	Receiver's action a_1	a_2	a_3
s_1	0.610	0.639	0.435
s_2	0.957	0.216	0.966
s_3	1	0.769	0.076

Figure 5.9: Three payoff matrices among the randomly generated ones that resulted in the lowest convergence-to-optimal percentage in terms of the mean over all algorithms.

Sender's state	Receiver's action a_1	a_2	a_3
s_1	1	0.157	0.065
s_2	0.682	1	0.240
s_3	0.591	0.008	1

Sender's state	Receiver's action a_1	a_2	a_3
s_1	1	0.031	0.715
s_2	0.180	1	0.734
s_3	0.028	0.307	1

Sender's state	Receiver's action a_1	a_2	a_3
s_1	1	0.105	0.715
s_2	0.086	1	0.668
s_3	0.467	0.136	1

Figure 5.10: Three payoff matrices among the randomly generated ones that resulted in the lowest final mean reward over all algorithms.

- Set $R(s_2, a_3) = x$.
- Set $R(s_2, a_3) = x$ and $R(s_3, a_2) = x$
- Set $R(s_2, a_3) = x$ and $R(s_2, a_2) = 1 + x$

Figure 5.11 shows the percentage of runs converging to the optimal policy across all the algorithms as x is varied. In the first case, as x approaches 1, the difference between $R(s_2, a_2)$ and $R(s_2, a_3)$ reduces and the average payoff of a_3 increases, which leads to the percentage of optimal runs dropping. In the second case, the average payoff is the same for all the actions, but the percentage of optimal runs still drops, showing that the difference between

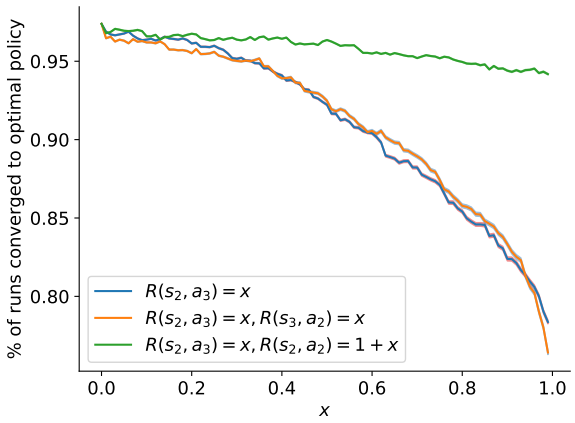

Figure 5.11: Effect of payoffs on the percentage of runs converging to the optimal policy across all the algorithms.

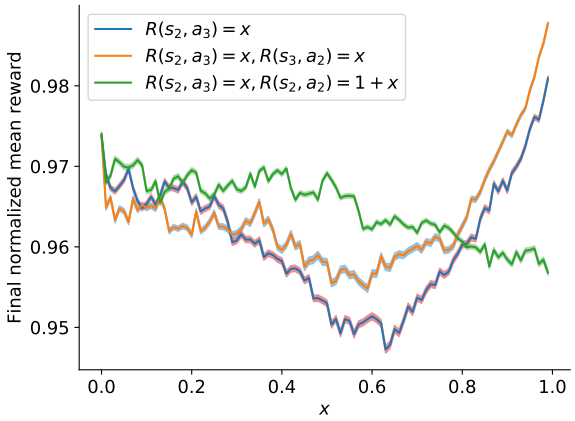

Figure 5.12: Effect of payoffs on the final mean normalized reward across all the algorithms.

rewards might be the bigger reason for poor performance. When the absolute difference between rewards is kept constant in the last case, the drop is smaller, which further reinforces the hypothesis of close rewards making it hard for the algorithms to converge to the optimal policy.

We repeated the experiments using the final reward as the comparison metric instead of the percentage of runs reaching the optimal policy. Figure 5.12 shows the effect of payoff matrix variation on the mean final reward obtained over all algorithms. The third case has a small drop in reward, while in the

first two cases, the reward drops initially before rising. The result is unsurprising since the percentage of optimal runs reduces as x increases but the sub-optimal reward also increases at the same time. Hence, the obtained reward decreases initially until the sub-optimal reward becomes large enough to make the received reward higher.

Thus, sub-optimal payoffs being close to the optimal payoff leads to most of the algorithms not finding the optimal policy. The difficulty increases as the sub-optimal payoff becomes closer, but the penalty for choosing the sub-optimal action also reduces. When the optimal and sub-optimal rewards are well separated (like in identity payoff matrix), the rewards obtained during exploration provide enough incentive to choose a unique message for each state. Our method, Info-Q, finds the optimal policy even in these difficult problems since it gives higher priority to less-used messages in addition to using the rewards obtained during training.

Chapter 6

Experiments with a Partially Observable Gridworld

In the previous chapter, we showed experiments on signaling games with a finite and relatively small number of states and actions. Real-world settings are often more complex in three ways. First, the problem generally has multiple time-steps in each episode, which requires the agents to reason about the sequence of messages. Second, the state and the action spaces can be infinite which means that function approximation needs to be used for maintaining the unscaled messaging policy as well to approximate the posterior probabilities. Finally, with increased problem complexity, the time required for an agent to learn to best respond to a message is much higher. As a consequence, credit assignment becomes harder for the sender since it cannot distinguish easily between a bad message and a bad response. To study the effects of these complexities on our algorithm, we focus on a gridworld environment in which two agents take actions as well as send messages to the other agent while cooperating towards a common goal.

6.1 Experimental Setup

We use the gridworld environment called "Treasure Hunt" introduced by Eccles et al. (2019) to test our algorithm in domains in which exact inference is infeasible. The previously discussed complexities are all present in this environment. There are multiple time-steps in the episodes and the agents

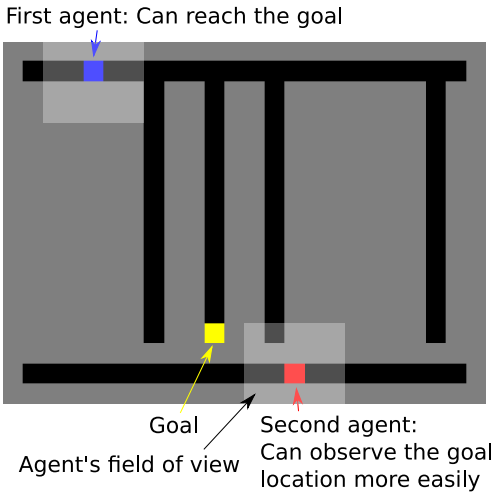

Figure 6.1: The Treasure Hunt environment.

communicate at each time-step. The problem is partially observable, which, combined with the large size of the gridworld, makes it necessary for the policies to be approximate. The learning time is also high, which means that the agents will not be able to quickly best respond to other agents' messaging policies.

6.1.1 Environment Details

There are two agents in the environment with the common goal of reaching the treasure at the bottom of one of the tunnels. Both agents have a limited field of view, and their positions make it so that one agent can easily see the goal location but cannot reach it, while the other agent can reach the goal, but potentially needs to explore all tunnels before reaching the goal. By allowing communication, one agent can easily locate the goal and signal the location to the other agent. Specifically, both agents receive the message that the other agent sent at the previous time step. Hence, both agents take the role of a sender and a receiver, in contrast to the signaling game, in which there is only one sender and one receiver.

Figure 6.1 shows a screenshot of the environment. For each training episode, an 18 high and 24 wide grid is created as follows:

- Create two tunnels between the second column and the second to last

column at the second and the second to last rows.

- Randomly choose four columns with a spacing of at least two columns between each of them. Create a tunnel of length 14 starting from the top tunnel in each of these columns.
- Place the first agent in a random cell in the top horizontal tunnel. Place the second agent in a random cell in the bottom horizontal tunnel.
- Place the goal at the bottommost cell of a random vertical tunnel.

The goal location is moved to the bottommost cell of a random vertical tunnel every time an agent reaches it. The episode is terminated after 500 time-steps and the grid is regenerated using the above rules.

At each time step, both agents observe a 5×5 area centered around their current location. The agents can move in any cardinal direction or take no action. The agents can also send one of the 5 messages that is shared with the other agent in the next time step. If an agent moves into a wall, it stays in its previous location. If it moves into the goal location, both agents get a reward of 1.

6.1.2 Training Method

The network architecture and the training method we used for our experiments is ~~kept~~ similar what is different? to that given by Eccles et al. (2019) to keep the comparisons fair. Agents were trained in an online manner using a modification of the Importance Weighted Actor-Learner Architecture (IMPALA) algorithm (Espeholt et al., 2018). During each time-step of an episode, both agents select an action and a message using its hidden state, the current observation and the message sent by the other agent in the previous time-step. Specifically, each agent uses its own network with the following architecture:

- Pass the 5×5 observations through a convolutional neural network (CNN) with 6 channels, kernel size 1, and stride 1.
- Flatten the CNN output and concatenate the received message. Pass it through two fully connected layers (MLP) with 64 units each.

- Concatenate the previous reward and action to the MLP output and pass it to an LSTM with 128 hidden units.
- Pass the LSTM outputs through a separate linear layer for action logits, message logits, and the value estimate.

The action policy and the unscaled message policies of agents are updated using the IMPALA algorithm.

Contrastive Predictive Coding (CPC) (Oord et al., 2018) is used to improve LSTM performance. The CPC loss is calculated as follows:

- Transform the LSTM inputs by passing it through a linear layer with 64 outputs to get the input projection.
- Get the output projection corresponding to predictions of inputs at 20 future time steps by passing the LSTM outputs through 20 different linear layers with 64 outputs each.
- Compute the dot product of the output projection at time t and input projection at time $t + k$ across all batches for $k = 1, 2, \ldots, 20$.
- Calculate the CPC loss as the cross-entropy loss using the dot products as logits and the prediction being true when the input and output projections are from the same batch.

In our method, the message selection uses inference simulation as shown in Algorithm 3 (Page 17) to compute $m^{(t)} = \text{argmax}_m \, p(s^{(t)}|m)$. As a baseline (called **No-Bias**), we used agents that picked messages based on the learned unscaled messaging policy and were trained independently using IMPALA. We also compared our method with the biases given by Eccles et al. (2019). With positive signaling bias, the sender was incentivized to send a message with higher information, using losses based on mutual information between private states and messages. With positive listening bias, the receiver was incentivized to use the received message for its action selection by maximizing the distance between the receiver's policy distribution and the distribution induced by not using the messages. It was achieved by maximizing the divergence between

action probabilities resulting when messages are used and those when the messages are zeroed out.

The algorithms were implemented using the RLLib (Liang et al., 2018) library. The hyper-parameters were selected using those used by Eccles et al. (2019) as the starting point and performing grid search over the hyper-parameters ~~written~~ listed below. The other hyper-parameters were unchanged or manually chosen such that the mean episode reward after 100 million time steps was maximized. Each experiment was repeated 12 times and each run lasted for 300 million time steps. RMSProp optimizer (Hinton et al., 2012) was used for updating the weights, with an initial learning rate of 10^{-3}, exponentially decayed by 0.99 after every million steps. ϵ for the optimizer was set to 10^{-6}. The optimizer was tuned between Adam (Kingma and Ba, 2015) with a learning rate of 10^{-4} and no decay, Adam with a learning rate of 10^{-4} and exponential decay by 0.99 after every million steps, RMSProp with a learning rate of 10^{-4} and no decay, RMSProp with a learning rate of 10^{-3} and exponential decay by 0.99 after every million steps. RMSProp ϵ was further tuned between $\{10^{-6}, 10^{-3}\}$. All gradients were scaled such that the gradient norm was at most 10. The maximum gradient norm was tuned between $\{10, 40\}$. Each rollout consisted of 100 time-steps and a batch size of 16 was used. 32 asynchronous parallel actors were used to collect data from the environment. The batch size was tuned between $\{16, 32\}$, but rollout length was not tuned. Policy, value and entropy losses were balanced by using a coefficient of 0.5 for the value loss and 0.006 for the entropy loss. The value loss coefficient was tuned between $\{0.5, 1\}$, while the entropy loss coefficient was tuned among $\{0.01, 0.006, 0.001\}$. The discount factor was set to 0.99. The hyper-parameters for the positive signaling and positive listening biases were the same as those given by Eccles et al. (2019). For our method, the hyperparameter μ in Algorithm 3 (Page 17) was set to 0.5 after tuning among $\{0, 0.25, 0.5, 0.9\}$.

Table 6.1: Comparison of our work with the biases given by Eccles et al. (2019)

Method	**Final reward**	**Fraction of good runs**
No-Bias (our implementation)	11.55 ± 1.03	0.42 (5 / 12)
No-Bias (Eccles et al., 2019)	12.45 ± 0.48	0.28
Positive signaling (our implementation)	16.45 ± 0.20	1 (12 / 12)
Positive signaling (Eccles et al., 2019)	14.22 ± 0.36	0.84
Positive signaling + listening (our implementation)	16.25 ± 0.20	1 (12 / 12)
Positive signaling + listening (Eccles et al., 2019)	15.14 ± 0.33	0.94
Inference-Based Messaging	14.29 ± 1.26	0.92 (11 / 12)
Inference-Based Messaging + positive listening	15.48 ± 0.76	0.92 (11 / 12)

6.2 Results and Discussion

Table 6.1 shows the results we obtained for methods presented in Eccles et al. (2019) and our inference-based method. Similar to their paper, we divide the runs into two categories: good runs, in which the final reward is greater than 13, and all runs. Good runs require efficient communication since the maximum reward achieved without communication is 13, as shown by Eccles et al. (2019). Due to discrepancies in the results of our implementation of the communication biases and the values reported by Eccles et al. (2019), we provide both the values in the table. We believe that the difference in the number of repetitions for each experiment and the number of training steps to be the reason for these discrepancies. Due to practical computational constraints, we could not perform longer runs or more repetitions.

Inference-based messaging performed significantly (more than one standard error of the mean difference) better than No-Bias. Adding positive listening bias further improved the performance of inference-based messaging. The performance of our method is similar to the reported value of positive signaling bias, but worse compared to our implementation of it. Our method combined with positive listening performs similarly to the reported value with both signaling and listening biases and is slightly worse (but not significant) when

compared to our implementation of the biases. Agents receive more than 13 reward per episode after training in 11 of the 12 runs with inference-based messaging, which, as described earlier, shows that the agents are learning to communicate in most of the runs.

The results indicate that inference-based messaging can also improve communication in complex multi-agent environments, and be complementary to methods that improve the receiver. The improvement not being as significant as in the signaling games could be a result of the inaccurate estimate of the posterior probabilities. As described in Section 4.1 (Page 16), the estimates given by the moving average of the empirical mean is biased since it uses data from older rollouts which were generated using previous policies. Finding a better estimation method is an open problem.

Chapter 7

Conclusions and Future Work

The contributions of this thesis are threefold. First, we demonstrated that state-of-the-art MARL algorithms often converge to sub-optimal policies in a signaling game. By analyzing random payoff matrices we found that a sub-optimal payoff being close to the optimal payoff, in combination with the sub-optimal action having higher average payoffs in other states can lead to such behavior.

We then proposed a method in which the sender simulates the receiver's Bayesian inference of its private state given a message to guide its message selection. Training agents with this new algorithm led to convergence to the optimal policy in nearly all runs, for a varied set of payoff matrices. The motivation to use the full communication channel irrespective of the reward appears to help the learning agents to converge to the optimal policy.

Finally, we applied our method to a more complex gridworld problem which requires probability approximations for the inference simulation process. In this domain, too, we could show performance gains.

However, with approximation, our method doesn't always reach an optimal policy. We believe that our algorithms can be further improved with more sophisticated inference approximation techniques.

Our algorithm can also be improved in cases where the message bandwidth is limited. One of the main assumptions in this thesis is that good messages contain information about the private state. When there is limited bandwidth, it is not possible to share full information about the state. In such cases,

messages with the highest amount of information about the private state may not always be optimal. For example, if multiple states have the same optimal action, then it is not necessary to be able to distinguish between them. A future extension to our work could take this into account while choosing the message.

Moreover, as shown in this thesis, learning to communicate using decentralized training is a hard problem. Currently, communication is used for sharing observations while acting to allow agents to make more informed decisions. One possible way to simplify decentralized training is to allow agents to send a second message during training that is used for sharing information useful for agents' policy updates. For example, in Simplified Action Decoder (Hu and Foerster, 2020), agents send the greedy action to other agents in a separate channel, which is used during the training process to differentiate between exploratory and non-exploratory actions. More generally, the agents could learn what message to send, allowing agents to simulate a centralized training setting while still being decentralized.

References

Balch, Tucker R. and Ronald C. Arkin (1994). "Communication in Reactive Multiagent Robotic Systems." In: *Autonomous Robots* 1.1, pp. 27–52.

Bowling, Michael and Manuela Veloso (2000). *An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning*. Tech. rep. School of Computer Science, Carnegie-Mellon University.

— (2002). "Multiagent Learning Using a Variable Learning Rate." In: *Artificial Intelligence* 136.2, pp. 215–250.

Claus, Caroline and Craig Boutilier (1998). "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems." In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference*, pp. 746–752.

Crawford, Vincent P and Joel Sobel (1982). "Strategic information transmission." In: *Econometrica: Journal of the Econometric Society*, pp. 1431–1451.

Eccles, Tom et al. (2019). "Biases for Emergent Communication in Multi-agent Reinforcement Learning." In: *Advances in Neural Information Processing Systems*, pp. 13111–13121.

Espeholt, Lasse et al. (2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." In: *Proceedings of the 35th International Conference on Machine Learning*, pp. 1406–1415.

Evtimova, Katrina et al. (2018). "Emergent Communication in a Multi-Modal, Multi-Step Referential Game." In: *Proceedings of the 6th International Conference on Learning Representations*.

Foerster, Jakob N. et al. (2016). "Learning to Communicate with Deep Multi-Agent Reinforcement Learning." In: *Advances in Neural Information Processing Systems*, pp. 2137–2145.

Fudenberg, Drew and Jean Tirole (1991). "Perfect Bayesian Equilibrium and Sequential Equilibrium." In: *Journal of Economic Theory* 53.2, pp. 236–260.

Fulda, Nancy and Dan Ventura (2007). "Predicting and Preventing Coordination Problems in Cooperative Q-learning Systems." In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 780–785.

Gibbons, Robert (1992). "A Primer in Game Theory." In:

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034.

Hernandez-Leal, Pablo, Bilal Kartal, and Matthew E. Taylor (2019). "A survey and critique of multiagent deep reinforcement learning." In: *Autonomous Agents and Multi-Agent Systems* 33.6, pp. 750–797.

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (2012). "Neural Networks for Machine Learning Lecture 6a: Overview of Mini-Batch Gradient Descent." In: *Coursera*.

Hu, Hengyuan and Jakob N. Foerster (2020). "Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning." In: *Proceedings of the 8th International Conference on Learning Representations*.

Jaques, Natasha et al. (2018). "Intrinsic Social Motivation via Causal Influence in Multi-Agent RL." In: *CoRR* abs/1810.08647.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." In: *Proceedings of the 3rd International Conference on Learning Representations*.

Lazaridou, Angeliki, Alexander Peysakhovich, and Marco Baroni (2017). "Multi-Agent Cooperation and the Emergence of (Natural) Language." In: *Proceedings of the 5th International Conference on Learning Representations*.

Lewis, David (1969). *Convention: A Philosophical Study*.

Liang, Eric et al. (2018). "RLlib: Abstractions for Distributed Reinforcement Learning." In: *Proceedings of the 35th International Conference on Machine Learning*, pp. 3059–3068.

Littman, Michael L. (1994). "Markov Games as a Framework for Multi-Agent Reinforcement Learning." In: *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 157–163.

Lowe, Ryan et al. (2019). "On the Pitfalls of Measuring Emergent Communication." In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 693–701.

Matignon, Laëtitia, Guillaume J. Laurent, and Nadine Le Fort-Piat (2007). "Hysteretic Q-Learning : An Algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams." In: *Proceedings of the International Conference on Intelligent Robots and Systems*, pp. 64–69.

— (2012). "Independent Reinforcement Learners in Cooperative Markov Games: A Survey Regarding Coordination Problems." In: *The Knowledge Engineering Review* 27.1, pp. 1–31.

Mnih, Volodymyr et al. (2015). "Human-Level Control Through Deep Reinforcement Learning." In: *Nature* 518.7540, pp. 529–533.

Oord, Aäron van den, Yazhe Li, and Oriol Vinyals (2018). "Representation Learning with Contrastive Predictive Coding." In: *CoRR* abs/1807.03748.

Panait, Liviu, Keith Sullivan, and Sean Luke (2006). "Lenience Towards Teammates Helps in Cooperative Multiagent Learning." In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems*. Citeseer.

Sen, Sandip, Stéphane Airiau, and Rajatish Mukherjee (2003). "Towards a Pareto-Optimal Solution in General-Sum Games." In: *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent Systems*, pp. 153–160.

Shapley, Lloyd S (1953). "Stochastic Games." In: *Proceedings of the National Academy of Sciences* 39.10, pp. 1095–1100.

Silver, David et al. (2018). "A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play." In: *Science* 362.6419, pp. 1140–1144. DOI: 10.1126/science.aar6404.

Sukhbaatar, Sainbayar, Arthur Szlam, and Rob Fergus (2016). "Learning Multiagent Communication with Backpropagation." In: *Advances in Neural Information Processing Systems*, pp. 2244–2252.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement Learning: An Introduction*. MIT press.

Sutton, Richard S., David A. McAllester, et al. (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems*, pp. 1057–1063.

Tan, Ming (1993). "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents." In: *Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337.

Vinyals, Oriol et al. (2019). "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning." In: *Nature*, pp. 1–5.

Wagner, Kyle et al. (2003). "Progress in the Simulation of Emergent Communication and Language." In: *Adaptive Behaviour* 11.1, pp. 37–69.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-Learning." In: *Machine Learning* 8.3-4, pp. 279–292.

Wei, Ermo and Sean Luke (2016). "Lenient Learning in Independent-Learner Stochastic Cooperative Games." In: *Journal of Machine Learning Research* 17, 84:1–84:42.

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." In: *Machine Learning* 8, pp. 229–256.

Yang, Xue et al. (2004). "A Vehicle-to-Vehicle Communication Protocol for Cooperative Collision Warning." In: *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems*, pp. 114–123.