MSc in Computer Science 2020-21

Project Dissertation

Project Dissertation title: Cheap Talk Discovery and Utilization in Multi-Agent Reinforcement Learning

Term and year of submission: Trinity Term 2021

Candidate Number: 1049264

Word Count: 19034

Abstract

In recent years, we have witnessed numerous successes in the field of Reinforcement Learning (RL) in tackling sequential decision-making problems, such as playing the Game of Go, autonomous driving, and robotics control. The advancement was catalyzed by the use of deep neural networks as function approximators. Most of these early successes assumed the single-agent setting, in which an agent learns the optimal behavior in an environment to maximize rewards. As the field progresses to solve more difficult problems, the domain of cooperative Multi-Agent Reinforcement Learning (MARL) has regained growing popularity given its resemblance to how human agents solve problems in a communicative and collaborative manner. These problems are significantly harder to solve given the more nonstationary dynamics. Precisely, MARL tries to tackle sequential decision-making problems of multiple agents that act in the same environment toward solving a common goal.

One of the key challenges in MARL is to develop agents that can communicate with each other efficiently. Most existing approaches tackle this task by enabling agents to send either discrete or continuous messages to each other through free communication channels, commonly known as "cheap talk channels". The knowledge of these channels' existence among agents allows a variety of methods to be employed to exploit the channels in learning how to communicate.

In this project, we lift the requirement for cheap talk channels to be known by the agents apriori. In other words, these channels have to be discovered by the agents within the environment before learning how to use them. Hence, the problem we tackle can be broken down into two main parts, namely Cheap Talk Discovery and Cheap Talk Utilization. By combining some of the recent methods like Off-Belief Learning and Differentiable Inter-Agent Learning, with our ideas based on mutual information maximization, we propose methods to effectively perform discovery and

utilization and examine them in our custom environments. We report promising results based on our proposed methods that can discover and utilize communication channels effectively while all existing baselines fail to solve such tasks.

Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a crucial field in machine learning that has advanced a lot along with the success of deep learning [59, 15]. Being a paradigm to solve sequential decision-making tasks, it shows promise in pushing machine learning from solely offering actionable data to offering actionable decisions, leading to a greater level of automation. In short, the RL paradigm trains agents that interact with the environment to collect experience. The agents learn from the experience to maximize their respective utility functions. Details will be provided in the next chapter. By using deep neural networks as its function approximators, deep RL (DRL) has made rapid progress in tackling complex tasks like playing Go [59] and performing autonomous driving [57]. Within the realm of real-world applications, it has demonstrated promising solutions to tasks including computer chip design, cooling system control in data centres, and personalized sepsis treatment policy [36, 31, 50].

In this thesis, we focus on multi-agent reinforcement learning (MARL), a subfield of RL. MARL tries to tackle sequential decision-making problems of multiple agents that act in the same environment. Each agent aims to maximize its own rewards by interacting with the environment and other agents [5]. Depending on the environment, the rewards can be shared among agents in a cooperative, competitive, or mixed setting. With advancements in DRL, more difficult problems can be tackled, giving rise to MARL's growing popularity as it resembles that of how human agents solve problems in a communicative and collaborative manner. Many real-life problems can be reframed in the MARL setting. Having agents that can work together to tackle tasks effectively would bring promise in applications over a variety of domains, including robotics, distributed control, telecommunications, and economics [5]. They include areas like large-scale fleet management and energy-sharing optimization [43].

Effective communication is essential for successful and effective multi-agent systems. Communicating the right information at the right time and the right place would facilitate efficient sharing of information to achieve a certain task quicker. In some cases, communication itself is required for task completion. In the MARL setting, communication is often provided as persistent channels (also known as cheap talk channels) between agents to send messages and the agents have to learn what to send by forming an effective protocol. However, existing work assumes agents' knowledge on these channels, sometimes including their properties like channel capacity and noise level, which is arguably something that should be learned by the agents for the sake of effective adaptation when we envision agents learning in a changing and open environment. At the same time, in many problems, the domain knowledge in assuming cheap talk channels' existence does not always hold. Hence, the main purpose of this thesis is to take away these assumptions on cheap talk channels. In other words, agents have to learn to discover these channels before they can learn how to use them. We formulate this problem into two sequential steps: *cheap talk discovery* and *cheap talk utilization*, and justify how this breakdown offers an opportunity to tackle such a difficult exploration problem. We develop a custom environment to benchmark MARL algorithms in tackling this problem. Then, we propose a novel approach based on information theory, combining with some of the latest advancements in MARL, to tackle the task. Through quantitative analysis of variants of our method in our custom environments, we offer insights into how our novel problem formulation and proposed approach can potentially solve this formidable problem in its entirety while existing work cannot, by first discovering cheap talk channels and subsequently learning how to use them. We further highlight future steps to strengthen our results and the limitations of this investigation.

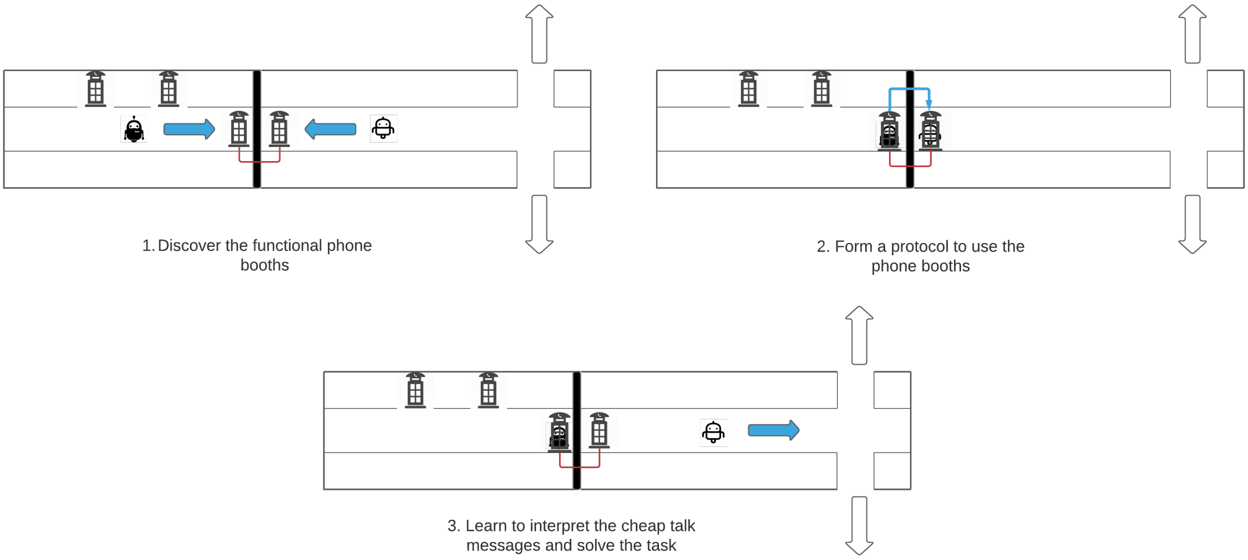
1.1 Motivation

As discussed above, effective communication among agents is the key to success for a multi-agent system. Particularly, in many cases, the capability of sharing information among agents is essential to completing a task when agents have different levels of accessibility to different information. Being able to exchange information can also potentially bring about better policies when agents can acquire information from other agents without having to acquire all information by themselves. The importance of effective communication is even more prominent in the partially observably setting where no agents have full visibility of the environment. Here, we mainly focus on this setting as it has a greater resemblance to real-life problems. By effectively communicating with each other through sending messages, agents would be able to interact, coordinate and negotiate successfully.

Prior work can only exploit cheap talk channels if their existence and capacity is known apriori, e.g. as part of domain-specific privileged information. In addition, prior work largely assumes that these cheap talk channels are persistent, i.e. that agents can access cheap talk channels at every time step (or at their respective turn). To the best of our knowledge, this work is the first to do away with both these assumptions through proposing and evaluating a novel method and framework for cheap talk discovery and utilization during the learning process. In more specific terms, without this assumption, these channels can only be used within a particular set of states. One can think of these states as "phone booths" in an environment that allows communication. Agents have to first find and enter a phone booth before they can "call" agents that are situated in other phone booths. Once in a connected phone booth, agents can then call other agents to send their messages. "Calls" or messages can only be received if an agent is in a connected phone booth. These "Calls" or messages can be unidirectional, bidirectional or even multi-directional. Here, we assume the unidirectional case in which a "call" or a message has a sender and a receiver, but our formulation and methods can easily scale to the other cases too.

We believe this is a natural subsequent question to ask in building effective and communicative agents after we have agents that can learn how to communicate reasonably well. One can think of real-life scenarios in which we would expect our agents to know where to communicate effectively. For instance, a fleet of drones going on a search and rescue mission in a remote area would have varying signal strength for communication within the area. They would need to know where to position themselves to communicate messages successfully. Another example could be a group of robots within an assembly plant of giant machinery. For maximal efficiency, they need to get regular updates on each other's progress and potentially exchange important parts to complete the task. This would require them to meet in a particular location to communicate, which is ideally something to be learned by the agent, as these locations should change flexibly for better efficiency based on factors like their respective current locations.

Besides the potential applications in solving this problem, it is also a difficult problem that covers some of the challenging open questions in MARL, which are the credit assignment problem and the combinatorial nature of MARL [25]. To illustrate the difficulty, here we use our custom environment where we conduct our study as



1. Discover the functional phone booths

2. Form a protocol to use the phone booths

3. Learn to interpret the cheap talk messages and solve the task

Figure 1.1: Visual illustration of the three learning stages based on the phone booth maze environment

an example, the phone booth maze. In this environment, we have two agents, a sender and a receiver, which are placed into two separate rooms. The goal is for the receiver to escape from the correct exits out of the two possible exits. Only the sender knows which one is the correct exit and the only way for the sender to communicate this information to the receiver is to have both of them going to their respective functional phone booths. This essentially leads to three required learning paths to solve the task. Firstly, they have to learn to get to the booth. Then, the sender has to learn to form a protocol by sending a particular message for a particular correct exit. Finally, there is a post-discovery exploration stage where the receiver has to learn to interpret the sender's protocol by trying out the exits. This makes credit assignment, which refers to the correct attribution of consequences (rewards) to actions, particularly difficult. Because communicative actions do not have an immediate effect on rewards or other agents' observations, meaning it takes a lot more steps for any consequence to manifest. Thus, it would be hard to reach the optimal policy given the difficulty to assign long-term credits. The difficulty is further compounded if rewards are sparse and delayed, which is often the case in many MARL environments. Figure 1.1 provides a visual illustration of these three needed learning stages in the phone booth maze environment. Regarding the combinatorial nature of MARL, it refers to the exponential increase in the size of solution space as the number of agents increases, as the joint action space is often considered. This leads to great difficulty in learning the correct solution, which is further exacerbated when communicative actions are also considered. These properties make the problem a very challenging and well-motivated one to solve.

1.2 Contributions

The contributions of this thesis are

(i) A clear formulation of the cheap talk discovery and utilization problem

(ii) A custom and configurable environment to benchmark MARL algorithms in solving cheap talk discovery and utilization

(iii) An approach to solve the cheap talk discovery and utilization problem based on information theory and recent advances in MARL including Off-Belief Learning (OBL) [21] and Differentiable Inter-Agent Learning [11].

(iv) A systematic and quantitative evaluation of the proposed approach and other MARL baselines in solving the problem

(v) A variety of ablation studies to understand various aspects of the problem

1.3 Project Scope

Given that the project addresses a problem that has not been well investigated before, a significant portion of the thesis work went into designing and implementing a proper environment so that any methods can be evaluated properly and fairly. More importantly, as the problem is essentially an encapsulation of a few hard problems, existing methods are found to be surprisingly ineffective in solving our custom GridWorld environment. Hence, we believe a tractable environment with a clean problem description would allow us to isolate the causes of issues and conduct extensive analysis more efficiently.

1.4 Thesis Overview

The goal of this project is to formulate the cheap talk discovery and utilization problem, provide a configurable environment to evaluate different algorithms, and propose a promising approach to solve this problem, as well as offer a detailed and systematic analysis of our results with clear directions for future extensions. The structure is as follows:

- *Chapter 2 - Background* lays the theoretical foundations employed in this thesis. It starts with covering the theoretical background of RL and MARL. Then, it goes over relevant pieces of information theory that are used in our proposed method. The level of detail in this chapter should allow readers to assess and understand our contribution in subsequent chapters.
- *Chapter 3 - Related Work* presents previous work that has attempted to solve problems similar to our problem or proposed approaches based on ideas that are similar to our proposed approach.
- *Chapter 4 - Methodology* defines our novel problem formulation - *the cheap talk discovery and utilization problem* and describes our novel proposed approach in tackling the problem. It provides all mathematical derivations, algorithmic details, and justifications of our approach in order to train such a model.
- *Chapter 5 - Experiments and Results* outlines the experimental setup and implementation details of the baselines we use and our proposed approach. It further provides an extensive quantitative evaluation of our results in solving the cheap talk discovery and utilization problem.
- *Chapter 6 - Conclusion* provides a summary of the contributions and results of this thesis, as well as future directions that are made possible by our work. It ends with summaries and a personal evaluation of the journey in completing this thesis including its challenges.

Chapter 2

Background

This section assumes the basic knowledge of deep neural networks and recurrent neural networks which are used in this work. Please refer to the appendix A.1 for an extensive discussion on these components.

2.1 Markov Decision Process

Markov Decision Process (MDP) is a formalism for decision-making processes. A MDP models a decision-making agent taking actions in its environment over a sequence of discrete time steps $t = 1, 2, 3, ...$ [66]. It has the Markov Property which holds if:

$$\mathbb{P}(s_{t+1}|s_t, a_t) = \mathbb{P}(s_{t+1}|s_t, a_t, ..., s_0, a_0) \tag{2.1}$$

$$\mathbb{P}(r_t|s_t, a_t) = \mathbb{P}(r_t|s_t, a_t, ..., s_0, a_0) \tag{2.2}$$

This means, at each time step t_i, the transition from s_t to s_{t+1} can be conditioned only on the current state, without the entire history of previous states. In other words, s_t serves a sufficient summary of the past. Based on [12], a MDP can then be defined as:

Definition 2.1. A MDP is a 5-tuple (S, A, T, r, γ) where:

S: state space

A: action space

T: $S \times A \times S \rightarrow [0, 1]$, is the transition function of the environment

r: $S \times A \leftarrow \mathbb{R}$, a reward function from the environment

γ: discount rate

At each time step t_i, interaction between the agent and its environment is characterized by a MDP. Given the current state s_i, the agent takes a action $a \in A$. Then, the environment emits a numerical reward $r(s_i, a)$ to the agent and transitions to a new state s_{i+1} based on the transition function $T(s_i, a, s_{i+1})$.

With the agent interacting with the environment, we get a sequence of states, actions, and rewards. The sequence is commonly referred as a trajectory τ in the form of:

$$\tau = s_i, a_i, r_{i+1}, s_{i+1}, a_{i+1}, r_{i+2}, ...s_t \tag{2.3}$$

where $s_i \in S, a_i \in A$ and s_t is the terminal state. An environment can either be episodic or continuing [66]. In the latter case, a terminal state does not exist, so the environment never terminates. In this work, we only consider environments that are episodic in which a trajectory naturally terminates as an episode. Most RL benchmarks are framed as episodic problems like the game of Go. Nonetheless, we expect our proposed approaches to work naturally with continuing environments too, as they do not depend on having terminal states.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning that focuses on learning to complete a task through interactions, or learning what to do by mapping situations to actions by maximizing a numerical reward signal [66]. In other words, RL algorithms try to learn how to make decisions, with a specific focus on solving problems that can be modelled as MDPs. It is inspired by the field of psychology when observing how animals learn through trial and error [52]. In this case, unlike supervised learning, the model does not have access to the "action labels". Precisely, the reinforcement learning setting assumes an agent which interacts with an environment over a period of time by taking actions. The goal is to maximize a scalar reward signal provided by the environment by learning what to do. To do so, the agent has to learn to map what it observes in the environment - *state* to an *action* that would maximize the reward signal. The mapping from state to action is often known as the agent's behavior policy [66]. Figure 2.1 illustrates the high-level learning loop of reinforcement learning.

This different mode of learning poses quite different, and sometimes unique challenges. One classic challenge is the exploitation/exploration dilemma. To discover actions

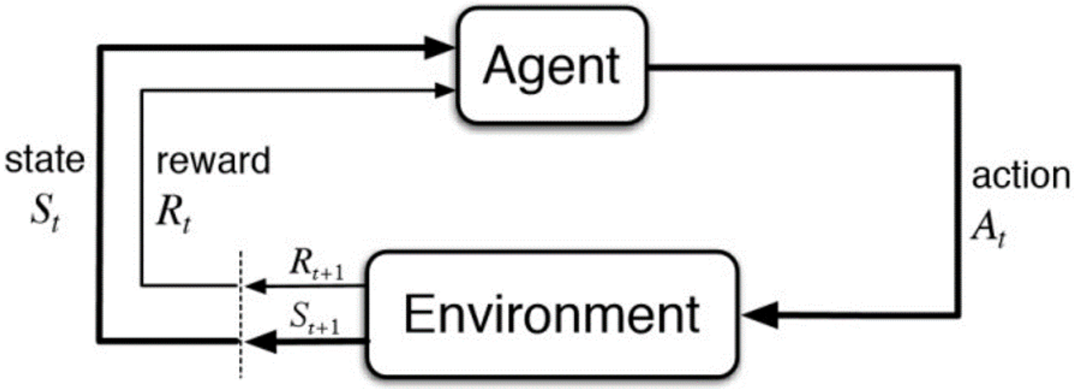


Figure 2.1: Learning loop of reinforcement learning on a high-level, taken from [66]

that lead to positive rewards, the agent has to explore by taking new actions in new scenarios. At the same time, to get more rewards, the agent has to exploit by taking actions that were shown to be effective in getting rewards. The dilemma is that neither can be pursued exclusively without failing the task [66]. There are also challenges regarding the reward signal itself. The rewards provided by the environment can be sparse and delayed, leading to the credit assignment problem in which the agent has to learn and assign credits to actions that bring rewards [66]. In practice, the design of reward functions is also difficult as it is not simple to engineer reward functions that can lead to desired behavior without misalignment, especially in more complicated tasks. Last but not least, issues like catastrophic forgetting and instability arise when RL algorithms are used with function approximators like deep neural networks, given that most foundational RL algorithms were designed to work in an online and non-independent-and-identically-distributed (non-iid) manner [6].

2.2.1 Fundamentals

To begin with, a policy π determines how an agent selects an action [12]. It can either be deterministic or stochastic, which we consider the latter. Formally, a policy is defined as:

Definition 2.2. A policy π can be either deterministic or stochastic where:

> A deterministic policy $\pi : S \to A$: a function that maps each state to a single action [66]
>
> A stochastic policy $\pi : S \times A \to [0, 1]$: a function that maps a state s_i and an action a_i to the probability of a_i being taken given the current state is s_i [66]

Policies can also be classified as stationary or nonstationary [12]. If a policy is nonstationary, even if a pair of state-action pairs are the same, the agent might not take the same action as the policy is conditioned on the time step. In this work, we consider stochastic and stationary policies exclusively.

Maximizing expected (discounted) returns is the goal of RL agents, so it plays a key role when designing RL algorithms. Let $M = (S, A, T, r, \gamma)$ be a MDP. The *discounted return* G_t at time step t is expressed as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.4}$$

where it corresponds to the cumulative reward attained from time step t onward. Based on equation 2.4, the expected return is expressed as follows:

Definition 2.3. The *expected return* for a policy π at time step t is defined as

$$J_t^{\pi} = \mathbb{E}[G_t] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, \pi\right] \tag{2.5}$$

where the expectation is taken over the distribution of trajectories τ generated by the policy π and the transition function T [12]. Being the discounted factor between 0 and 1, γ discounts future rewards, determining the extent that immediate rewards are preferred over future rewards. We note that having $\gamma < 1$ provides guarantees for an infinite sequence to converge [12]. But given that we only focus on the episodic setting, discounting is not strictly necessary for convergence and can be set to 1. Consequently, the optimal policy π^* is the policy that maximizes the expected return [66], in the form of:

$$\pi^* = \arg\max_{\pi} J^{\pi} \tag{2.6}$$

A value function is one of the most important components in many RL algorithms to learn based on the expected return. It gives you the value of a state or a state-action pair, which is the expected return the agent would get starting from that state or state-action pair and act according to a particular policy thereafter [66]. The value function and action-value function corresponds to functions that give the value of a state and state-action pair respectively. They are defined as follows:

Definition 2.4. The value function of a MDP is a function $V^{\pi} : S \rightarrow \mathbb{R}$ that gives the expected return given the current state s with all the future actions picked according to the policy π:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi\right] \tag{2.7}$$

Definition 2.5. The action-value function of a MDP is a function $Q^{\pi} : S \times A \rightarrow \mathbb{R}$ that gives the expected return given the current state s and the action selected a with all the future actions picked according to the policy π:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right] \tag{2.8}$$

The optimal value function and action-value function can be defined similarly except the subsequent actions are selected according to the optimal policy π^*:

Definition 2.6. The value function of a MDP is a function $V^{\pi} : S \rightarrow \mathbb{R}$ that gives the expected return given the current state s with all the future actions picked according to the policy π:

$$V^{*}(s) = \max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi\right] \tag{2.9}$$

Definition 2.7. The action-value function of a MDP is a function $Q^{\pi} : S \times A \rightarrow \mathbb{R}$ that gives the expected return given the current state s and the action selected a with all the future actions picked according to the policy π:

$$Q^{*}(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right] \tag{2.10}$$

Another important value function is the advantage function which outputs the difference between the expected return of taking a as the first action or when the action is selected from π. It is defined as:

Definition 2.8. The advantage function is a function $A^{\pi} : S \times A \rightarrow \mathbb{R}$:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \tag{2.11}$$

2.2.2 Tabular Reinforcement Learning

This section gives an overview of one of the most widely used RL algorithms to learn an action-value function, which is the base algorithm used in this work. In the tabular setting, action-value functions are represented as lookup tables. For the sake of conciseness, RL algorithms like dynamic programming (DP), Monte Carlo (MC) methods, and policy gradients methods will not be covered. However, we note that our conceptual contributions like our novel problem formulation and proposed methods based on information theory are algorithm-independent and should generalize across any algorithms used.

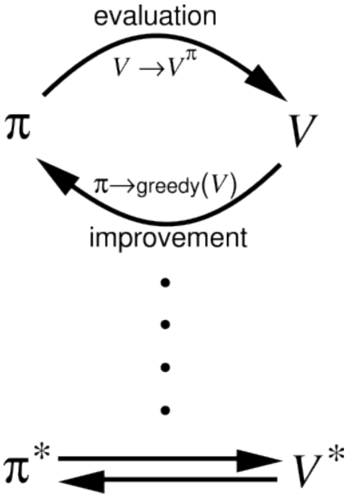


Figure 2.2: Generalized Policy Iteration, taken from [66]

Most, if not all, RL algorithms fall under the paradigm of generalized policy iteration (GPI), which consists of two simultaneous and interacting processes, namely, policy evaluation and policy improvement [66]. Policy evaluation computes the value function of the current policy while policy improvement updates the policy to be greedy with respect to the current value function to improve it. Figure 2.2 depicts the two interacting processes in GPI.

Q-learning is a control algorithm based on Temporal Difference (TD) learning. TD methods learn based on a technique called bootstrapping, which computes estimates based on previous estimates [65]. In policy evaluation, TD methods update by sampling one-step rewards and bootstrapping the current value function estimate instead of using the complete episode returns used in MC methods. To learn a value function $V(s_t)$ using TD learning, the update can be expressed as:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.12}$$

where:

α: learning rate

$r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$: TD error [66]

TD error is the difference between the current estimate and the updated estimate using the one-step reward from the environment. For Q-learning, its update step is written as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.13}$$

Q-learning is an off-policy learning algorithm as the maximization operation used when computing the new estimate uses an action that is not necessarily from the behavior policy, estimating the optimal state-action value function directly. It is also worth noting that the approach fits with the GPI paradigm with the policy improvement step happening implicitly in maximization operation.

TD approaches have several advantages when compared with methods like DP and MC approaches. It does not require models of the environment, unlike DP. It also converges faster than MC methods because updates are performed in an online and fully-incremental fashion. Please see [66] for more detailed comparisons. However, approaches like Q-learning do suffer from the maximization bias as the maximization operation tend to bias towards overestimated values [66]. With reference to [66], pseudocode for Q-learning in the tabular setting is provided below.

Algorithm 1: Q-learning pseudocode

```
Hyperparameters: $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a) \forall s \in S, a \in A$, with terminal state $s_{terminal}$ having $Q(s_{terminal}, \cdot) = 0$
for $e$ = 1, Max_Episode do
    Initialize $s$
    while $s \neq s_{terminal}$ do
        select $a$ from $s$ based on policy derived from $Q$, e.g. $\epsilon$-greedy policy
        take action $a$ to observe reward $r$ and next state $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
        $s = s'$
```

Note that ϵ-greedy policy here is a common approach to deal with the exploration-exploitation dilemma in RL. Precisely, at each time step, there is a probability ϵ that the agent would take a random action. Over the years, many different exploration policies have been proposed to better handle this dilemma, which is beyond the scope of this work [2, 35, 68].

2.2.3 Reinforcement Learning with Function Approximation

Tabular RL solutions like tabular Q-learning discussed above can only handle MDPs with finite and discrete state and action spaces. However, in most complex problems of interest, the state space and action space are often combinatorially huge or even continuous. Thus, tabular methods would face the issue of impossible memory requirements and the impossibility of visiting every state sufficiently. Generalization is needed across states that are similar to each other [66].

To obtain such generalization and handle problems with combinatorially huge problem spaces, it has become common to use a function approximator to estimate the action-value function - $Q(s, a; \theta) \approx Q^*(s, a)$. In the case of deep reinforcement learning, deep neural networks are used as non-linear function approximators to the value functions with θ being the weights in the neural networks.

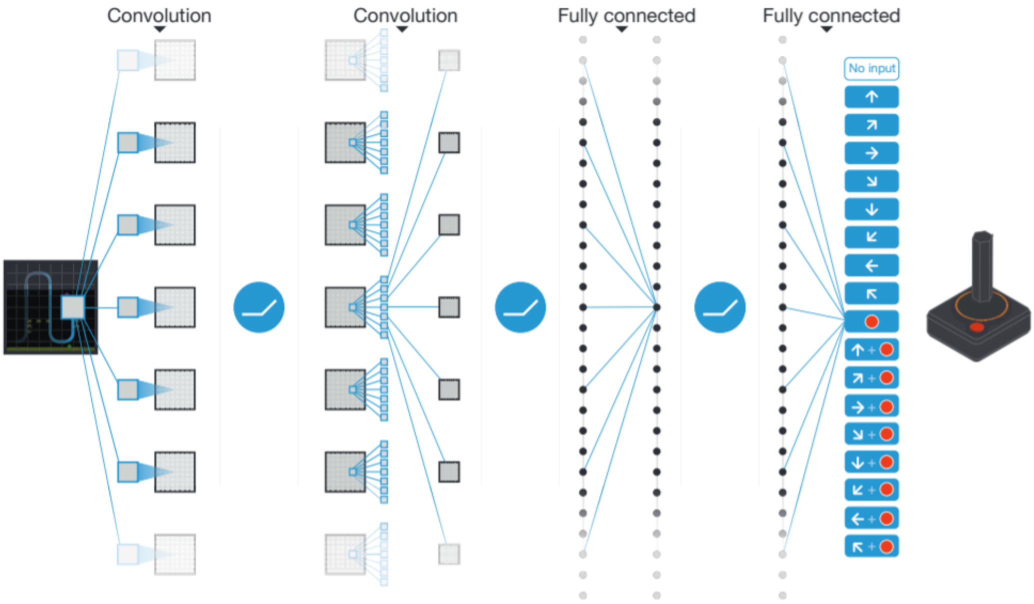


Figure 2.3: Deep Q-network architecture, taken from [38]

Deep Q-network (DQN) is a seminal work by [37] that popularises the use of neural networks as function approximators by showing their power in training powerful RL agents to perform human tasks like playing video games. The Q-network's last layer outputs the action-value for each action with a state as input. See figure 2.3 for a visual depiction. The Q-network is learned by minimizing a sequence of loss functions $L_i(\theta_i)$ that is different for each iteration i, which they are expressed as:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot),s'\sim\xi}\left[((r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})) - Q(s, a; \theta))^2\right] \tag{2.14}$$

where $\rho(s,a)$ is the behaviour distribution, a probability distribution over sequences of states s and actions a and ξ is the distribution for s' based on the emulator or the environment. The weights in the network can then be updated using SGD by differentiating the loss function with respect to the weights, which would give us:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot),s'\sim\xi}\left[(y_i - Q(s,a;\theta))\nabla_{\theta_i} Q(s,a;\theta_i)\right] \tag{2.15}$$

where $y_i = (r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}))$.

Two additional innovations were introduced in this work that are significantly responsible for the model's success, namely, experience replay and a target network. Experience replay is a memory buffer that stores an agent's experiences at each time step as a transition tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ [33] into a dataset $D = e_1, e_2 ... e_N$. In each iteration, a minibatch of experience is sampled randomly from the buffer to perform a deep Q-learning update. By doing so, it alleviates the issues of correlated data and nonstationary distributions, smoothing the training distribution over many past behaviours [37]. For the target network, it is a separate neural network to the Q-network that copies the Q-network's weight periodically [38]. It was experimentally shown to offer more stability during training by reducing correlations to the target. Algorithm 2 shows the pseudocode for Deep Q-learning, in courtesy of [38].

Algorithm 2: Deep Q-learning pseudocode

```
Initialize replay memory $D$ with capacity $N$
Initialize action value function Q with random weights $\theta$
Initialize target action value function $\hat{Q}$ with random weights $\theta^- = \theta$
for $e$ = 1, Max_Episode do
    Initialize $s$
    while $s \neq s_{terminal}$ do
        Select $a$ from $s$ based on policy derived from $Q$, e.g. $\epsilon$-greedy policy
        Take action $a$ to observe reward $r$ and next state $s'$
        Store the transition into $D$
        Sample random minimatch of transitions from $D$
        Set the targets $y_i$ for the sampled data:
            $\begin{cases} r_i \text{ if the episode terminates at this time step} \\ r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \theta_{i-1}) \text{ otherwise} \end{cases}$
        Perform a gradient descent step on $(y_i - Q(s,a;\theta_{i-1}))^2$ with respect to the network weights $\theta$
        Every C steps, set $\hat{Q} = Q$
```

Since the seminal work of DQN, many new algorithms and improvements have been proposed. Two relevant ones are prioritized experience replay and Dueling DQN. Prioritized experience replay is an intuitive modification to the experience replay, which prioritizes transitions with greater learning error so they are sampled more frequently [54]. It outperforms DQN by a significant margin. For dueling DQN, it proposes a dueling architecture that explicitly separates the representation of state values and state-action values into two separate heads of neural networks with a shared backbone [71]. The intuition is to allow the agent to learn which states are useful without directly learning the effect of each action for each state, which is informative in places where actions do not influence the environment in any relevant way. The action-value function is computed as follows:

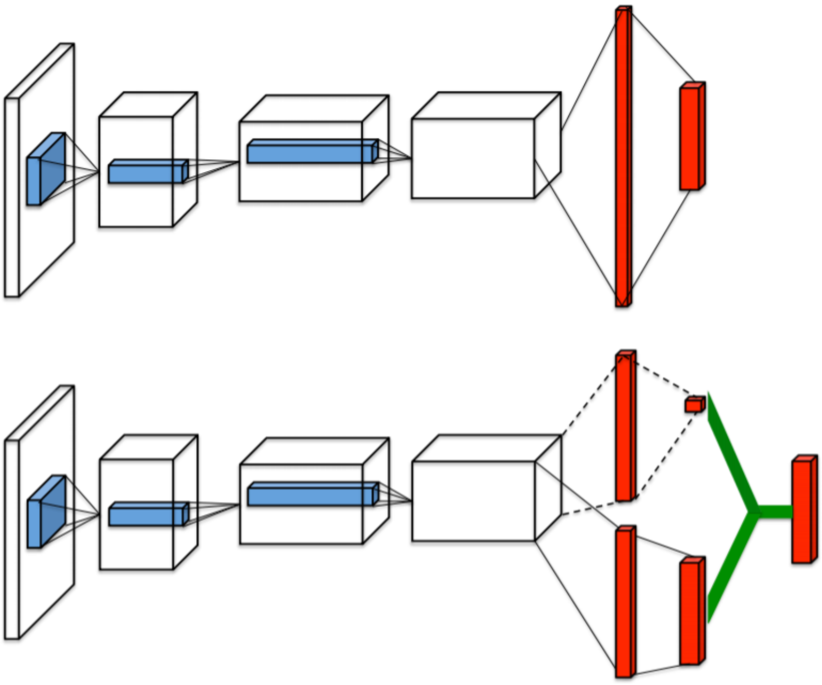


Figure 2.4: Architectural differences between DQN and Dueling DQN, taken from [71]

$$Q(s, a) = V(s) + A(s, a) \tag{2.16}$$

The architecture showed superior performance in the Atari-2600 benchmarks. Figure 2.4 shows the differences in architecture between DQN and dueling DQN. These are only the tip of the iceberg regarding advances in Deep RL. For instance, deep policy gradient approaches like Deep Deterministic Policy Gradient [32] and Proximal Policy

Optimization [56] have shown state-of-the-art performances in environments with continuous action space like robotics.

2.3 Partially Observable Markov Decision Process

In the case of MDPs, we assume an environment's state space is entirely observable which is a stringent assumption if we consider real-life problems. Considering the world we live in as an environment, there are way too many variables for us to have access to a complete state, let alone processing it. Therefore, our agents ought to be able to perform when the environment is only partially observable, leading to the notion of the observation space $O \in S$ [66]. Such partial observability is captured by the formalism of Partially Observable MDP, which we will use in this work. Under this formalism, our trajectories will no longer have states which are replaced with observations - $o_1, a_1, r_1, o_2, a_2, r_2, ...o_t, a_t, r_t$, which is sometimes called a history. Formally, a partially observable MDP (POMDP) [24] is defined as follows:

Definition 2.9. A POMDP is a 7-tuple $(S, A, T, R, \Omega, O, \gamma)$ where:

S: state space

A: action space

T: $S \times A \times S \rightarrow [0, 1]$, is the transition function of the environment

R: $S \times A \times S \rightarrow \mathbb{R}$, a reward function from the environment

Ω: observation space

O: $S \times \Omega \rightarrow [0, 1]$, is the function that gives the conditional probabilities of an observation

γ: discount rate

At each time step t, given a state $s_t \in S$ from the environment, the agent receives the observation $\omega_t \in \Omega$ with probability $O(s_t, \omega_t)$. Then, the agent takes an action $a_t \in A$ with the environment transitioning to s_{t+1} based on the transition function $T(s_t, a_t, s_{t+1})$ emitting a reward r_t from the reward function $R(s_t, a_t, s_{t+1})$. Note that as the observation does not have the full information of the state, it might not be rich enough to capture all the crucial information or the system dynamics. In other words, the formalism is likely to be no longer Markovian, which the observation no longer captures the history sufficiently. In that case, it is common to use a history of past observations to better estimate the system dynamics or the state of the environment [12].

2.4 Reinforcement Learning in Handling Partial Observability

As this work bases our investigations on deep RL algorithms in environments with partial observability, this section briefly covers some common techniques in deep RL to handle partial observability. One key technique to handle partial observability in deep RL is the introduction of recurrent components in the architecture. This was first utilized in [17], which Deep Recurrent Q-Network was proposed (DRQN). Unlike DQN, The architecture has an LSTM network as the penultimate layer with a linear final layer to output action values. By adding recurrency, it is able to learn to retain and summarize the past, offering more information to capture the underlying system state. In other words, it is able to better estimate the action-values from the sequence of observations, closing the gap between $Q(o, a|\theta)$ and $Q(s, a|\theta)$. Figure 2.5 shows the architecture of DRQN. Another notable difference in this line of methods is the use of the experience replay buffer. As the sequential property of an episode has to be preserved for the recurrent component to learn properly, the replay buffer stores data samples on the granularity of episodes instead of transitions. In each update iteration, a minibatch of episodes is sampled to perform learning.

In this work, we use a variant called Recurrent Replay Distributed DQN (R2D2), one of the more recent works in this line of research [26]. It also uses an LSTM layer. But instead of storing regular transition tuples or episodes, it stores fixed-length sequences of s, a, r in the replay with adjacent sequences overlapping by m time steps, which is a hyperparameter. Note that we do not use the distributed version since it is not needed in our work. The work further explores ways to initialize the starting hidden state of RNN as the typical zero start state strategy limits its ability to fully learn long temporal correlations [26]. Therefore, it proposes two strategies:

1. **Stored state**: stores the recurrent state in the replay buffer to initialize the network at training time. It alleviates the issue with zero start state strategy but may suffer from representational drift. This would lead to staleness in the recurrent state as the stored ones could be generated by a quite old version of the network.

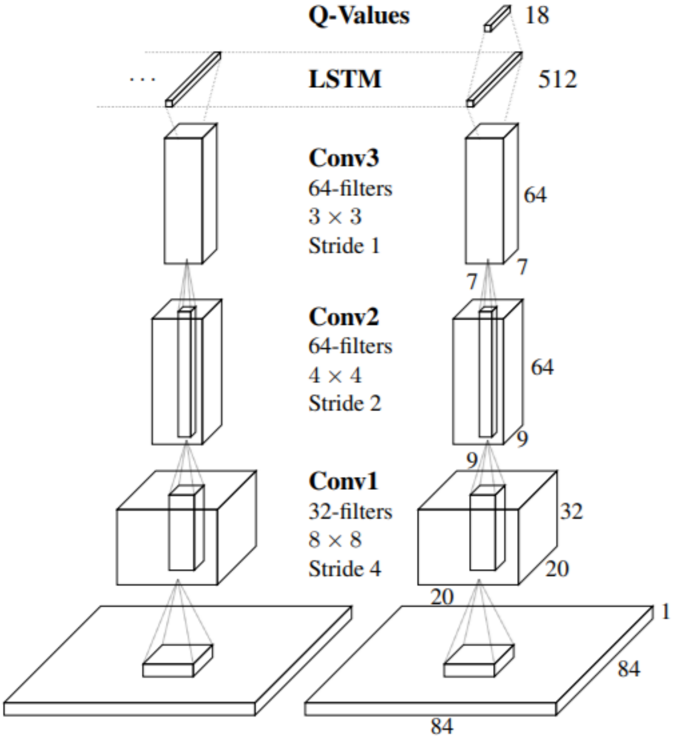


Figure 2.5: The architecture of DRQN, taken from [17]

2. **Burn-in**: allows the network a 'burn-in' period by using a portion of the replay sequence to unroll the network and get a start state. The update is only performed in the remaining part of the sequence. [25] hypothesizes that this allows the network to sometimes recover from a poor start state before needing to produce accurate outputs.

By examining the recurrent states in the replay buffer, they demonstrated how the mixture of the stored state and burn-in strategies mitigates the issue of staleness and improves learning of long temporal correlations.

2.5 Decentralized Partially Observable Markov Decision Process

Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is a popular framework in modeling multi-agent scenarios, specifically for the decision-makings of a team of cooperative agents [45]. In this setting, each agent can only learn to make decisions based on its observation which does not contain the information of the entire environment. Together with multiple agents acting in the environment, agents often have to learn to reason about other agents and learn how to coordinate or communicate by observing each other in order to achieve decent performance. This effectively means that the other agents' policies become part of an agent's observation function in a Dec-POMDP setting. The learning task is then to find policies that both produce high reward actions and communicate information effectively to the other agents. Figure 2.6 depicts how a Dec-POMDP works with two agents.

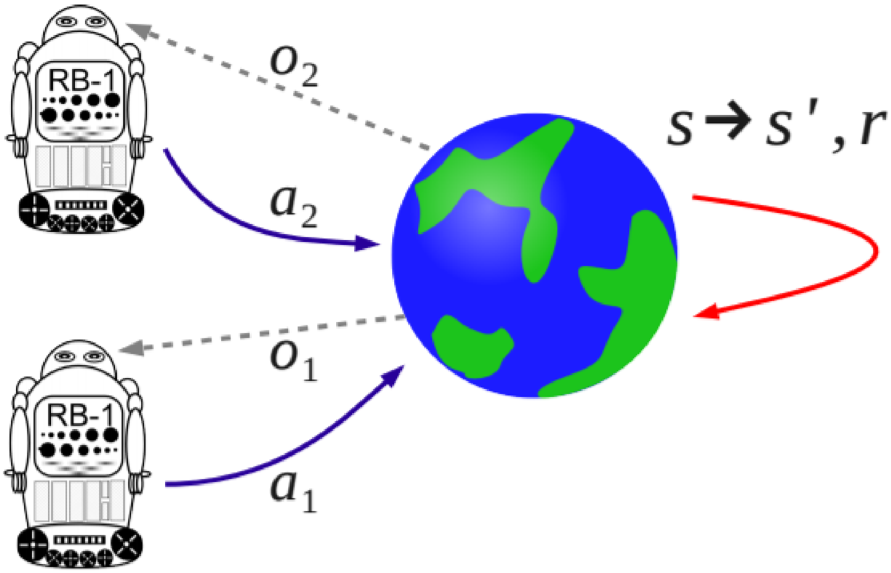


Figure 2.6: Schematic representation of a Dec-POMDP, taken from [45]. At every time step, each agent takes an action based on its own observation

Formally, assume a Dec-POMDP with N agents, with each assigned an index $i \in N$, it can be defined as follows:

Definition 2.10. A Dec-POMDP is a 9-tuple $(D, S, \mathbb{A}, T, R, \Omega, \mathbb{O}, O, Z, \gamma)$ where:

D: set of N agents - [1, ... N]

S: state space

$\mathbb{A}$: joint state space, $A_1 \times A_2 \times ...A_N$

T: $S \times A \times S \rightarrow [0, 1]$, is the transition function of the environment

R: $S \times A \times S \leftarrow \mathbb{R}$, a joint reward function from the environment

Ω: observation space

$\mathbb{O}$: joint observation space, $\Omega_1 \times \Omega_2 \times ...\Omega_N$

O_i: $S \times \Omega_i \rightarrow [0, 1]$, is the function that gives the conditional probabilities of an observation

Z: $S \times D \rightarrow \Omega$, is the observation function. If it is stochastic, it produces an observation based on the observation probability function

γ: discount rate

The environment trajectory and the action-observation history (AOH) of an agent i are denoted as $\tau_t = s_0, \mathbf{a_0}, ...s_t, \mathbf{a_t}$ and $\tau_t^i = o_0^i, a_0^i, ...o_t^i, a_t^i$ respectively. Each agent learns a policy $\pi^i(\tau^i)$ conditioned on its AOH, with the goal to learn a joint policy $\pi = \pi^i$ that maximizes the total expected return $J_\pi = \mathbb{E}_{\tau \;\; P(\tau|\pi)} R(\tau)$, where $R(\tau)$ is defined as:

$$R(\tau) = \sum_{t'>=t} \gamma^{t'-t} r(s_t, \mathbf{a_t}) \tag{2.17}$$

where $\mathbf{a_t}$ corresponds to the joint action of the agents at time step t.

2.6 Multi-Agent Reinforcement Learning

Multi-agent Reinforcement Learning (MARL) refers to a subfield of RL developing methods to handle situations when there are more than one agent acting in the environment. This includes problems that can be modelled as Dec-POMDPs. Similar to RL, each agent in MARL also has the goal of maximizing a scalar reward signal. Depending on the environment, it can be shared among agents or each has its own reward function. The environment can also be cooperative, adversarial, or mixed. However, the reward that each agent receives depends on the joint action of all the agents. The goal of a MARL algorithm, in general, is to learn control policies π_i for each agent i that could maximize reward. This work focuses on the cooperative setting where agents share the same reward function.

There are 4 major areas of focus in the field of MARL, namely, analysis on emergent behaviors, agents modeling agents, learning cooperation, and learning communication [18]. Firstly, analysis on emergent behaviors looks at how different behaviors, including cooperative and competitive behaviors, emerge from learning. Many of the works in the area propose methods to induce these behaviors, independently of the learning algorithm and architecture, including several using various forms of reward shaping. Secondly, for agents modeling agents, it is about the ability of agents reasoning about other agents, making predictions about their behaviors, By forming beliefs about other agents, an agent takes into account of other agents' influence on the environment to make more informed decisions. Thirdly, to learn how to cooperate, this domain looks at architectural and algorithmic approaches to induce cooperation without communication. Lastly, to learn how to communicate, agents try to maximize rewards by sharing information through communication channels in the environment, which can range from signaling actions to sending discrete or continuous messages through a communication channel. A seminal work would be Differentiable Inter-Agent Learning (DIAL) [11] which facilitates communication through means of gradients passing across agents.

This section provides a non-exhaustive overview of some common MARL algorithms used to handle Dec-POMDP problems. Here, we focus on settings with centralised training but decentralized execution (CTDE). Under this setting, communication among agents is not restricted during learning, performed by a centralized algorithm. During execution, the agents' communication becomes limited only via the limited-bandwidth channels [11]. We also use some decentralized learning methods as baselines. Only the algorithms directly used in this work will be elaborated on in detail.

2.6.1 Independent Q-Learning

One commonly used MARL method, a natural extension to a classic RL method, is independent Q-learning (IQL) [67]. In this case, each agent learns its own action-value function from its own states and actions. If the environment is partially observable, the function is conditioned on its action-observation history instead. When applied to deep RL, this can be done by having each agent learning using DQN with a recurrent component to account for partial observability. It has been shown to perform well empirically in many environments [72, 34]. This will be used as a baseline for comparison in this work

IQL has the appealing property of circumventing the combinatorial nature of the MARL problem mentioned previously. This is because each agent learns its value function independently and the model does not have to learn from a joint state space and/or a joint action space. Therefore, it is quite scalable. Yet, given such independence in information among agents, the environment could easily appear to be non-stationary to the agents as other agents are also learning and taking actions. This leads to instability in convergence and the nonexistence of any convergence guarantees.

2.6.2 Off-Belief Learning

OBL is a recently proposed method in handling the issue of interpreting the actions of other agents and accounting for how they will interpret our actions in turn [21]. The approach sets to learn optimal grounded policies, which do not interpret the actions of other agents and assumes other agents do the same to their actions. This is often desirable as making wrongful assumptions can often be a source of coordination failure. Thus, learning or starting with grounded policies is important to avoid reasoning about partners' actions.

To learn a grounded policy, the *grounded belief* B_G has to be defined as a modified belief that conditions only on the observations but not any other agents' actions:

$$B_G(\tau|\tau^i) = \frac{P(\tau)\Pi_t P(o_t^i|\tau)}{\sum_{\tau'} P(\tau')\Pi_t P(o_t^i|\tau')}. \tag{2.18}$$

With the grounded belief, an *optimal grounded policy* π_G can be formulated to be any policy that plays an action that maximizes the expected reward at each AOH τ^i, with the state distribution at τ^i drawn from the grounded belief B_G and π_G being played thereafter. The goal of OBL is to learn this π_G.

Here, we assume all agents to be playing a common knowledge policy π_0 up to τ^i. Then, an agent's belief distribution B, conditioned on their AOH can be computed as:

$$B_{\pi_0}(\tau|\tau^i) = P(\tau|\tau^i, \pi_0) \tag{2.19}$$

The belief distribution fully describes the effect of the history on the current state. We assume each agent to play a 'counterfactual' policy π_0 (e.g. uniformly random policy) to reach τ^i and π_1 thereafter, which the latter corresponds to the optimal policy under the same assumptions. We denote the return for this to be $V^{\pi_0 \to \pi_1}(\tau^i)$, which precisely means the return of sampling τ from $B_{\pi_0}(\tau^i)$ with all players playing π_1 from this trajectory. The counterfactual value function and state-action value function are defined as follows:

$$V^{\pi_0 \to \pi_1}(\tau^i) = \mathop{\mathbb{E}}_{\tau \sim B_{\pi_0}(\tau^i)} [V^{\pi_1}(\tau)], \tag{2.20}$$

$$Q^{\pi_0 \to \pi_1}(a|\tau^i) = \sum_{\tau_t, \tau_{t+1}} B_{\pi_0}(\tau_t|\tau_t^i)[R(s_t, a) + \mathrm{T}(\tau_{t+1}|\tau_t) V^{\pi_1}(\tau_{t+1})]. \tag{2.21}$$

$Q^{\pi_0 \to \pi_1}$ corresponds to the return from playing a at τ^i with the assumption of π_0 was played to reach τ^i and π_1 is played in future steps.

The OBL operator is defined as follows to map π_0 to π_1:

$$\pi_1(a|\tau^i) = \frac{\exp(Q^{\pi_0\to\pi_1}(a|\tau^i)/T}{\sum_{a'}\exp(Q^{\pi_0\to\pi_1}(a'|\tau^i)/T)} \tag{2.22}$$

where T is a temperature hyperparameter. There are theoretical details to provide support in how OBL can lead to an optimal grounded policy. Please see [21] for more details. It is also worth noting that, although learning an optimal grounded policy is crucial, some settings do benefit from counterfactual reasoning and convention formation. Fortunately, the repeated application of OBL leads to direct control of the number of counterfactual reasoning loops to be carried out by an agent.

To compute an OBL policy using value iteration methods, the Bellman equation for $Q^{\pi_0\to\pi_1}(\tau^i)$ for each agent i can be put as follows:

$$\begin{aligned} Q^{\pi_0\to\pi_1}(a_t|\tau_t^i) = \mathop{\mathbb{E}}_{\tau_t\sim B_{\pi_0}(\tau_t^i),\tau_{t+k}\sim(\mathrm{T},\pi_1)} \Bigg[\sum_{t'=t}^{t+k-1} R(\tau_{t'}, a_{t'}) + \\ \sum_{t+k} \pi_1(a_{t+k}|\tau_{t+k}^i) Q^{\pi_0\to\pi_1}(a_{t+k}|\tau_{t+k}^i) \Bigg] \end{aligned} \tag{2.23}$$

where τ_{t+k} is the next k steps of history in which player i acts. It is sampled from the distribution of successor states where all players play according to π_1.

To overcome the issue that the states reached by π_1 may be reached at very low probabilities under π_0, the variant Learned-belief OBL was proposed [21]. In this variant, an approximate belief $\hat{B}_{\pi_0}$ that takes τ_i as input and can sample a trajectory from an approximation of $P(\tau|\tau^i, \pi_0)$. The belief model is computed based on [20]. Q-learning is then performed with a modified approach in computing the target value $Q(a|\tau_{t+2}^i)$. In particular, a new τ' is resampled from $\hat{B}_{\pi_0}(\tau_t^i)$. Then, a transition to $\tau_{t+1}^{i'}$ is simulated with other agents playing policy π_1. The bootstrapped value is then $\max_a Q(a|\tau_{t+2}^i)$. Figure 2.7 shows the differences in the learning rules of independent Q-learning (IQL) and LB-OBL. The latter only involves fictitious transitions. Precisely, in addition to applying the action a_t^i given AOH τ_t^i of the active player i to the actual environment, the action is also applied to a fictitious state sampled from the belief model. The fictitious environment is then forwarded to the next state to compute the target after the other agent also takes a fictitious action. The learning target becomes the sum of fictitious rewards r'_t, r'_{t+1} and the fictitious bootstrapped value $\max_a Q(a|\tau_{t+2}^i)$. Practically, similar to what was done in [21], this target is computed during rollouts and stored along with the actual trajectory τ into the replay buffer. This may lead to the targets getting stale as they are precomputed

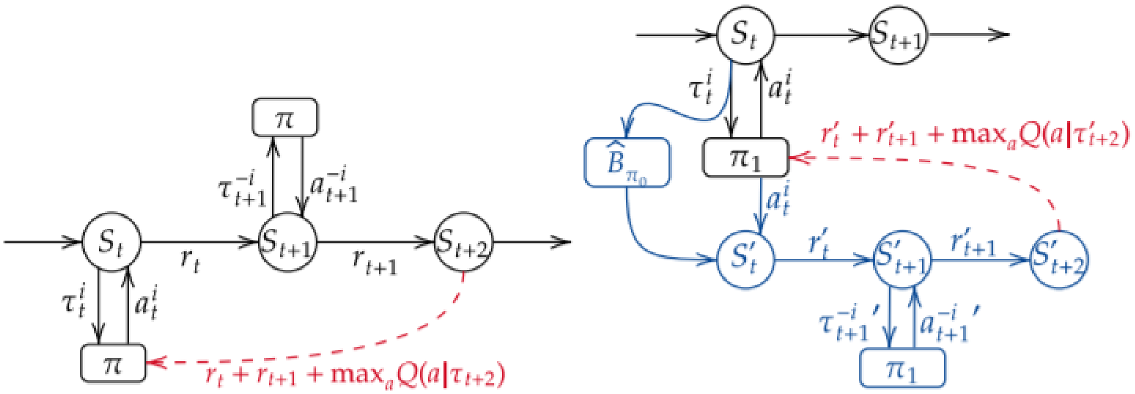


Figure 2.7: Learning rule differences between IQL and LB-OBL, taken from [21]

by an older target network. However, the authors find it to be fairly stable. Hence, we follow the same practice.

2.6.3 Differentiable Inter-Agent Learning

In [11], two methods were proposed to demonstrate end-to-end learning of protocols in complex multi-agent environments, namely, Reinforced Inter-Agent Learning (RIAL) and Differentiable Inter-Agent Learning (DIAL). The former uses deep Q-learning and the latter proposes backpropagating the learning gradients through communication channels across agents. Hence, the approach is based on centralized learning but decentralized execution, meaning these gradients are only allowed to flow across agents during training. Here, we will focus on DIAL.

DIAL addresses the idea of allowing agents to give each other feedback about their communicative actions, by opening up the communication channels for gradients to be pushed through from one agent to another. This provides richer feedback to the agent, leading to a reduction in the amount of learning by trial and error with more efficient discovery of effective protocols.

More precisely, during centralized training, direct connections between the output of one agent's network and the input of another agent's are established through communicative actions. In other words, agents can send real-value messages during this stage which are only restricted to discrete messages during execution. These real-value messages are generated from the networks, gradients can then be propagated along the channel, leading to end-to-end propagation across agents.

The proposed network is called C-net. It produces two different types of values, namely the Q-values $Q(\cdot)$ of the environment actions A_{env} and the real-valued messages m_t^a. The former is used by an action selector module to decide on an action

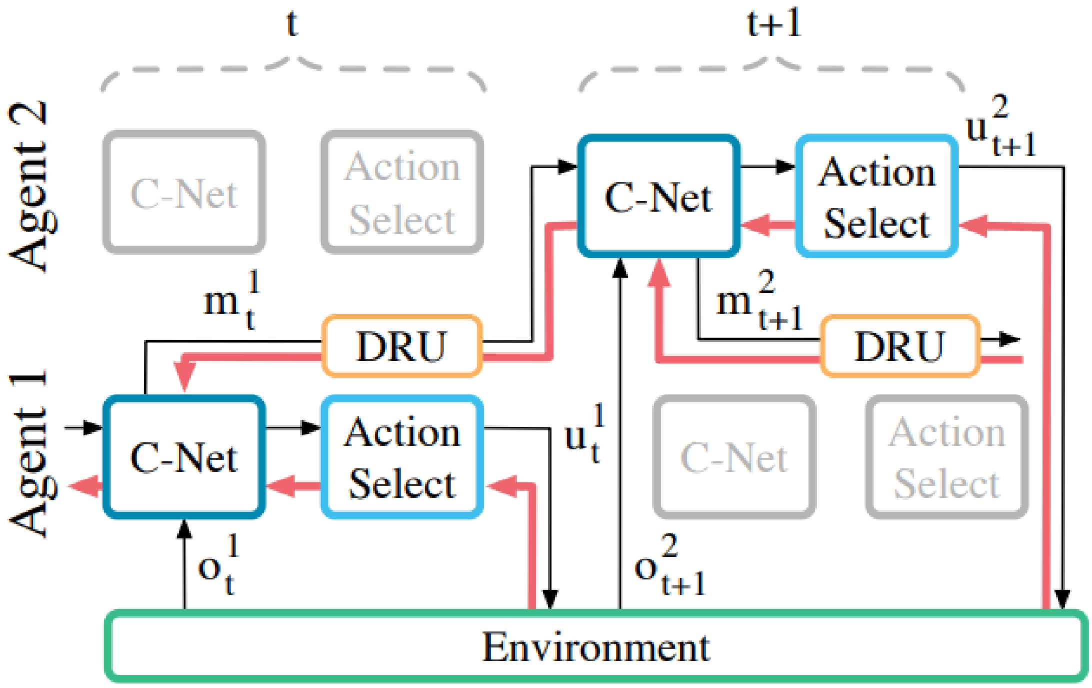


Figure 2.8: How gradients flow in DIAL, taken from [11]

while the latter bypasses that module and is passed to another agent after being processed by the *discretize/regularize unit* $DRU(m_t^a)$. The DRU is a function that takes in a real-valued message that is expressed as follows:

$$DRU(m_t^a) = \begin{cases} Logistic(N(m_t^a, \sigma)), & \text{if centralized learning} \\ \mathbb{1}\{m_t^a > 0\}, & \text{if decentralized execution} \end{cases} \tag{2.24}$$

where σ is the standard deviation of the noise added to the cheap talk channel. Figure 2.8 shows how the gradients flow with DIAL. We can see that in addition to the DQN loss, the gradient term for m is also backpropagated based on the error from the recipient of the message to the sender. This means the network can directly adjust the messages so as to minimize the downstream loss, making it easier to learn and explore good protocols. Note that we are using the OBL loss here instead of DQN loss. Given the approach handles real-value messages during training, it can clearly scale with settings that uses continuous messages in addition to discrete messages.

2.7 Information Theory

Information theory is a field of study focusing on quantifying, storing, and communicating information [58]. This section lays out the concepts from information theory that are used in this work.

2.7.1 Entropy

Entropy $H(\cdot)$ is considered as the average level of information or uncertainty inherent in a random variable's all possible outcomes [58]. It is expressed as:

$$H(X) = \underset{X \sim p(x)}{\mathbb{E}} - \log p(x), \tag{2.25}$$

where X is a random variable. The entropy of a random variable is maximized when the probability mass is spread as evenly as possible and minimized when it is concentrated at a single point.

2.7.2 Conditional entropy

Conditional entropy $H(\cdot|\cdot)$ of two random variables are expressed as follows:

$$H(X|Y) = H(X, Y) - H(Y), \tag{2.26}$$

where X and Y are random variables and $H(X, Y)$ is defined as:

$$H(X, Y) = \underset{X \sim p(x), Y \sim p(y)}{\mathbb{E}} - \log p(x, y). \tag{2.27}$$

In RL, conditional entropy better represents a policy's stochasticity as it is conditioned on the state or observation [61].

2.7.3 Mutual information

Mutual information measures the strength of the dependence between two random variables. The greater the mutual information between two random variables, the more the outcome of one variable affects the conditional distribution of the other. Mutual information is defined as:

$$I(X; Y) = H(Y) - H(Y|X) = H(X) - H(X|Y). \tag{2.28}$$

It can be interpreted as the amount of uncertainty about one variable that is eliminated by observing the realization of another variable.

Chapter 3

Related Work

This chapter reviews some of the related work that is relevant to our proposed method. Specifically, we would like to cover works that solve Dec-POMDP problems and propose similar ideas in using mutual information like ours in cheap talk discovery, and methods that learn how to communicate similar to the one we propose in cheap talk utilization. We note that our setting is quite different from many existing works in learning how to communicate given that we require the agent to discover the channels themselves, which many off-the-shelf protocol learning algorithms would not work without additional treatment for discovery.

3.1 MARL algorithms for Dec-POMDPs

Many MARL methods have been proposed to deal with cooperative multi-agent decision-making problems under the Dec-POMDP formalism [45]. We will cover methods that tackle cooperative multi-agent problems with a focus on approaches that tackle the credit assignment problem.

To begin with, approaches like [62] transform a Dec-POMDP problem into a simpler formalism that can be solved with planning algorithms, which is more computationally tractable than the original formalism. In recent years, the use of deep neural networks as function approximators to solve Dec-POMDP problems has found remarkable successes. [16] extends existing algorithms based on policy gradient, temporal-difference error, and actor critic methods to cooperative multi-agent settings, showing how deep MARL algorithms can scale successfully in continuous action spaces to perform complex continuous tasks like bipedal walking. In this approach, agents are solely trained based on their respective observations with no channels or ways for them to communicate but parameters are shared across agents. [55] proposes Multi-Agent Common Knowledge Reinforcement Learning by allowing a group of agents to condition on their common knowledge, in addition to their respective observations. By exploiting common knowledge, agents can demonstrate more complex decentralized coordination in benchmarks like StarCraft II unit micromanagement tasks. [10] proposes counterfactual multi-agent policy gradients which uses a centralized critic to Q-function estimation and decentralized actors for policies learning. The approach addresses the multi-agent credit assignment problem by using a counterfactual baseline that marginalizes out an agent's action with the other agents' actions fixed. This provides information on the contribution an agent's action has on an outcome which was shown to significantly improve performance. [64] proposes value decomposition network (VDN) that decomposes the team value function into agent-wise value functions, with the latter conditions only on the individual agent's observations and actions. This leads to better credit assignment across agents, addressing the issue of the "lazy agent" problem in which agents are credited for not doing anything useful because some other agents performed useful actions. [51] proposes QMIX which takes the previous approach further by having the team value function as a nonlinear combination of each agent's action-value function instead of a simple sum. This allows different weighing to be attributed to each agent. Note that in this case, the team value function is conditioned on the state. [9] takes another direction by learning explicit intrinsic reward for each agent instead of changing the value function architecture like VDN and QMIX. By learning individual intrinsic reward, agents are diversely stimulated at each time step, leading to a more effective and informative credit assignment per time step.

3.2 Mutual Information in Reinforcement Learning

In this section, we will cover previous work in RL that leverages the idea of mutual information. Mutual information has been used for many purposes across different variables in the RL paradigm including improving exploration and regularization. In [3], mutual information maximization was proposed to improve state representation learning, which maximizes mutual information across spatially and temporally distinct features of observations. They showed that this approach leads to the capturing of the underlying factors of a state, a more powerful representation. [40] proposed using mutual information as empowerment to improve exploration. Specifically, the approach looks for maximal mutual information, conditioned on a starting state s, between a sequence of K actions $\mathbf{a}$ and the final state reached s'. An agent that maximizes this value will go for the states from which it can reach the largest number of future states within its planning horizon. [40] further derived a variational information lower bound, so maximizing mutual information can be more tractable. [14] proposed using mutual information for regularization to improve upon entropy-based regularization. Precisely, entropy-based regularization improves exploration by encouraging policies to place probability mass on all actions while some actions may be undesirable. Thus, [14] used mutual information to weigh the importance of actions, leading to better action selection than entropy regularization approaches.

Mutual information has also been explored in the MARL setting. [69] proposed a reward shaping approach based on mutual information to improve exploration in the multi-agent setting. Unlike our approach which maximizes the mutual information between an agent's actions and the other agent's observations, they proposed maximizing the mutual information between two agents' transitions $MI(S_2'; S_1, A_1|S_2, A_2)$. This encourages an agent to visit critical points where it can influence the transition probability of another agent. Using this quantity might have a similar effect as our proposed mutual information quantity, but it might be harder to compute. They showed that the approach gives rise to more coordinated exploration with better policies in optimizing team performance. However, unlike our method, this approach requires access to the full environmental state during training. [28] proposed a similar framework as [14] but in the multi-agent setting. The approach maximizes the mutual information between the actions of agents which showed state-of-the-art results in several benchmarks. [61] proposes a method to discover implicit communication protocols via minimum entropy coupling, separating communication and non-communicative decision-making. Unlike cheap talk channels which facilitate communication explicitly, implicit communication refers to communication through actions that have non-communicative effects on the environment. [23] proposed rewarding agents for having causal influence over other agents' actions using mutual information to achieve better coordination and communication. Each agent is required to learn a model of other agents to compute the influence reward in a decentralized manner. The method shows improved performances in sequential social dilemma environments. [23] has the closest resemblance to our approach but it still assumes the omnipresence and knowledge of communication channels and uses a different mutual information quantity. They propose maximizing the mutual information of actions among agents at each time step, which is not necessarily applicable to our situation in terms of achieving cheap talk discovery.

3.3 Communication Protocol Learning in Multi-Agent Reinforcement Learning

In this section, we will cover previous work in MARL that focuses on algorithms that learn how to communicate among agents. Methods that we use like DIAL will be omitted here as they will be covered in detail in section 2.6.3. [63] proposes CommNet, a similar approach to DIAL in which agents are connected by differentiable continuous communication channels but the messages are processed by a mean-pooling method, making it suitable to deal with environments with a dynamic number of agents. [47] proposes using a bidirectional recurrent neural network for communication channels. Unlike DIAL and CommNet, BiCNet focuses on continuous action space and actor critic methods. The choice of architecture for communication channels also allows an agent to deliver information to others one by one. [48] proposes a policy gradient method with a shared memory device for communication. Specifically, during training, agent learns to perform read and write operations on the memory device to form a common representation of the world and as an explicit communication protocol. They showed superior performance on tasks that require coordination and synchronisation skills. [27] proposes the message-dropout technique which drops the received messages through a communication channel at a specified probability. The technique is shown to improve the robustness of learned protocols. [8] proposes a targeted communication architecture to tackle the issue of what messages to send and who to send them to. By including the consideration of the latter question, it leads to more flexible collaboration strategies in complex environments. The method also enables multiple rounds of communication before taking actions in the environment. They showed superior performance with communication protocols that are more interpretable and intuitive. [60] proposes a different approach that also considers the question of when to communicate. A gating mechanism was proposed which can control when to communicate. Together with the addition of individual rewards, the proposed approach is also able to handle semi-cooperative and competitive settings along with cooperative settings with better training efficiency. [30] focuses on referential games and showed that deep RL agents can learn communication protocols in different cases, including when the inputs are symbolic data and pixel data. They also find agents' inability to learn more compositional communication protocols if the input data are more entangled. [7] examines two main types of communication protocols used in MARL, namely grounded channel that is restricted by the semantics of the game and ungrounded channel in form of cheap talk. Interestingly, results reveal that self-interested agents can use grounded channels to negotiate fairly but are unable to use ungrounded ones, while prosocial agents are able to. This suggests cooperation could be necessary for a language to emerge.

Chapter 4

Methodology

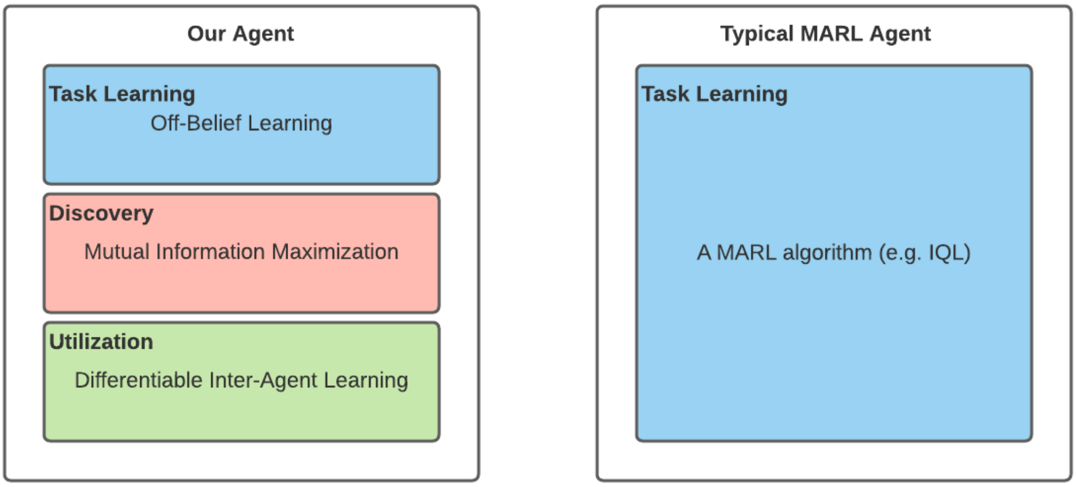

Our Agent
Task Learning
Off-Belief Learning
Discovery
Mutual Information Maximization
Utilization
Differentiable Inter-Agent Learning
Typical MARL Agent
Task Learning
A MARL algorithm (e.g. IQL)

Figure 4.1: Comparative overview of our proposed method with a typical MARL method. Task learning here refers to the learning of the main task.

In this thesis, we look at one of the most commonly used communication methods in MARL, known as cheap talk channels, which allow agents to send discrete or continuous messages to each other without considerable cost. To the best of our knowledge, existing approaches assume agents to have knowledge of these channels and can be accessed whenever they decide to. Our work does away with this assumption and requires the agents to learn the knowledge of these channels. In other words, we ask the question of whether we can develop agents that can learn where to best communicate with each other before learning what and how to communicate. As mentioned in section 1.1, not only does this have a lot of potential applications, it also requires the address of some of the open problems in MARL like credit assignment given how the problem requires coordination and joint deep exploration among agents.

Given that this is a problem that has not been explored before, in this section, we would first provide a formulation of the problem and motivate how the decomposition of the problem in this formulation would make the problem more solvable. Then, we will go over in detail our proposed method in tackling cheap talk discovery and utilization that is based on information theory and some of the most recent advances in the field. Figure 4.1 shows a high-level comparison between our proposed method and a typical MARL agent, which our method breaks the problem down into smaller and more solvable components.

4.1 Problem Formulation

To begin with, we consider each agent's action space A to be decomposable into two subspaces, namely, the environment action space A_{env} and communication action space A_{comm}, where:

> A_{env}: environment action space, actions that have an impact on the state of the environment.
>
> A_{comm}: communication action space, actions that have no immediate impact on the state of the environment or the reward that other agents receive but can have an influence on other agents' observations. They are used to communicate information, commonly known as cheap talk actions.

Using our phone booth maze environment as an example, A_{env} for the sender would be actions that move its locations like *Up*, *Down*, *Left* and *Right*, while A_{comm} would be actions that send messages to the receiver including *Hint Up* and *Hint Down*.

As discussed, in this setting, agents do not know where to best communicate, meaning only certain places are ideal or possible for communication. We refer to these places in the state space as the communicative state space $S_{comm} \in S$, which is defined as:

Definition 4.1. The communicative state space S_{comm} of an environment is a subspace of its state space S. Assume the environment is in a state $s_c \in S_{comm}$, at least one agent in the environment can modify another agent's observation by taking an action $a \in A_{comm}$.

Similarly, the communicative joint observation space $\mathbb{O}_{comm}$ can also be defined as follows:

Definition 4.2. The communicative joint observation space $\mathbb{O}_{comm}$ of an environment is a subspace of its joint observation space $\mathbb{O}$. Assume the environment is in a state $s_c \in S_{comm}$ with a joint observation $\mathbb{O}_c$, at least one agent, say agent i, in the environment can modify another agent's observation, say agent j's observation, o_j^{t+1} by taking an action $a \in A_{comm}$.

With these conceptualizations, the cheap talk discovery and utilization problem can then be defined. S_{comm} corresponds to the places where the cheap talk channels are and an agent has to learn to discover them and utilize them. Formally, the problem can be further formulated as follows:

Definition 4.3. Cheap talk discovery refers to the problem of learning a policy $\pi_{discover}$ in which agents take action in A_{env} that could lead to states or observations that are in S_{comm} or $\mathbb{O}_{comm}$.

Definition 4.4. Cheap talk utilization refers to the problem of what message to send and how to send a message once the cheap talk channels are discovered, i.e. an agent has a policy π to reach states in S_{comm}. In other words, what messages m should an agent send to another agent in order to convey the correct meaning that could lead to task completion.

To tie it back to a more concrete example, our phone booth maze environment, cheap talk discovery would be the agent's capability in discovering functional phone booths while cheap talk utilization would be the agent's ability to form a communication protocol so the task can be solved by having the sender conveying the goal consistently and the receiver interpreting the goal from the sender's message correctly.

Even with a problem seemingly as simple as the phone booth maze environment, the joint exploration problem is still severely difficult. This is because just for communication to happen, it requires agents to stumble upon states that are in S_{comm} (i.e. both the sender and receiver each happen to be in a functional phone booth). As the main task's reward signal does not directly motivate going to these states, the motivation or signal to go to the booth is very weak, let alone having sufficient experience to form a protocol among agents. Hence, the problem is quite formidable for the agents to solve. Thus, we hypothesize that by breaking down the problem that first learns to discover the communicative states before learning how to use the discovered channels would greatly alleviate the joint exploration problem.

4.2 Cheap Talk Discovery

In this section, we will present our proposed method to achieve cheap talk discovery. The method is composed of two components, namely Off-Belief Learning (OBL) [21] and mutual information maximization. The former serves as our base learning algorithm with sound theoretical properties that are beneficial for the subsequent cheap talk utilization step. The latter is our idea to induce agents to discover these cheap talk channels based on mutual information.

4.2.1 Off-Belief Learning

As mentioned, the approach sets to learn optimal grounded policies, which do not interpret the actions of other agents and assumes other agents do the same with their actions. This is often desirable as making wrongful assumptions can often be a source of coordination failure. Thus, learning or starting with grounded policies is important to avoid reasoning about partners' actions.

4.2.1.1 From Learned Belief to Exact Belief

Learning the beliefs of other agents is an open and hard problem in MARL. However, since this is not the focus of the project, we would like to take this factor out of our consideration in order to focus on the performance of the methods of interest. Thus, to make sure the belief model does not influence our assessment, we use exact belief models for our agents instead of learned ones. Given that our environments have relatively straightforward dynamics, we are able to hard code the belief models without significant time and effort. Specifically, for OBL, we need the belief model of the policy π_0 to sample from, which is a random policy. Our exact belief models are initialized with a transition matrix $\mathbf{T}$ and a starting belief vector $\mathbf{b}$. At each time step t, the belief is updated as follows:

$$\mathbf{b}_{t+1} \leftarrow \mathbf{b}_t \cdot \mathbf{T} \tag{4.1}$$

These belief models are reset at the end of every episode. Figure 4.2 shows how the belief model of the sender in the phone booth maze environment evolves over time, which demonstrates how the probabilities propagate as the belief is being updated. Note that they are essentially belief probabilities of where the sender is after assuming it follows a uniformly random policy π_0. For the sender's belief model, it has a special case when it successfully sends a message, which narrows down its possible location

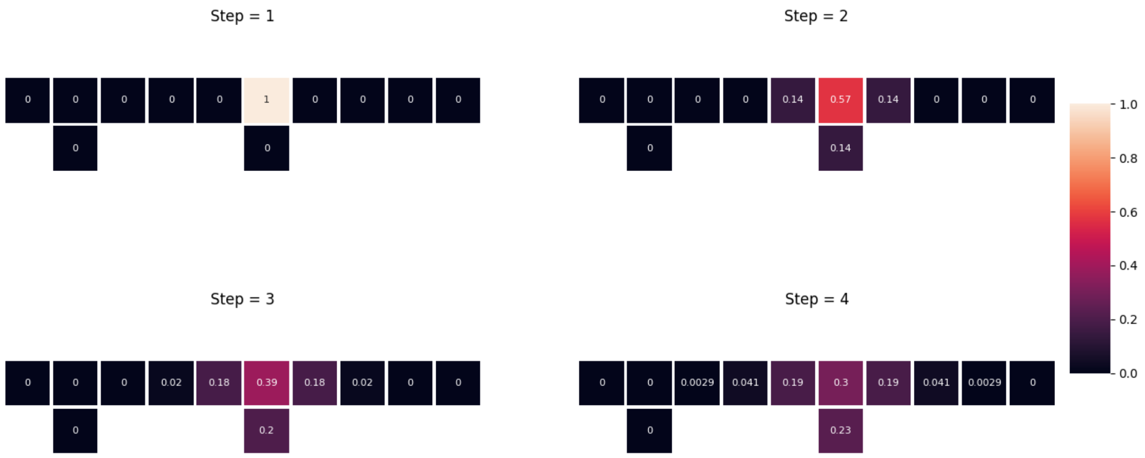


Figure 4.2: Heatmap illustration of how the sender's belief model in the Phone Booth Maze environment evolves over time

to the functional phone booths. Our exact belief models handle this special case explicitly and would recompute the belief probabilities.

4.2.1.2 Why Off-Belief Learning?

OBL has appealing theoretical properties that are very useful for learning communication. Two properties derived in [21] are particularly of interest:

Theorem 4.1. For any temperature $T > 0$ and starting policy π_0, *OBL* computes a unique policy π_1.

Theorem 4.2. Applying OBL to any constant policy $\pi_0(a|\tau^i) = f(a)$ or policies that only condition on the public state yields an optimal grounded policy in the limit as temperature T tends to zero.

Theorem 4.1 offers an advantage for policies learned from OBL as they are unique, unlike most MARL methods. In other words, OBL always converges to the same policy regardless of the random initialization of weights and other hyperparameters. What's more, theorem 4.2 is especially beneficial for cheap talk discovery. Given that we do not want any conventions (i.e., a communication protocol) to form during the learning process in cheap talk discovery which would affect the rewards received, learning an optimal grounded policy over communicative actions would be preferable. This means it is preferred to learn a policy that discovers the channels, or cheap talk actions that could lead to outcomes observable by other agents, while not having a preference over any cheap talk actions. Advantageously, OBL theoretically guarantees to yield such policy. In the case of the Phone Booth Maze, the actions of *Hint Up* and *Hint Down* after the cheap talk discovery learning process should be equally preferred by the agent. Such a grounded policy is preferable as we separate out the protocol learning process into a separate cheap talk utilization problem. This makes it easier to learn different protocols for different types of communication channels. More importantly, by having the policy grounded, it allows more flexible adaptation when properties of channels alter. Hence, better performance in zero-shot coordination can be expected. In the case of the phone booth maze, imagine the agent is put into a slightly different version of the environment with the phone booth sending negated versions of whatever messages the agent sends, the agent with the grounded policy should adapt to this environment quicker given that it has an optimal grounded policy.

4.2.2 Mutual Information Maximization

OBL offers us the ability to learn optimal grounded policies, which makes cheap talk discovery easier by learning a uniform policy over cheap talk actions. However, OBL alone is not sufficient given the lack of incentives to visit cheap talk channels. As using these channels is not immediately rewarded, not to mention they have to be used properly in order to be consistently rewarded in expectation (i.e., forming a protocol), agents are simply not well motivated to find these channels.

To discover cheap talk channels that can facilitate communication with other agents, an agent has to know which channel can actually affect another agent when used. In other words, useful cheap talk channels are the ones that can influence another agent's observations. In the case of the phone booth maze, the sender should discover that the phone booth connected to the receiver's end is the one to be used in order to communicate with the receiver, given that it is the only phone booth which cheap talk communication can be transmitted to the receiver. To induce such discovery, we propose a different reward function and loss function based on mutual information as defined in 2.28.

4.2.2.1 Mutual Information Reward

Using the phone booth environment as an example, we propose using mutual information as an extra term to the reward function to encourage greater mutual information between the sender's (denoted as agent 1) actions A^1 and the receiver's (denoted as agent 2) observations O^2. This can also be considered as a form of reward shaping [42]. The reward function is modified as follows:

$$R'(\tau_t, \pi_0, \pi_1) = R(\tau_t, a) + \alpha H(\pi_1) + \beta I(A^1, O^2). \tag{4.2}$$

where α and β are hyperparameters. Here, the first term is the environmental reward and the second term corresponds to the entropy of π_1 which encourages the policy to be more stochastic, akin to the maximum entropy RL objective [73]. The third term is the mutual information term mentioned above which can be explicitly expressed as:

$$I(A^1, O^2) = \mathop{\mathbb{E}}_{a^1 \sim A^1, o^2 \sim O^2} [\log(p(a^1|o^2) - \log(p(a^1))] \tag{4.3}$$

We assume we have a perfect environment model which allows us to estimate $p(a^1|o^2)$. We would need density-based approaches to estimate this term as we move on to more difficult environments that do not have easily accessible and perfect environment models [49, 39]. The proposed reward function in equation 4.2 can be directly substituted into the reward function in the update equation 2.23. Once we have agents that can discover cheap talk channels, we can employ off-the-shelf cheap talk utilization algorithms like DIAL [11]. To estimate $p(a^1|o^2)$, the term can be further expressed as follows:

$$\begin{aligned}
I(A^1, O^2) &= H(A^1) - H(A^1|O^2) \\
&= H(A^1) + H(O^2) - H(A^1, O^2) \\
&= \mathop{\mathbb{E}}_{a^1 \sim A^1} [-\log(p(a^1))] + \mathop{\mathbb{E}}_{o^2 \sim O^2} [-\log(p(o^2))] - \mathop{\mathbb{E}}_{a^1 \sim A^1, o^2 \sim O^2} [-\log(p(a^1, o^2))] \\
&= \sum_{a^1 \sim A^1} [-p(a^1) \log(p(a^1))] + \sum_{o^2 \sim O^2} [-p(o^2) \log(p(o^2))] \\
&- \sum_{(a^1, o^2) \sim (A^1, O^2)} [-p(a^1, o^2) \log(p(a^1, o^2))]
\end{aligned}$$

where $p(o^2) = \sum_{a^1 \sim A^1} [p(a^1, o^2)]$ and $p(a^1, o^2) = p(o^2|a^1)p(a^1)$. We note that this approach generalizes to any pair of agents in a MARL environment.

Practically, using the perfect environment model (we simply give the agent access to a replica of the environment), we first save the current configuration of the environment and load it into the replica. Then, we perform every action possible for the agent and reset the replica to the current configuration while keeping track of the observations seen. By doing so, we are able to compute the mutual information reward term.

4.2.2.2 Mutual Information Loss

Throughout our experiments, we noticed that the mutual information reward term might not be sufficient in retaining an optimal grounded policy over communicative actions. Although the additional reward does motivate the agent to discover and go to these communication channels, it does not help in distinguishing communicative actions A_{comm} from actions that keep the agents within S_{comm}. This leads to a policy that favors all these actions equally. Ideally, we prefer a policy that only favors the communicative actions equally when in S_{comm}. This issue arises because the reward term is computed based on taking all the actions from a particular state. Therefore, for the same policy, the mutual information reward term for staying at communication channels and that of taking communicative actions are actually the same, resulting in these actions being treated equally. In the case of the phone booth environment, this would mean the communicative actions - *Hint Up* and *Hint Down* will be treated equally with environment actions A_{env} that keep the agent in the functional phone booth.

To deal with this issue, we propose adding a mutual information loss to the loss function which uses the policy from the model. By having this extra term in the loss function, the model would update to maximize the mutual information between an agent's action and the other agent's observation, which is maximized when the policy takes the communicative actions more. For each iteration i, extending equation 2.14, the loss function is expressed as:

$$L_i(\theta_i) = L_i^{OBL}(\theta_i) - \kappa I(A^1, O^2; \pi_\theta)_i \tag{4.4}$$

where:

$L_i^{OBL}(\theta_i)$: OBL loss at iteration i

$I(A^1, O^2; \pi_\theta)_i$: Mutual information loss at iteration i

π_θ: the policy from the model

κ: Hyperparameter to weigh the mutual information loss term

Note that a minus term is used here as we are trying to maximize the mutual information term. By combining this loss term with the mutual information reward, the agent is then able to discover cheap talk channels and learn to prefer communicative actions that correlate with high mutual information.

4.3 Cheap Talk Utilization

In this section, we will go over our proposed component to achieve cheap talk utilization. After discovering cheap talk channels, agents have to learn how to use them by forming a proper protocol. Here we refer to learning a protocol as agents forming a consensus in terms of how messages are interpreted. In the case of the phone booth maze, an example protocol would be *Hint Up* and *Hint Down* being interpreted by the receiver as 1 and 0 respectively. In this work, we propose using Differentiable Inter-Agent Learning (DIAL) [11] as our algorithm for cheap talk utilization, which has been shown to learn protocols efficiently. We will cover how we adapt DIAL to our setting. Note that there could be many other approaches to perform cheap talk utilization. However, due to the limitation of time and computing resources, we leave the benchmarking of different cheap talk utilization methods for future work and focus on how our proposed framework serves as a promising starting framework to handle such problems.

4.3.1 Differentiable Inter-Agent Learning with Experience Replay

As DIAL backpropagates gradients across agents, in [11], an entire trajectory of an episode is generated to maintain the gradient chain. This is inherently not fully compatible with our problem for two reasons. Firstly, OBL itself is an experience-replay-based method in which updates are performed based on batches of transitions sampled from the replay buffer. Furthermore, unlike in the environments used in [11] which messages are sent between agents in every time step, messages can only be sent when the environment is in a state $s \in S_{comm}$. In other words, direct connections can only happen occasionally within an episode when a connection is established between agents by being in the communication channels (i.e., in the phone booths for the phone booth maze environment).

As a result, we have adapted DIAL to make it compatible with our setting and work well with experience replay buffers. To do so, we have a separate replay buffer that is designed to work with DIAL. Specifically, during an episode, we keep track of the transitions and only add the trajectory to the buffer when communicative actions are taken successfully. Hence, the last two transitions of an added trajectory would be an agent taking a communicative action and another agent taking an action after receiving a message. So the target would be the learning error of the latter agent which would be backpropagated through to the sender's sequence actions. Additionally. we

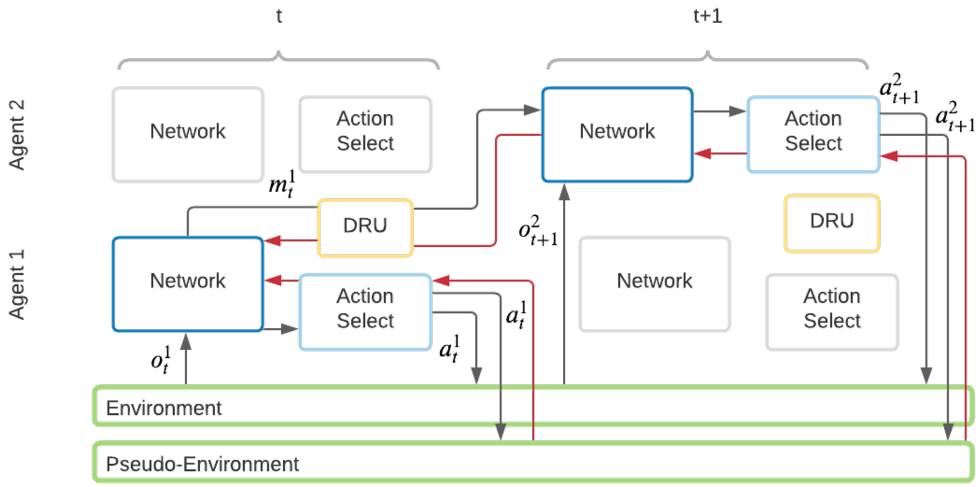


Figure 4.3: How gradients flow through our proposed architecture, with reference to [11]

also tried a version of a buffer that stores entire episodes with flags that indicate which transitions to allow direct connections. Experimentally, the latter appears to be able to learn a protocol much more effectively, most likely due to less noisy gradient information by having full trajectories, leading to agents assigning credits to communicative actions more accurately.

We notice that this replay buffer for DIAL would require the implicit assumption of knowing when the agents are using the communication channels. But we envision that this can be known by the mutual information value we obtain from the discovery process, acting as a trigger to the utilization process in a heuristic manner. Hence, whenever high mutual information is observed, the trajectory will be added to this special buffer for DIAL. We leave the exploration of different types of replay buffer for DIAL as future work.

4.4 Putting It All Together

Figure 4.3 provides a complete overview of how gradients flow when putting all components of our proposed method together. Notice that when comparing it with figure 2.8, an extra *Pseudo-Environment* object is present which corresponds to the replica environment that is used to perform OBL sampling. The figures show the updates when the agents are in the communication channels in which both OBL and DIAL would be in action. The message is omitted for agent 2 to match with the phone booth maze setting where the receiver has nothing to send to the sender. These different learning paths do not activate at the same time due to their different roles as pointed out in the 3 learning stages illustrated in Figure 1.1. Precisely, during the discovery stage, only OBL with mutual information maximization is activated to discover communication channels. Next, with the discovery policy, DIAL will be performed with normal Q-learning to learn a communication protocol and solve the task. Note that normal Q-learning is used here instead of OBL because the latter prevents formation of conventions, which is needed here once an optimal grounded policy over communicative actions is learned with communication channels discovered. Algorithm 3 further provides a detailed step-by-step description of how our proposed method functions with all components working together.

Algorithm 3: Pseudocode for our proposed method

```
for each agent do
    Initialize replay memory for OBL $D_{OBL}$ with capacity $N$
    Initialize replay memory for DIAL $D_{DIAL}$ with capacity $N$
    Initialize action value function Q with random weights $\theta$, using architecture C-Net based on R2D2
    Initialize target action value function $\hat{Q}$ with random weights $\theta^{-} = \theta$ , using architecture C-Net based on R2D2
for $e$ = 1, Max_Episode do
    Initialize $s$
    while $s \neq s_{terminal}$ do
        for each agent do
            Select $a$ from $o$ based on policy derived from $Q$, e.g. $\epsilon$-greedy policy or stochastic OBL policy
            Take action $a$ to observe reward $r$ and next state $o'$
            Take action $a$ in Pseudo-Environment to perform OBL sampling with mutual information computation
            Store the transition into $D_{OBL}$
            if High mutual information is observed then
                Store trajectory into $D_{DIAL}$
            Sample random minimatch of transitions from $D_{OBL}$
            Sample random minimatch of transitions from $D_{DIAL}$
            if Discovery stage then
                Perform a gradient descent step on $L_i^{OBL}(\theta_i) - \kappa I(A^1, O^2; \pi_\theta)_i$ with respect to the network weights $\theta$
            else
                Perform a gradient descent step on $(y_i - Q(o, a; \theta_{i-1}))^2$ for DIAL with respect to the network weights $\theta$
                Perform a gradient descent step on $L_i^{IQL}(\theta_i)$ with respect to the network weights $\theta$
            Every C steps, set $\hat{Q} = Q$
```

Chapter 5

Experiments and Results

5.1 Experimental Setup

5.1.1 Environment Design

As part of our contribution, we designed and implemented a configurable environment to evaluate agents in cheap talk discovery and utilization, called the (multi-) phone booth maze. The environment has two agents in the classical sender-receiver setting interacting in a discrete GridWorld. These two agents are placed into separate rooms with the goal being the receiver escaping successfully from the correct exit in one of the doors in its room. Importantly, only the sender is given the correct hint on which door is the correct one and it is the sender's job to communicate the goal to the receiver. Although they are isolated in different rooms, the rooms are connected by phone booths that would allow transmissions of messages. Some of the booths are functional while some of them are not. Therefore, the sender and the receiver have to learn to go to connected booths in order for the sender to communicate the goal information to the receiver with a learned protocol so the receiver can exit correctly.

A positive reward is given if the receiver exits correctly and a negative reward is given if the receiver exits incorrectly. Here, we use rewards of 1.0 and -0.5, respectively. If the episode times out, no reward is given. Both agents have the environment actions of *UP*, *DOWN*, *LEFT*, *RIGHT*, *NO-OP* with the last action as not moving. The sender has two extra communicative actions, namely *HINT-UP* and *HINT-DOWN* which only has an effect when both agents are in the functional phone booths. The observations for the agents consist of a tensor of 3 channels plus some role-specific information. The channels are the wall channel, phone booth channel, and the agent channel, which are essentially binary grid encoding of each position for the existence of wall, phone booth, and the agent respectively. For the sender, it has

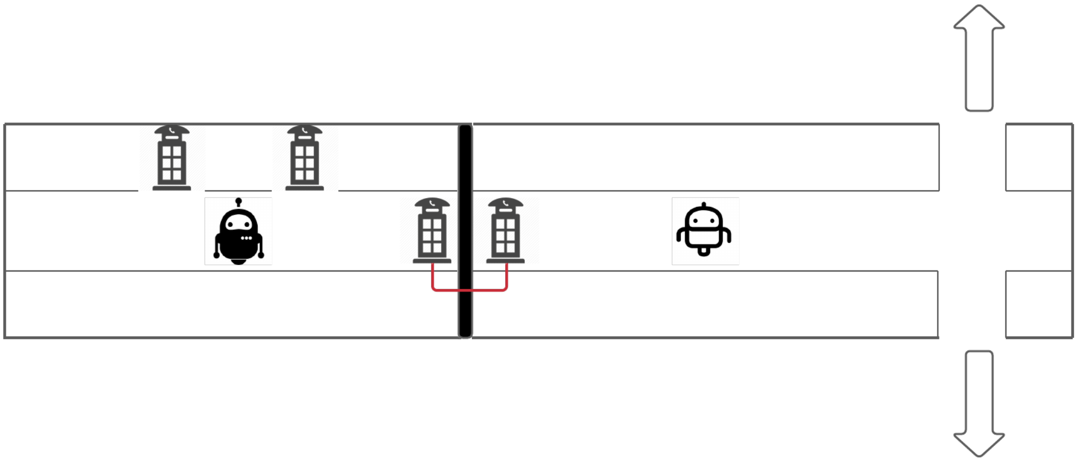
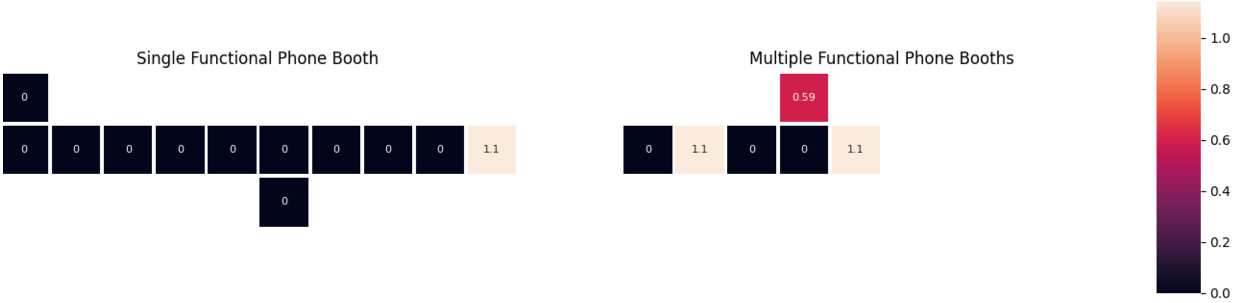


Figure 5.1: Visual illustration of an instance of the Phone Booth Maze environment

an additional 2-bit encoding vector as goal information. Specifically, if the receiver should go up, the sender would get a vector of $\begin{bmatrix}1 & 0\end{bmatrix}$. If the receiver should go down, the sender would get $\begin{bmatrix}0 & 1\end{bmatrix}$ instead. For the receiver, it has a communication token vector. If the sender performs a *HINT-UP* with both of them at the functional booths, it would be a vector of $\begin{bmatrix}1 & 0\end{bmatrix}$. On the other hand, if the sender performs a *HINT-DOWN*, it would be $\begin{bmatrix}0 & 1\end{bmatrix}$.

Figure 5.1 shows an instance of what the environment would look like when visualized. There are multiple phone booths in the environment, acting as cheap talk channels, which allow the sender to communicate the goal to the receiver. In this instance, there is only one phone booth in the sender's room that is connected to the phone booth in the receiver's room. The connected phone booths are indicated by the connecting red cable in the figure. Many aspects of this environment are configurable to allow extensive investigation and exploration of different algorithms, some key adjustable features are:

- **lengths**: the length of each of the agent's rooms. The longer it is, the harder the exploration becomes.
- **starting points**: the starting location of each of the agents.
- **correct reward**: The reward that is given when the receiver exits from the correct door.
- **wrong reward**: The reward that is given when the receiver exits from the wrong door.


Single Functional Phone Booth

Multiple Functional Phone Booths

Figure 5.2: Mutual information heatmaps of the sender's room, based on two different configurations of our environment

- **use intermediate reward**: If set as true, it gives an intermediate reward for if both agents are at the functional booths, used for debugging purposes.
- **episode limit**: Episode length, meaning the episode terminates if it reaches this number of steps.
- **booth types**: Among functional phone booths, they are configurable in two major ways, namely, cost and noise. The former refers to the cost of using the booth and the latter is modeled as the probability of a phone booth dropping a message.
- **booth locations**: Locations of functional booths.
- **number of decoy booths**: The number of decoy booths in the sender's room. Decoy booths refer to booths that are not functional and not connected to the booth in the receiver's room. They appear as booths in the agent's observation (i.e., the booth channel).
- **booth reinitialization**: If set true, the decoy phone booths' locations are reinitialized whenever the environment is reset for the next episode, making the problem more difficult.
- **use mutual information reward**: If set true, the environment would compute the mutual information reward, used in the proposed method.
- **use mutual information loss**: If set true, the environment would return the necessary information (i.e., tensor masks for each term in equation 4.2.2.1 to compute the mutual information loss in a batch-based manner.

Such a high level of configurability allows users to create a wide variety of scenarios to test their algorithms. For instance, different booth types give rise to phone booths with different levels of mutual information. Figure 5.2 demonstrates two different instances that are used in our experiments. The one on the left shows a sender room with only one functional phone booth. The one on the right shows a sender room with three functional booths with one of them having a noise probability of 0.5. Note that the mutual information computed on the right figure was computed based on 1000 different random seeds.

5.1.2 Network Structure and Training

As briefly mentioned, all our methods use an architecture modified based on the R2D2 architecture by [26]. Instead of using convolutional layers, to lower the training time due to limited computing resources, we use fully-connected neural networks with a recurrent component as our neural network model. Hence, we flatten the observation from the environment into a vector by first fattening the 3-channel tensor and then concatenate it with the role-based information (i.e. goal encoding or communication token). Precisely, the input is first processed by an LSTM layer to handle partial observability. Then, it is followed by a two-layer and two-headed fully-connected neural network of hidden size 128. The two heads are used to compute the action-value function and the advantage function respectively. All neural network components are implemented using the neural network library PyTorch [46], with the weights initialized using Xavier initialization [13].

In terms of training, we use the Adam optimizer to train our models [29]. Since the baselines we use and variants of all our proposed methods use Deep Q-Learning as the backbone algorithm when training, we performed a hyperparameter sweep over common hyperparameters and fixed them across all the methods. See Appendix A.2.1 for details of the sweep values of other common parameters and A.2.2 for values of method-specific parameters. All methods were trained for 12000 episodes (80000 episodes for cheap talk utilization) and evaluated every 20 episodes by taking the corresponding greedy policy.

5.2 Results

This section reports and analyzes the results obtained from our experiments in cheap talk discovery and utilization. We use IQL and OBL as our baselines to compare with different variants of our proposed method. All results are obtained from test episodes during evaluation. Each algorithm reports results from 4 random seeds. For curves, the mean values are plotted with standard errors in the form of shaded areas, which are defined as $\mu \pm \sigma/\sqrt{N}$, where μ, σ and N are the mean value, standard deviation and number of runs respectively. Since we could only run a limited number of runs due to limited time, some of the shaded areas are visually obstructive. Therefore, we will show the exponentially smoothed results in the main results with versions without smoothing placed in the appendix.

5.2.1 Cheap Talk Discovery

In this section, we will look at different algorithms' performance of cheap talk discovery. To quantify this performance, we look at the number of steps it takes for the sender to first reach the functional phone booth. As we are assessing the performance of cheap talk discovery, the fewer number of steps it takes to reach the functional phone booth the better the algorithm is in cheap talk discovery. At the same time, we also examine the sender's policy at the functional phone booth to see an agent's preference between environment actions A_{env} and communicative actions A_{comm}. The instance of the Phone Booth Maze environment we use here has one functional phone booth with the following properties:

- **lengths**: 10 and 3 for sender and receiver respectively
- **starting points**: $[5, 1]$ and $[1, 1]$ for sender and receiver respectively
- **correct reward**: 1.0
- **wrong reward**: -0.5
- **episode limit**: 40
- **booth types**: 1 functional phone booth
- **booth locations**: $[9, 1]$
- **number of decoy booths**: 2
- **booth reinitialization**: False

5.2.1.1 Sanity check

As a sanity check to make sure our environment and backbone model are performing as expected, we first look at the performance of our baselines (i.e., IQL and OBL). We also include their variants with intermediate rewards to ensure that this form of reward shaping does induce the cheap talk discovery behavior. Note that although the intermediate reward appears to be simpler than our proposed mutual information reward, it is less flexible and requires knowledge and explicit hard-coding of where the cheap talk channels are, making it less transferable and applicable to different environments.

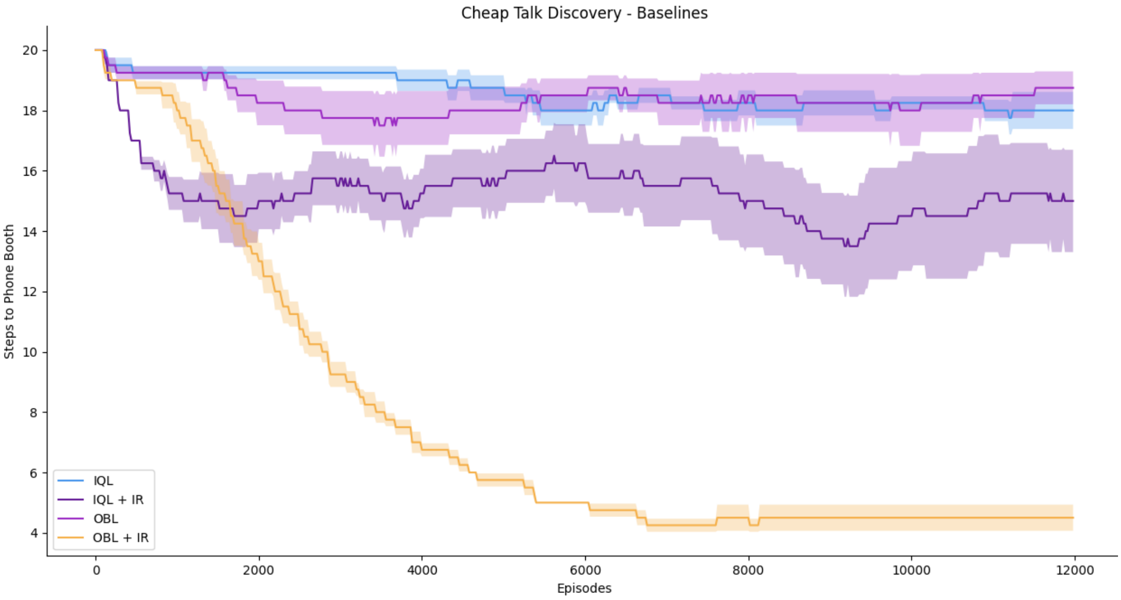


Figure 5.3: Baselines' performance on cheap talk discovery, IR stands for Intermediate Reward. See Appendix A.11 for the same plot without smoothing

Figure 5.3 shows the performances of the baselines mentioned. As we can see from the figure, the variants with intermediate reward reach the functional phone booth much faster than their counterparts without the reward as they have no motivation to use the phone booth at all. Particularly, OBL with intermediate reward is able to reach functional phone booth the quickest. This shows promises to our more flexible proposed method of mutual information reward. One interesting observation is the difference between IQL and OBL when using the intermediate reward (IQL + IR, OBL + IR). The OBL variant is able to reach the optimal discovery behaviour much quicker than IQL. We hypothesize this is due to the fact that the trajectory sampling based on a random belief in OBL offers better exploration properties than an ϵ-greedy policy in IQL. We leave the investigation of this additional benefit of OBL for future work.

5.2.1.2 Time step to cheap talk channel

After the sanity check, we are ready to examine how our proposed methods perform in cheap talk discovery. Specifically, we examine how variants of our proposed methods based on mutual information maximization (i.e., mutual information reward and mutual information loss) compare with the baselines.

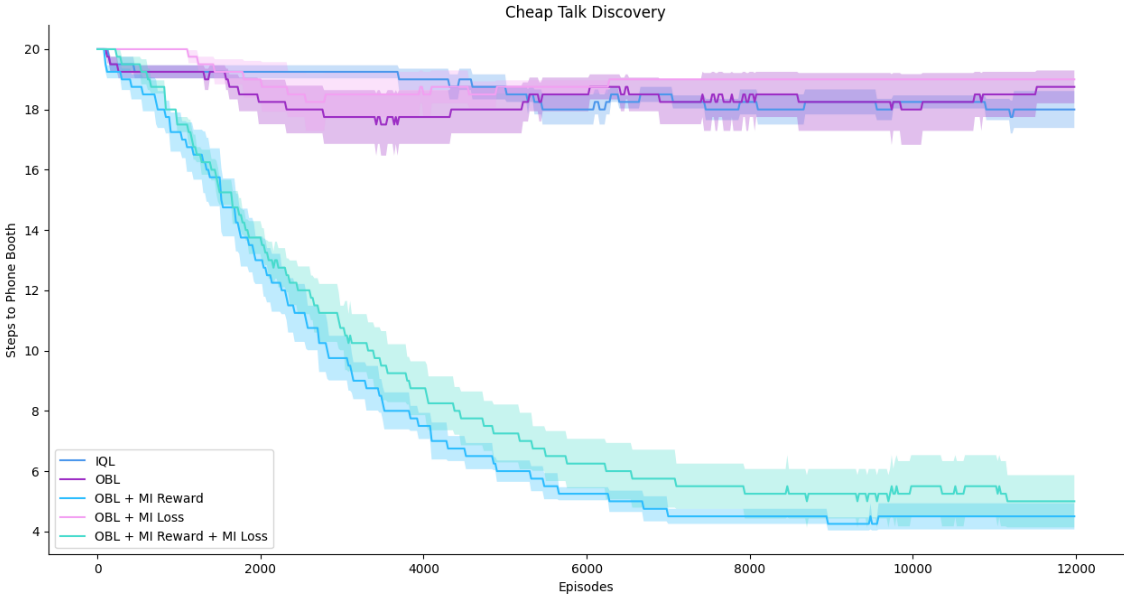


Figure 5.4: Baselines' and our proposed approaches' performance on cheap talk discovery, MI stands for Mutual Information. See Appendix A.11 for the same plot without smoothing

Figure 5.4 shows the performances of the baselines and our proposed methods. From the figure, we can see that our proposed methods (OBL + MI Reward, OBL + MI Reward + MI Loss) which use mutual information reward are able to discover the functional phone booth quickly while the baselines are not able to. It does appear to be the case that the addition of the mutual information loss slightly slows down the learning process with a bit higher variance. What's more, using only mutual information loss fails to discover the functional phone booth. We hypothesize that without the mutual information reward, OBL + MI Loss simply does not generate enough data for the agent to learn to reach the functional phone booth. After all, our proposed approach (i.e. OBL + MI Reward + MI Loss) is able to discover the functional phone booth with ease.

5.2.1.3 Policies at cheap talk channel

With our proposed methods based on mutual information maximization being able to achieve the discovery of communication channels, we would like to look at their respective policies to examine whether they can achieve the kind of optimal grounded policy we prefer. To reiterate, with cheap talk discovery, we are looking to propose methods that can uniformly prefer communicative actions over the rest of the actions when using a functional communication channel. We specifically look at their policies when both the sender and receiver are in the functional phone booth after training.

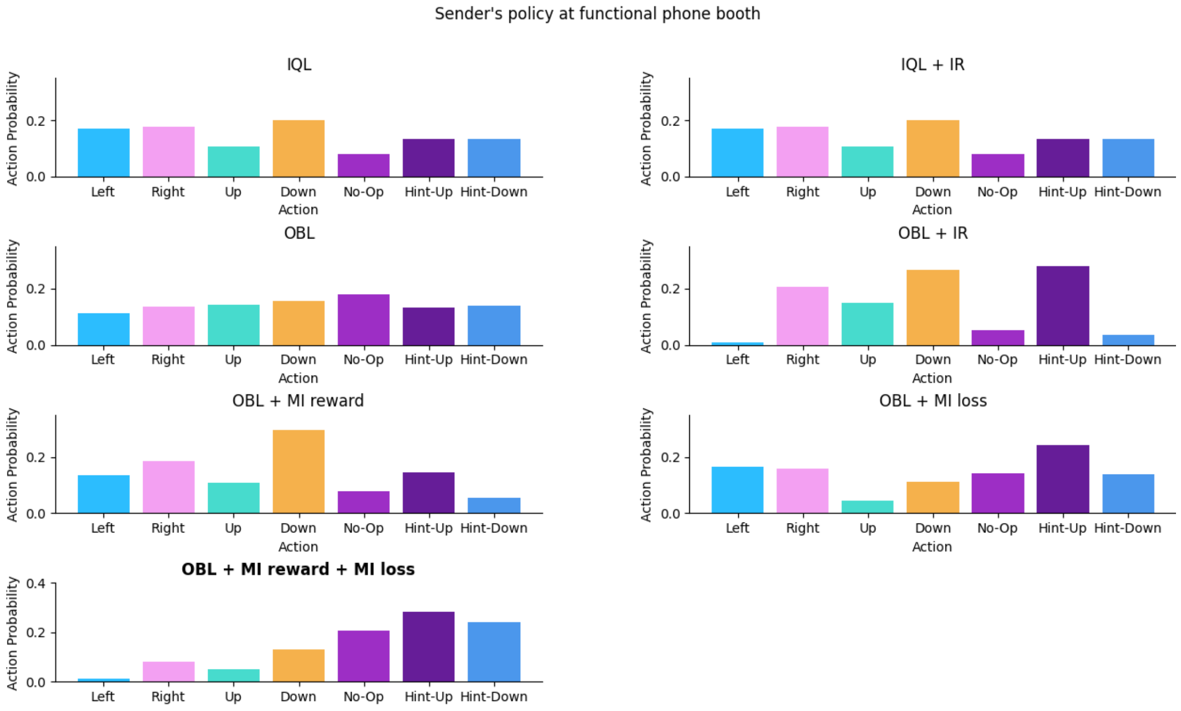


Figure 5.5: Different algorithms' sender policy when both sender and the receiver are at the functional phone booth. This shows our proposed methods in learning a more grounded policy, preferring communicative actions over environmental actions that keep the sender in the booth

Figure 5.5 shows different algorithms' sender policy when both agents are at the booth. To begin with, pure OBL almost learns a uniform policy across all actions, which is expected and supports the correctness of our implementation of OBL. More importantly, as discussed in section 4.2.2.2, reward shaping itself, no matter whether it is the hard-coded intermediate reward or our proposed mutual information reward, does not lead to an optimal grounded policy over communicative actions. This can be observed in IQL + IR, OBL + IR and OBL + MI Reward. Even though they are able to discover the cheap talk channels (especially the latter two) as demonstrated in the last section, they do not learn an optimal grounded policy over communicative actions. We can see that these policies end up learning to prefer actions that keep the sender in the booth, including *UP*, *DOWN*, *RIGHT* and *NO-OP*. On the other hand, we can see that our use of mutual information in OBL + MI Loss and OBL + MI reward + MI loss leads to policies that are more optimally grounded over communicative actions. Specifically, with OBL + MI reward + MI loss, it learns a policy that prefers all communicative actions over all environment actions, demonstrating the effectiveness of our proposed mutual information loss.

We note that given limited time and computing resources, these results are not optimally tuned and we expect even more significant differences if we could train the agents for a longer time, average the results over more random seeds and properly tune the mutual information loss factor κ.

5.2.1.4 Can the agent discover the best cheap talk channel?

To further examine our proposed method, we ask the question of whether the agent can discover the best cheap talk channel when there are multiple channels in the environment. In our Phone Booth Maze environment, this means there are multiple functional phone booths with varying properties. In this case, the best cheap talk channel would be the one that costs the least to use and the most reliable (i.e., less noisy, less dropping of messages). The instance of the Phone Booth Maze environment we use here has 3 functional phone booths with the following properties:

- **lengths**: 5 and 3 for sender and receiver respectively
- **starting points**: $[3, 1]$ and $[1, 1]$ for sender and receiver respectively
- **correct reward**: 1.0
- **wrong reward**: -0.5
- **episode limit**: 40
- **booth types**: 3 functional phone booths. The first one has a cost of -0.4 and 0 noise factor. The second one has 0 cost and 0 noise factor. The third one has 0 cost and $x \in [0.1, 0.3, 0.5]$ as noise factor across 3 sets of experiments
- **booth locations**: $[4, 1], [0, 1], [3, 0]$
- **number of decoy booths**: 0

- **booth reinitialization**: False

Figure 5.6 illustrates what the instance of this environment would look like. The golden phone booth corresponds to the costly phone booth while the one with the dotted red connection cable is the one that has a nonzero noise factor. To investigate whether our method chooses the best cheap talk channel, we measure the number of booth visits for each booth during a test episode.

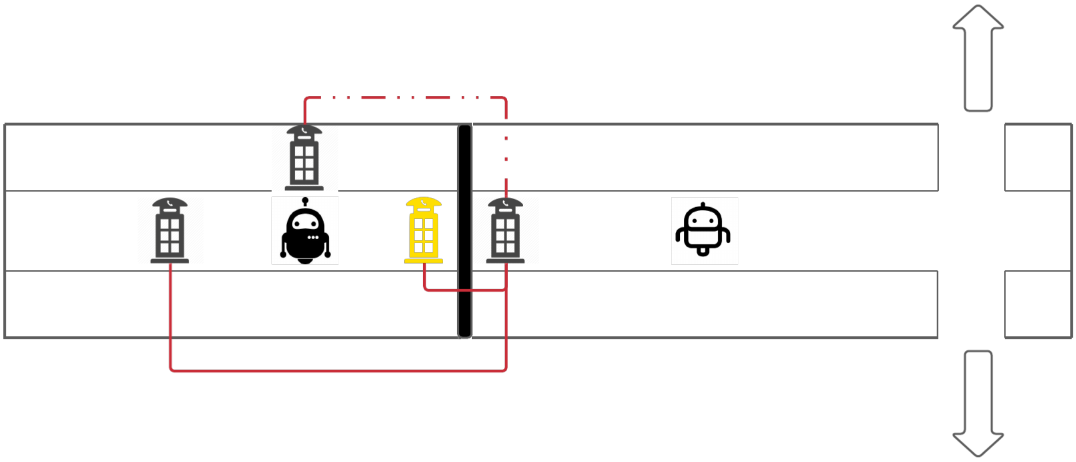


Figure 5.6: Visual illustration of an instance of the Phone Booth Maze environment with multiple functional phone booths used in experiments in this section.

Figures 5.7, 5.8 and 5.9 showcase booth visits of different booth types across different algorithms and 3 different noise factors for the noisy booth. The optimal behavior is to visit the Perfect Booth the most. To begin with, it is obvious that the Costly Booth is easily avoided by all methods given its cost to use. What is more, We can see that OBL barely has any booth visits regardless of noise levels, which is expected given the lack of motivation. We also see that OBL + IR is consistently visiting the Noisy Booth. This is also within our expectation as the intermediate reward does not offer any distinction between the Perfect Booth and the Noisy Booth, so the model would latch onto a closer booth very quickly.

Unlike the previously discussed methods, the mutual information reward does offer an interpretation of noise or information capacity because a channel with a higher noise factor would necessarily mean a lower expected mutual information, as we have already shown in figure 5.2. Therefore, we can see that OBL + MI Reward has no issue visiting the Perfect Booth until the noise factor becomes too low (i.e. $x = 0.1$) for it to distinguish between the Perfect Booth and the Noisy Booth. We hypothesize longer training time and better hyperparameter tuning would allow the

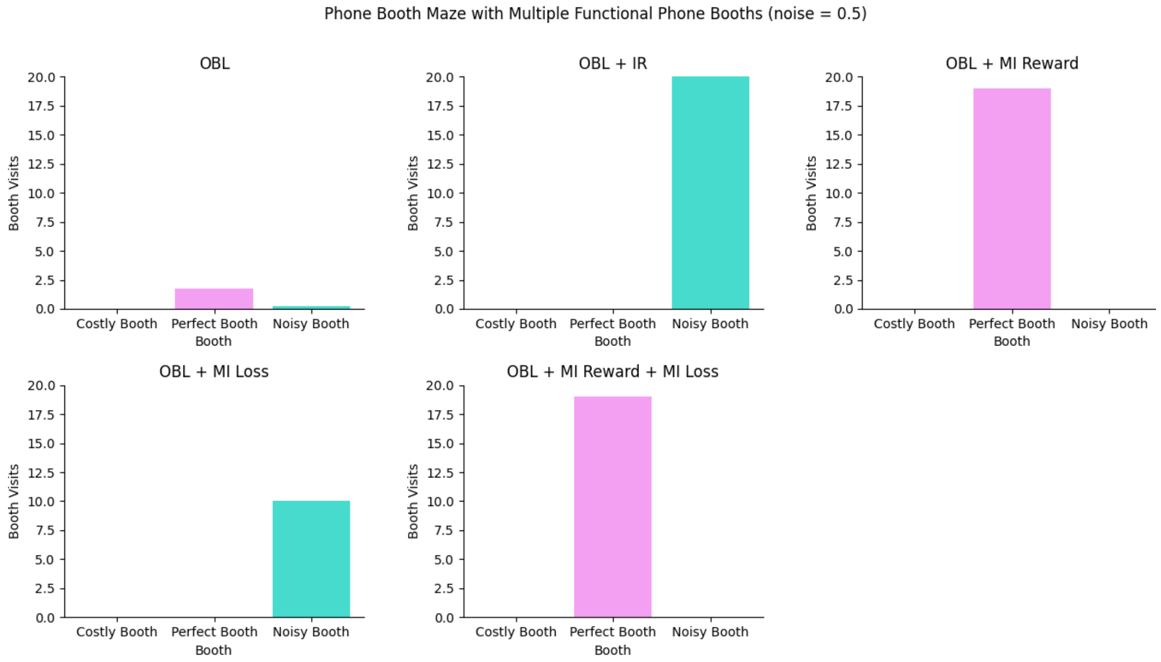


Figure 5.7: Bar plot of booth visits of each phone booth for different algorithms. The noise factor for the noisy booth is 0.5.

agent to avoid Noisy Booths with low noise factors. This provides strong support to how our proposed mutual information reward can aid agents in choosing between cheap talk channels, going for the best one, potentially offering a solution to some real-life applications mentioned in section 1.1.

However, it does appear that the addition of the mutual information loss makes it difficult for the agent to distinguish between the Perfect Booth and the Noisy Booth as the noise factor goes down, especially when $x \in [0.1, 0.3]$. The poor performance in OBL + MI Loss is expected as it does not appear to have enough motivation to generate enough booth visitation data to learn to choose the correct one. Yet, the relatively poor performance of OBL + MI Reward + MI Loss is a bit out of our expectations, especially the mutual information loss's adverse effect on the performance as the noise factor decreases. We do not have a conclusive answer for this phenomenon at the time of writing. We hypothesize that longer training or better hyperparameter tuning could resolve this issue, or having the mutual information loss simply leads to the agent converging to the noisy booth quickly before having sufficient exploration. We aim to concretely resolve this question in our continuing work post-submission.

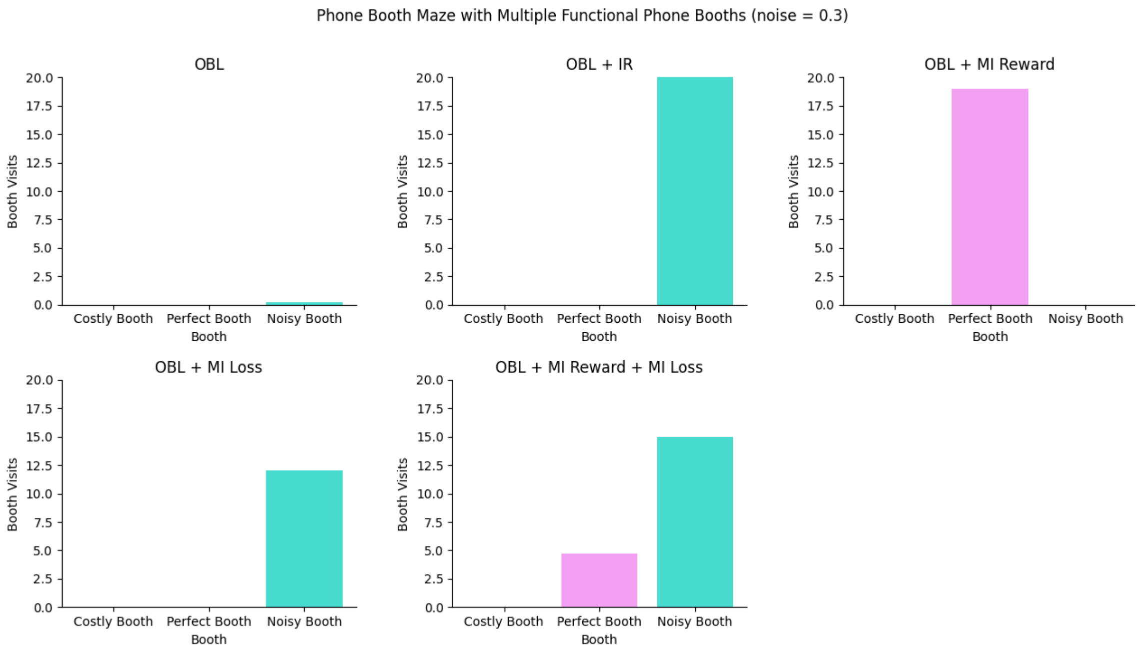


Figure 5.8: Bar plot of booth visits of each phone booth for different algorithms. The noise factor for the noisy booth is 0.3.

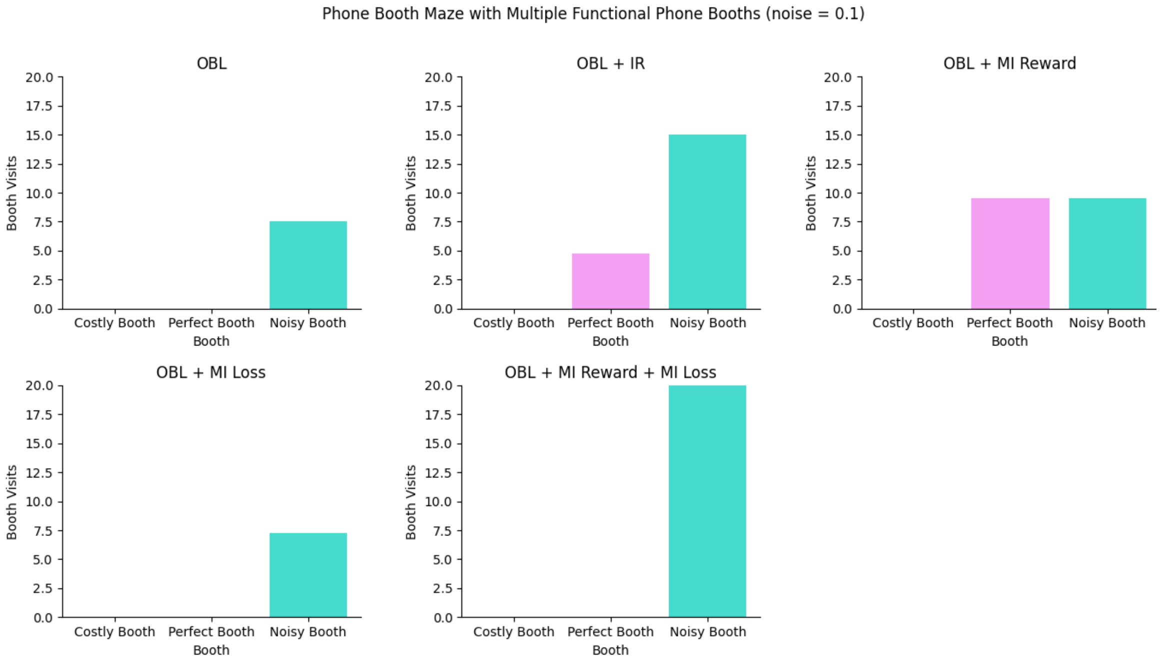


Figure 5.9: Bar plot of booth visits of each phone booth for different algorithms. The noise factor for the noisy booth is 0.1.

5.2.2 Cheap Talk Utilization

In this section, we will look at different algorithms' performance in solving the full task of the Phone Booth Maze environment. Specifically, we would like to look at how our novel problem formulation with our proposed method fair with the baselines as all baselines we use are not effective on this problem. To assess cheap talk utilization, we will be using the running task reward of test episodes.

The settings of the environment in this experiment are mostly the same as in section 5.2.1, except a slight modification to the architecture and action space. Precisely, instead of having two communicative actions, *HINT UP* and *HINT DOWN*, we only use one communicative action - *SEND*, with a separate head that produces a message similar to the architecture in [11]. In this case, when an agent takes the *SEND* action, the message head would produce a message to be sent to the receiving agent if both of them are in connected phone booths. These changes are introduced because we find that having messages acting as q-values at the same time detrimentally affects the learning of a protocol. In other words, having the outputs for the *HINT UP* and *HINT DOWN* actions acting as q-values for OBL and messages for DIAL experimentally appears to affect the learning of a protocol. Therefore, we follow the paradigm in previous work in having two heads, one for action-value function and one for messages [11]. We note that the same discovery approach is used in this section, so all conclusions drawn from the last section would still hold.

5.2.3 Performance based on running task reward

Figure 5.10 shows the performance of baselines and our proposed methods in solving the Phone Booth Maze environment. Without a proper protocol between the sender and the receiver to communicate the goal information, the receiver can only guess by taking one of the exits randomly. Hence, in this setting, the methods that learn to take an exit randomly should achieve an expected reward of $0.25 = 1.0 \times 0.5 + (-0.5) \times 0.5$. As we can see, both IQL and OBL achieve the expected reward of guessing randomly. By combining what we see in this figure and figure 5.4, we can infer that the senders in these methods do not learn to go to the functional phone booth to send messages, so no protocols can be formed between the sender and the receiver to communicate. This leads to the receiver only learning to exit randomly.

On the other hand, the two variants of our proposed method are able to obtain significantly higher levels of rewards than randomly guessing. OBL + MI + IQL Util uses IQL during utilization while OBL + MI + DIAL Util uses DIAL during

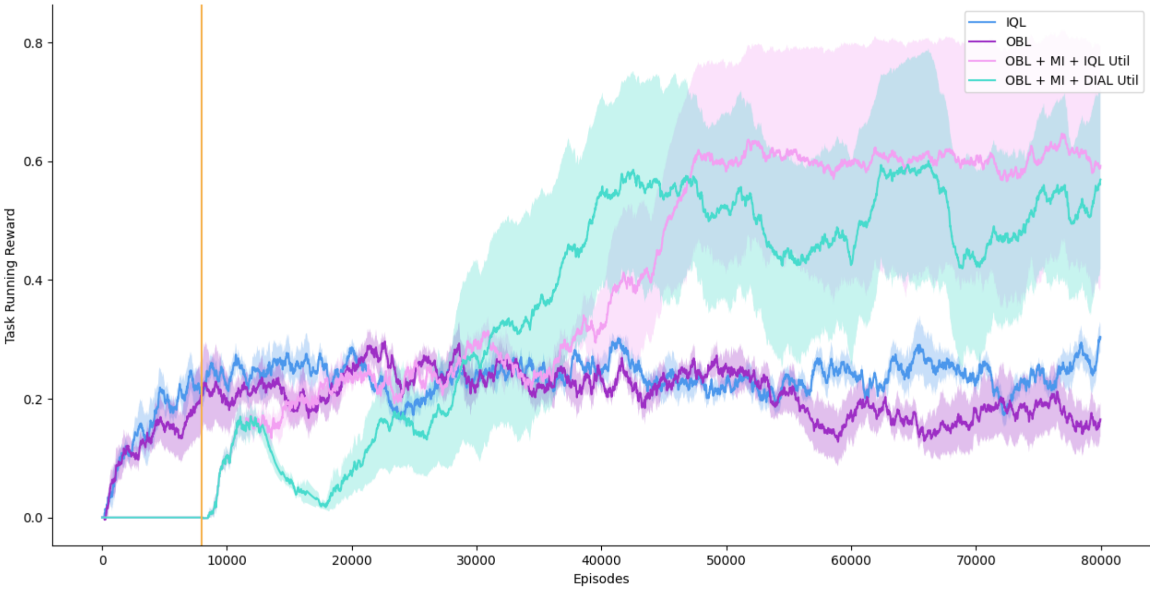


Figure 5.10: Baselines' and our proposed methods' performance on the Phone Booth Maze environment

utilization. Both of them use the same discovery method based on mutual information we propose in the first 8000 episodes as indicated by the yellow vertical line. Qualitatively, we also observe how these methods are able to solve the problem in which the receiver learns to wait at the functional booth until receiving the message from the sender, and then it would go to the the correct exit accordingly based on the protocol learned. What is more, the results show how our novel problem formulation in decomposing this hard communication and exploration problem helps to make the problem easier to solve, together with the effectiveness of solving cheap talk discovery with mutual information maximization.

There are two interesting observations on the results. Firstly, we can see that the method that uses IQL for utilization performs better than the one that uses DIAL for utilization. We hypothesize that IQL could be working better than DIAL because the message space is too small, making it easy for IQL to function well, as pointed out in [11]. We leave the investigation on this for future work to look at how these methods scale with the message space and communication difficulty. Furthermore, noticing the instability and high variance of DIAL despite performing significantly better than the baselines, we believe a more extensive tuning over all its hyperparameters would improve the stability which could not be done at this stage due to time and computing resource constraints.

Chapter 6

Conclusion

6.1 Summary

Along with Deep Learning, Multi-Agent Reinforcement Learning has become one of the frontiers in Artificial Intelligence research. The overarching goal is to develop agents that work and communicate with each other similar to how humans collaborate to accomplish tasks. One key line of research in this area is learning how to communicate given how essential it is in completing tasks efficiently through information exchange. Existing works (e.g. [11] and [63] have shown promising results by equipping agents with communication channels (i.e., cheap talk channels) to send messages to each other at every time step. By exchanging messages, agents can coordinate much more effectively, especially in partially observable settings.

Despite numerous successes, existing work assumes that these communication channels are already known or built-in among agents and can be used anytime they want. In this work, we take a step back to a more realistic point of view and ask the question: *Can an agent learn to discover these communication channels before learning how to use them?* In other words, we take away this common assumption to challenge agents/existing algorithms in finding and using these communication channels, which we coin as the *cheap talk discovery and utilization problem.* To the best of our knowledge, no existing work has addressed this problem before and we find the capability of an agent to locate where to best communicate to be crucial in many real-world settings.

In this thesis, we first formulate the *cheap talk discovery and utilization problem* based on the Dec-POMDP framework. As there is no existing environment to assess algorithms on this problem, we then design and implement a highly configurable environment, called the Phone Booth Maze. The environment requires agents to discover specific areas in the state space where communication can happen and learn a protocol to solve the task. Subsequently, we propose a framework that could potentially solve this problem. For cheap talk discovery, we propose having a mutual information reward and a mutual information loss to maximize the mutual information between an agent's actions and the other agent's observations, on top of the base algorithm OBL, allowing the learning of an optimal grounded policy over communicative actions. By doing so, agents can discover communication channels as they are places where actions can influence another agent's observation. Then, we propose using a modified version of DIAL for cheap talk utilization to learn how to use the discovered channels (i.e., forming a protocol among agents).

Experimentally, using our custom environment, we evaluated our proposed method and various baselines in a series of experiments. We show how our mutual information maximization approach leads to agents that can effectively discover communication channels and select the best one among them. The policies learned with the proposed method are also optimally grounded over communicative actions, ideal for cheap talk utilization. Then, we further show how our novel problem formulation and the proposed framework have the potential of solving the full problem in its entirety, in which our agents are able to first discover where to communicate, and then learn a protocol using the discovered channels.

6.2 Future Work

This thesis can be considered as the very first stride of work on this problem. As elaborated in 1.1, we see a lot of utility in this line of work with a lot of potential future research avenues.

The very first future step would be to conduct further investigations on our proposed framework. Specifically, it would be helpful to conduct a thorough investigation and hyperparamter tuning on different parts of the framework. This would aid us in isolating the causes of instability and errors to make the framework more robust. Examples include trying out different ways of updating the replay buffers for DIAL.

With a more robust proposed framework, there remains a lot of unexplored territories and questions within the proposed method itself. Here is a non-exhaustive list:

- Can the framework still function well when the belief model is learned?

- The framework assumes the knowledge of the time when agents are at the channels when performing DIAL. Can we get rid of this assumption using the mutual information reward?
- Are there better approaches than mutual information maximization? Can we use some pseudomeasure or approximation of mutual information that does not require access to the model of the environment?
- As covered early on in Figure 1.1, there are 3 learning stages. During the discovery stage, we are employing reward shaping to discover the channels, so the value function of the environment with reward shaping is learned which is different from the value function of the task reward. Therefore, there might be issues of interference or unlearning as learning happens in the 2nd and 3rd stages. More modular architecture can be explored to retain what has been learned during training with the dynamic triggering of each module during execution.
- Investigations on the mutual information loss anomaly observed in section 5.2.1.4
- Can our framework handle nonstationarity? E.g., phone booths are reinitialized in every episode
- Working under the centralized training and decentralized execution setting, we can explore how the access to extra state information during centralized training could help discover and utilize cheap talk channels
- Benchmarking different cheap talk utilization algorithms by varying factors like channel noise and channel capacity

Regarding environments, there are also many potential extensions. For our Phone Booth Maze environment, there are many scenarios one can explore beyond the current setting. For instance, the environment can be modified so that the sender also has to escape from a door, meaning the receiver would have to send messages to the sender. Another example would be introducing multiple booths in the receiver's room to increase its difficulty. On the code level, it would be ideal to perform proper refactoring on our codebase so that we can open-source a user-friendly software for practitioners to test their algorithms on our environment. On the other hand, it is also worth looking into higher dimensional and more complex environments beyond GirdWorlds, which would be helpful in assessing different algorithms' capability in generalizing to harder environments on this problem.

6.3 Critical Evaluation

This section covers some of the limitations of our work in our methodology and experimental results that would strengthen our discussion. Our proposed method and its results do rely on access to the environment model and a perfect belief model. These can be difficult to obtain in more complex problems, which we leave it as future work to explore these components interactions when they are learned. Additionally, since our method involves reward shaping, it can be sensitive to the overall task reward and would have to be manually scaled accordingly for different tasks. Moreover, not much consideration was paid to the choice of representation (e.g., flattened representation vs channel-based tensor), which could have affected the results too.

In terms of our experimental setup and results, there are things that could not be done due to limited time and computing resources, which would improve the overall quality and robustness of our results and conclusion. To begin with, more runs should be performed over more random seeds to strengthen the findings. Furthermore, a more thorough and systematic hyperparameters tuning process should be done, especially for our proposed method with the introduction of new hyperparameters, so our results can be more robustly comparable. This also applies to our baselines in which aspects like the exploration policy could have been better tuned and tested. More importantly, given the complexity and interconnectedness of our proposed framework, more unit tests should have been created to aid debugging and analysis.

6.4 Relation to Materials Studied in the MSc Program

Several courses I took during the MSc Program have offered a solid and relevant foundation to pursue the topics covered in this thesis. The Computational Game Theory course was very helpful in laying the groundwork and mindset for me to think in terms of the multi-agent setting, despite not directly covering RL or MARL. More importantly, many of the concepts covered in the course have been widely studied and adopted in the MARL setting which has helped significantly when reading MARL papers. Furthermore, the Advanced Topics in Machine Learning and Bayesian Statistical Probabilistic Programming have also offered me a solid probability foundation in machine learning, which is crucial in understanding aspects in MARL like belief modeling. Last but not least, the Computers in Society and Topics in Computational

Biology courses provided me with sufficient opportunities to practice research writing with solid feedback, preparing me to write this thesis in a more professional and research-oriented manner.

6.5 Personal Developments and Challenges

Throughout this project, I have learned a lot and have encountered many challenges. This section aims to summarize these learning experiences and hurdles.

To begin with, before working on this project, I only had experience working on RL projects with one single agent interacting in an environment. Venturing into MARL was indubitably a challenge given all the different formalism and sets of considerations. After some extensive reading in the beginning, with some incredible recommendations on papers and survey papers by my supervisors, I was able to establish a solid understanding and coverage of the field, which I am excited to further pursue as a research focus after this program.

Furthermore, as this project tries to tackle a problem that has not been investigated before, a lot of components have to be developed from scratch including the environment and some of the algorithms. Most of them were coded from the ground up except for a few more foundational libraries like PyTorch [46] and Numpy. This was challenging as it took more time than projects that built on top of a plethora of existing work. Instead of testing our proposed algorithms on an existing dataset right away, we had to start with designing a software that could test algorithms on our problem. It was a challenge and a fruitful learning opportunity for me to better my time management skills, especially in prioritizing tasks and assessing progress. Besides, some of the algorithms like OBL are some of the most recent work in the field and have no off-the-shelf codebase to take reference from. Thus, these components require diligent attempts in reproducing the algorithms solely based on the papers. Luckily, with close guidance from my supervisors, I was able to reproduce most of these algorithms in our custom environment. Implementing research papers was surely not easy and the process has nurtured my skills in dissecting research papers in order to gain full understanding and identify details that could be crucial in reproducing the algorithms.

What's more, the entire research process has provided me with essential lessons that I am certain will be advantageous throughout my career. Towards the beginning and the middle of the project, whenever things are not working, I used to spend hours going through the codebase again and again to try to identify errors. An important lesson I have learned from my supervisors is to always start simple in order to isolate the causes of problems. They were able to come up with settings to experiment with that can eliminate factors of concerns in an iterative manner. Later on, I started to acquire that mindset which allowed me to fix things and identify the source of issues much quicker. I hope to further nurture this skill moving forward.

Last but not least, having able to go through the rigorous process of research with experienced researchers has further consolidated my foundation in conducting scientific research, from asking the right research questions to explaining complex technical ideas in a reader-friendly manner. I am wholeheartedly grateful for my supervisors' time and effort in mentoring me throughout the process.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021.

[3] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. *arXiv preprint arXiv:1906.08226*, 2019.

[4] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 2021.

[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[6] Andy Cahill. *Catastrophic forgetting in reinforcement-learning environments*. PhD thesis, University of Otago, 2011.

[7] Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *arXiv preprint arXiv:1804.03980*, 2018.

[8] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR, 2019.

[9] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. 2019.

[10] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[11] Jakob N Foerster, Yannis M Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*, 2016.

[12] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[14] Jordi Grau-Moya, Felix Leibfried, and Peter Vrancx. Soft q-learning with mutual-information regularization. In *International conference on learning representations*, 2018.

[15] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[16] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[17] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[18] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, pages 473–479, 1997.

[20] Hengyuan Hu, Adam Lerer, Noam Brown, and Jakob Foerster. Learned belief search: Efficiently improving policies in partially observable settings. 2021.

[21] Hengyuan Hu, Adam Lerer, Brandon Cui, Luis Pineda, David Wu, Noam Brown, and Jakob Foerster. Off-belief learning. *arXiv preprint arXiv:2103.04000*, 2021.

[22] Alexander Iversen, Nicholas K Taylor, and Keith E Brown. Classification and verification through the combination of the multi-layer perceptron and auto-association neural networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 1166–1171. IEEE, 2005.

[23] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2019.

[24] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[25] Sanyam Kapoor. Multi-agent reinforcement learning: A report on challenges and approaches. *arXiv preprint arXiv:1807.09427*, 2018.

[26] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[27] Woojun Kim, Myungsik Cho, and Youngchul Sung. Message-dropout: An efficient training method for multi-agent deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6079–6086, 2019.

[28] Woojun Kim, Whiyoung Jung, Myungsik Cho, and Youngchul Sung. A maximum mutual information framework for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.02732*, 2020.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls, and Stephen Clark. Emergence of linguistic communication from referential games with symbolic and pixel input. *arXiv preprint arXiv:1804.03984*, 2018.

[31] Yuanlong Li, Yonggang Wen, Dacheng Tao, and Kyle Guan. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE transactions on cybernetics*, 50(5):2002–2013, 2019.

[32] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[33] Long-Ji Lin. *Reinforcement learning for robots using neural networks.* Carnegie Mellon University, 1992.

[34] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Twenty-sixth AAAI conference on artificial intelligence*, 2012.

[35] Roger McFarlane. A survey of exploration strategies in reinforcement learning. *McGill University*, 2018.

[36] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[39] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[40] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. *arXiv preprint arXiv:1509.08731*, 2015.

[41] Andrew Ng. Cs229 lecture notes. *CS229 Lecture notes*, 1(1):1–3, 2000.

[42] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[43] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

[44] Christopher Olah. Understanding lstm networks. 2015.

[45] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[47] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[48] Emanuele Pesce and Giovanni Montana. Improving coordination in multi-agent deep reinforcement learning through memory-driven communication. *arXiv preprint arXiv:1901.03887*, 2019.

[49] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[50] Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. *arXiv preprint arXiv:1711.09602*, 2017.

[51] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[52] Robert A Rescorla. Pavlovian conditioning: It's not what you think it is. *American psychologist*, 43(3):151, 1988.

[53] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[54] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[55] Christian Schroeder de Witt, Jakob Foerster, Gregory Farquhar, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. *Advances in Neural Information Processing Systems*, 32:9927–9939, 2019.

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

[57] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[58] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[59] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[60] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755*, 2018.

[61] Samuel Sokota, Christian Schroeder de Witt, Maximilian Igl, Luisa Zintgraf, Philip Torr, Shimon Whiteson, and Jakob Foerster. Implicit communication as minimum entropy coupling. *arXiv preprint arXiv:2107.08295*, 2021.

[62] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. *Advances in neural information processing systems*, 26:1772–1780, 2013.

[63] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.

[64] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[65] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

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

[67] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[68] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *31st Conference on Neural Information Processing Systems (NIPS)*, volume 30, pages 1–18, 2017.

[69] Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. *arXiv preprint arXiv:1910.05512*, 2019.

[70] Xin Wang, Yi Qin, Yi Wang, Sheng Xiang, and Haizhou Chen. Reltanh: An activation function with vanishing gradient resistance for sae-based dnns and its application to rotating machinery fault diagnosis. *Neurocomputing*, 363:88–98, 2019.

[71] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[72] Erik Zawadzki, Asher Lipson, and Kevin Leyton-Brown. Empirically evaluating multiagent learning algorithms. *arXiv preprint arXiv:1401.8074*, 2014.

[73] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendix A

Appendix

A.1 Deep Neural Networks

A.1.1 Perceptron

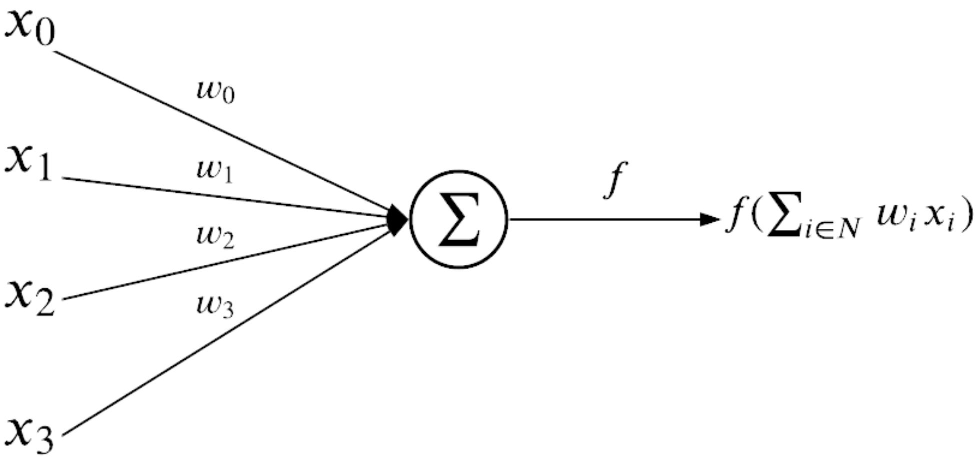


Figure A.1: A diagram of a perceptron

The basic building block of a neural network is a perceptron. It was first proposed by [53], modeled after biological neurons in animal brains. As shown in Figure A.1, it has multiple inputs and one output. The output of a perceptron is the linear combination of the inputs weighted by the weights with some nonlinear function applied to it:

$$o = f(\sum_{i \in N} w_i x_i) \tag{A.1}$$

where:

x_i: inputs

w_i: corresponding weight to each input

f: non-linear function

o: output

We note that a bias term b is commonly added to the linear combination sum in the machine learning literature as follows:

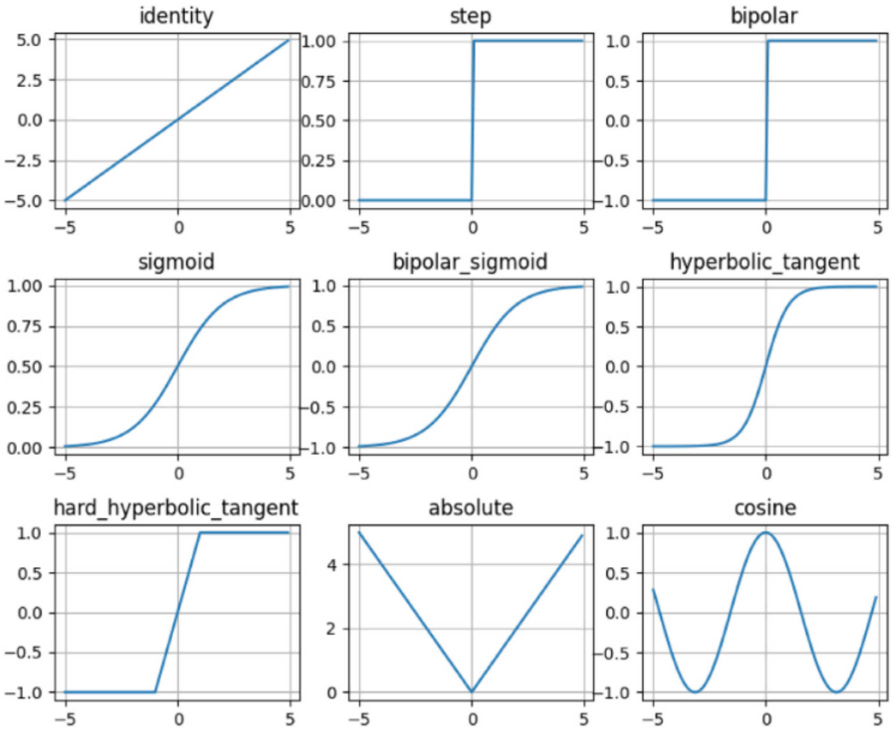


Figure A.2: Examples of activation functions taken from [4]

$$o = f(\sum_{i \in N} w_i x_i + b) \tag{A.2}$$

Here, we include the bias term as part of the weight by adding it as part of the input vector with its input being 1 always. The non-linearity f corresponds to the activation function. The common activation functions include sigmoid, tanh, and rectified linear unit (ReLU). These activation functions are selected by the network designers and are often application-specific. The introduction of such non-linearity is

arguably a major key to the power of neural networks in approximating complex and non-linear functions. Each of them has different properties. For instance, ReLU is known to be more resistant to the problem of vanishing gradients [70]. Some common activation functions are illustrated in figure A.2.
One perceptron may appear simple. But a perceptron can already be used as a linear classifier for binary classification. Assuming that we have a dataset:

$$D = (x_i, y_i)_i \tag{A.3}$$

where:

x_i: an input vector of dimension $\mathbb{R}^N$

y_i: class to which x_i belongs to, between 0, 1

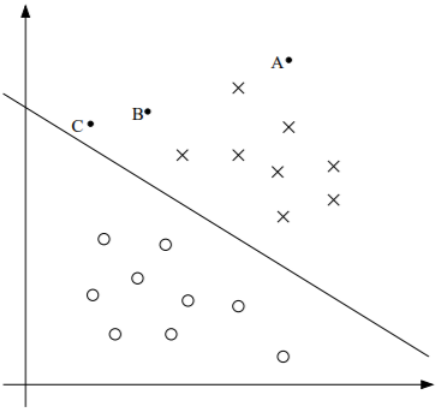


Figure A.3: Example of a perceptron being a classifier, taken from [41]

The probability of an input vector x_i belong to class 1 can then be written as:

$$P(y = 1|x, w) = f(\sum_{i \in N} w_i x_i) \tag{A.4}$$

This can solve classification problems with samples that are linearly separable. An example of this is given in Figure A.3, where we can see a linear decision boundary.

A.1.2 Multilayer perceptron (MLP)

Arranging perceptrons together into different layers gives us a multilayer perceptron (MLP). See figure for an illustration. By stacking them into layers, the input of an MLP is the input to the first layer of perceptrons, while the inputs to the next layer would be outputs of the first layer, and so on. Therefore, outputs of each layer are computed sequentially starting from the first layer, a process commonly known as feedforwarding, until the final layer is reached. The output of each layer is naturally a vector given that it is the output of every perceptron in the previous layer.

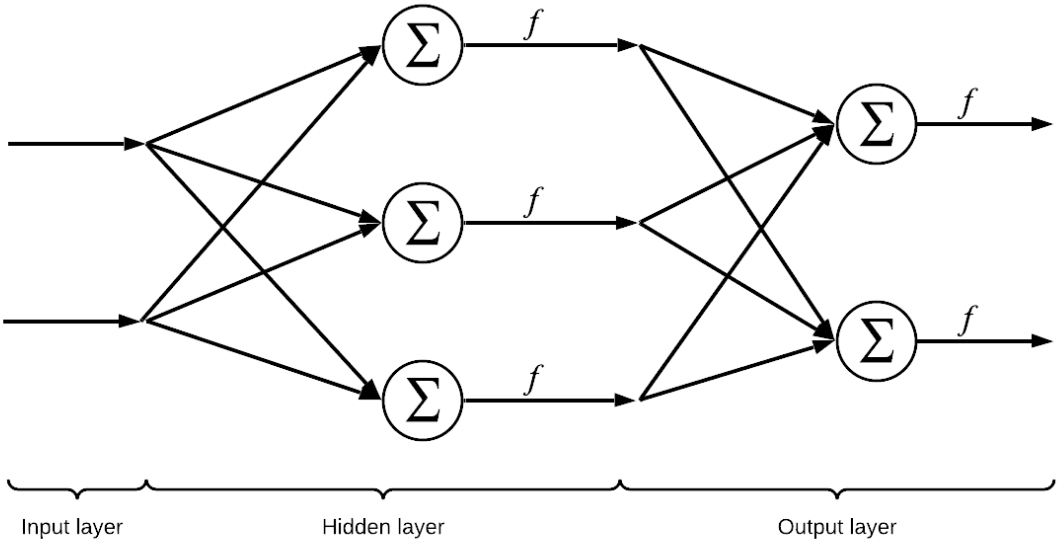


Figure A.4: Diagram of a multilayer perceptron

For illustration, considering the i-th layer of an MLP, we can put all the weights of the perceptrons in that layer as rows in a matrix, denoted as $\boldsymbol{W_i}$. Then, based on section A.1.1, the output of the i-th layer can be expressed as:

$$\boldsymbol{o} = f(\boldsymbol{W_i o_{i-1}}) \tag{A.5}$$

where $\boldsymbol{o_{i-1}}$ corresponds to the output of the last layer or the input to the MLP if it is the first layer (i.e. i = 1). Note that we use bold symbols to indicate the use of vectors or matrices.

MLP can be adapted to most, if not all, machine learning tasks like classification, regression, and reinforcement learning. Differences would be the targets y_i one tries to predict and the nonlinearity functions used, especially in the final layer. For instance, softmax is often used when the outputs of the final layer need to sum to 1 to have a probability interpretation. on the other hand, tanh is often used for training reinforcement learning agents with a continuous action space which limits the outputs to be in the interval of [-1, 1], corresponding normalized range of actions.

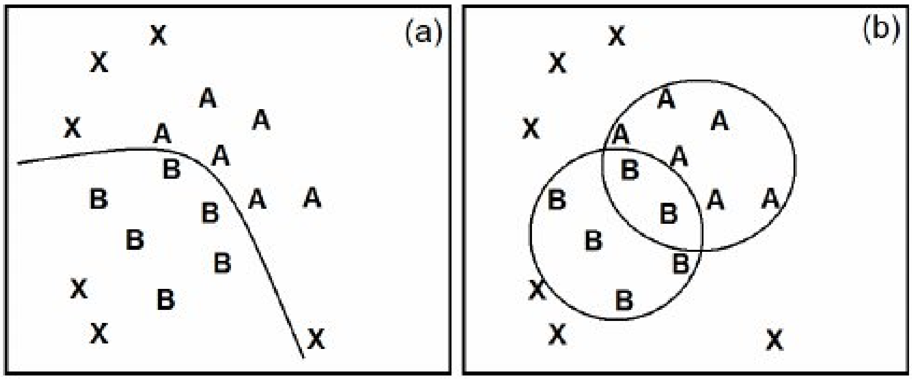


Figure A.5: Example of an MLP being a non-linear classifier, taken from [22]. A and B are class samples while X denotes non-class samples

This arrangement of perceptrons has led to the necessary capacity to learn complex and nonlinear functions Figure A.5 shows how an MLP can learn a non-linear decision boundary. The success of MLP serves as a foundation to field-changing innovations like convolutional neural networks and recurrent neural networks that are crucial to most AI applications nowadays.

A.1.3 Training a neural network

To train a neural network, for the sake of simplicity, a fully connected neural network/ an MLP, we can apply the same procedure irrespective of the task. Assuming we have a regression dataset in the same form as equation A.3, we need a cost function to minimize in order to improve the model's prediction. For a simple illustration, here we use the squared loss:

$$L = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2 \tag{A.6}$$

where:

L: loss function

$\hat{y}_i$: output of the neural network for the i-th data sample

Gradient-based optimization methods are often used to minimize a loss function like the one above. One of the most widely used approaches is stochastic gradient descent (SGD). The idea is to compute the gradient of the loss function with respect to every parameter of the network. Then, the parameters are updated in the direction opposite of the gradient to lower the loss:

$$\boldsymbol{W_i} = \boldsymbol{W_i} - \alpha(\frac{\partial L}{\partial \boldsymbol{W_i}})^T \tag{A.7}$$

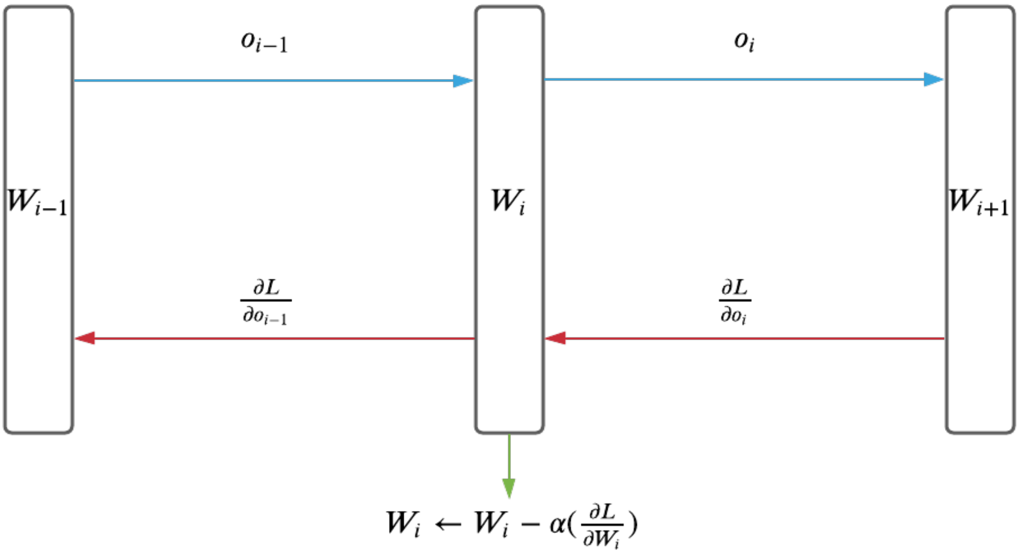

o_{i-1}
o_i
W_{i-1}
W_i
W_{i+1}
$\frac{\partial L}{\partial o_{i-1}}$
$\frac{\partial L}{\partial o_i}$
$W_i \leftarrow W_i - \alpha(\frac{\partial L}{\partial W_i})$

Figure A.6: Backpropagation through one layer. Blue arrows, red arrows, and green arrows are forward pass, backpropagation, and update respectively

where α is the learning rate. For each update, the gradient of each parameter in the network has to be computed using a method called backpropagation.

Backpropagation computes these gradients using the chain rule. For example, using the output of layer i in equation A.5, we can express the gradient with respect to the weight - $\frac{\partial L}{\partial \boldsymbol{W_i}}$ as:

$$\frac{\partial L}{\partial \boldsymbol{W_i}} = \frac{\partial L}{\partial \boldsymbol{o_i}} \frac{\partial o_i}{\partial \boldsymbol{W_i}} = \frac{\partial L}{\partial \boldsymbol{o_i}} f'(\boldsymbol{W_i o_{i-1}}) o_{i-1}^T \tag{A.8}$$

To further compute gradients of the layer before, we would need $\frac{\partial L}{\partial \boldsymbol{o_{i-1}}}$ that can be expressed as:

$$\frac{\partial L}{\partial \boldsymbol{o_{i-1}}} = \frac{\partial L}{\partial \boldsymbol{o_i}} \frac{\partial o_i}{\partial \boldsymbol{o_{i-1}}} = \frac{\partial L}{\partial \boldsymbol{o_i}} f'(\boldsymbol{W_i o_{i-1}}) \boldsymbol{(W_i)^T} \tag{A.9}$$

For this to work, we need the gradient $\frac{\partial L}{\partial \boldsymbol{o_i}}$ for the final layer. Then, for a data sample i we have:

$$L_i = \frac{1}{2}(y_i - o_F)^2 \tag{A.10}$$

where:

F: number of layers

o_F: output of the final layer

Immediately we have:

$$\frac{\partial L}{\partial \boldsymbol{o_F}} = o_F - y_i \tag{A.11}$$

Using $\frac{\partial L}{\partial \boldsymbol{o_i}} = \frac{\partial L}{\partial \boldsymbol{o_F}}$, both $\frac{\partial L}{\partial \boldsymbol{W_i}}$ and $\frac{\partial L}{\partial \boldsymbol{o_i}}$ can be computed. This can be repeated from the last layer to the very first layer by passing the corresponding gradient terms backward. Figure A.6 shows how backpropagation is performed for one layer in terms of how the gradients are passed. Modern deep learning libraries like Tensorflow [1] and PyTorch [46] have automated this process so practitioners no longer have to worry about these computations.

By combining all these steps, the backpropagation can be put into the form of an algorithm as follows:

Algorithm 4: Backpropagation pseudocode

```
Compute $\frac{\partial L}{\partial \boldsymbol{o_F}}$
for $i = F, 1$ do
    $\frac{\partial L}{\partial \boldsymbol{W_i}} \leftarrow \frac{\partial L}{\partial \boldsymbol{o_i}} f'(\boldsymbol{W_i o_{i-1}}) o_{i-1}^T$
    $\frac{\partial L}{\partial \boldsymbol{o_{i-1}}} = \frac{\partial L}{\partial \boldsymbol{o_i}} f'(\boldsymbol{W_i o_{i-1}}) \boldsymbol{(W_i)^T}$
    $W_i \leftarrow W_i - \alpha(\frac{\partial L}{\partial W_i})$
```

A.1.4 Recurrent neural network

Recurrent neural network (RNN) is a type of neural network that specializes in processing sequential data. It has shown incredible promise in fields like natural language processing. They are essentially neural networks with loops to allow information of the past to be remembered as a sequence is being processed. This is done by having an additional input, called the hidden state h_t, in addition to the input data, which provides you a memory or context of the recent past. t here corresponds to the time step. An output y_t is also produced in every time step. A simple way to visualize it is with the application of machine translation. An input of word is given in every time step which the model spits out a translated word while h_t is used to provide context to the model of what has been observed so far. Figure A.7 shows a visual depiction of an RNN with an unrolled version to demonstrate how the self-loop works.

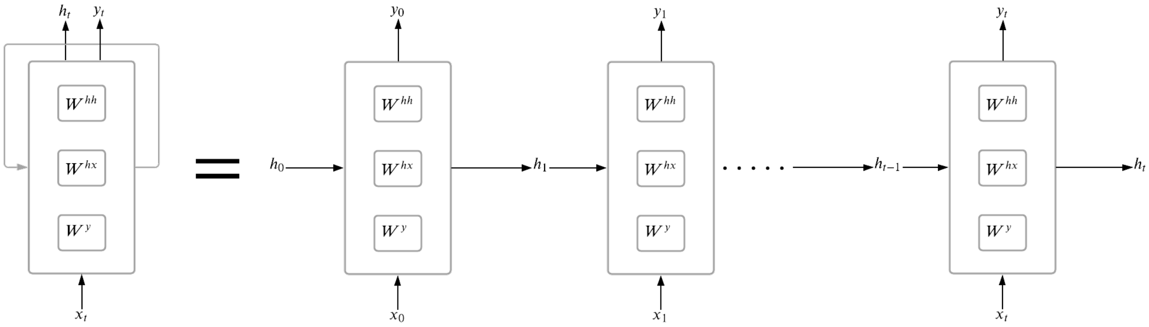


Figure A.7: Visual illustration of an RNN and its unrolled version

$\boldsymbol{W^{hh}}$, $\boldsymbol{W^{hx}}$ and $\boldsymbol{W^{y}}$ are weights that are shared temporally across time steps. They are used to update the hidden state and produce output as follows:

$$h_t = f(\boldsymbol{W^{hh}} h_{t-1} + \boldsymbol{W^{hx}} x_t) \tag{A.12}$$

$$y_t = softmax(\boldsymbol{W^{y}} h_t) \tag{A.13}$$

To learn an RNN, a variant of backpropagation is used, known as backpropagation through time (BPTT). The approach is essentially the same as backpropagation, except the gradients are flowing through the unrolled version of the RNN based on a loss function defined on the outputs.

In addition to being able to take historical information into account, such structure allows RNN to process data of arbitrarily any length with the network size not increasing with the size of the input. However, a vanilla RNN does suffer certain issues. The most prominent ones are exploding gradients and vanishing gradients. This happens due to the difficulty in capturing long-term dependencies as the multiplicative gradients can be exponentially increasing or decreasing with respect to the number of layers or the number of time steps in the rollout. A variant of RNN, named Long Short Term Memory networks (LSTM) was proposed to tackle this issue [19]. Given its robustness and is widely used in the context of RL too, LSTM is the choice of RNN for this work. The key to LSTM is the introduction of a cell state which corresponds to longer term memory that updates more slowly while the hidden state serves as shorter term memory or working memory. The cell state is updated by 3 functions called gates, namely, the forget gate, the input gate, and the output gate. They determine what to forget, what to include, and what to output, respectively. Without going much into the details, each iteration performs these operations:

$$f_t = \sigma(\boldsymbol{W^f} \cdot [h_{t-1}, x_t]) \tag{A.14}$$

$$i_t = \sigma(\boldsymbol{W^i} \cdot [h_{t-1}, x_t]) \tag{A.15}$$

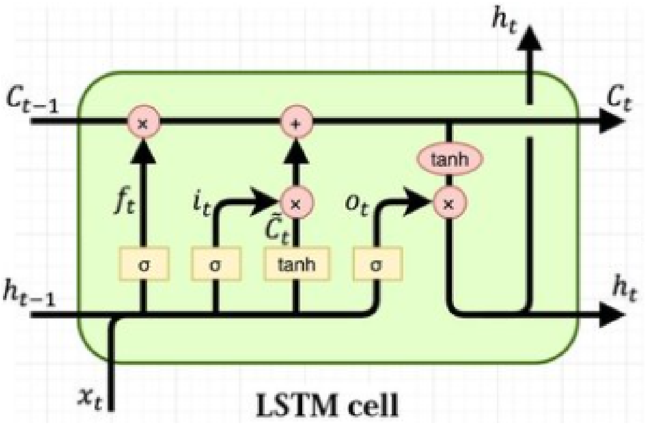


Figure A.8: Graphical illustration of an LSTM cell taken from [44]

$$\hat{C}_t = tanh(\boldsymbol{W^C} \cdot [h_{t-1}, x_t]) \tag{A.16}$$

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{A.17}$$

$$o_t = \sigma(\boldsymbol{W^o}[h_{t-1}, x_t]) \tag{A.18}$$

$$h_t = o_t * tanh(C_t) \tag{A.19}$$

As you can see, each gate has its own weight matrix and the cell state is updated based on the forget and input gate in removing old information and adding new information. Figure A.8 shows a graphical representation of an LSTM network.

A.2 Hyperparameters Setting

This section covers details in setting hyperparameters for various methods used in this work. They are covered in two separate sections depending on whether a parameter is common to all used methods or not.

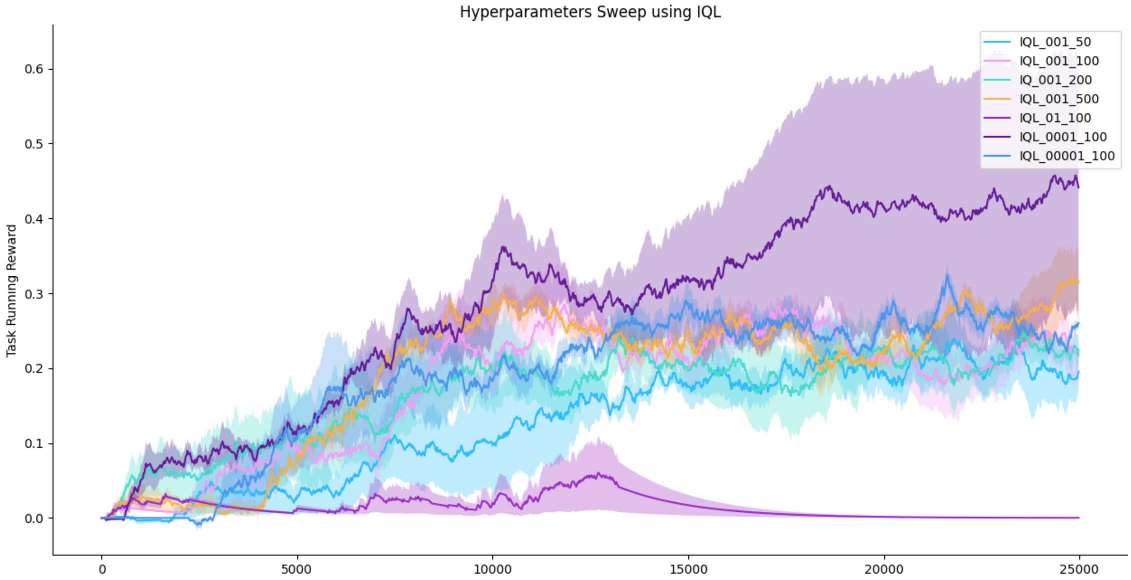


Figure A.9: Hyperparameter sweep results using IQL, the labels on the legend are named based on the "method_decimals_target network update frequency" format

A.2.1 Common parameters

To determine the best parameters that are common to all the methods used, we performed a hyparameter sweep over some key common parameters. Specifically, the search was performed on the learning rate and target network update frequency in the lists of $[0.01, 0.001, 0.0001, 0.00001]$ and $[50, 100, 200, 500]$. The results are averaged over 3 random seeds, each trained for 25000 episodes. Figure A.9 shows the results of the sweep and we can see that the best set of parameters for learning rate and target network update frequency are 0.0001 and 100, respectively. Other common parameters are set to values in table A.2.1.

Hyperparameter	Value
Discount Factor γ	0.99
Batch Size	32
Replay Buffer Size	10000
Temperature	1.0
Non-Linearity	ReLU

A.2.2 Method-specific parameters

Method-specific parameters are set to values in table A.2.2.

	IQL	IQL + Intermediate Reward	OBL	OBL + Mutual Information Maximization
Starting ϵ	1.0	1.0	N/A	N/A
ϵ Decay Step	0.00001	0.00001	N/A	N/A
Minimum ϵ	0.1	0.1	N/A	N/A
Initial Exploration Step	1000	1000	1000	1000
Intermediate Reward	N/A	1.0	N/A	N/A
N-step Reward	N/A	N/A	2	2
α Entropy Factor	N/A	N/A	N/A	0.0
β Mutual Information Reward Factor	N/A	N/A	N/A	2.0
κ Mutual Information Loss Factor	N/A	N/A	N/A	1.0

A.3 Results Figures without Smoothing

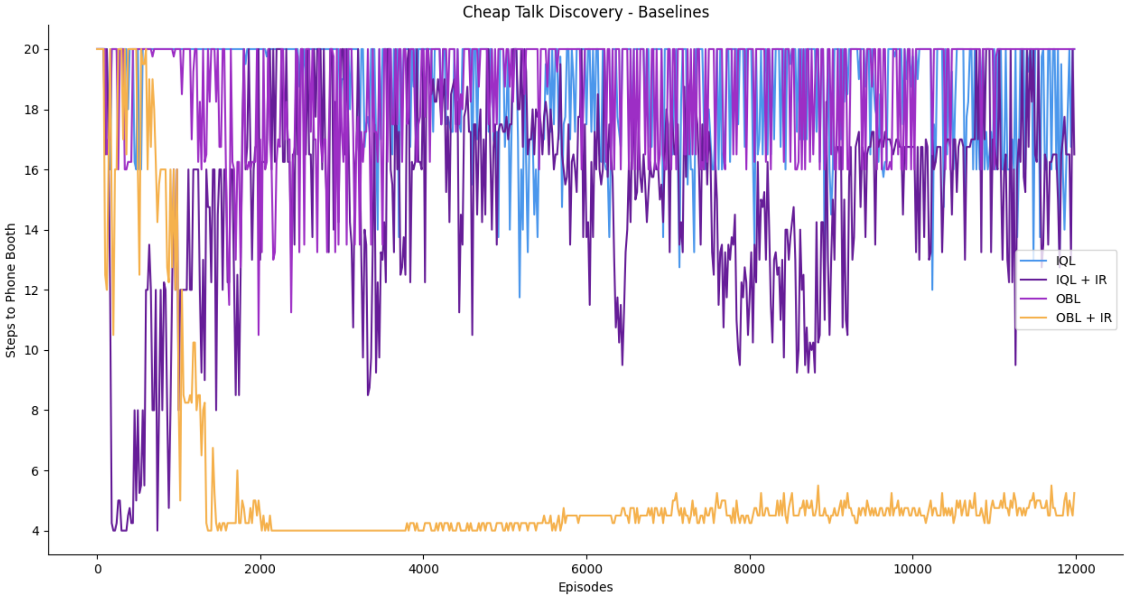


Figure A.10: Baselines' performance on cheap talk discovery without standard errors and smoothing, IR stands for Intermediate Reward. Corresponding section: 5.2.1.1

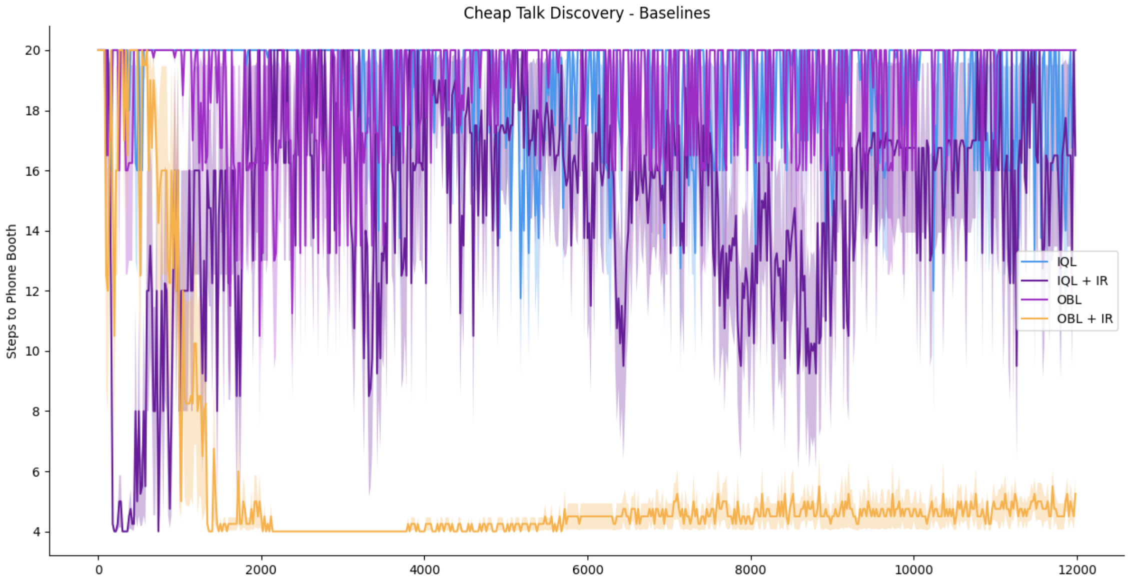


Figure A.11: Baselines' performance on cheap talk discovery with standard errors and without smoothing, IR stands for Intermediate Reward. Corresponding section: 5.2.1.1

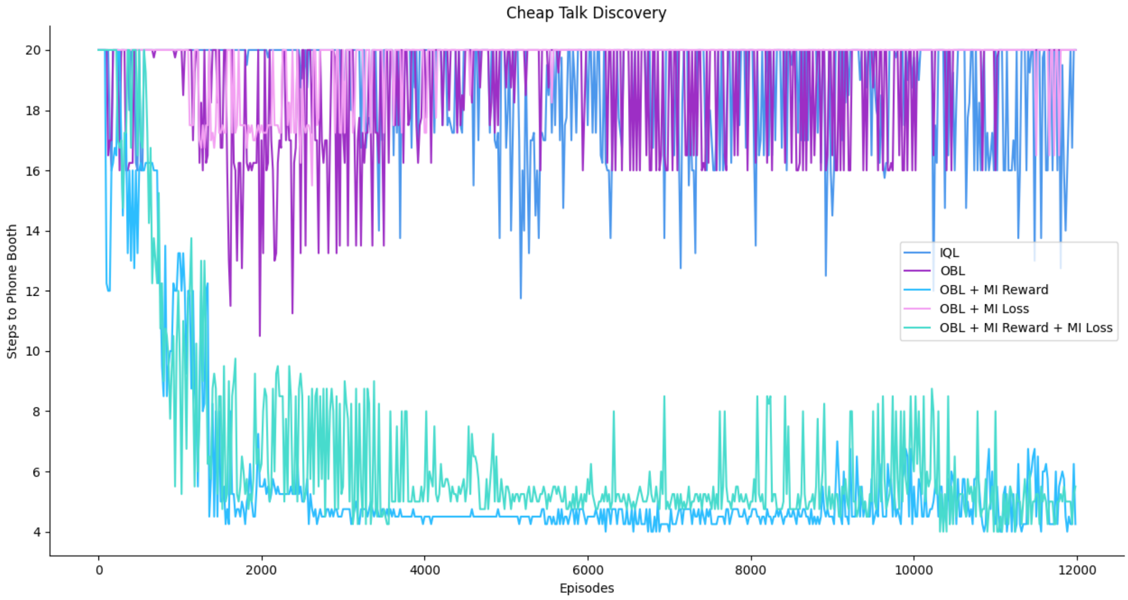


Figure A.12: Baselines' and our proposed approaches' performance on cheap talk discovery without standard errors and smoothing, MI stands for Mutual Information. Corresponding section: 5.2.1.2

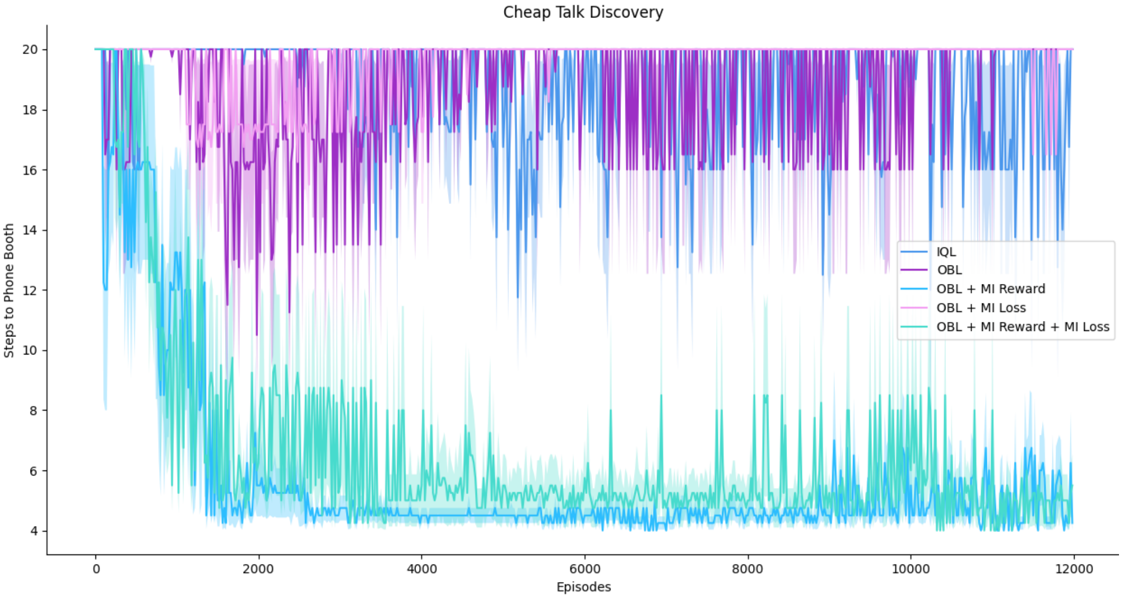


Figure A.13: Baselines' and our proposed approaches' performance on cheap talk discovery with standard errors and without smoothing, MI stands for Mutual Information. Corresponding section: 5.2.1.2