Universiteit Leiden

Master Computer Science

Count-Based Exploration for Multi-Agent Learning

Name:	Jianing Wang
Student ID:	s2434938
Date:	29/07/2021
Specialisation:	Data Science:Computer Science
1st supervisor:	Aske Plaat
2nd supervisor:	Matthias Müller-Brockhausen

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Two-dimensional navigation is a common task in reinforcement learning. To find the target and learn the task, the agent needs to explore the environment thoroughly and gather enough information. In comparison with single-agent tasks, collaborative multi-agent environments contain more complex information that necessitates effective exploration strategies. Inspired by the human's curiosity to explore the novelty, the Count-Based method counts the number of the occurrences of each state-action pair and encourages the agent to probe the less-visited states. In this thesis, we propose two adaptions of the Count-Based method for multi-agent learning, which we call Distributed Multi-Agent Count-Based (Dis-MACB) and Centralized Multi-Agent Count-Based (Cen-MACB). The Dis-MACB strategy focuses on individual exploration, while the Cen-MACB strategy takes into account both the exploration of individuals and cooperation with teammates. In the continuous state and action space environments, both Distributed and Centralized exploration strategies will encounter a problem, where each state-action pair only appears one time. To address this problem, we use a hash function to map similar states and actions to the same hash value. Our results show that with the help of the Distributed and Centralized MACB strategies, the agents only require less than half of the training episodes to master the tasks. In addition, the methods we propose are easy to implement with already existing multi-agent learning algorithms since there are only two main hyper-parameters that influence the performance.

Contents

1 Introduction

Learning can be associated with exploration and exploitation. Exploration refers to gaining new information and focusing on long-term gains. Exploitation utilizes current information to maximize short-term benefits. In reinforcement learning (RL), the trade-off between exploration and exploitation is a critical challenge. To make an optimal decision, the RL agents must thoroughly probe the environment to gather sufficient information. If we want the agents to find the best strategy as fast as possible, the speed and effectiveness of the agents to explore the environment play a decisive role. A certain amount of exploitation can also help to reduce useless exploration by providing the searching direction. In this thesis, we focus on two problems: how to effectively explore an environment, and how to balance the trade-off between exploration and exploitation. We study these two problems in 2-dimensional multi-agent navigation environments, where the agents need to cooperate in order to solve the tasks.

Efficacious exploration is crucial, especially in hard to explore environments. The agents in these environments, with random exploration, barely achieve the tasks and receive learning signals, which is known as the sparse reward problem. One of the well-known methods for this problem uses intrinsic reward as a bonus to encourage the agents to explore, which has been studied a lot in single-agent environments. The intrinsic reward comes from intrinsic motivation [31, 40, 41]: humans are intrinsically motivated to explore something they are curious about. Inspired by this, some studies quantify the uncertainty [33], novelty [30, 46], and information gained [14, 27] as the intrinsic rewards and show that with the intrinsic bonuses, the agents can find optimal solutions faster.

Since using intrinsic reward works well in single-agent problems, we are interested in its performance in the multi-agent domain. Multi-agent reinforcement learning (MARL) has intrigued the interest of many researchers in recent years because many real-world applications are naturally modeled as multi-agent learning problems, such as team sport [20], multi-robot control [47] and autonomous vehicles [37], etc. The agents in cooperative tasks need to explore the environment, just like in single-agent tasks. Moreover, they need to seek possible collaboration with their teammates [7]. For complex multi-agent environments, some studies have proposed different intrinsic reward methods to encourage exploration. Jaques et al. [16] strengthen team coordination and communication by encouraging agents to choose actions with more social impact. Böhmer et al. [4] introduce a centralized agent for exploration, and other agents can share its exploration results. Iqbal et al. [15] use a hierarchical structure to select one of the five intrinsic reward methods to help multi-agent learning. In this thesis, we extend a classic but effective method, Count-Based strategy to address multi-agent exploration problems. This method is inspired by the tendency that humans have, which is being intrinsically interested in novelty. It uses the inverse-square-root state-action-visitation count as the intrinsic reward. The agents receive a higher bonus when visiting the less-visited states which encourages them to explore new states. The idea behind Count-Based is that agents that can visit more different states in a limited time have a higher chance to find an optimal policy.

The idea of the Count-Based method is straightforward, but to apply it in the multi-agent task has the problem of non-stationarity, as well as high-dimensional state and action space problem. We aim to find a suitable method to apply Count-Based strategy for two multi-agent problems: N-Agent Navigation and Communicative Navigation. The multi-agent learning algorithms we use are Distributed Multi-Agent Deep Deterministic Policy Gradient (Dis-MADDPG) [21] and

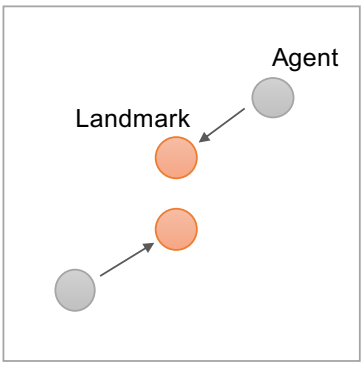
Centralized Multi-Agent Deep Deterministic Policy Gradient (Cen-MADDPG) [23]. For the Distributed MADDPG, each agent only considers its local information for both evaluation and execution. However, in cooperative multi-agent tasks, the reward received by an agent is not only based on the performance of its own policy, but also considers the performance of other agents' policies. At the same time, the policy of other agents is changing during the training process, causing the environment to become non-stationary. A policy that only considers local information cannot explain the change in rewards. For the Centralized MADDPG, the agent takes other agents' observations and actions into account for evaluation while only using local information for execution. This is known as centralized training and decentralized execution, which can solve the non-stationarity problem since each agent knows the information of other agents. Inspired by these two methods, we extend the Count-Based method to work with these two learning algorithms and propose Distributed Multi-Agent Count-Based (Dis-MACB) and Centralized Multi-Agent Count-Based (Cen-MACB). The high-dimensional state and action space causes all state-action pairs to only occur once, which makes it impossible to determine which state is relatively novel based on counting methods. This further leads to high memory storage and large search time issues, because Count-Based method uses a table to store the counts. To solve this problem, we consider using a hash function [46] to map the similar state-action pairs to the same hash code before counting.

The main contribution of this thesis is to adapt the Count-Based strategy to effectively explore the collaborative multi-agent environments. We provide solutions for the problems that happen during the adaption. The non-stationarity problem does not only happen in the learning algorithm, exploration strategies for multi-agent tasks also have this problem. The Centralized MACB strategy solves this problem by taking the state-action pairs of all agents into account, similar to the centralized learning algorithms. In addition, by using centralized exploration, agents can simultaneously explore the environment and different ways of cooperation with their teammates. We solve the high-dimensional state and action space problem using a hash function to map the similar pairs into the same hash value. In addition, we do a lot of empirical evaluation of our methods in two multi-agent navigation tasks. Our results show that both Distributed and Centralized MACBs can reduce the total training episodes that the agents need to master the task by at least half.

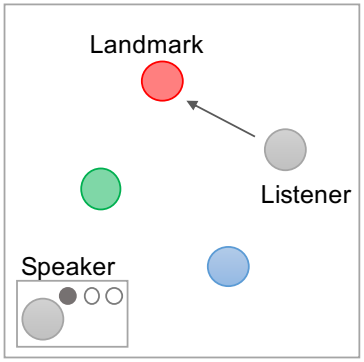
The structure of this thesis is as follows. First, we introduce two multi-agent tasks and explain why the agents in these two environments need an effective exploration strategy in section 2. Next, we summarize some basic exploration strategies and the intrinsic reward methods for both single-agent and multi-agent scenarios in section 3. In section 4, we gradually introduce from RL basic knowledge, to learning algorithms, and then the exploration methods we propose. In section 5, we conclude the measurements we used to evaluate our methods and the plan of our experiments. We empirically show the results of our methods in section 6. Lastly, we give a high level discussion of the exploration in multi-agent tasks in section 7 and a conclude in section 8. The detailed hyper-parameters we choose for experiments and the steps to reproduce this thesis can be found in Appendix A.

2 Problem Statement

In this thesis, we study how to effectively explore two cooperative multi-agent environments, N-Agent Navigation (Figure 1a) and Communicative Navigation (Figure 1b) [28]. Both of the



(a) N-Agent Navigation



(b) Communicative Navigation

Figure 1: We study how to effectively explore these two multi-agent environments. (a) N agents aim to cover N landmarks without collision. (b) One of the landmarks is the target landmark. The speaker utters signals to guide the listener to cover the target landmark.

environments are 2-dimensional with a continuous state and action space.

2.1 N-Agent Navigation

There are N number of agents and N landmarks in this environment. In Figure 1a, N is equal to 2. The agents need to cooperate together in order to cover all the landmarks without collision. If an agent moves to a landmark, the other agent is expected to move to the other landmark. The observation of each agent includes the location of the other agents as well as the position of the landmarks. Based on this information, they decide which direction to move toward. Before learning, they do not know what their task is. They learn about the task once they start receiving rewards under different scenarios. For example, the agents receive a positive reward when all the landmarks are covered and negative otherwise. They move to different positions to gather information (reward) under different states and learn to move to the positions that provide positive rewards. The more information they get, the better decisions they can make. As the agents explore all the scenarios, they can make the optimal decision, meaning that the learning time depends on the effectiveness of exploration. If an agent repeatedly explores negative reward states that have been seen before, it will slow down the learning process. In a sparse reward setting, only by quickly finding the positive reward scenarios, the agent can learn the task faster. One solution to this issue is to reduce the exploration for states that have already been visited.

An environment is harder to explore if it has many different states. In comparison with the 1-Agent navigation task, in the multi-agent task the agent needs to explore more complex scenarios, including N landmarks and $N-1$ teammates. The problem we address in this thesis is how to effectively explore such complex multi-agent environments.

2.2 Communicative Navigation

In the N-Agent Navigation task, the agents can fully observe the environment, which means that each agent can finish the task based solely on their observation (the position of landmarks and teammates). However, some multi-agent environments are partially observable which means that the observation of each agent only holds part of the information. The partially observable environments require more cooperation between the agents in order to complete the task.

Communicative Navigation is a partially observed environment. As shown in the Figure 1b, there are 2 agents (a speaker and a listener) and 3 landmarks in 3 different colors. A specific color landmark is chosen to be the target landmark for each episode. In this task, the listener needs to go to the target landmark. However, only the speaker knows which landmark is the target. The speaker needs to guide the listener to navigate to the target landmark by uttering a communication signal to the listener. The listener decides where to move based on the positions and the colors of the landmarks, and the communication signal. By receiving the signal from the speaker, the listener learns which landmark is the target. For this task, the speaker needs to learn how to utter the correct signal to show which landmark is the target, and the listener needs to understand the signal and move to that target.

Since the speaker and listener have different tasks, what they need to explore is different. The speaker needs to explore how to utter signals to receive positive rewards. The exploration for the listener is more complicated, because one part of its observation comes from the environment and the other part comes from the speaker, which may provide a wrong signal. This leads to a complex and unstable environment for the listener to explore. To address this, we utilize Centralized MADDPG algorithm [23] for learning and propose Centralized MACB strategy for exploration.

3 Related Work

In the introduction, we explained the importance of exploration in the learning process. In this section, we summarize some basic exploration strategies and some more powerful exploration methods that utilize intrinsic rewards, for both single-agent and multi-agent environments. In addition, we show the difference between our method and other existing methods.

3.1 Basic exploration strategies

In some simple RL problems, the basic exploration strategies guarantee finding the optimal decision [54]. The ϵ-greedy method [25, 50] uses a probability of ϵ to randomly select an action for exploration and a probability of $(1-\epsilon)$ to choose an optimal action. Its extension is the decay ϵ-greedy method, which starts with a large probability for ϵ in the beginning of the training process to emphasize exploration, and gradually reduces it as training progresses to emphasize to exploitation. Instead of choosing a random action with a certain probability, the noise-based methods [54] add random noise to action or parameter space directly [10, 34]. The common noise functions for action are Gaussian noise and Ornstein Uhlenbeck (OU) noise [49]. Random exploration is easy to apply, but it is the least efficient strategy [24, 47]. Another method for the multi-armed bandit problem [17] is the Upper Confidence Bound (UCB). To reduce the uncertainty of the environments, it utilizes an upper confidence bound function $U(a)$ to encourage the agent to choose the less performed action. Based on this idea, the UCB1 method [1] uses $U(a) = \sqrt{\frac{2\log t}{n(a)}}$, and $a_t = \arg\max_{a \in A}(r_a + U(a))$ to choose an action a at time t, where $n(a)$ is the number of times this action was selected before t, and r_a is the average reward for taking action a.

3.2 Intrinsic reward strategies

Intrinsic reward strategies are commonly used in hard to explore environments, where the agents barely receive learning signals. The intrinsic reward strategies provide bonus rewards as learning signals to the agents through other criteria and therefore boost the progress of learning. Since most of the hard to explore environments are high-dimensional, most of the intrinsic reward methods use deep neural networks. Burda et al. [6], Stadie et al. [42] and Pathak et al. [33] use the prediction error of different feature spaces to encourage the agents to visit the uncertain parts of the environment. Variational Information Maximizing Exploration (VIME) encourages the agents to visit the states which can minimize the uncertainty of the environment dynamics distribution [14, 27]. Count-based methods such as MBIE [43] and MBIE-EB [44] encourage the agent to discover novel states by using the state and action count. In the continuous state and action space, some extension of this method in order to solve the high-dimensional state and action space problem include [3, 30], where they propose using a density model to generate pseudo-count and Tang et al. [46], where they use a hash function to decrease the dimension. Instead of the hash function, they also propose a learned hash model (an autoencoder [29, 18]) to extract features from the state and reduce the dimensionality. Besides autoencoders, a convolutional neural network which can recognize pattern of high-resolution images to solve classification tasks from Krizhevsky et al. [19] can also be used to extract the features.

As people raise interest in multi-agent problems, intrinsic reward methods that encourage multi-agent exploration have emerged. Böhmer et al. [4] use an extra centralized agent to receive intrinsic rewards for exploration and the other agents share the replay buffer with this agent in order to share the exploration results. The intrinsic reward for the centralized agent is the largest uncertainty over all the agents. The uncertainty is the variance of Bayesian posterior distribution over Q-values [32]. Wang et al. [52] and Jaques et al. [16] introduce the use of social influence as the intrinsic reward. In Jaques et al. [16], the agents are encouraged to select the action which can influence the behavior of the other agents the most. The influence is calculated by how much the selected action can change the distribution over other agents' next actions. In Wang et al. [52], the agent is encouraged to visit the states where that influence the transition distribution of other agents the most. Iqbal et al. [15] use a hierarchical policy where the top-level agent chooses the best among 5 intrinsic reward functions, and the low-level agents follow this bonus to learn. Baker et al. [2] apply count-based strategy in a different problem: hide-and-seek. The hiders can get use of tools like boxes or walls to block the way of the seekers. The seekers can also use ramps to jump across obstacles. They let two teams compete against each other (self-play) to create a self-supervised autocurriculum and that results in emergent cooperation and tool usage. They show that the count-based strategy encourages the movement of the agents and the tool usage, but the performance of the count-based strategy decreases when the state includes more information (high-dimension). This is also the problem we address in this thesis.

3.3 Multi-Agent Environments

The environments in our experiments are N-Agent Navigation and Communicative Navigation environments which are introduced by [28]. Lowe et al. [23] use these environments to evaluate their centralized actor-critic learning algorithm which we also use to apply our exploration strategy on. Ryu et al. [35] also evaluate their exploration strategy on the N-Agent Navigation

environment. They propose an end-to-end encoder-decoder model that generates the initial state from easy to hard which form a curriculum for agents. In comparison to their environment where the position of the landmarks is static, we randomly initialize the landmarks and the agents' positions for each episode, which is harder for agents to learn. We did not find any work that aims at effectively explore the Communicative Navigation environment.

In this thesis, the two exploration strategies we use are noise-based methods and an adaption of the Count-Based method. The noise function in our experiments is a Gaussian action noise strategy. We extend the Count-Based strategy to Distributed and Centralized Multi-Agent Count-Based exploration (Dis-MACB and Cen-MACB). The Centralized MACB strategy is inspired by the centralized learning algorithm, in which all the agents' states and actions are joint together into a new observation for evaluation. With the joint observations, the Centralized strategy encourages the agents to visit the joint observations that have not been visited before which encourage the cooperation. The main problem is that the dimensionality of the joint observations becomes larger as the number of agents increases. To address this, inspired by Tang et al. [46], we use a hash function to decrease the dimension of the joint state-action before counting. There are other models that can reduce the dimensionality, but our environments contain up to 2 agents, and a simple hash function has good results.

4 Methods

Reinforcement learning (RL), a type of machine learning, is a paradigm where the RL agents learn a task by constantly interacting with the environment [45]. At each time step t, the agent decides an action a_t based on the current state s_t. The action is performed on the environment and leads to a new state s_{t+1}. Afterwards, the environment returns a reward r_t based on this new state. In practice, the environment is episodic and each episode consists of a number of time steps from $t = 0$ to T. So each episode can be captured as a sequence of states, actions and rewards $(s_{t=0}, a_{t=0}, r_{t=0}, \ldots, s_{t=T}, a_{t=T}, r_{t=T})$, where $s_{t=0}$ is the initial state.

4.1 Markov Decision Process (MDP)

An RL problem can be modelled as a Markov Decision Process (MDP) [45] using a 5-tuple (S, A, P, R, γ). A finite-MDP is defined by a set of finite states S and actions A. The state captures all the information about the environment. For example, the state of a board game can be an array, where each element uses a specific number to represent a different entity. In Atari games from the OpenAI gym framework [5], the states are represented by pixels. Navigational tasks include the location of the target in the environment as well as the agent's information, such as the velocity and position. Depending on the environment, the action space can either be discrete or continuous. For example, we can use four different numbers to represent moving up, down, left, and right as discrete actions. In robotic environments, the actions are usually continuous, such as the angle of the joints of the robot's hand or leg in MuJoCo environments [48]. In navigation tasks, actions include moving direction and velocity, both of which are continuous.

Choosing an action a_t in state s_t, produces the next state s_{t+1} according to the state transition probability function $P(s_{t+1}|s_t, a_t)$. The important thing about MDPs in RL is that the current state can completely characterise all the history states, which means that agent can only base

its decision on the current state. The reward $r_t = R(s_t, a_t)$ based on how well for choosing action a_t in the state s_t. The goal of the RL problem is to maximise the cumulative reward $R = \sum_{t=0}^{T} \gamma^t r_t$ where $\gamma \in [0, 1]$ is the discount factor. We use γ to control the effect that future rewards have on current decisions. Small values of γ focus on the near-term gains while larger values emphasize far-sighted rewards.

4.2 Markov Games (MGs)

The framework of Markov Games (MGs) [22] is an extension of MDP for multi-agent systems. For an N agent RL problem, MGs are defined by a set of states S for all agents, sets of actions $A_1, ..., A_N$ and sets of observations $O_1, ..., O_N$ for each agents. In comparison with MDP, the state transition function for MGs considers actions from all the agents $P(s_{t+1}|s_t, a_1^t, ..., a_N^t)$. Each agent i receives its own reward $r_i^t = R(s_t, a_i^t)$ which only considers their action a_i. They aim to maximize their own expected reward $R_i = \sum_{t=0}^{T} (\gamma^t r_i^t)$.

N-Agent Navigation
In the N-Agent Navigation problem, each state s_t corresponds to the concatenated observations of all the agents $o_1^t, ..., o_N^t$ at time-step t. For each agent i, their observation o_i includes the position of all the landmarks, teammates' positions, and the velocity and the position of themselves. Based on the observation o_i^t, the agent i needs to decide the best moving direction and velocity as its action a_i^t in order to maximize their own cumulative reward R_i over an episode. Each value in the observations and the actions are in range of -1 to 1. The reward function considers the number of landmarks that are covered and whether the agents have collided with each other. Specifically, the reward function is $r_i^t = R(s_t, a_i^t) = -(n+m)/N$ where n is the number of landmarks that are not covered and m is the amount of collisions happened at time t. The rewards returned by the environment are called extrinsic rewards. In section 4.5, we cover intrinsic rewards from the exploration method.

Communicative Navigation
Similar to the N-Agent Navigation, each state of the Communicative Navigation task is the concatenated observations of all the agents in the environment, but the observations of the speaker and the listener are different. The observation of the speaker is the color of the target landmark and its action is a communication signal. The signal is an array that consists of 3 continuous entries which represent the RGB value of the color. The listener's observation consists of the positions and the colors of the 3 landmarks. The observation of the listener also includes the communication signal from the speaker. The listener is movable and its action is the direction and the velocity. We use two different reward functions for this environment in our experiments. The first reward setting is sparse, meaning that it returns 0 when the listener covers the target landmark and -1 otherwise. The second reward setting uses the negative distance between the listener and the target landmark. The maximum reward is 0 when the listener exactly cover the target landmark. More specifically, the distance between them is measured by Euclidean distance between the position of their centers.

These two environments have no boundaries, which makes learning difficult because there's no limit for the maximum distance between the agents and the landmarks. The further the agents move away from the landmarks, the probability of never reaching them grows exponentially. Therefore, to simplify the learning tasks, we set a time-step horizon for each episode. After limited time-steps, the environment resets, and the position of the agents go back to an acceptable range. The episode doesn't terminate if the agents reach the landmarks. Therefore,

if the agents succeed before the episode termination, it is expected that they remain on the landmarks in order to maximize their rewards.

4.3 Action-Value Function

In the MDP and MGs sections, we showed how to model an RL problem with a 5-tuple (S, A, P, R, γ). In this section, we talk about the learning process using this tuple. Before moving to the multi-agent tasks, we first introduce some popular methods that are used for single-agent learning.

Recall that the goal of the RL problem is to maximize the cumulative reward, which requires the agent to make the best decisions throughout an episode. In practice, the best decision corresponds to the best action at a given time-step. A decision can be decomposed into two parts, evaluating all the actions for a current state and choosing the one with the best result [38].

The agent's behavior is controlled by a policy $\pi(a|s)$, which returns a probability distribution over the actions given a state. The action-value function $Q^\pi(s_t, a_t)$ for policy π is used to evaluate an action a_t on a state s_t at time t:

$$Q^\pi(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t] \tag{1}$$

where $R_t = \sum_{k=0}^{T-t}(\gamma^k r_{t+k})$ is the accumulated reward from time t to T. Since the action is chosen by following the policy π, the RL problem can be seen as finding the optimal policy π^* which maximizes the cumulative expected reward. The action-value function can be rewritten in a recursive form known as the Bellman Equation. The Bellman equation decomposes the action-value function into two parts, the current reward and the discounted future expected return [45]:

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}[R_t|s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{T-t} r_T|s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma(r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T)|s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma R_{t+1}|s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma \mathbb{E}[R_{t+1}|s_{t+1}, a_{t+1}]|s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma \mathbb{E}_{a_{t+1}\sim\pi}[Q^\pi(s_{t+1}, a_{t+1})]|s_t, a_t] \end{aligned} \tag{2}$$

After evaluating each action $a \in A$ at state s with $Q^\pi(s, a)$, we can apply a greedy policy to choose the action with the maximum Q value:

$$\pi(a|s) = \arg\max_{a\in A} Q^\pi(s, a) \tag{3}$$

To conclude, we first use the action-value function $Q^\pi(s, a)$ to do the evaluation, and then use the greedy policy to choose the action with the maximum Q value. At each time step, we choose the greedy action which can lead to the maximum cumulative reward.

4.3.1 Deep Q-network (DQN)

Deep Q-network (DQN) [26] is a well-known algorithm that uses the action-value function. It utilizes a deep convolutional network [11] to approximate the action-value function $Q(\cdot|\theta^Q)$

with parameters θ^Q. They optimize the function by minimizing the loss:

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}}[(Q(s_t,a_t|\theta^Q) - y_t)^2], \text{where } y_t = r_t + \gamma \max_{a_{t+1}} \bar{Q}(s_{t+1},a_{t+1}|\theta^{\bar{Q}}), \quad (4)$$

where y_t is the target value calculated using the Bellman equation. To stabilize the learning process, the DQN method utilizes a target network $\bar{Q}(\cdot|\theta^{\bar{Q}})$ to calculate the target y_t. The target network is a copy of $Q(\cdot|\theta^Q)$ with a delayed update. Another stabilization method that DQN uses is the experience replay. During episode simulations, we store the transitions (s_t, a_t, r_t, s_{t+1}) in the replay buffer. During training, a batch size of transitions is sampled at random and used for optimization. Sampling randomly from the experience replay buffer, the correlation between the samples breaks, which reduces the variance of the updates. In addition, the data can be reused which makes DQN sample efficient.

4.4 Policy Gradient (PG)

Greedy policy is not applicable when facing continuous action spaces. Equation 3 shows that a full-scan of actions is required in order to make a decision, which is impossible when the number of actions is infinite. To address this, we can use a parameterized function instead of a greedy policy to directly choose an action with a method called Policy Gradient (PG).

Policy Gradient directly models the agent's behavior using function $\pi_\theta(a|s)$ with parameters θ^π [45]. We can use $\pi_\theta(a|s)$ to directly predict an action given a state. The goal is to find the best parameters θ^π that maximize the objective function which calculates the average reward per time-step [53, 38]:

$$J(\theta^\pi) = \sum_{s\in S} p^\pi(s) \sum_{a\in A} \pi_\theta(a|s) R_{s,a} \quad (5)$$

where $p^\pi(s)$ is the state distribution for π_θ and $R_{s,a}$ is the instantaneous reward of taking action a in state s. We maximize the objective function $J(\theta^\pi)$ by gradient ascent:

$$\begin{aligned} \bigtriangledown_{\theta^\pi} J(\theta^\pi) &\propto \sum_{s\in S} p^\pi(s) \sum_{a\in A} \bigtriangledown_{\theta^\pi} \pi_\theta(a|s) R_{s,a} \\ &= \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\bigtriangledown_{\theta^\pi} log \pi_\theta(a|s) R_{s,a}] \end{aligned} \quad (6)$$

There are different ways to calculate $R_{s,a}$. The REINFORCE method [55] uses the discounted reward $R_t = \sum_{k=0}^{T-t} (\gamma^k r_{t+k})$ as $R_{s,a}$. The Actor-Critic method [9] introduces a parametarized action-value function $Q(s,a|\theta^Q)$ (critic) that estimates $R_{s,a}$. In the actor-critic framework, the update direction of the actor $\pi_\theta(a|s)$ is provided by the critic. The gradient of the objective function for the Actor-Critic method can be written as Equation 7. More ways to calculate $R_{s,a}$ can be found in [36].

$$\bigtriangledown_{\theta^\pi} J(\theta^\pi) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\bigtriangledown_{\theta^\pi} log \pi_\theta(a|s) Q(s,a|\theta^Q)] \quad (7)$$

4.4.1 Deep Deterministic Policy Gradient (DDPG)

The policy in Equation 7 is stochastic $\pi_\theta(a|s) \to [0,1]$ which assigns a probability to each action given a state s. The Deterministic Policy Gradient (DPG) [39] utilizes a deterministic policy $\mu(s|\theta^\mu) \to a$ which directly maps a state to an action. The gradient of the deterministic policy $\mu(s|\theta^\mu)$ is:

$$\bigtriangledown_{\theta^\mu} J(\theta^\mu) = \mathbb{E}_{s\sim p^\mu, a\sim\mu_\theta}[\bigtriangledown_a Q^\mu(s,a)|_{a=\mu(s|\theta^\mu)} \bigtriangledown_{\theta^\mu} \mu(s|\theta^\mu)] \quad (8)$$

where p^μ is the state distribution following the policy μ. Deep DPG (DDPG) [21], is a method that combines techniques of DPG and DQN in order to extend DQN method to continuous action space environments. In contrast with DPG, the DDPG parameterized the action-value function $Q^\mu(s, a|\theta^Q)$ with the deterministic policy μ. The update of it can be written as:

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}}[(Q^\mu(s_t, a_t|\theta^Q) - y_t)^2], \text{where } y_t = r_t + \gamma \bar{Q}^\mu(s_{t+1}, \bar{\mu}_\theta(s_{t+1})|\theta^{\bar{Q}}). \quad (9)$$

DDPG also utilizes a replay buffer and target networks to stabilized learning. There are 4 function approximators in the DDPG method, the policy $\mu(s|\theta^\mu)$, the action-value function $Q^\mu(\cdot|\theta^Q)$ and their corresponding target networks $\bar{\mu}(s|\theta^{\bar{\mu}})$ and $\bar{Q}^\mu(\cdot|\theta^{\bar{Q}})$. After collecting a number of transitions, the critic and the actor update after each time-step. The targets copy the parameters of the main networks every few time-steps. DDPG uses a soft update to slowly change the actor and critic networks:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \quad (10)$$

where $\tau \ll 1$ is the hyper-parameter that controls the update amount. Since the policy is deterministic, DDPG adds noise to the policy as shown in Equation 11 in order to maintain exploration. The noise is sampled from a noise process $\mathcal{N}$ which can be Gaussian action noise or Ornstein Uhlenbeck action noise.

$$a = \mu(s|\theta^\mu) + \mathcal{N} \quad (11)$$

4.4.2 Distributed MADDPG

So far we explained how a single agent learns. In this section, we explain a simple method that adapts DDPG into multi-agent tasks. The first way to extend the DDPG algorithm for multi-agent learning is to simply treat each agent in the environment as an individual agent, which is known as distributed training [12]. It means that each agent i has its own actor $\mu_i(o_i)$ and critic $Q_i^\mu(o_i, a_i)$ networks and only considers their local information (observation o_i and action a_i). We refer to it as Distributed Multi-Agent Deep Deterministic Policy Gradient (Dis-MADDPG, Algorithm 1). Another way is to use a shared actor and critic that control all the agents' behaviors, but this requires the structure of the agents' observation and action to be the same. For example, the agents in the N-Agent Navigation use the same information and actions, so sharing an actor and a critic can be applicable. In the Communicative Navigation, the speaker and the listener observe different information and have different type of actions. For the Dis-MADDPG method we initialize an actor and a critic for each agent for both environments.

Before training the networks $\boldsymbol{\mu} = \{\mu_1, ..., \mu_N\}$ and $\boldsymbol{Q} = \{Q_1^\mu, ..., Q_N^\mu\}$, the agents interact with the environment and store training data in the replay buffer. The environment provides an initial state s_0 which consists of N local observations $(o_1^0, ..., o_N^0)$ for each agent. At time-step t, each agent i selects an action a_i^t that corresponds to the output of its current policy μ_i given local observation o_i^t and exploration action noise. Next, a list of actions $x_t = (a_1^t, ..., a_N^t)$ from all the agents is executed on the environment, and the environment returns a reward r_t and a new state $s_{t+1} = (o_1^{t+1}, ..., o_N^{t+1})$. Here the reward is not a list because we only consider the cooperation tasks, where all the agents receive the same reward. In the case of competitive tasks, the environment would return a list that contains the rewards for each agent. The transitions (s_t, x_t, r_t, s_{t+1}) are stored in a shared replay buffer B.

Since the Distributed MADDPG algorithm only considers the local information of agents, we can extend the loss of the action-value function in Equation 9 to multi-agent learning. For each agent i, the loss function can be written as:

$$\mathcal{L}(\theta^{Q_i}) = \mathbb{E}_{o_i^t, a_i^t, r_t, o_i^{t+1}}[(Q_i^\mu(o_i^t, a_i^t|\theta^{Q_i}) - y_t)^2], \text{where } y_t = r_t + \gamma \bar{Q}_i^\mu(o_i^{t+1}, \bar{\mu}_i(o_i^{t+1})). \quad (12)$$

When calculating y_t, the target networks $\bar{Q}_i^\mu$ and $\bar{\mu}_i$ with parameters $\theta^{\bar{Q}_i}$ and $\theta^{\bar{\mu}_i}$ are used for stabilizing the training. Extending Equation 8, the gradient of the expected return of each agent i can be written as:

$$\bigtriangledown_{\theta^{\mu_i}} J(\theta^{\mu_i}) = \mathbb{E}_{o_i \sim \rho^{\mu_i}, a_i \sim \mu_i}[\bigtriangledown_{a_i} Q_i^\mu(o_i, a_i)|_{a_i = \mu_i(o_i)} \bigtriangledown_{\theta^{\mu_i}} \mu_i(o_i)] \quad (13)$$

We use $\mu_i(o_i)$ instead of $\mu_i(o_i|\theta^{\mu_i})$ for simplification.

Algorithm 1: Distributed Multi-Agent DDPG algorithm (Dis-MADDPG)

for *agent* $= i$ *to* N **do**
  Randomly initialize critic $Q_i^\mu(o_i, a_i|\theta^{Q_i})$ and actor network $\mu_i(o_i|\theta^{\mu_i})$
  Initialize the target networks $\bar{Q}_i^\mu$ and $\bar{\mu}_i$ with weights $\theta^{\bar{Q}_i} \leftarrow \theta^{Q_i}$, $\theta^{\bar{\mu}_i} \leftarrow \theta^{\mu_i}$
  Initial replay buffer B_i
end
for *episode* $= 1$ *to* M **do**
  Initialize a random process N for action exploration
  Receive initial observation state $s_0 = (o_1^0, ..., o_N^0)$
  for $t = 0$ *to* T **do**
    Each agent selects action $a_i^t = \mu_i(o_i^t|\theta^{\mu_i}) + N_t$ using current policy and exploration
    Execute actions $x_t = (a_1^t, ..., a_N^t)$ and observe reward r_t and new state s_{t+1}
    Store the transition $(o_i^t, a_i^t, r_t, o_i^{t+1})$ in the replay buffer R_i
    for *agent* $i = 1$ *to* N **do**
      Sample a random minibatch of S samples (o_j, a_j, r_j, o_{j+1}) from B_i
      Set $y_j = r_j + \gamma \bar{Q}_i^\mu(o_i^{j+1}, \bar{\mu}_i(o_i^{j+1}|\theta^{\bar{\mu}_i})|\theta^{\bar{Q}_i})$
      Update critic by minimizing the loss: $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j (Q_i^\mu(o_i^j, a_i^j|\theta^{Q_i}) - y_j)^2$
      Update actor using the sampled policy gradient:
      $\bigtriangledown_{\theta^{\mu_i}} J \approx \frac{1}{S}\sum_j \bigtriangledown_a Q_i^\mu(o, a|\theta^{Q_i})|_{o=o_i^j, a=\mu_i(o_i^j)} \bigtriangledown_{\theta^{\mu_i}} \mu_i(o|\theta^{\mu_i})|_{o=o_i^j}$
    end
    Update target network for each agent i:
    $\theta^{\bar{Q}_i} \leftarrow \tau\theta^{Q_i} + (1-\tau)\theta^{\bar{Q}_i}$
    $\theta^{\bar{\mu}_i} \leftarrow \tau\theta^{\mu_i} + (1-\tau)\theta^{\bar{\mu}_i}$
  end
end

4.4.3 Centralized MADDPG

One problem of the Dis-MADDPG algorithm is that the agents only use their local information which causes the agents to neglect cooperation with their teammates. If the environment is fully observable, like the N-Agent Navigation task, the Dis-MADDPG algorithm may work. For the communication navigation task (partially observable), since the policies of other agents change during the training process, an agent that only considers its local information cannot learn based on the reward it receives. This is also known as non-stationarity problem. An

example is that the speaker can utter the same signal given the same observation but receive a different reward. This can be caused if the listener moves to different positions given the same signal by the speaker as a result of its policy change. In this scenario, a learning algorithm that considers all the agents' information is required.

Inspired by the framework of centralized training and decentralized execution, the Centralized MADDPG [23] algorithm utilizes the information from other agents when training the action-value function (centralized training) but only uses the local information when choosing actions (decentralized execution). Same as the Distributed MADDPG, each agent has its own actor and critic networks in the Centralized MADDPG algorithm. This also allows it to be applied in both collaborative and competitive environments.

More specifically, the centralized action-value function $Q_i^\mu(s_t, x_t)$ now considers the states $s_t = (o_1^t, ..., o_N^t)$ and actions $x_t = (a_1^t, ..., a_N^t)$ from all the agents. Each state s_t and action x_t can simply be the concatenated observations and actions of all agents. We can change Equation 12 to the update of the centralized action-value function as follows:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s_t, x_t, r_t, s_{t+1}}[(Q_i^\mu(s_t, x_t|\theta^{Q_i}) - y_t)^2], \text{where } y_t = r_t + \gamma \bar{Q}_i^\mu(s_{t+1}, \bar{\mu}_1(o_1^{t+1}), ..., \bar{\mu}_N(o_N^{t+1})) \tag{14}$$

, where $\bar{\mu}_1, ..., \bar{\mu}_N$ are the target policies of the agents and they only consider the local observations $o_1, ..., o_N$. The gradient of the policies given as:

$$\bigtriangledown_{\theta^{\mu_i}} J(\theta^{\mu_i}) = \mathbb{E}_{s\sim\rho^\mu, x\sim\mu}[\bigtriangledown_{a_i} Q_i^\mu(s, a_1, ..., a_i, ..., a_N)|_{a_i=\mu_i(o_i)} \bigtriangledown_{\theta^{\mu_i}} \mu_i(o_i)], \tag{15}$$

where only a_i is predicted by the policy μ_i, other agents' actions are from the replay buffer transitions.

4.5 Count-based Exploration

Exploration is important for learning because it can help to find an optimal policy faster. The DDPG method adds random action noise to achieve exploration. When the environment is complex, we can not rely on random noise to help us find an optimal reward. Intrinsic reward methods add a bonus to the extrinsic reward to encourage the agent to explore more states and actions. The criteria of the intrinsic reward is different from the extrinsic one. Instead of requiring the agents to complete a task, intrinsic rewards may be given to the agents if they visit some new states or gather effective information. When training the policy, r_t' is used to calculate the loss function of the action-value function. It includes the extrinsic reward r_t from the environment and the intrinsic bonus r_t^+ [54]:

$$r_t' = r_t + \beta r_t^+ \tag{16}$$

where $\beta > 0$, is the bonus coefficient that balance exploration and exploitation. We can use $\beta = 0$ to represent the baseline. The Count-based exploration strategy uses the state-action count to encourage the agent to visit new state-action. At time t, the bonus r_t^+ equals to the inverse square root count of the state-action pairs:

$$r_t^+(s_t, a_t) = \frac{1}{\sqrt{n(s_t, a_t)}} \tag{17}$$

where $n(s_t, a_t)$ is the number of times this state-action pair has appeared before. With the inverse count bonus, the agent is encouraged to visit the less-visited states. This helps the

agent reduce the uncertainty of the environments with fewer interactions and therefore learn an optimal policy faster. The count is stored in a tabular C. In practice, the table is a dictionary in which the key is the state-action pair and the value is its total number of appearances.

4.5.1 Count-Based exploration for multi-agent learning

Similar to the Dis-MADDPG method where each agent is regarded as an individual agent in a multi-agent environment, the simplest way to apply Count-Based method is to keep a count table C_i for each agent i and use its local information $n(o_i, a_i)$ for counting (Distributed Multi-Agent Count-Based). If we want to explore the environment and encourage cooperation using the Distributed MACB method, the environments need to be fully observable. For example, the observation of each agent in the N-Agent Navigational task includes the position of the landmarks and the teammates.

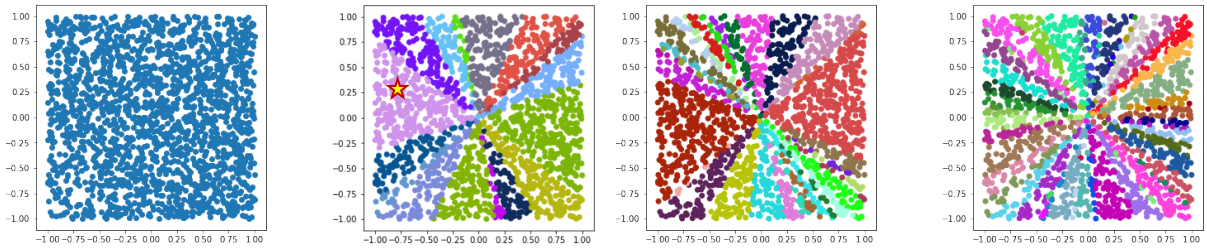
But some of the multi-agent environments are partially observed. The Dis-MACB strategy has the same drawbacks as the Dis-MADDPG learning method. The agents may focus on individual exploration and neglect the search of different types of cooperation with the teammates. For example, in the Communicative Navigation task, we suppose that the signal of the speaker is an array with 3 values where $(1, 0, 0)$ is the signal that shows that the target landmark is the red color, $(0, 1, 0)$ is the green, $(0, 0, 1)$ is the blue. The agents receive an extrinsic reward of -1 when the listener has not covered the landmark. One possible scenario is that the speaker utters $(1, 0, 0)$ 100 times when it observes the red landmark, but the listener never covers the red landmark. Applying Dis-MACB bonus on Equation 16, the reward r' for the speaker to utter $(1, 0, 0)$ when the landmark is red is small $(-1 + \beta\frac{1}{\sqrt{100}})$. This will result in encouraging the speaker to utter other signals when the target landmark is red again which is not what we expect.

Inspired by the Centralized MADDPG method, we propose a Centralized Multi-Agent Count-Based (Cen-MACB) strategy, which takes the joint observations of all the agents for counting $n(o_1, ..., o_N, a_1, ..., a_N)$. The joint observation covers all the information about the environment and the cooperation. With the Centralized MACB, if the listener chooses different actions during the 100 times in the above example, the reward r' for the speaker to utter $(1, 0, 0)$ signal is $(-1 + \beta\frac{1}{\sqrt{1}})$ which will not decrease the probability of speaker to utter $(1, 0, 0)$ when it observes the red landmark next time.

Both Distributed and Centralized MACB exploration strategies have the high-dimensional space problem in the continuous state and action space environments. The Count-Based method becomes meaningless if all the state-action pairs only appear one time. The count of each state will always be 1. This further causes higher storage memory and searching time problems. In order to apply Dis-MACB and Cen-MACB, we need to find a method to solve the high-dimensional space problem.

4.5.2 Hash Function

Learning from [46], we utilize a hash function to discretize the concatenated state and action $s||a$ into a k length hash code in the form of $\{-1, 0, 1\}^k$ which is used for counting. The main idea is to alleviate the one-time appearance problem by mapping similar state-action pairs into the same hash code. At the same time, the non-similar pairs should map to different hash code. The hash function used in [46] is called SimHash [8], which discretizes the states by the



(a) 2000 points (b) $k = 8$ (c) $k = 16$ (d) $k = 32$

Figure 2: Using a SimHash function to group 2000 points with $k = 8, 16, 32$. The points that map to the same hash code are given the same color. With k increasing, there are more groups and fewer points that map to the same code.

angular distance. The SimHash function $\phi(s||a)$ returns a k length hash code of the states and actions by:

$$\phi(s||a) = sgn(A \cdot s||a) \in \{-1, 0, 1\}^k, \tag{18}$$

where A is a predefined $k \times D$ matrix with i.i.d. entries which sample from a standard Gaussian distribution, D is the size of the state-action $s||a$ and k is the length of the hash code which controls the granularity. With a larger k, the hash code becomes longer which leads to less state-action pairs mapping to the same code. The function $sgn()$ is used to map a real number into $\{-1, 0, 1\}$:

$$sgn(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases} \tag{19}$$

Following Equation 18, the SimHash function returns a $\{-1, 0, 1\}^k$ code. To demonstrate how the SimHash function maps similar states into the same code, we randomly draw 2000 points in range $(-1, 1)$ and show the grouping results with $k = 8, 16, 32$. The points that map to the same hash code are grouped with the same color. Figure 2 shows how the SimHash function groups 2-dimensional points angularly. If the size of the concatenated state is large, a smaller k can be used to map similar state-action pairs into the same code. However, if the hash code is too short, useful information can be lost which can affect the learning process negatively. For example, if the agent's task is to navigate to the star position in Figure 2b as fast as possible, a Count-Based strategy can help the agent avoid exploring areas that have been visited before. In our example, the problem that occurs when $k = 8$ is that the area with the star covers a large portion of the states. If the agent visits this area and doesn't find the star, it becomes less probable to visit that area again as a result of the Count-Based exploration strategy. Therefore, a suitable k needs to be chosen for optimal results.

After decreasing the scale of state-action pairs using the SimHash function, we can use the corresponding hash code in the MACB strategy. The intrinsic reward is calculated by:

$$r_t^+(s_t, a_t) = \frac{1}{\sqrt{n(\phi(s_t||a_t))}} \tag{20}$$

The pseudo-code of Centralized MACB with the SimHash function is shown in Algorithm 2. After initializing the learning algorithm, we prepare the empty count table. Instead of initializing N tables for each agent, only 1 table is initialized and shared by all the agents to count the

joint state-action pairs. The benefit of sharing only 1 table is that all the agents can utilize it to share information. This ensures that the exploration progress of each agent is consistent and finds the best cooperation method more effectively.

Next, we set the hyper-parameters β for the exploration and exploitation trade-off and k for the granularity of the hash code. With k and D we can initialize the matrix A. After the initialization, we can begin the learning process. For each time-step, after collecting a transition, we update the count of the joint state-action pair in the table. During the training process, we randomly sample a batch of transitions and then map the $s_j||x_j$ of each sample to a hash code. Next, we use the count of this hash code to calculate the new reward r'_j for each sample. The new reward is used to update the critic and actor networks. The MACB strategy may fail if the update of the count and the calculation of the intrinsic rewards are not separated. This means that we have to update the count after collecting a transition, not when sampling this transition during training. A transition can be sampled many times during training, which results into increasing the count of the corresponding state-action pairs too fast, making the intrinsic bonus vanish after the first few episodes. In addition, the intrinsic reward of a state-action pair should be calculated after sampling a transition, rather than including them in the transitions in the replay buffer. The latter would cause identical state-action transitions in the replay buffer to have different rewards. When the random samples have earlier transitions, the reward does not correspond to the count. This would result into inaccuracies during training, as earlier transitions would not have the corresponding rewards for the current situation.

Algorithm 2: Centralized Multi-Agent Count-based strategy (Cen-MACB)

Initialize multi-agent learning algorithm (e.g. Cen-MADDPG)
Initialize an empty hash tables C where the new key initialize with value 0
Initialize hyper-parameters β for trade-off and k for hash code granularity
Initialize matrix $A \in \mathbb{R}^{k \times D}$ with i.i.d. entries sample from a normal distribution
for *episode* $= 1$ *to* M **do**
  for $t = 0$ *to* T **do**
    Collect transition (s_t, x_t, r_t, s_{t+1}) and store in the replay buffers
    Compute hash code using SimHash function
     $\phi(s_t||x_t) = sgn(A \cdot s_t||x_t) \in \{-1, 1\}^k$
    Update the count in the table C, $n(\phi(s_t||x_t)) = n(\phi(s_t||x_t)) + 1$
    for *agent* $i = 1$ *to* N **do**
      Sample a random minibatch of S samples (s_j, x_j, r_j, s_{j+1}) from replay buffers
      Compute hash code of each state-action pair $\phi(s_j||x_j)$
      Calculate the new reward $r'_j = r_j + \beta r_j^+$ where $r_j^+ = \frac{1}{\sqrt{n(\phi(s_j||x_j))}}$
      Update critic and actor using the new reward r'_j
    end
  end
end

We conclude all the methods introduced above in Table 1. There are two types of multi-agent learning algorithms: Distributed and Centralized MADDPG. The Distributed MADDPG only

considers the local observations and actions while Centralized MADDPG utilizes the framework of Centralized training (update actor and critic with a joint state-action for evaluation) and decentralized execution (only use local observation to choose action). We also have two types of multi-agent exploration strategies: Distributed and Centralized MACBs. The Distributed MACB strategy uses N count table for each agent to record their local observation-action. Before updating the count, a SimHash function ϕ is applied to the pairs. The same as the centralized training, the Centralized MACB strategy counts the joint state-action of all the agents. But in this case, we only initialize 1 table for all the agents to share information.

Table 1: This table concludes the learning algorithms and exploration strategies we introduced above. The description is the main difference between them.

Methods		Description
Learning Algorithm	Dis-MADDPG	Update with o_i and a_i
	Cen-MADDPG	Update with s and x
Exploration Strategy	Dis-MACB	Count $\phi(o_i\|\|a_i)$ using N tables
	Cen-MACB	Count $\phi(s\|\|x)$ using 1 table

5 Experimental design

So far, we have introduced two multi-agent learning algorithms (Distributed MADDPG and Centralized MADDPG) and two adaptions of the Count-Based strategy for multi-agent exploration (Distributed MACB and Centralized MACB). Note that in the following sections, when we mention Distributed MACB and Centralized MACB, they include the SimHash function. We use "without Hash" for not including the SimHash function. Recall that the two main problems we study in this thesis are 1) how to effectively explore an multi-agent environment and 2) how to balance the exploration and exploitation. Rather than relying on random action noise, we propose Dis-MACB and Cen-MACB in order to achieve a more effective exploration. In this section, we explain in detail how we investigate the impact of using MACB strategies with multi-agent learning algorithms for the first problem. The trade-off between exploration and exploitation is controlled by hyper-parameter β and also correlates to k. The detail of the experiments for finding the suitable parameters to balance exploration and exploitation are included in section 5.2.4. Our code can be found in Github [1] and the reproduction details can find in Appendix A.

5.1 Measurements

5.1.1 Success Rate

The success rate is the most direct measurement to evaluate an exploration strategy. With a better exploration strategy, the agents can probe the environments faster and therefore find optimal solutions with fewer episodes. For the navigational task, success means that all the landmarks are covered by the agents before an episode finishes. After each episode of training, we run 10 more episodes for evaluation. In the evaluation episodes, the actions applied to

[1] https://github.com/JianingWang99/CentralizedCountBased

the environment are without random noise. The success rate of an episode we use in all the experiments is the number of times that the agent succeeded over 10 evaluation episodes $\frac{\#success}{10}$. We can measure the effectiveness of an exploratory strategy from the trend of the success rate and the speed at which it converges.

5.1.2 Time Complexity

Additional exploration methods require more time to process and the time complexity differs between different strategies. We cannot say that an exploration strategy is effective when it helps the success rate converge with fewer episodes but each episode requires much more time to process. We measure the time complexity with the average time-step being processed per second. If this value is high, it means that there are more time-steps that can be processed in a second, and each episode costs less time. In comparison to the success rate where the data is from the episode evaluation, the time complexity records the speed of the algorithms during training. All of the experiments run on the same machine with the same version of software.

5.1.3 Count-1 Percentage

For the Count-Based strategy, we are also interested in the performance of the SimHash function to alleviate the high-dimensional space problem. Therefore, Count-1 Percentage is used to calculate the percentage of key $s||a$ in the count table that only appears one time (value $= 1$) over all the pairs.

5.1.4 Number of Collisions

In the N-Agent Navigation task, we want the agents to cover all the landmarks without collision. If a collision occurs, the agent gets punished by receiving a negative reward. Besides the success rate, we can also use the number of collisions to reflect the performance of agents. The number of collisions for an episode is the accumulated collisions from episode 0 up to the current episode.

5.2 Experiments

The general plan of our experiments is to evaluate the exploration strategies from simple to complex environments. For each experiment, we mention what exploration strategy we evaluate with which learning algorithm and environment. In addition, we include the expected results for some of the experiments.

5.2.1 Count-Based with 1-Agent Navigation

Before moving to the multi-agent tasks, we first test the Count-Based strategy with the DDPG learning algorithm on the N-Agent Navigation environment where $N = 1$. This experiment aims to find a general idea of how to choose β for the multi-agent tasks. We compare the success rate of the agent with a different amount of exploration where $\beta = \{0.0, 0.2, 0.4, 0.6, 0.8, 1.2\}$. When $\beta = 0.0$, the agent only uses random action noise to explore the environment, which provides a baseline for the Count-Based strategy. According to [46] using Count-Based and SimHash function for single-agent exploration, when β is too large, the intrinsic reward will overwhelm the extrinsic reward. Recall that the extrinsic reward

for this environment is 0 when the agent covers the landmark and -1 otherwise. As noted in [46], β should be smaller than 1.0. We increase β with 0.2 intervals from 0.2 to 0.8 in our experiment and compare their performances. In [46], they also show that $\beta = 0.2$ has a better performance. We want to see if this is true in our experiments. We also want to see the performance of the Count-Based method when the intrinsic reward overwhelms the extrinsic reward and choose $\beta = 1.2$ to evaluate.

5.2.2 MACB with Fully Observed Environment

In this experiment, the environment is N-Agents Navigation where $N = 2$. This environment is fully observable by the agents. We first evaluate MACB with the Distributed MADDPG learning algorithm. We evaluate whether the MACB can help accelerate the learning process and whether it is effective according to the time complexity. In addition, to show the effectiveness of the SimHash function, we compare the performance of Dis-MACB with and without Hash function. In paper [46] they run experiments with $k = \{64, 128, 256\}$. Regardless of the value of k, their method always outperforms the random noise exploration strategy. For our experiments, we use $k = 64$ for this experiment.

The second experiment on this environment is to evaluate MACB with the Centralized MADDPG learning algorithm. We want to evaluate whether our exploration strategies can improve the performance of different learning algorithms. In addition, according to [46], only using states for counting has almost the same result as using state-action pairs. We also run experiments to compare these two cases, because if using states to count has the same performance, we can further reduce the dimensionality. The parameters are the same as the experiments with Distributed MADDPG.

5.2.3 MACB with Partially Observed Environment

The experiments we explained so far aim to evaluate the performance of our MACB strategies with different leaning algorithms. We also want to know their performance in a partially observed environment, therefore we evaluate the effect of MACB with Centralized MADDPG in the Communicative Navigation environment. This environment is partially observed and the Distributed MADDPG method does not show learning tendency [23], therefore we do not include Dis-MADDPG in this experiment. An exploration strategy can only help a learning algorithm perform better, but can not help a learning algorithm that cannot learn at all. Recall that this environment has two types of reward functions, so in this experiment we evaluate whether the MACB strategies can accelerate the learning of Centralized MADDPG with sparse and dense reward settings. To do so, we modify the original dense reward function into a sparse reward function that returns 0 if the task is achieved and -1 otherwise.

5.2.4 Exploration and Exploitation

The second problem of this thesis is to balance the trade-off between exploration and exploitation. Learning from [46], a large β emphasizes exploration and may lead the agent to neglect the extrinsic reward from the environment. A small β may cause the agent to not explore enough, leading to never obtaining a positive reward. The minimum value of β is 0 in our experiments which means that the algorithm completely neglects the advantage of MACB exploration.

In order to address the high-dimensional space problem, we utilize a SimHash function that maps the pair with continuous value into a k length binary code. With a bigger k, there are more possible $\phi(s||a)$ that need to be visited and this may require a larger β to encourage more exploration. So the exploration and exploitation trade-off correlates to both β and k. In our final experiment, we would like to see how different $\beta = \{0.05, 0.2, 0.4, 0.8\}$ with different $k = \{32, 64, 256, 512\}$ influence the performance of Cen-MACB with Cen-MADDPG on the Communicative Navigation problem. We want to see if our results can prove that with the same β, a higher k leads to a better result [46]. We use a table to conclude the average success rate after 4×10^4 episodes of training for a different combinations of β and k.

6 Results

In this section, we show the results of our experiments. For visibility purposes, all the graphs are smoothed and averaged over 3 random seeds with standard deviation and 75% confidence interval. We include the parameters for the learning algorithms in Appendix A.

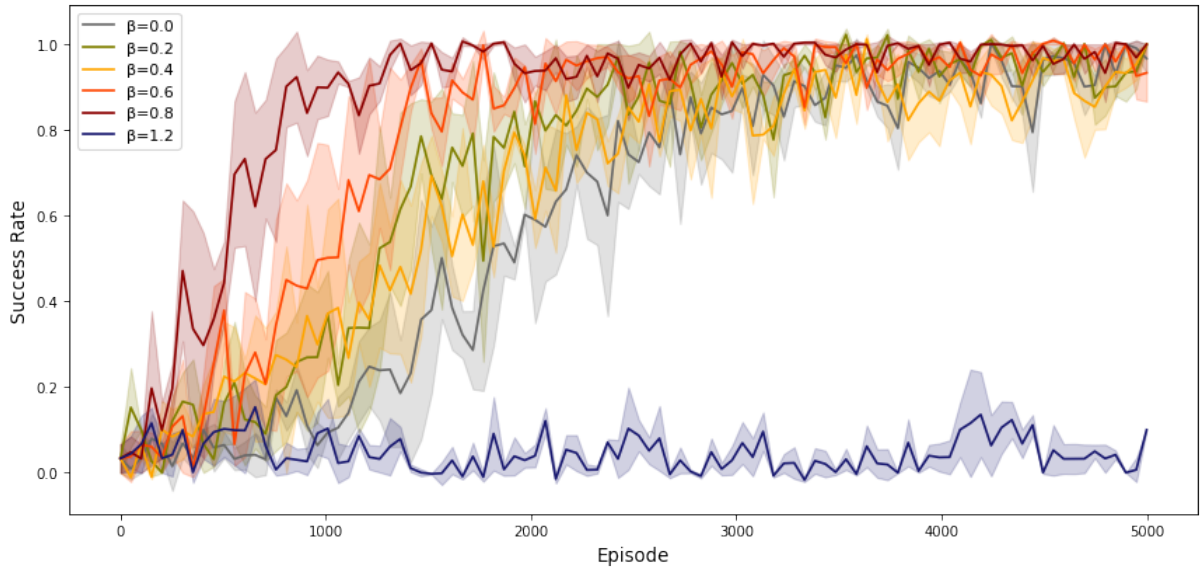


Figure 3: The success rate of DDPG algorithm on the 1-Agent Navigation problem with and without Count-Based exploration strategy. $\beta = 0.0$ is DDPG without Count-Based strategy. When $0 < \beta < 1$, the maximum value of intrinsic reward is less than the maximum value of extrinsic reward. When $\beta > 1$, the maximum intrinsic reward is larger than the maximum extrinsic reward.

6.1 Count-Based Exploration with DDPG

Figure 3 shows the average success rate (details in section 5.1.1) of the Count-Based exploration strategy with the DDPG learning algorithm on the 1-Agent Navigation problem. We compare the success rate of the Count-Based strategy with $\beta = \{0.0, 0.2, 0.4, 0.6, 0.8, 1.2\}$. The agent randomly explores the environment with $\beta = 0.0$ which provides a baseline for the Count-Based strategy. As we see, the success rate of $\beta = 0.0$ starts to increase steadily from around 1000 episodes and converges at around 4000 episodes. When β becomes larger than

0.0 and smaller than 1.0, the success rate increases faster than $\beta = 0.0$ before 1000 episodes and converges faster, especially when $\beta = 0.6$ and $\beta = 0.8$. Our results are different than [46], where they conclude that the agents' performance peaked at around $\beta = 0.2$ in the single-agent Atari game: Frostbite. In our experiment, a higher value of β can help the agent effectively probe the environment and find the landmark quickly. However, when $\beta = 1.2$, the agent shows no learning. This is because the exploration bonus overwhelms the extrinsic reward, which makes the agent always focus on exploring new areas, rather than reaching the landmark. The results of this experiment show us that the Count-Based strategy can help the agent find the landmark and converge faster, as long as the maximum intrinsic reward is smaller than the maximum of the extrinsic reward.

6.2 MACB Strategies with Dis-MADDPG

In this experiment, we show the results of the MACB strategies with Distributed MADDPG algorithm on the 2-Agent Navigation problem (fully observable). Figure 4 shows the success rate of the Dis-MADDPG algorithm with 3 different Count-Based strategies but has the same amount of exploration ($\beta = 0.7$). First, we notice that by only increasing 1 agent in the navigation task, the number of episodes that the agents need to tackle the task increases dramatically. For the single-agent task, the DDPG algorithm itself can converge before 5×10^3 episodes. For the 2 agents problem, although the Dis-MADDPG algorithm increases steadily, it requires around 5×10^4 episodes to converge, 10 times more than the single agent problem. This is because there are more states that need to be explored in the multi-agent environment. The length of the state is increasing from 4 for one agent to 18 for two agents and each value in the state is continuous. Since multi-agent environments have more states that need to be explored, it needs an effective exploration strategy.

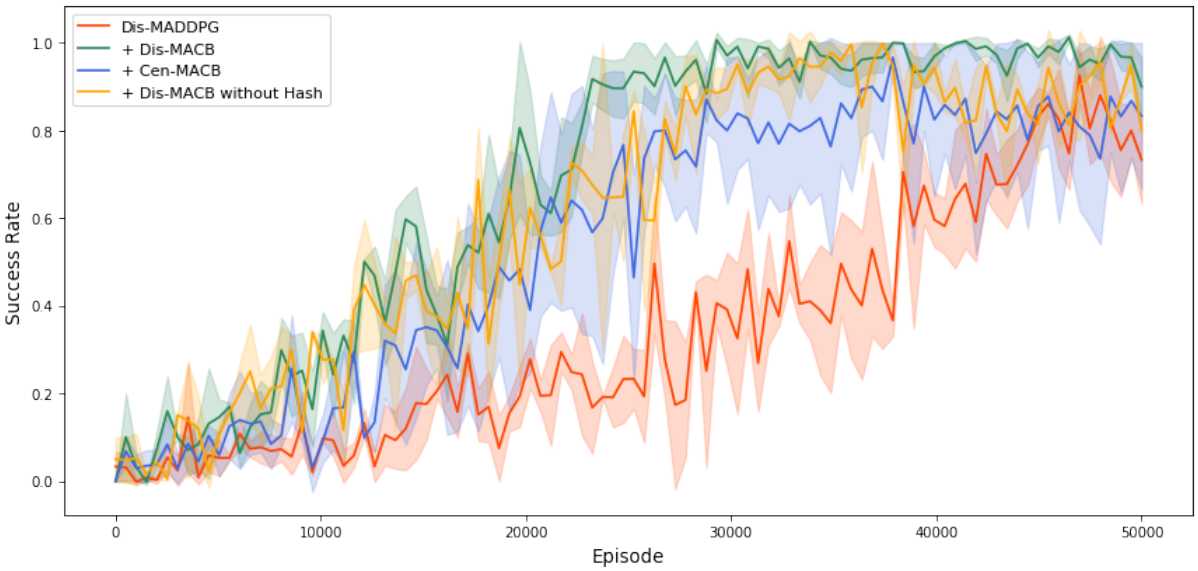
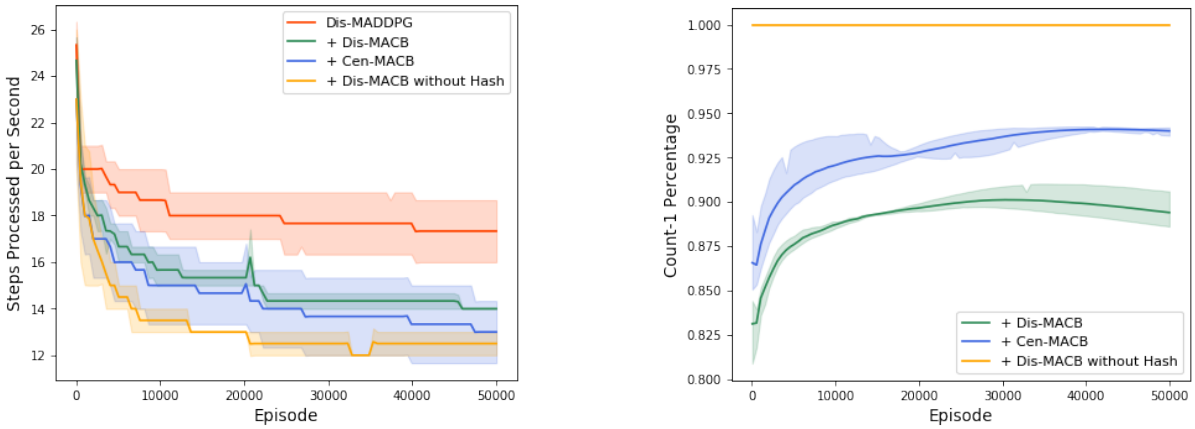


Figure 4: The success rate of Dis-MADDPG learning algorithm on the 2-Agent Navigation problem with 3 different MACB exploration strategies. All 3 exploration strategies can help to accelerate the learning, and the Dis-MACB has the most stable effectiveness when we apply it on the Dis-MADDPG algorithm comparing to the other two MACB.

When we add multi-agent count-based strategies on the Dis-MADDPG, from Figure 4 we see that with the help of 3 different MACBs the success rate increases faster and converges with much less training episodes. Specifically, they only require around half of the episodes to reach the same success rate as the Dis-MADDPG. Furthermore, we can see that the trend of the success rate of these 3 MACB strategies is almost the same. The surprising result is from the Dis-MACB without Hash, because it shows that it performs almost the same as the Dis-MACB despite the fact that there is no grouping of similar states, which makes the Count-Based strategy meaningless. We believe this may be because of different extrinsic reward settings. The extrinsic reward in this environment is calculated by $-(n+m)/2$ where n is the number of landmarks that are not covered, m is the number of collisions. We normalize the reward by dividing the result with the number of total agents, which is 2 in our case. Since $n \in \{0, 1, 2\}$ and $m \in \{0, 1\}$ for 2 agents, the extrinsic reward is $[0, -0.5, -1.0, -1.5]$. By adding a bonus of 0.7 to all the scenarios, the reward becomes $[0.7, 0.2, -0.3, -0.8]$. We change the extrinsic reward setting to the later. The Dis-MADDPG with the later reward as extrinsic reward has the same performance as the Dis-MACB without Hash. The Dis-MACB without Hash method improves the performance because of the different extrinsic reward settings, not because of the Count-Based strategy. We also notice that with Cen-MACB, the performance is slightly worse than the Dis-MACB and converges at around 80% success rate. This may be because the loss function of the action-value function (Equation 12) in the Dis-MADDPG only considers the local observation and action, but the intrinsic reward considers the joint pairs of all the agents, which leads to a mismatch of information. So when using the Dis-MADDPG learning algorithm in the 2-Agent Navigation problem, Dis-MACB is the most stable strategy that stimulates the learning over these 3 MACB strategies.



(a) Time Complexity

(b) Count-1 Percentage

Figure 5: The time complexity of MACB strategy. Figure (a) shows the number of steps that a method can process per second over the training episodes. Figure (b) shows the percentage of $\phi(s||a)$ that only appears one time in the count table which can reflect the effect of SimHash function.

Before moving to the second learning algorithm, we want to talk about the time complexity of these 3 exploration strategies. Additional exploration requires more time to process. If a strategy requires much more time than simply using random exploration, it is not considered effective. Figure 5a shows the average time steps processed in a second of each methods. The large drop for all the methods at the beginning is because the agents only collect data

and do not train the policies during the first 100 time steps. It is understandable that the Dis-MADDPG without any MACB strategies can process more time steps in a second. The time efficiency of the Dis-MACB and Cen-MACB are close to each other. Most importantly, without the hash function, it spends more time processing each time-step. Therefore, the hash function does not only alleviate the states that only appear one time, but also improves the time efficiency.

For a more obvious comparison, we calculate the required time for Dis-MADDPG and Dis-MACB without Hash by combining the success rate and the time-step per second together. If the lowest time-efficient method costs less time to converge, we can say that all 3 MACB are effective based on the success rate and the time efficiency. To calculate that, the agents use around 2.5×10^4 episodes to handle the task with MACB, and around 5×10^4 without the help of MACB. The minimum time steps that are processed for the Dis-MACB without Hash is around 12, and 18 for the Dis-MADDPG. Each episode has 20 time steps. So the time for Dis-MACB without Hash is $\frac{2.5\times10^4\times20}{12} \approx 41666$ seconds, which is less than $\frac{5\times10^4\times20}{18} \approx 55555$ for the Dis-MADDPG. Therefore, we can say that the MACB are effective based on the success rate over the episodes and the time it takes to process extra exploration.

The reason why the hash function can improve the time efficiency of the MACB strategy is that it helps to map similar state-action pairs to the same key. This decreases the number of items in the table and saves searching time. The count-1 percentage in the Figure 5b also reflects this. With a higher count-1 percentage (Dis-MACB without Hash $>$ Cen-MACB $>$ Dis-MACB), there are fewer steps that can be processed per second (Dis-MACB without Hash $<$ Cen-MACB $<$ Dis-MACB). Without the hash function, all the state-action pairs only appear one time and less steps be processed per second. With a hash function and the same k, the possible combinations of state-action pairs of the Cen-MACB is more than those of the Dis-MACB, and therefore the count-1 percentage of Cen-MACB is higher than Dis-MACB.

6.3 MACB Strategies with Cen-MADDPG

In the previous section, we evaluate the performance of MACB with the Dis-MADDPG algorithm. In this section, we show the results of them when applied to the Centralized MADDPG learning algorithm. We first evaluate on the 2-Agent Navigation (fully observed environment) and then on the Communicative Navigation (partially observed environment).

2-Agent Navigation

Figure 6 shows the success rate of the MACB methods on the 2-Agent Navigation problem. We first observe that the Cen-MADDPG agents learn gradually over episodes and fully grasp the problem after around 4×10^4 episodes, which slightly outperforms the Dis-MADDPG agents. All of the MACB strategies can promote the speed of learning and converge before 3×10^4 episodes. Because the MACB converges at different episodes over 3 random seeds, the average success rate does not reach to 100% in Figure 6. In addition, if we continue training after convergence, the success rate gradually decreases. Therefore, it is better to use Early Stopping when training the agents using MACB strategies. Since 2-Agent Navigation environment is fully observable, Dis-MACB and Cen-MACB have almost the same performance when counting the state-action pairs. Furthermore, in comparison to [46] where the state-action pairs counting has almost the same result as only counting the state, our results show that counting the state-action pairs has a better performance. The count with actions include a more detailed

information that describe the current exploration situation, which may be the reason for a slightly better performance.

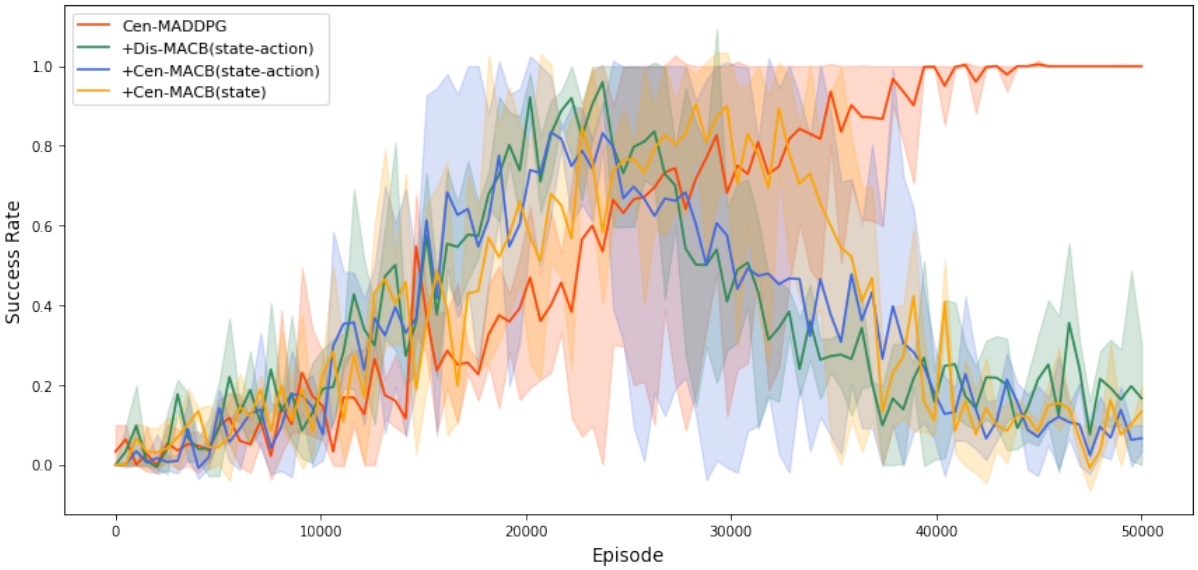


Figure 6: The success rate of the Cen-MADDPG learning algorithm on the 2-Agent Navigation problem with 3 different MACB exploration strategies. We compare the performance of Distributed MACB, Centralized MACB strategies, and the difference performance of counting the state-action pairs and only the states. All the MACB strategies can accelerate the learning of the Cen-MADDPG algorithm.

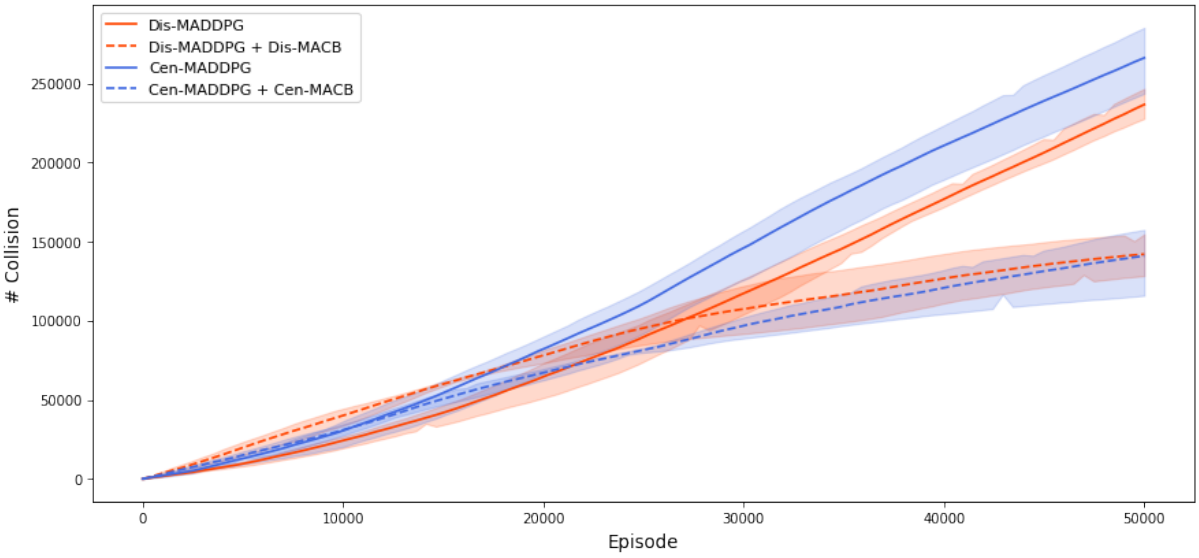


Figure 7: The accumulated number of collisions during episodes on the 2-Agent Navigation environment. With the help of MACB strategies, the agents with both Distributed and Centralized MADDPG learn how to cooperation faster and avoid colliding.

Except the success rate, we compare the accumulated number of collision between Distributed and Centralized MADDPG with the MACB strategies in Figure 7. According to [23], their result shows that the position of the Cen-MADDPG agents is a little closer to the landmarks than the

Dis-MADDPG agents. Although we do not have the policy ensembles as they do, our results are the same as theirs. The success rate of the Cen-MADDPG agents slightly outperforms the Dis-MADDPG. The policy ensembles can improve the stability of the Centralized MADDPG. In addition, [23] also mentions that the Cen-MADDPG agents only have half of the number of collisions. As shown in Figure 7, our result shows that the number of collisions of the Cen-MADDPG is more than that of Dis-MADDPG, which is contrary to their results. However, after we apply the MACB strategies, the accumulated number of the collisions after 5×10^4 episodes vastly decreases, especially for the Cen-MACB, which decreases the number to less than half. These results reflect that the MACB exploration can help the agents find the optimal cooperation strategies with less episodes and therefore the total number of collisions becomes less.

Communicative Navigation

In section 2.2 we introduce the Communicative Navigation environment which is partially observed. In this environment, the agents require a centralized method that takes the information of other agents into account to solve the non-stationarity problem. Figure 8 shows the success rate of MACB strategies in 2 different reward settings.

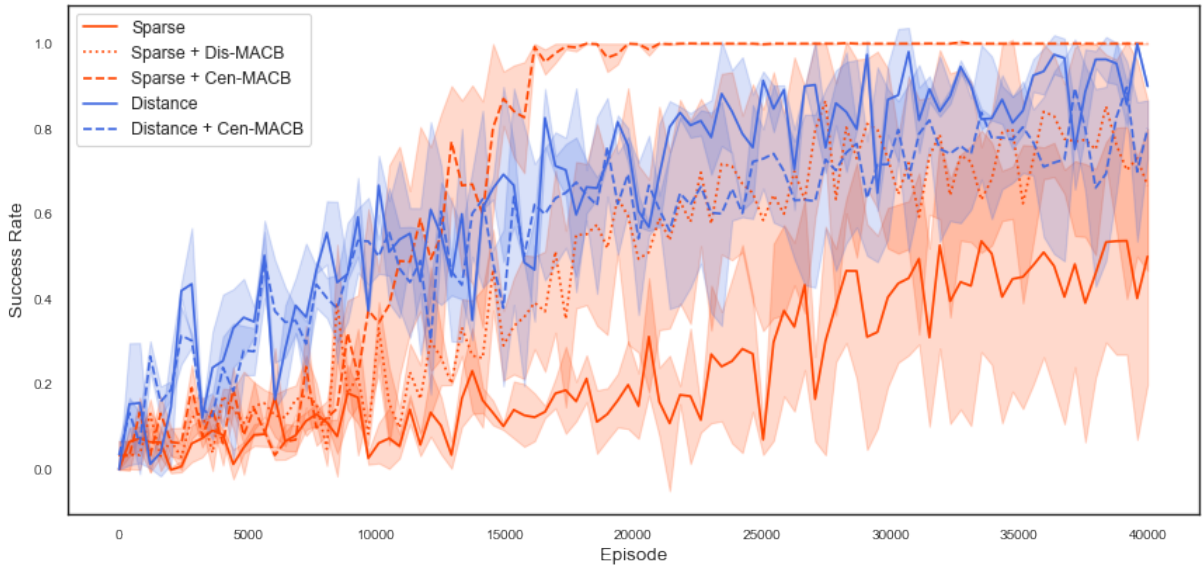


Figure 8: The success rate of Cen-MADDPG learning algorithm and MACB strategies on the Communicative Navigation problem with sparse reward and distance reward settings. In the sparse reward environment, the Cen-MACB strategy can help the agents find the optimal solution faster. It's success rate converges to 100% with less than 2×10^4 episodes.

In the sparse reward environment, the Centralized MADDPG agents learn slowly and only surpass 40% success rate after 4×10^4 episodes of training. While with the help of Dis-MACB the success rate increases to around 70% at 4×10^4 episodes, the agents with Cen-MACB strategy can reach 100% success rate with only 1.5×10^4 episodes. Both MACB strategies have the same β (0.8) and k (512). This proves that in the partially observed environment, a centralized method is required to achieve a better performance. In addition, with the help of both MACB strategies, the success rate starts to rapidly increase at around 1×10^4 episodes. This shows that the Count-Based strategy requires a number of episodes to encounter many

different scenarios, and once the agents receive positive rewards, the success rate starts to increase rapidly.

In the environment with distance reward, the agents start to learn when the training begins, and the success rate increases steadily until converge. Under the sparse reward setting, the agents receive the same negative reward at the start of training and have no learning signal until the speaker covers the target landmark. Once the agents in the sparse reward environment receive the non-negative reward, the success rate increases faster than the agents in the distance reward environment. If the agents have an effective exploration strategy which can help them to find the optimal solution with less episode, their success rate can converge at almost the same episode as the agents in the distance reward environment. We believe this is a very important feature, as it can be applied in many sparse reward environments without the need of reward shaping. We also observe that after tuning the parameters, the learning process for the agents in the sparse reward setting with an effective exploration strategy (Sparse + Cen-MACB) can even outperform the agents in the dense reward (Distance). In addition, we apply Cen-MACB on the Distance reward. Following the finding that the intrinsic reward has to be smaller than the extrinsic reward, we use $\beta = 0.1$ to control the amount of intrinsic reward. The results show that the Cen-MACB strategy makes Cen-MADDPG slightly worse in the dense reward setting. Both distance and count bonus confuse the agent. Most importantly, if an environment does not have a sparse reward setting, it does not need an intrinsic reward to boost learning.

6.4 Exploration and Exploitation

In this section, we show the results of the trade-off between exploration and exploitation with Centralized MACB and Centralized MADDPG algorithms. We evaluate on the Communicative Navigation problem. Table 2 concludes the average success rate and the count-1 percentage (sr, c-1) after 4×10^4 episodes of training with different combinations of β and k. With the help of

Table 2: The table concludes the success rate (sr) and count-1 percentage (c-1) of the Cen-MACB with Cen-MADDPG on the Communicative Navigation with different ratio of exploration β and length of hash code k.

k \ β	**0.0**	**0.05**	**0.2**	**0.4**	**0.8**
	sr, c-1	sr, c-1	sr, c-1	sr, c-1	sr, c-1
-	47%, -	-	-	-	-
32	-	59%, 67%	60%, 68%	56%, 71%	53%, 74%
64	-	**72%**, 91%	60%, 91%	34%,91%	**72%**, 94%
256	-	25%, 99%	**82%**, 99%	43%,99%	**81%**, 99%
512	-	**75%**,100%	48%,100%	**75%**,100%	**100%**,100%

random noise exploration ($\beta = 0.0$), the agents reach around 47% success rate. From the table we see that the MACB method can help the agent obtain a higher success rate in most of the parameter combinations. When $k = 32$, the performance of MACB just slightly outperforms random noise method. As we discuss in section 4.5.2 with the star example when $k = 8$, if the hash code is too short, useful information can be lost which affects the performance of MACB. When $k \geq 64$, the success rate has at least two peaks with different values of β. The peaks

are highlighted in a bold style. When k increases, there are more different hash codes which requires a larger β to emphasize exploration. Therefore, the first peak is at a relatively larger β when k is bigger. For three different k, they all have a peak when $\beta = 0.8$. This is contrary to [46] where their agent reaches a higher success rate when $\beta = 0.2$. This means that the best combination of β and k is different in different environments. We also observe that the count-1 percentage increases when k increases and converges to 100% with $k = 512$ in this environment. Importantly, when $c - 1 = 100\%$, the success rate can surpass 75% in most of the time. With the help of count-1 percentage, we can know whether we need to increase k or not. Note that the count-1 percentage here is the one after 4×10^4 episodes of training. It is not always at 100% as the MACB without hash function we show above.

In conclusion, when tuning k, we can increase the value of k until its count-1 percentage converges at 100%. When k is too big, the hash function may loose its meaning and the search time and storage memory becomes high. With a larger value of k, the agents need to explore more state-action pairs and require a larger β. When choosing β, we also need to make sure that the maximum value of the intrinsic reward is smaller that the maximum value of extrinsic reward.

7 Discussion

In this section, we discuss the problems we encountered in adapting the single-agent exploration method to the multi-agent exploration method. In addition, we further discuss how to balance the trade-off between exploration and exploitation based on our experimental results. We also summarize the weakness of the MACB method, as well as our future work.

7.1 From single-agent to multi-agent exploration

Count-Based is an effective method in single-agent exploration. The main purpose of our thesis is to adapt it to multi-agent exploration strategy. In addition to Count-Based method, there are many other single-agent exploration methods that can be adapted to multi-agent exploration, such as VIME [14, 27] and prediction error of feature encoder [33, 42], which we mention in the related work section (section 3). We encountered two problems in the process of adapting the Count-Based method. These two issues are also worth noting in the adaptation process of other methods. These issues include the high-dimensional state and action space problem, and the non-stationarity caused by the multi-agent environment.

The causes of high-dimensional space problem are different in fully observable environments and partially observable environments. In a fully observable environment, an observation includes all the information the agent needs in order to make a decision. The observation of a multi-agent environment needs to include information about teammates, which leads to the problem of high-dimensional space. If we can solve this problem, we can easily apply the single-agent exploration method to the multi-agent problem. A fully observable multi-agent problem can be simply broken down into multiple independent single-agent tasks. We only need to treat each agent as a single agent and design their learning algorithms and exploration strategies, just like Distributed MADDPG and Distributed MACB.

Partially observable environments do not only have the high-dimensional space problem, they also face the problem of environmental non-stationarity. Different from the fully observable

environment, the observation of the partially observable environment does not need to include all the information that the agents need to make decisions. The high-dimensional space problem arises when we address the problem of the non-stationarity environment. The Centralized MADDPG method uses a centralized evaluation framework to solve this problem. Inspired by them, we extend the exploration method to a centralized one. For each agent in the partially observable environment, in order to solve the non-stationarity problem, they consider the observations and actions of their teammates, which causes the high-dimensional space problem. In addition to addressing the non-stationarity problem, considering the information of other agents makes Count-Based a method that can explore environment and come up with different ways of cooperation with their teammates at the same time, which is important for the collaborative multi-agent tasks.

The non-stationarity problem has been solved through centralized framework and there are many ways to address high-dimensional space problems. In this thesis we utilize a SimHash function that maps similar state-action pairs to the same code. However, if the state is in pixel space (such as StarCraft II [51]), another method is required (such as autoencoder) to extract the important features from the pixels. A simple hash function can not appropriately map similar pixel states into the same code [46]. Besides the methods to decrease the dimensionality for the Count-Based method, in the related work section (section 3) we also summarized other methods that utilize neural networks to predict intrisic reward for the high-dimensional space environments.

After addressing these two problems, our MACB methods effectively promote the agent's learning speed, whether it's in a fully observable environment or a partially observable environment. Our results show that Distributed MADDPG with the help of our Distributed MACB can reduce the number of episodes required for convergence by half in the 2-Agent Navigation task. In Communicative Navigation tasks, the Centralized MACB method increases the success rate of the Centralized MADDPG from around 47% (with 4×10^4 episodes of training) to 100% (with only 1.75×10^4 episodes of training).

7.2 Intrinsic Reward and Extrinsic Reward

In the intrinsic reward method, the trade-off between exploration and exploitation is controlled by the ratio of intrinsic reward and extrinsic reward. We use β to control this ratio as shown in Equation 16. If we have a clear goal, the maximum value of intrinsic reward cannot exceed the maximum value of extrinsic reward, otherwise the agent pays too much attention to the intrinsic reward and ignores the original task. From [33] we learn that we can also use intrinsic rewards for training even without extrinsic rewards from environment. Their results show that the agent can learn to play Super Mario game with only intrinsic rewards. Another rule for choosing β is based on the number of different states that your tasks have. If the agent needs to explore a particularly large number of states, then it needs a large β to focus on more exploration.

In section 4.5.2 we discuss about the bonus vanishing problem when explain Algorithm 2. When we apply the Count-Based method on the multi-agent learning algorithms which utilize a replay buffer, the best time to update the count for a state-action pair is after collecting them during the simulation as shown in the Algorithm 2. The intrinsic reward vanishes if we update the count after sampling transitions during training. When using a replay buffer for sample efficiency, a transition may be sampled a lot of times during training and the count of

this sample gets incremented too fast. Therefore, when using intrinsic rewards for exploration, we should pay attention to the vanishing problem. Check the update speed of the intrinsic reward, and avoid all of the intrinsic reward approaching to zero in the first few episodes of training.

7.3 Drawback of MACB

The main drawback of our MACB methods comes from the use of the SimHash function that address the high-dimensional space problems. Due to limited computing power, we did not experiment with more than two agents to evaluate our methods. We are not sure whether the hash function can summarize the information well after increasing the number of agents. In addition, the hash function cannot extract information well when the state is represented by raw pixels [46]. For future work, an autoencoder can be used to decrease the dimensionality of the state-action pairs. What's more, we use a table to store the count which greatly increases the computational time of the MACB methods. For future work, a neural network can be used to predict the pseudo-count [3, 30] of a state-action pair. Combining the above two future work, we can extend MACB methods to multi-agent exploration in pixels. In that case, a possible approach can be that the state-action pair gets passed to the autoencoder and the output code gets passed to a neural network that predicts its pseudo-count. We use this pseudo-count to calculate the intrinsic reward.

Recall that Cen-MACB is not stable in the 2-Agent Navigation task. The success rate of using Cen-MACB decreases after converging in Figure 6. We suggest that we can use Early Stopping to solve this problem. Another possible solution to address this can change the MACB method with a decaying β.

Another disadvantage is that the exploration method can only help the learning method learn faster. If computing power is not a problem, we only need to choose a good learning algorithm to solve the task. However, Count-Based is easy to implement and only needs to tune two parameters. Finding the best parameters is not difficult, k can be adjusted according to count-1 percentage, and β can be selected according to the number of the state-action pairs and the extrinsic reward.

8 Conclusion

In order to effectively explore the multi-agent environments with sparse reward setting, we extend the Count-Based method to Distributed and Centralized Multi-Agent Count-Based. We use the SimHash function to solve the high-dimensional space problem, and at the same time use a centralized method to solve the non-stationarity problem of the multi-agent environment. The centralized method also helps the Count-Based strategy to achieve simultaneously exploring the environment and the cooperation in the cooperative multi-agent task. After selecting suitable parameters β and k, both Distributed and Centralized MACB methods can dramatically promote the learning process of the multi-agent learning algorithm in fully and partially observable environments, respectively. However, we only evaluate our methods in environments with a maximum of 2 agents where the states are not represented by pixels. Our future work is to extend the MACB to work for environments whose states are represented by pixels using an autoencoder instead of SimHash and a neural network instead of table to predict the count.

References

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

[2] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[3] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*, 2016.

[4] W. Böhmer, T. Rashid, and S. Whiteson. Exploration with unreliable intrinsic reward in multi-agent reinforcement learning. *arXiv preprint arXiv:1906.02138*, 2019.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[7] L. Busoniu, R. Babuska, and B. 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.

[8] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[9] T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[10] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[11] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[12] S. Gronauer and K. Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, pages 1–49, 2021.

[13] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[14] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. *arXiv preprint arXiv:1605.09674*, 2016.

[15] S. Iqbal and F. Sha. Coordinated exploration via intrinsic rewards for multi-agent reinforcement learning. *arXiv preprint arXiv:1905.12127*, 2019.

[16] N. Jaques, A. Lazaridou, E. Hughes, C. Gulcehre, P. Ortega, D. Strouse, J. Z. Leibo, and N. De Freitas. Social influence as intrinsic motivation for multi-agent deep

reinforcement learning. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2019.

[17] M. N. Katehakis and A. F. Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[18] A. Krizhevsky and G. E. Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, volume 1, page 2. Citeseer, 2011.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[20] K. Kurach, A. Raichuk, P. Stańczyk, M. Zajac, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet, et al. Google research football: A novel reinforcement learning environment. *arXiv preprint arXiv:1907.11180*, 2019.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[23] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

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

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. *arXiv preprint arXiv:1509.08731*, 2015.

[28] I. Mordatch and P. Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[29] A. Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[30] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[31] P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

[32] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845, 2018.

[33] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

[34] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[35] H. Ryu, H. Shin, and J. Park. Remax: Relational representation for multi-agent exploration. *arXiv preprint arXiv:2008.05214*, 2020.

[36] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[37] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[38] D. Silver. Lectures on reinforcement learning. URL: https://www.davidsilver.uk/teaching/, 2015.

[39] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[40] S. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. Technical report, MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE, 2005.

[41] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

[42] B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[43] A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pages 856–863, 2005.

[44] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[45] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[46] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. 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.

[47] S. B. Thrun. Efficient exploration in reinforcement learning. 1992.

[48] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[49] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[50] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[51] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[52] T. Wang, J. Wang, Y. Wu, and C. Zhang. Influence-based multi-agent exploration. *arXiv preprint arXiv:1910.05512*, 2019.

[53] L. Weng. Policy gradient algorithms. *lilianweng.github.io/lil-log*, 2018.

[54] L. Weng. Exploration strategies in deep reinforcement learning. *lilianweng.github.io/lil-log*, 2020.

[55] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

A Appendix

A.1 Training Detailed

The networks for actors and critics are with two hidden layers with the size of 400 and 300. The activation function for both actor and critic is ReLU. After collecting 100 transitions, we begin updating the network parameters at each time step. An episode consists of 20 time steps. After an episode of training, we evaluate the algorithms with 10 more episodes without exploration action noises. All the results are averaged over 3 random seeds. The hyper-parameters used in the experiments are summarized in Table 3.

Table 3: Hyper-parameters used in experiments

Hyper-parameter	Value
Buffer size	10^6
Batch size	100
Time-step per Episode	20
Learning rate for optimizer	0.001
γ	0.99
τ	0.005

A.2 Reproduction

We conclude all the information to reproduce the learning algorithms and exploration strategies in this thesis. Before that, we talk about the changes we made in the environment. The description of the environments is in the problem statement (section 2) and the Markov Games sections (section 4.2). The code for environments are in the "ma_env/" folder. Communicative Navigation environment uses "simple_speaker_listener" file. The original extrinsic reward function is the minus distance of the listener to the target landmark. We change the reward to sparse based on whether the listener covers the target landmark or not. Both agents and landmarks in the environment are represented with a circle. If the distance between the center of the listener and the center of the target landmark is smaller than the sum radius of them, reward function returns 0 and -1 otherwise. The distance here is the euclidean distance. To calculate the success rate, we add a function to return 1 if the listener covers the target and 0 otherwise. The code to show the communication signals can be found in "environment.py" file. We use 3 small circles to represent the signals, and the signal that has the maximum value is showed in a darker color. For example, if the signals are with values $[0.99, 0.5, 0.4]$, the first circle is shown in a darker color. To simplify the task, we also change the size of the agents and landmarks.

For the learning algorithms, we do not write our code from scratch. Instead, we start by using the code of DDPG algorithm from Stable-Baselines3 [13]. Our first step is applying this DDPG algorithm on 1-Agent Navigation environment which is provided together with other multi-agent environments in [28]. They name this single-agent environment "simple". We do not change the default hyper-parameters settings of DDPG method from Stable-Baseline3.

After DDPG algorithm learns the single-agent task, we adapt it to Distributed MADDPG following Algorithm 1. The basic 3 parts of code we need to change from DDPG are initialization, transitions collection, and training. During initialization, we need to initialize actors, critics, and their target networks for all the agents. Each agent has a replay buffer. During transitions collection, the multi-agent environments return a list of states, next states, actions, rewards, dones which includes the data for all the agents. When executing actions in the environment, we need to collect all of the actions from the agents first, and then execute them together in the environment. Since each agent has its own replay buffer, we store the transitions of each agent in their own replay buffer. During training, we train the critics and the actors of each agent one by one. For the Centralized MADDPG, we joint all the observations and actions of all the agents together for training following the update Equation 14 and 15. The MADDPG authors [23] also provide the code, where they use TensorFlow packages for their neural networks. The code of Baselines3 we adapt from uses Pytorch. We only reproduce the centralized training and decentralized execution part from the MADDPG paper [23]. They also come out with Policy Ensembles which we have not reproduced. Besides, we assume that the agents can know the observations and policies of other agents.

The code of Count-Based also includes 3 parts: initialization, update count and calculation of intrinsic reward. The table we use is "defaultdict" which value's format is integer. When a new key gets store in this dictionary, the default value of this key is 0. The key in our case is $s||a$ array and the value is the number of times that this pair appears. In addition, before using $s||a$ as the key we convert it to bytes. When updating a count, the update function receives a state-action pair, converts it to bytes format, searches this key in the dictionary and adds 1 on its value. When calculating the intrinsic reward of a state-action pair, we convert it to bytes format first, and then search the value of this key, and apply it in Equation 16. The SimHash function receives a pair and converts it to hash code using Equation 18 and then the intrinsic reward is calculating by 20. In practice, before using SimHash function, we round the continuous value of pair to decimal 2.

All of the code are using Python programming language. We plot our results with the help of NumPy, Matplotlib, SciPy and seaborn packages. SciPy is used to smooth the line plot and seanborn helps us plot the results over 3 random seeds.