Learning transferable cooperative behavior in multi-agent teams

Akshat Agarwal

CMU-RI-TR-19-10

April 2019

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Katia Sycara, Chair
Timothy Verstynen
Wenhao Luo

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

This work has been funded by AFOSR award FA9550-15-1-0442 and AFOSR/AFRL award FA9550-18-1-0251

For my parents

Abstract

A multi-agent system comprises of multiple interacting intelligent agents, collaborating together to solve problems that are difficult or impossible for a single agent to solve, with the goal of maximising their shared utility. We study the emergence of cooperative behavior and communication protocols in multi-agent teams, for collaboratively accomplishing tasks like resource allocation and formation control for swarms. While multi-agent interactions can be naturally modeled as graphs, the environment has traditionally been considered as a black box. We propose creating a shared agent-entity graph, where agents and environmental entities form vertices, and edges exist between the vertices allowed to communicate with each other, allowing agents to selectively attend to different parts of the environment, while also introducing invariance to the number of agents/entities as well as permutation invariance, desirable properties for any multi-agent system representation. We present state-of-the-art results on coverage and formation control for swarms in a fully decentralized execution framework, and show that the learned policies have strong zero-shot generalization to scenarios with different team sizes. Additionally, we introduce communication dropout for robustness to glitches, and find that it also aids learning as a regularizer. This is an important step towards swarms which can be realistically deployed in the real world without assuming complete prior knowledge or instantaneous communication at unbounded distances.

Acknowledgments

I would like to thank my advisor, Prof. Katia Sycara for her unwavering support over these two years. She has been a solid mentor to me, and her enthusiasm for good research, especially in multi-agent systems, served as an inspiration to me. Her ability to place trust and responsibility in the hands of her students, allowing them to follow through on their own research ideas while providing support, is rare and very valuable.

I would also like to thank Sumit Kumar, who collaborated with me on the work presented in this thesis, and whose support helped keep the project together in times of failure. I'm grateful to Swaminathan Gurumurthy for his interest in and constructive criticism of my ideas. Apart from that, I would like to thank the many people who I have had the good fortune of working with on various projects, including Prof. Timothy Verstynen, Erik Peterson, Prof. Michael Lewis and Kyle Dunovan. I would like to thank Wenhao Luo and Dana Hughes for their help and support, and other members of my lab for stress-busting conversations.

Finally, I am deeply grateful to my family and friends for always believing in me, even when I did not myself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A multi-agent system can be defined as a loosely-coupled network of agents that interact with each other to collaboratively solve problems or accomplish tasks that are beyond the individual capacities of each agent. Cooperative multi-agent systems find a lot of applications in domains as varied as robotics, distributed control, telecommunications, resource management, decision support systems, economics, operations research etc. The complexity of these tasks often preclude them from being solved with pre-programmed agent behaviors in many dimensions, including but not limited to, inability to craft good heuristics which implement desired behavior, and inability to faithfully model the dynamic and stochastic operational environment.

A reinforcement learning (RL) agent [36] learns by interacting with its environment, receiving a reward or penalty for its actions, and accordingly shaping its behavior, over many such interactions, to maximize the reward it receives. While deep reinforcement learning techniques have contributed to huge successes in recent years [20, 32], multi-agent RL poses an entirely new set of challenges. From the perspective of any one agent, the other agents which are also learning make the environment non-stationary, which in turn requires the agent to adapt its own evolving behavior to theirs, leading to a reward only if all the agents' behaviors converge to produce a meaningful interaction. At the same time, both the joint action and state space of the agents grow linearly (or even combinatorially) in the number of agents, making it difficult for single-agent RL to scale up to a large number of agents. Hence, we require the agents to learn cooperative behavior conditioned only on local observation and communication, where locality could be defined as physical vicinity or other metrics as appropriate, but imposing a constraint on how many agents each agent observes at each time step. This argument is strengthened by the fact that most real world environments will have partial observability (due to limited range and/or noisy sensors) and limited communication (due to latency, finite bandwidth, lost packets), necessitating the use of decentralised policies. Focusing on fully cooperative settings also lets us avoid the thorny (and, we believe, orthogonal) issue of choosing an appropriate opponent to learn and evaluate against in competitive (zero-sum) or general settings.

Recent work on multi-agent deep RL in fully cooperative settings [21, 34, 37] has led to some success in learning cooperative behaviors in simple tasks with two or three agents. Building upon this, Foerster et al. [6], Lowe et al. [17] demonstrated emergence of intelligent behavior with up to 3 agents in a paradigm of centralized training and decentralized execution, where agents have access to global information during training but do not need the same for execution. This

approach was scaled up by Foerster et al. [7], Rashid et al. [27] who showed results with up to 8 agents micromanaging units in StarCraft. Though some of these approaches have modeled multi-agent systems as interaction networks, the environment has traditionally been treated as a black box, with agents still receiving information about all the entities in the environment as a single vector, which is a gross under-utilization of the natural structure present in our environments. For example, an agent might want to focus on obstacles close to it, or landmarks salient to its objective, and an environment representation which provides this flexibility gives a strong inductive bias to the multi-agent learning problem. Additionally, to the best of our knowledge, there has been no study of how well the learned policies generalize and extrapolate to addition or deletion of team members, when such situations have not been seen previously during training.

Hence, in this thesis we propose incorporating this environmental information directly in the learning framework and creating a shared agent-entity graph, making learning more controlled and effective. This approach to multi-agent reinforcement learning capitalizes on the network structure inherent in these domains, and provides a strong inductive bias to the agents' policy, enabling them to learn robust cooperative behavior with large teams. We represent each agent and environmental entity as a node in a graph, with edges between the nodes whose agents/entities can communicate with each other. We then run a convolution operation over this graph [8, 13, 29], which allows us to model multi-agent interactions naturally, and we take the results of this network and feed it into standard feedforward/recurrent policy and value networks for each agent. This shared agent-entity graph, along with suitable message passing functions allow us to set up our learning framework such that it is completely invariant to the number of agents or entities in the environment and can easily generalize and extrapolate learned behavior. Our method works under partial observability, and we present results for both centralized and decentralized communication, setting state-of-the-art results in both settings. To the best of our knowledge, this is the first work to demonstrate emergence of cooperative behavior without assuming availability of full observability and/or global knowledge during training. The agents learn to communicate with each other, learning both what to send as well as how much attention to pay to the messages they receive from their neighbors, with the use of multi-head attention [39]. Finally, we present results on strong zero-shot generalization of learned policies to environments with additional team members, which is then fine-tuned very quickly to further improve performance to the limit.

Chapter 2

Preliminaries

In this chapter we introduce preliminary concepts required to understand this thesis, and expose related prior work to put our work in context.

2.1 Reinforcement Learning

2.1.1 Markov Decision Processes

In the real world, an agent's actions influence not just the immediate rewards it receives, but also the subsequent state of the environment, which in turn influences future rewards. This kind of sequential decision-making process is often formalized as a Markov Decision Process (MDP). An MDP is defined by a tuple $(S, A, P, r, p_0, \gamma)$ where S is a finite set of states, A is a finite set of actions, $P : S \times A \times S \to \mathbb{R}$ is the state transition probability distribution, $r : S \to \mathbb{R}$ is the reward function, $p_0 : S \to \mathbb{R}$ is the probability distribution over the agent's initial state s_0, and γ is the discount factor which determines how much the agent values immediate rewards over future rewards. The name derives from the Markov property, which describes processes whose future state depends only on the present state, not on the sequence of events that preceded it. This can be formalized as $p(s_{t+1}|s_t, a_t, \cdots, s_0, a_0) = p(s_{t+1}|s_t, a_t)$

The policy is a probability distribution from states to actions $\pi : S \times A \to [0, 1]$ which encodes the agent's probability of taking each action in each state. Under policy π, the state value function $V_\pi : S \to \mathbb{R}$, action-value function $Q_\pi : S \times A \to \mathbb{R}$ and advantage function $A_\pi(s_t, a_t)$ are defined as:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots} \sum_{k=0}^{\infty} \gamma^k r(s_{t+k})$$

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \cdots} \sum_{k=0}^{\infty} \gamma^k r(s_{t+k})$$

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

where $a_t \sim \pi(a|s_t), s_{t+1} \sim p(s_{t+1}|s_t, a_t)$, and the subscripts $t, t+1$ etc. refer to time.

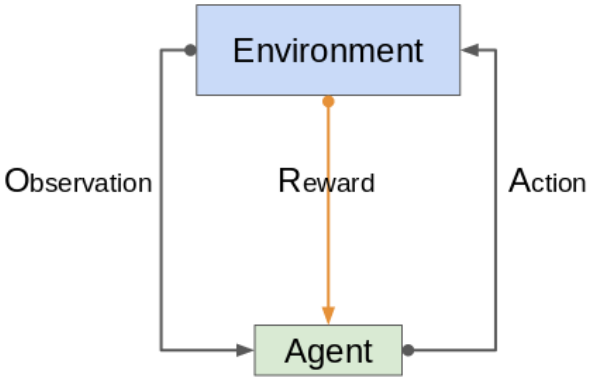

Figure 2.1: The classic reinforcement learning "Observation - Action - Reward" loop. An agent observes its environment and uses this information to act appropriately upon the environment, which in turn transitions to a new state and rewards or penalizes the agent based on the objective at hand. The agent learns to choose its actions such that the long-term discounted reward it receives is maximized.

2.1.2 Policy Gradient methods for learning in MDPs

An agent's objective is to maximize the expected total discounted reward $\sum_{k=0}^{\infty} \gamma^k r(s_{t+k})$. Let the policy π be parameterized by θ. Policy gradient methods [42] usually optimize this by estimating the gradient $g = \nabla_\theta \mathbb{E} \sum_{k=0}^{\infty} \gamma^k r(s_{t+k})$. This has multiple formulations which tradeoff bias and variance in the resulting gradient estimate, however for the purpose of this thesis we will use the following:

$$\hat{g} = \hat{\mathbb{E}} \sum_{t=0}^{\infty} \hat{A}_{\pi,\gamma}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

where $\hat{A}$ is an estimate of the advantage function, using the Generalized Advantage Estimator (GAE) [30], and $\hat{\mathbb{E}}$ denotes the empirical average over a finite batch of samples.

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{2.1}$$

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.2}$$

where γ is the discount factor, and λ is a hyperparameter which contributes to the bias-variance tradeoff. Note that the agent needs to maintain an estimate of the value function V, along with its estimate of the policy π, in order to use this formulation of the policy gradient method.

To improve stability and robustness to hyperparameter settings, we use the PPO algorithm [31] which optimizes a clipped, surrogate policy gradient objective function. This algorithm has subsequently been used in many works [1, 25] for its stability and robustness to hyperparameter settings.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{2.3}$$

$$r_t(\theta) = \frac{\pi_\theta\left(a_t|s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t|s_t\right)} \tag{2.4}$$

where θ_{old} is the vector of policy parameters before the update. The second term inside the $\min$ removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$, preventing potentially harmful major updates to the policy and lending to stability of training. We also add a MSE (L-2) loss function for the value estimate, and add an entropy bonus to encourage exploration. This leads to the final objective function for PPO:

$$L := L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right]\left(s_t\right) \right] \tag{2.5}$$

where c_1, c_2 are coefficients, S denotes the entropy bonus, and L_t^{VF} is a squared error loss $\left(V_\theta\left(s_t\right) - V_t^{\text{targ}}\right)^2$. For completeness, we present the PPO algorithm from Schulman et al. [31].

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize π_θ, V_θ
for iter $= 1, 2, \cdots$ **do**
  Run policy $\pi_{\theta_{\text{old}}}$ in N parallel environments for T timesteps.
  Compute GAE advantage estimates $\hat{A}$ for each timestep in each environment.
  for k $= 1, \cdots, K$ **do**
    Construct L using Eqn. 2.5 and optimize wrt θ, with minibatch size $M \leq NT$.
    $\theta_{\text{old}} \leftarrow \theta$

2.1.3 Markov Games

The standard single-agent RL setting assumes that a single agent acts on the environment, and has to, by definition, treat other agents as part of the environment. Instead, we model multi-agent environments as Markov (or stochastic) games, which are a generalization of MDPs to the multi-agent setting first proposed by Littman [16]. We note that in the fully cooperative case, this formulation is consistent with the decentralized partially observable MDPs (Dec-POMDP) formulation Oliehoek et al. [23].

Defined by a tuple $(S, \mathbf{A}, P, r, \gamma, N, \mathbf{Z}, O)$ where N is the number of agents, S is the set of states, $P : S \times \mathbf{A} \times S \rightarrow [0, 1]$ is the state transition probability function. At each time step, each agent chooses an action $a^i \in \mathbf{A}$, to create a joint action $\mathbf{a} \in \mathbf{A}^n$, and receives an immediate reward from the environment $r^i(s, \mathbf{a}) : S \times \mathbf{A} \rightarrow \mathbb{R}$, along with an observation $z^i \in Z$ according to observation function $O(s, a) : S \times A \rightarrow Z$. Each agent's policy π^i is conditioned on its history of observations and actions, $h^i \in H \in (Z \times A)$. We consider fully cooperative games, where all agents share the same reward function and discount factor.

2.2 Related Work in Multi-Agent RL

Following the success of deep RL in single-agent domains [20, 32], there has been a resurgence in multi-agent RL led by the use of deep neural networks to parameterize agent policies or value functions, allowing methods to tackle high-dimensional state and action spaces. Having looked at single-agent RL in the previous section, we now discuss various formulations of the multi-agent reinforcement learning problem.

2.2.1 Centralized Multi-Agent RL

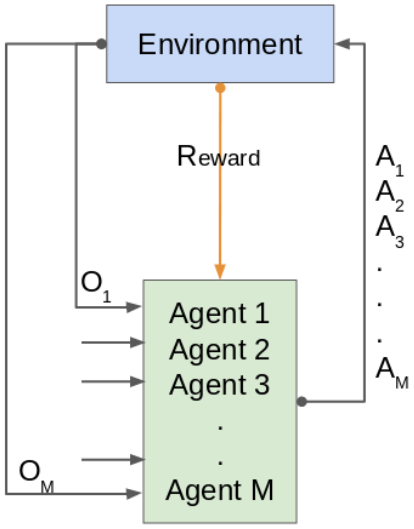

Figure 2.2: Multi-Agent Reinforcement Learning with a Hive Mind (centralized controller)

Centralized multi-agent RL (see Figure 2.2) is the simplest, most obvious interpretation of multi-agent RL. There is a centralized controller, or a hive mind, which receives information from all the agents in the system, and then takes a collective decision about what action each agent should take. Being able to consider a wide variety of information from all the agents and then order each agent around makes this very powerful. It can be thought of as a single-agent system with multiple arms that it can directly control, the same way we have full control over our own two arms. Hence, from a learning perspective, this is effectively single-agent reinforcement learning. The recent paper on AlphaStar by DeepMind [40] which has demonstrated superhuman performance on the challenging multi-agent game of StarCraft treats the problem in such a fully centralized manner. It is worthwhile to note that such a centralized approach is hard to scale to many agents, since the observation and action spaces grow combinatorially, leading to an explosion of the policy search space, meaning that converging to a good policy becomes very difficult, and computationally expensive, often out of reach for academic research labs. Another disadvantage of such a system is that it has a single point of failure. Any event impacting the central controller's ability to command the agents renders all the agents useless. This fragility is obviously undesirable.

2.2.2 Decentralized Multi-Agent RL

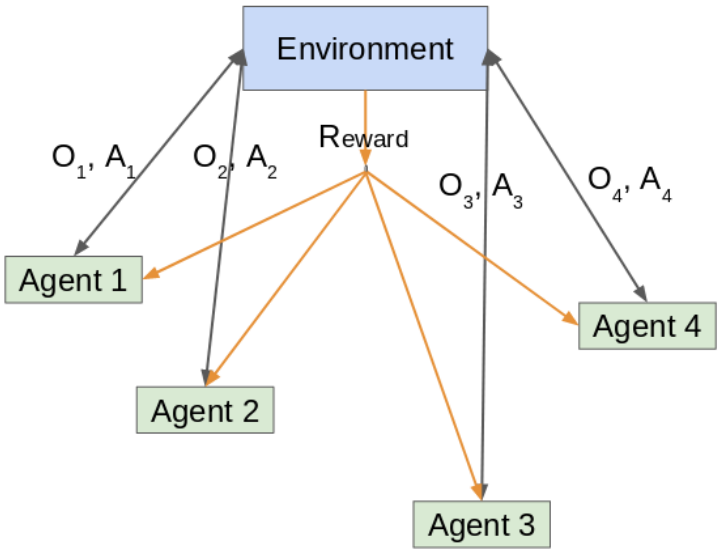

Figure 2.3: Fully decentralized multi-agent reinforcement learning - each agent is an independent decision making entity, relying solely on its own observation O_i to choose an action A_i.

To mitigate these drawbacks, we consider a decentralized setting now, where each agent is its own decision making entity, which receives its own observation from the environment and chooses to act solely on this observation. The observation received by each agent usually consists of information within a certain observability radius of the agent. If there are other agents within that radius, they can observe each other. The benefits of this approach include the absence of a single point of failure, which means that even if anything happens to one agent - the rest can carry on. Decentralization also makes scaling (up to larger teams) easier, since each agent is only observing its local area.

There has been a lot of work in this problem setting, which we now discuss. Independent Q-learning [37, 38] requires training independent Q-value functions for each agent using regular Q-learning [41], while assuming that the other agents are a part of the environment. Since the other agents are also learning, the environment becomes non-stationary and the resulting instability has prevented these methods from working with more than 2 agents in simple settings. Other works like [19, 24] present a formalization of multi-agent decentralized RL under partial observability and also address learning stabilisation, and present results on normal form games or predator-prey games with up to 3 agents. However, these methods do not scale to a larger number of agents. Under the paradigm of centralised learning with decentralised execution, a multitude of recent works have trained actor-critic algorithms where the critic is centralized and makes use of global information during training, allowing the agents to use only the actor network to execute in a fully decentralized manner. MADDPG [17] learns a centralised critic for each agent by providing the actions of all agents to the critics, and train different policies for each agent using the DDPG algorithm. They present results in cooperative and competitive environments with up to 3 agents, and we compare our results with this algorithm in Section 4. COMA

[7] also uses a centralised critic but estimates a counterfactual advantage function that helps with multi-agent credit assignment by isolating the effect of each agent's action. They present results with up to 5 agents in Starcraft unit micromanagement tasks. These fully centralized critics becomes impractical as we increase the number of agents. VDN [35] counter this by decomposing a centralized state-action value function into a sum of individual agent specific functions and present results on 2-agent grid world environments. The decomposition introduced imposes a strict prior which is not well justified, and limits the complexity of the agents' learned value functions. Q-Mix [27] improves upon this by removing the requirement of additive decomposition of the centralised critic, instead imposing a less restrictive monotonicity requirement on agents' individual state-action value functions, and allowing a learnable mixing of the individual functions which does not limit the complexity of functions that could be learned. This paper also presents results with up to 8 agents in a variety of StarCraft unit micromanagement tasks. To the best of our knowledge, all these works using a centralized critic use network architectures which prevent any transfer of learned behavior to environments with greater or fewer number of team members. These papers do not study robustness of learned behavior to addition or deletion of team members, or, more broadly, generalization via extrapolation to any domain not seen during training.

A significant drawback of this approach is that the agents cant actually coordinate with each other during execution - they depend myopically on their own observations. If the agents encounter a situation never before seen during training, there is no mechanism which allows them to adapt to that.

2.2.3 Decentralized Multi-Agent RL with Communication

To ameliorate this significant weakness, we can allow the agents to start communicating with each other, along the red arrows between the agents. This communication is usually learned, and could consist of symbols from a discrete vocabulary or, of vectors of real numbers. While the agents still observe local information, they are able to communicate with each other during execution, which provides a mechanism for them to adapt to scenarios never seen before during training. This communication is often restricted, either due to bandwidth considerations, or to be within a certain radial distance only.

CommNet [34] is one of the earliest works to learn a differentiable communication protocol between multiple agents in a fully cooperative setting, and presented results on interesting cooperative tasks like a traffic junction, however they work with a fully centralized communication architecture and full observability. The paper does not explicitly model interactions between agents, instead each agent receives the averaged states of all its neighbors. VAIN [9] remedied this by using an exponential kernel based attention to weigh the other agents' states, and effectively demonstrated predictive modeling of multi-agent systems using supervised learning. Mordatch and Abbeel [21] also demonstrated emergence of compositional language in multi-agent systems with up to 3 agents in both cooperative and competitive settings, and released a multi-agent particle environment which we make use of in this work. They, however, learned discrete communication using symbols from a limited vocabulary, and made it end-to-end differentiable by using the Gumbel-softmax estimator in a fully centralized approach. ATOC [10] proposes an attentional communication model that learns when communication is needed and dynamically

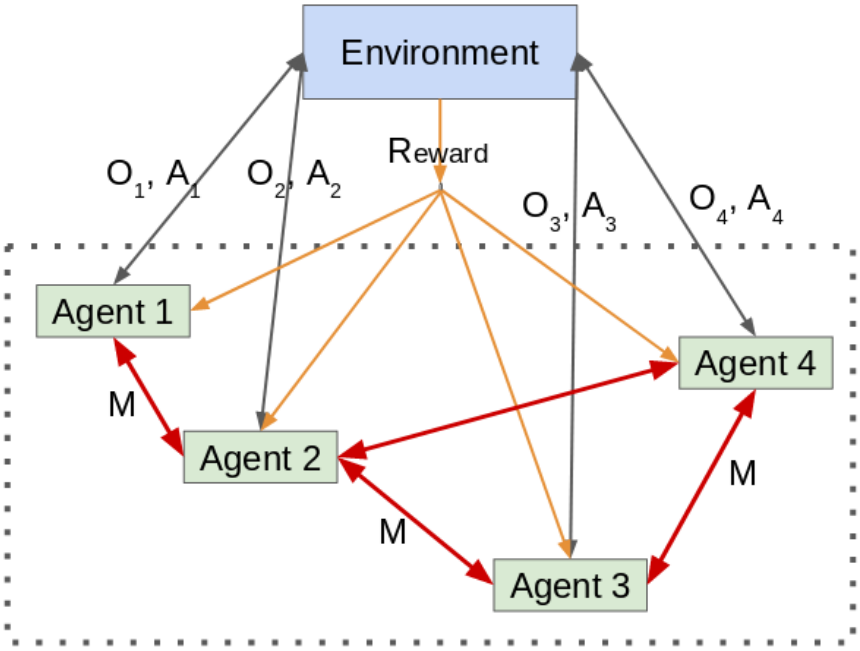

Figure 2.4: Decentralized multi-agent reinforcement learning, where the agents are allowed to communicate with each other. Agent i receives observation O_i, communicates with other agents and gathers information M, and then chooses an action A_i.

creates cliques which can communicate among each other using a bidirectional LSTM unit as the channel.

In work done concurrently to ours, TarMAC [5] uses a scaled dot-product attention mechanism for inter-agent communication, and present results in a 3D indoor navigation task. They do not impose any restrictions on communication, leading to a centralized execution paradigm. DGN [11] also uses a similar mechanism for inter-agent communication, using Q-learning for training. They restrict each agent to communicate with its 3 closest neighbors. From a practical consideration, communication between agents is usually restricted by distance, meaning that an agent can communicate only with neighbors within a certain radius. For example, this is the standard practice in the swarm robotics community. We would also like to emphasize that being able to communicate with the 3 closest neighbors ensures that the agents' graph is always a single connected component and no agent is ever disconnected from the others, while having a distance-based restriction leads to formation of several different connected components in the agents' graph, none of which can communicate with each other - leading to a significantly more difficult learning (to cooperate) problem.

Chapter 3

Methods

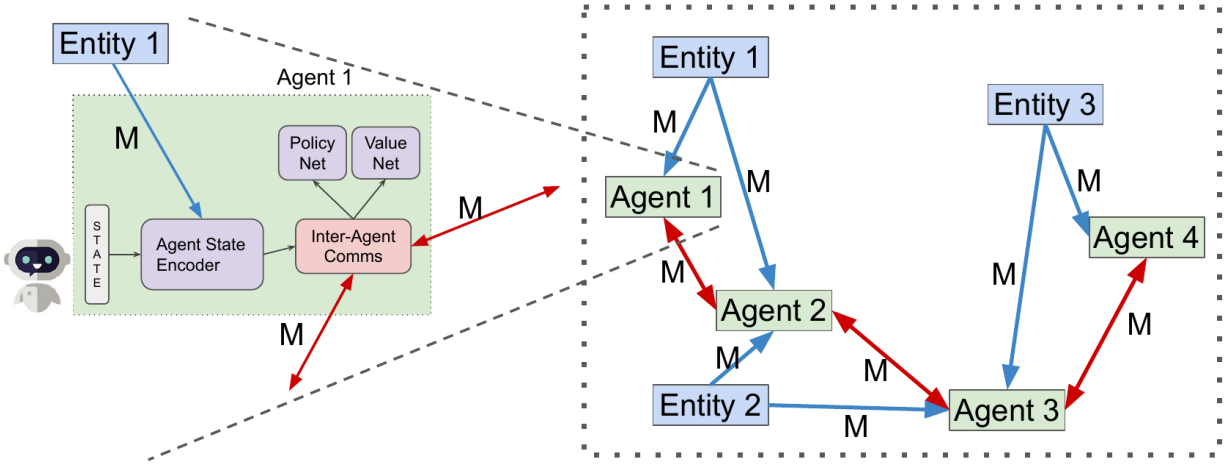

Figure 3.1: The proposed shared agent-entity graph on the right, and a detailed look at the internal architecture of each agent on the left. Agents are in green, environmental entities in blue. Messages are exchanged between agents along the red edges, and are sent from entities to agents along the blue edges. Many agent pairs ($< 1, 3 >, < 1, 4 >, < 2, 4 >$) do not communicate directly, however information from one agent is still able to propagate to others due to the use of multihop communication. The agent state encoder is described in Section 3.3.1, communications in Section 3.2 and policy/value heads in Section 3.3.2.

There are N agents interacting and jointly performing a task, sharing a policy parameterized by π. Agent i encodes its own state (consisting of information like position, velocity etc.) into a H-dim vector, and combines it with information about environmental entities (like landmarks, obstacles etc.) to form an agent embedding H_i. The agents then communicate with their neighbors to update their embeddings with the received messages. This is the crucial step which allows the emergence of coherent cooperative behavior in the team. Finally, this updated state is fed to an actor-critic network, whose policy network is used by the agent to choose an action for the environment. The agents then act simultaneously on the environment, and receive a reward which is used to train, in an end-to-end fashion, the entire network architecture, using reinforcement

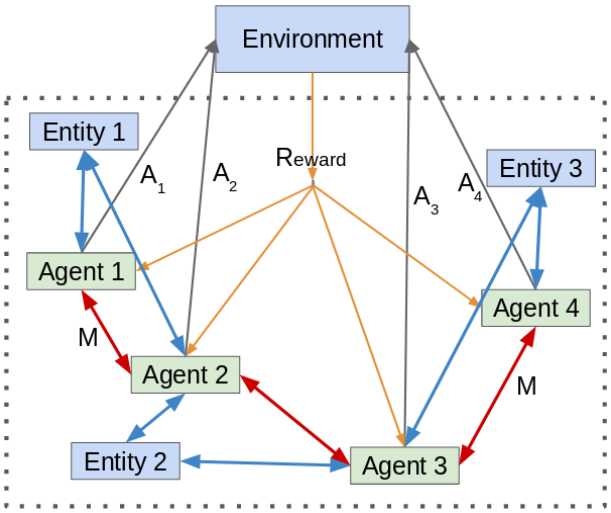

Figure 3.2: Agents and entities embedded in a shared graph, with the agents acting upon and receiving a reward from the environment. This figure should be compared to those in Section 2.2.

learning. In this work, we consider scenarios where all the agents form a homogeneous team and share policy and value network parameters. Since each agent receives different observations (their own state), sharing parameters does not preclude them from behaving differently, as is appropriate.

Figure 3.1 provides a high-level overview of the proposed multi-agent learning framework, with the shared agent-entity graph on the right and the internal architecture of each agent on the left. We now describe each part in detail.

3.1 Agent-Entity Graph

The salient parts of our environment can often be described as a collection of different entities. For a vehicle, this might include traffic lights, pedestrians or obstacles on the road. Any cooperative multi-agent system will also have M agents sharing the environment, who collectively need to accomplish a given objective. We would like each agent to be able to learn to focus selectively on particular entities salient to it (such as obstacles close to it, or objects which align with its goal). Hence, we let the agents receive messages from each entity, where the agent can use attention to selectively focus on all the messages it is receiving.

Hence, we define a graph $G := (V, E)$ where each node $n \in V$ is an agent or an environment entity, and there exists an edge $e \in E$ between two nodes if the agents or entities on those nodes communicate with each other. This can be seen in Figure 3.1. A shared agent-entity graph has these agents and entities occupying vertices of the graph, with edges existing between those vertices whose occupants can communicate with each other. In a real world scenario, communication is often restricted to within a certain radial distance only, as illustrated in Fig. 3.3. In the case there is no restriction on communication radius, this graph is fully connected, whereas

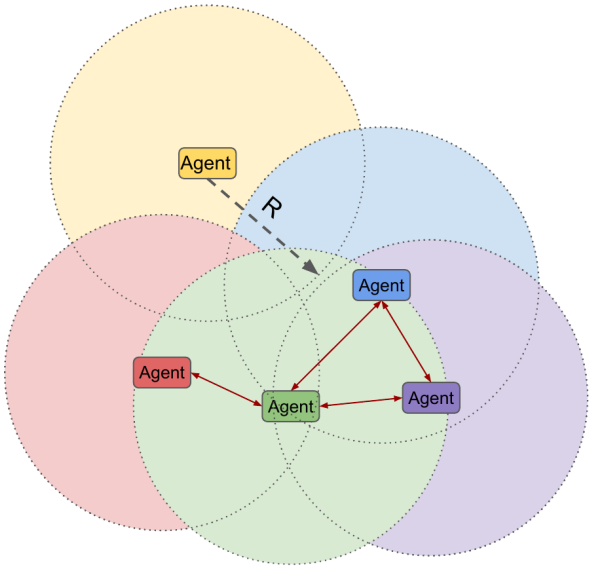

Figure 3.3: Illustration of restricted communication. The circles around each agent indicate its radius of communication R. Note that the yellow agent is disconnected from the rest of the graph, creating multiple disconnected components which can not communicate with one another. This is a situation encountered frequently in reality and the learned behavior needs to be robust to such situations.

in the case that communication is restricted to neighbors (agents within the communication radius of each agent), the graph can have arbitrary connectivity, including multiple disconnected components. Modeling the agent network as a graph provides a very strong inductive bias to the learning algorithm. In practice, agents which are closer to each other have a greater impact on each others' behavior, and this crucial information is baked into the architecture of the graph, which greatly aids learning. Using appropriate message-passing functions in the graph (as discussed below) also enables the learned policies to work in environments with a different number of agents, which is necessary for enabling the agents to learn robust and generalizable strategies. We now discuss what messages are sent and received between vertices on this graph, and how they use that information to make a decision.

3.2 Learning Communication

Communication between agents is crucial for emergence of cooperative behavior. While there has been prior work [2] on goal-oriented communication between two agents grounded in natural language, our focus here is to allow groups of agents to develop a communication protocol, and we do not impose the constraint of natural language grounding.

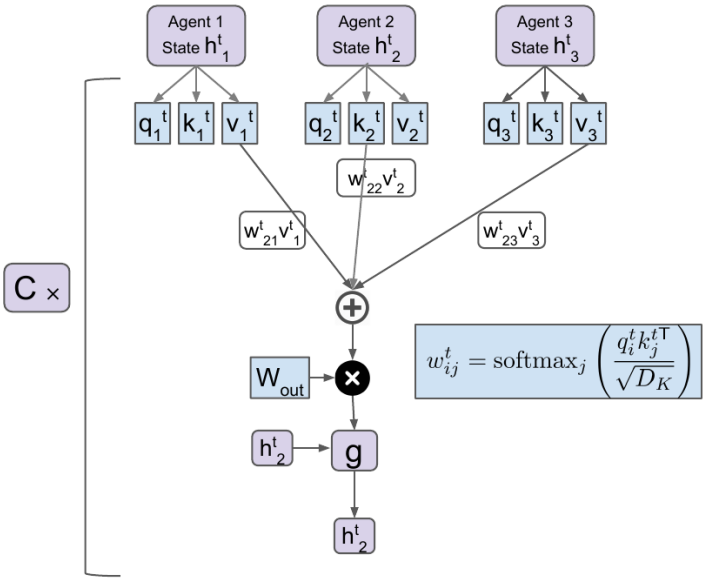

Figure 3.4: Illustration of communication between agents and/or entities, using dot-product attention and multihop routing.

Calculating a message to send

All nodes n in the graph (agents and entities) compute a message v^t_n to send, conditioned on their state h^t_n:

$$v^t_n = W_V h^t_n \in \mathbb{R}^{D_V}$$

where $W_V \in \mathbb{R}^{D_V \times H}$ is a learnable matrix.

Aggregating received messages

Only agents receive messages. Agent m receives multiple messages from its neighbors in the graph and aggregates them using a weighted linear combination:

$$M^t_m = \sum_n \left(\frac{\exp\left(w^t_{mn}\right)}{\sum_n \exp\left(w^t_{mn}\right)}\right) v^t_n$$

where n indexes over all agents (including m), and the `softmax` normalization of the weights w^t_{mn} is done to ensure invariance to the number of agents. We now discuss how these weights are calculated.

Calculation of attention weights w^t_{mn}

We use the dot-product multi-head attention mechanism proposed by Vaswani et al. [39] to calculate the weight (or attention) that each agent assigns to each message received by it. This weight is a function of the internal state of the agent sending the message and the agent receiving the message. This provides us with two benefits: (1) allows each agent to learn to selectively attend

to entities that are more relevant to it, and (2) ensures invariance and robustness to number of entities in the environment, since the `softmax` operation in the attention calculation normalizes over the number of agents or entities.

More specifically, for each vertex in the graph, we calculate a key-value pair $< k_n^t, v_n^t >$ where

$$k_n^t = W_K^{AE} e_n^t \in \mathbb{R}^{D_K}$$
$$v_n^t = W_V^{AE} e_n^t \in \mathbb{R}^{D_V}$$

where $W_K^{AE} \in \mathbb{R}^{D_K \times H}$ and $W_V^{AE} \in \mathbb{R}^{D_V \times H}$. Each agent vertex m also calculates a query vector $q_m^t = W_Q^{AE} a_m^t \in \mathbb{R}^{D_K}$, where $W_Q^{AE} \in \mathbb{R}^{D_K \times H}$ and a_m^t is the state of the vertex at time t. Intuitively, W_K^{AE} and W_Q^{AE} are learnable matrices which embed the states of the two agents (or entity-agent pair) sending and receiving the message, respectively, into a shared embedding space where we can use the dot-product between the two embeddings (key and query, respectively) to compute their similarity. W_V^{AE} is also a learnable matrix which computes a message (value) to be sent from vertex n to the vertices it is connected to (able to communicate with).

$$w_{mn}^t = \begin{cases} \left(\frac{q_m^t k_n^{t\mathsf{T}}}{\sqrt{D_K}}\right) & \text{if m and n are connected} \\ -\infty & \text{otherwise} \end{cases}$$

where the scaling factor $\frac{1}{\sqrt{D_K}}$ is added for stability [39] and j indexes over all the agents or entities in the environment. This computation is also depicted in Figure 3.4.

We note that our shared agent-entity graph is also compatible with two previously proposed functions for inter-agent communication, which we specify below for completeness:
Exponential Kernel Attention (VAIN [9]): Each agent computes a key value $k_n^t \in \mathbb{R}^1$:

$$k_n^t = W_K h_n^t \in \mathbb{R}^1$$

We then calculate the L_2 distance between the key values of agents m and n to use as the attention weight w_{mn}^t:

$$w_{mn}^t = \begin{cases} -||k_m - k_n||^2 & \text{if m and n are connected} \\ -\infty & \text{otherwise} \end{cases}$$

Uniform Attention (CommNet [34]): The agent focuses uniformly on all the messages it receives, i.e.

$$w_{mn}^t = \begin{cases} 1 & \text{if m and n are connected} \\ -\infty & \text{otherwise} \end{cases}$$

Agent State Update

The aggregated message M_m^t is now passed through the agent update function g, which takes as input the concatenation of the current agent state h_m^t and the aggregated message, to update the agent state.

$$h_m^t \leftarrow g(h_m^t | M_m^t)$$

where $|$ denotes concatenation. We emphasize that the matrices W_K, W_Q, W_V and function g are shared (have the same value) across all agents.

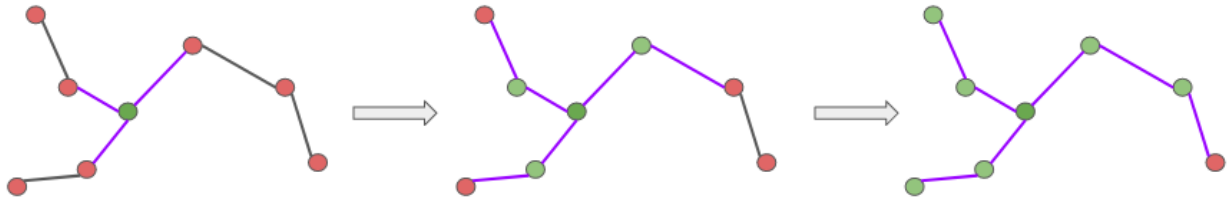

Figure 3.5: An illustration of information propagation through multi-hop communication. In the first step, the central node has some information (green) which it has to propagate to the other nodes in the graph. In this step, the edges connected directly to the green node carry this information. In the second step, we see that the nodes directly connected to the central node have received this information (have turned green), and now all the edges connected to any green node carry this information. In the third step, we see that nodes which are **not directly connected** to the central node have also received the information (turned green). This is the basic premise of multi-hop communication, where information can pass between nodes that are not directly connected to each other.

Multi-Hop Communication

Since the agent graph is sparsely connected, we use multihop (or peer-to-peer) communication to allow information to propagate between agents that might not be directly connected with each other. At each timestep t, the message passing algorithm outlined above is repeated C times, after which the hidden state of each agent (potentially) incorporates information from agents up to C hops away. The premise of multi-hop communication is explained in Figure 3.5

3.3 Internal Agent Architecture

3.3.1 Agent State Encoder

At each timestep t, agent i observes its local state $x_i^t \in \mathbb{R}^{D_I}$, which consists of information like its own position and velocity, and provides it as input to the agent state encoder $q : \mathbb{R}^{D_I} \to \mathbb{R}^H$, to produce an embedding $a_i^t \in \mathbb{R}^H$.

$$a_i^t = q(x_i^t)$$

3.3.2 Policy and Value Output Heads

As discussed in Section 2.1.2, actor-critic reinforcement learning algorithms require the agent to produce (1) a probability distribution over actions, and (2) a value estimate of the current state. Hence, each agent is equipped with a policy network and a value network. These networks both take as input the agent state h_i^t obtained after C rounds of communication with the other agents. This input contains information about other agents' expected behavior which is crucial for correct estimation of value and producing a policy distribution which leads to a high probability of

coherent and cooperative behavior. While each agent's policy and value network can be independent in a scenario where each agent is unique and heterogeneous, a more likely scenario is that agents have specific roles, for e.g., medic, firefighter and police (in a fire response scenario), and all the agents in a particular role share policy and value networks, since they would be expected to behave interchangeably, which desirable behavior results directly from sharing parameters. In this document, we consider only scenarios where all the agents involved are homogeneous, and hence share policy and value network parameters. Since each agent receives different observations (their own state), sharing parameters does not preclude them from behaving differently, as is appropriate.

3.4 Training

3.4.1 Multi-Agent PPO

We adapt the Proximal Policy Optimization (PPO) [31] actor-critic algorithm for multiple agents. During training, we collect the following data from all agents at each timestep:

1. Local state and environment observation
2. Action taken
3. Log probability value of the policy head for the action chosen
4. Reward received from the environment
5. The agent's value estimate for that state
6. Did the episode end? (True/False)

The reward and value estimates are later used to calculate returns and advantage estimates for each agent, using Generalized Advantage Estimation [30], as described in Section 2.1.2. Then, we sample a mini-batch of the collected data for the same time steps for all agents, and use that to do a gradient update. Since PPO updates are stable, we are able to use the same experience for multiple updates, improving sample efficiency.

3.4.2 Dropout Communication

Real world communication networks are prone to glitches, with lost or corrupted information packets being a regular occurrence. Usually, these dropped connections are not just momentary but do persist for a short duration of time.

To operate robustly in the real world, agents have to learn to not be too dependent on continuous communication with their neighbors. We induce this behavior by exposing them to artificially severed communications during training. A fraction p of edges in the agent graph are dropped at each timestep, with the particular edges to be dropped being randomly sampled every K timesteps. This technique also draws inspiration from the dropout technique [33] used to reduce overfitting and co-adaptation between neurons in deep neural networks, since our objective is also to reduce co-dependence between agents in the agent graph.

3.4.3 Curriculum Training

We make use of curriculum learning [4] to train policies in progressively tougher environments as the number of agents increases. Our graph network framework which is adaptable to varying number of agents and landmarks allows us to transfer learned behavior, and shows both strong zero-shot generalization as well as ability to quickly improve in new scenarios with very few gradient updates. We deploy a curriculum over the number of agents, i.e. a policy is trained with $M = 3$ agents, then transferred (and further trained) with $M = 5$ agents, and so on.

This allows us to vastly simplify the multi-agent credit assignment problem which arises with a large number of agents in cooperative teams, where it is not clear which agent's behavior might be leading to a reward or penalty from the environment. In the 3 agent case, the credit assignment problem is not very difficult - and then the use of a curriculum when scaling to larger teams makes sure that the agents always have a very strong policy to bootstrap from.

Chapter 4

Experiments and Results

4.1 Environment Description

We present results on two standard swarm robotics tasks [3, 22]: coverage control and formation control. These are implemented inside the multi-agent particle environment [17], which is a 2-D world consisting of agents and landmarks in a continuous space with discrete time.

Observation Space

At each time step t, each agent observes only its own state $x_i^t \in \mathbb{R}^4$, comprising of 2D position and velocity. At the beginning of each episode, the agents also observe the positions of all entities (landmarks) in the environment, which are embedded as virtual nodes in the agent graph and used as described above.

Action Space

The agent has a discrete action space and chooses from 5 possible actions: (1) no-op, (2) $a_x = 1$, (3) $a_x = -1$, (4) $a_y = 1$, (5) $a_y = -1$ where a_x, a_y represent accelerations in x and y dimensions, respectively, and $\mathbf{a}^t = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$ is used as input to the agent dynamics as shown below.

Agent Dynamics

Agents can move around in the 2-D space, and we provide a state space representation of the agent transition dynamics, where the state is denoted by $\mathbf{y}$

$$\mathbf{y}_i^t = \begin{bmatrix} \mathbf{p} \\ \mathbf{p}' \end{bmatrix}_i^t = \begin{bmatrix} \mathbf{p}^{t-1} + \mathbf{p}' \Delta t \\ \gamma \mathbf{p}'^{t-1} + \mathbf{a}^t \Delta t \end{bmatrix}$$

where $\mathbf{p} \in \mathbb{R}^2$ is the 2-D position coordinates of the agent and $\mathbf{p}' \in \mathbb{R}^2$ is the 2-D velocity of the agent. $\gamma = 0.5$ is a damping coefficient used in double integrator robot dynamics models in the literature [26]. The agent chooses action $\mathbf{a}^t$ at each time step t, and runs the simulation for $\Delta t = 0.1$s. We do not model collision dynamics.

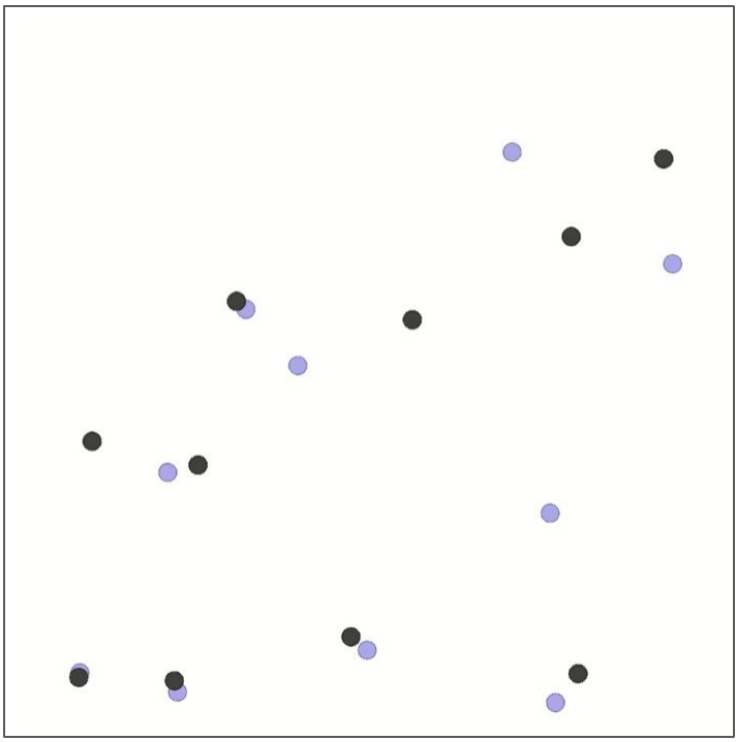

Figure 4.1: Coverage Control: 10 agents and 10 landmarks. Some landmarks have been covered, others have an agent on the way to cover them.

4.2 Task Description

We now describe both the coverage and formation control tasks.

4.2.1 Coverage Control

There are M agents (blue) and M landmarks (grey) in the environment, see Figure 4.1. The objective is for the agents to deploy themselves in a manner to ensure that each landmark is covered, i.e. each landmark has at least one agent within a certain distance of it. The size of the environment is 4×4 unit square, and all the agent and landmark positions are randomly initialized at the beginning of each episode, so the learned policy does not depend on any particular configuration of the landmarks. Note that we **do not assign particular landmarks to each agent**, as is the standard practice in the swarm robotics literature, but instead let the agents communicate with each other and develop a consensus about which agent will cover which landmark.

Reward Function

At each time step t, we consider an agent-landmark bipartite graph, where the agents form one set of nodes A and the landmarks for the other set of nodes L. There are weighted edges from every node in $A \to L$, with edge weights being specified by the L_2 distance. We then find the minimum weight matching, which is a set of edges without common vertices such that the combined weight

of the edges is minimized. Intuitively, this finds the agent closest to each landmark and considers the distance between this pair, while ensuring that the same agent does not count as being the closest agent to two different landmarks (i.e., we do not want the same agent to be able to cover two different landmarks at the same time). We use the Hungarian algorithm [15] for linear sum assignment to find this minimum weight matching. Finally, we use the negative of the clipped mean distance of a landmark to its closest agent (as specified by the linear assignment) as a shared reward function for the team. This incentivizes the team to cooperatively cover all landmarks.

Goal Specification

The task is episodic, and can terminate for one of the following two reasons:

1. Each landmark has at least one agent within a distance of 0.1 units from it, where the overall size of the environment is 4×4 unit square. This is a success case.
2. 50 time steps have lapsed since the beginning of the episode without the agents having completed the task. This is a failure case.

4.2.2 Formation Control

There are M agents (blue) and 1 landmark (grey) in the environment, see Figure 4.2. The objective is for the agents to deploy themselves into a M-sided regular polygonal formation, with the

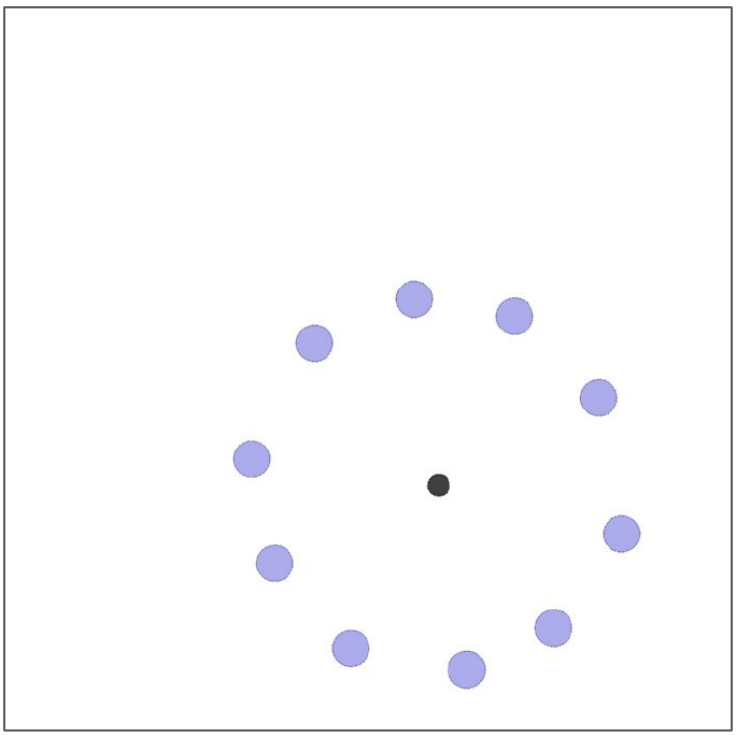

Figure 4.2: Formation Control: 10 agents coalescing to form a 10-sided regular polygon around the landmark.

landmark at its centre. The size of the environment is 2×2 unit square, and all the agent and landmark positions are randomly initialized at the beginning of each episode. Again, note that we do not specify any particular orientation to the polygon, or assign agents to specific points along that polygon - but instead let the agents communicate with each other to spread out uniformly.

Reward Function

The reward provided by the environment to the agents at each timestep is a sum of two terms: a radial separation reward, and an angular separation reward.

1. **Radial Separation:** We specify a desired distance (0.5 units) that all agents should be from the landmark, and provide a clipped negative of the absolute value of the deviation from that distance as a reward to each agent. This term ensures that all agents are at the correct distance from the landmark, however it does nothing to spread them out uniformly.
2. **Angular Separation:** Agents should be spread out at an angle of $\frac{2\pi}{N}$ to create a regular polygon (N is the number of agents). We penalize the agents for deviating from this angle, ensuring that they spread out at the appropriate angle. We scale this term by $\frac{1}{\pi}$ so that both radial and angular separation terms have similar magnitude.

Goal Specification

The task is episodic, and can terminate for one of the following two reasons:

1. Each agent is within ± 0.05 units of the target distance from the landmark, **AND** within ± 0.1 radians of the target angular separation. This is a success case.
2. 50 time steps have lapsed since the beginning of the episode without the agents having completed the task. This is a failure case.

4.3 Experimental Specifications

Network Architecture

1. Embedding dimension $H = 128$
2. Agent state encoder, q: Single fully-connected layer (FC): $\mathbb{R}^4 \rightarrow \mathbb{R}^{128}$, followed by a rectified linear unit (ReLU) non-linearity
3. Entity state encoder, r: Single FC layer + ReLU, $\mathbb{R}^2 \rightarrow \mathbb{R}^{128}$
4. Environment encoder $D_K, D_V, D_Q = 128$
5. Inter-agent communication $D_K, D_V, D_Q = 128$ for multi-head attention based communication with $C = 3$ hops.
6. Agent state update function, g: Single fully-connected layer (FC) layer $\mathbb{R}^{256} \rightarrow \mathbb{R}^{128}$
7. Value output head: FC layer + ReLU + FC layer, overall $\mathbb{R}^{128} \rightarrow \mathbb{R}^1$
8. Policy output head: FC layer + ReLU + FC layer, overall $\mathbb{R}^{128} \rightarrow \mathbb{R}^5$

All parameters are initialized using orthogonal initialization [28].

Hyperparameters

1. For dropout communication, we drop 50% edges which are randomly resampled every 10 timesteps.
2. We restrict communication to a radius of $R = 2$ units in the coverage control task, where arena size is 4×4, and to $R = 1$ unit in the formation control task, where arena size is 2×2.
3. During training, agents collect experience from 32 environments in parallel, all initialized with different seeds
4. Each evaluation is in a newly initialized environment with different random seeds, following best practices for evaluation of reinforcement learning [18].
5. We use the Adam optimizer [12] with learning rate $1e - 4$.
6. Future rewards are discounted with $\gamma = 0.99$
7. The GAE parameter from Equation 2.1 $\lambda = 0.95$
8. Entropy coefficient c_2 from Equation 2.5 is 0.01
9. Value loss coefficient c_1 from Equation 2.5 is 0.5
10. Gradients were clipped to keep L_2 norms below 0.5
11. Each batch of experience was used for 4 PPO updates
12. PPO clipping parameter ϵ from Equation 2.3 is 0.2

The entropy coefficient and value loss coefficient values were obtained after minimal hyperparameter tuning on coverage control, while the other values were chosen based on previous work and publicly available implementations of RL algorithms. Note that we did not need to tweak any parameters for formation control. [14].

4.4 Results

We evaluate various learning methods on the following metrics:

1. **Success Rate** $(\%)$: In what percentage of episodes does the team achieve its objective? (Higher is better)
2. **Time**: How many time steps does the team require to achieve its objective? (Lower is better)
3. **Average Distance**: (for coverage control only) what is the average distance of a landmark to its closest agent?

For evaluation, we pause each training run after 50 update steps and evaluate on 30 episodes in a newly seeded environment, with each agent performing greedy decentralized action selection.

We present results of our graph network based learning framework on both tasks, with $N = \{3, \cdots, 10\}$ agents, using the following abbreviations while reporting results with all 3 inter-agent communication functions discussed in Section 3.2: `MHA`: Multi-Head Attention, `EXP`: Exponential Kernel Attention [9], `UNIFORM`: Uniform Attention [34]. We also present

results in 2 communication settings: (1) Unrestricted Communication (UC), where all agents can communicate with each other, and (2) Restricted Communication (RC), where agents can only communicate with other agents that are within their radius of communication ($R = 1$ unit distance).

4.4.1 Comparisons with previous work

While we could not find previous work on multi-agent reinforcement learning in coverage or formation control and hence do not have previously published results to compare with, we use publicly available implementations[1] to compare with Q-Mix [27], VDN [35], IQL [37] and MADDPG [17]. These methods rely on access to global state during training (and local state during execution), instead of inter-agent communication, for emergence of cooperation. Hence, at first, we restricted each agent to observe other agents only within a certain radius (while the global state with all agents and landmarks was available during training). However, none of the methods learn to complete the task (0% success), hence we relaxed the observability to be global for all agents - after which Q-Mix got to a 20% success rate, with the other methods still unable to learn to cooperate successfully, see Table 4.1. By contrast, our methods easily outperforms previously proposed methods even with partial observability and restricted communication. We emphasize that the invariance of our method to the number of agents allows us to use curriculum learning and transfer learned policies from 3 to 6 agents, while the previous works use architectures that do not allow such transfer. Hence, in the presented results, with $M = 6$ agents, our method has been bootstrapped from the $M = 3$ case, while the previous works have been trained from scratch. However, considering the poor performance of those methods in the $M = 3$ case, we do not believe this to be significantly affecting the results.

None of the previously proposed methods learned anything on the formation control task, in the partial observability/restricted communication setting, or with $M \geq 6$ agents, hence we do not report those results in the table.

Table 4.1: Comparisons with previous work on coverage control with $M = 3$ and $M = 6$ agents. `Av.Dist.` refers to the average distance of a landmark to its closest agent (lower is better).

Method	Observ-ability	Comms	$M = 3$ Av. Dist.	$M = 3$ S%	$M = 6$ Av. Dist.	$M = 6$ S%
MADDPG[2]	Full	N/A	0.59	0	0.56	0
Q-Mix	Full	N/A	0.19	20	2.87	0
VDN	Full	N/A	0.41	0	0.46	0
IQL	Full	N/A	0.351	0	0.529	0
Ours: MHA	Partial	RC	**0.067**	**91**	0.103	55
Ours: Exp	Partial	RC	0.074	85	**0.078**	**73**
Ours: Uniform	Partial	RC	0.082	84	0.11	53

[1] `https://github.com/oxwhirl/pymarl`

4.4.2 Flexibility in message passing

Having established that our proposed method with the shared agent-entity graph and learned communication clearly outperforms previously proposed MARL algorithms, we now compare variations of the proposed method. Table 4.2 presents the performance of the 3 different inter-agent message passing functions in both the restricted and unrestricted communication scenarios, as well as the coverage and formation control tasks. It is clear that all 3 functions perform very well, with no function being clearly the best (or worst) in all scenarios.

Using multi-head attention (MHA) for message passing outperforms exponential kernel and uniform attention, since it gives agents greater selectivity in choosing which neighbors' messages to attend to. We note that MHA has a greater advantage in the unrestricted communication settings, where agents are receiving a large number of messages (from all other agents, rather than only from those within a neighborhood) and hence find the ability to listen selectively more useful, as compared to the restricted communication setting, where agents receive fewer messages, hence ameliorating the effectiveness of MHA.

Table 4.2: Comparing (1) Multi-Head Attention (MHA), (2) Exponential Kernel Attention (Exp) and (3) Uniform Attention (Uniform) for inter-agent communications on coverage and formation control task, with both unrestricted (UC) and restricted (RC) communication, for $M = \{3, 5, 6, 10\}$ agents.

Task	M	Comms	MHA		Exp		Uniform	
			S%	Time	S%	Time	S%	Time
Coverage	3	UC	96	24.14	83	29.94	89	26.14
Coverage	5	UC	100	18.41	96	23.90	75	33.01
Coverage	6	UC	**99**	**20.03**	87	26.15	53	41.20
Coverage	10	UC	**98**	**19.84**	80	29.90	0	50
Coverage	3	RC	91	27.15	85	28.44	84	27.8
Coverage	5	RC	79	33	78	32.69	74	33.84
Coverage	6	RC	55	39.77	**73**	**35.98**	53	40.52
Coverage	10	RC	0	50	0	50	0	50
Formation	3	UC	99	14.21	94	17.98	90	18.63
Formation	5	UC	99	16.55	79	26.76	76	28.56
Formation	6	UC	**99**	**19.7**	79	28.86	77	29.53
Formation	10	UC	0	50	**61**	**36.75**	33	43.25
Formation	3	RC	89	20.74	96	17.38	96	16.49
Formation	5	RC	98	17.78	0	50	90	21.35
Formation	6	RC	**97**	**20.52**	0	50	66	35.55
Formation	10	RC	8	47.69	0	50	**61**	**35.13**

Table 4.3: Zero Shot Generalization: What happens when up to two team members are added or removed from the team? S% is success rate, and R is the average reward received by the agents at the final step of each evaluation episode. The policies were trained with N agents, and the 2 columns marked $N-1$ and $N-2$ indicate performance when 1 and 2 agents, respectively, are removed, while columns marked $N+1$, $N+2$ indicate performance when 1 and 2 agents, respectively, are added.

TASK	COMMS	N-2		N-1		N		N+1		N+2	
		S%	R	S%	R	S%	R	S%	R	S%	R
COVERAGE	UC	99	-0.57	97	-0.52	96	-0.47	78	-0.36	39	-0.29
COVERAGE	RC	43	-0.839	60	-0.629	79	-0.59	29	-0.46	6	-0.48
FORMATION	UC	0	-0.34	0	-0.26	100	-0.43	0	-0.36	0	-0.47
FORMATION	RC	0	-0.33	0	-0.26	97	-0.52	0	-0.36	0	-0.47

4.4.3 Zero-Shot Generalization to Additional Team Members

This subsection looks at what happens when agents are removed or added to the team. Table 4.3 presents results of how policies learned using our framework (using MHA for inter-agent communication) generalize to scenarios when up to 2 team members are added or removed. The presented results are with **no additional fine-tuning** on the new scenario.

In the coverage task we see the agents adapting very well to the removal or addition of team members, especially in the unrestricted communication case - achieving up to 99% accuracy after the loss of 2 teammates. In the formation task, while zero-shot transfer success rates are 0%, this is because our evaluation of success is a very strict condition. The reward values show us that there is significant transfer of learning, and the policies are very quickly able to adapt to the new scenarios (with only a few gradient updates). We emphasize that this generalization is a direct result of our proposed learning framework, not a result of having exposed the agent to such diversity (in the number of agents) during training (which is the usual practice in multi-task learning).

4.4.4 Robustness to Communication Drops

We now look at the performance of learned policies under conditions where the communications networks are glitchy. To encourage robustness to such problems, we trained the agents with dropout communication (Section 3.4.2). In both coverage and formation control, agents trained with dropout communication remain robust to random packet loss. More interestingly, we also observed an (unexpected) regularization effect, where training with communication dropout actually improved learning to be better than in the case with perfect communication! This can be seen in Table 4.4, where the success rate of formation control with $M = 10$ agents, in both unrestricted and restricted communication settings, goes from low single-digit success rates to near-perfect (100%) success rates. Both our technique, and this regularization effect, are similar to dropout [33] used for reducing overfitting in deep neural networks.

Table 4.4: Regularization effects of communication dropout

TASK	M	COMMS	DROPOUT	SUCCESS %	TIME
FORMATION	10	UC	NO	0	50
FORMATION	10	UC	YES	97	22.85
FORMATION	10	RC	NO	8	47.69
FORMATION	10	RC	YES	100	20.95

Chapter 5

Conclusion

This thesis presented a new method for cooperative multi-agent reinforcement learning. We propose embedding the agents and environmental entities into a shared agent-entity graph, which allows the agents to communicate with each other and selectively focus on aspects of the environment salient to them. We propose the use of multi-head attention for communication between these agents, and our state of the art results on swarm coverage and formation control tasks for swarms in a fully decentralized execution framework show that our method easily outperforms previous work in this domain, learning cooperative behavior with teams of up to 10 agents. We also show strong zero-shot generalization (without any fine-tuning) of the learned behavior policies to scenarios with more and fewer agents, which indicates robustness to addition or removal of agents. Additionally, introducing dropout communication during training allows the learned behavior to be robust to communication glitches and random packet loss, a common occurrence for real-world communication networks. Surprisingly, dropout communication also acts as a regularizer for learning, with policies trained using it *outperforming* policies trained with perfect communication.

Future Work

We identify the following areas of immediate interest for future exploration:

1. Investigate robustness to measurement and control noise: We have assumed perfect observability and controls without any noise, an assumption which will almost certainly be violated in the real world. Investigating the robustness of the methods to both measurement and control noise would be valuable.
2. Have agents with specialized roles: In this work, all the agents were homogeneous and entirely interchangeable, which resulted in the agents sharing policy and value head parameters. However, each agent's policy and value network can be independent in a scenario where each agent is unique and heterogeneous. A likely scenario is that agents have specific roles, for e.g., medic, firefighter and police (in a fire response scenario), and all the agents in a particular role share policy and value networks, since they would be expected to behave interchangeably, which desirable behavior results directly from sharing parameters. Exploring the emergence of cooperative behavior in such heterogeneous teams is of great interest to the community.

3. Investigate the use of meta-learning to improve zero shot generalization: Our results on zero-shot generalization to scenarios with additional or fewer teammates are a direct byproduct of the shared agent-entity graph, which is invariant to the number of agents and/or entities in the environment. The use of meta-learning for improved zero-shot generalization to out of distribution scenarios should be explored.

Bibliography

[1] Akshat Agarwal, Ryan Hope, and Katia Sycara. Challenges of context and time in reinforcement learning: Introducing space fortress as a benchmark. *arXiv preprint arXiv:1809.02206*, 2018. 2.1.2

[2] Akshat Agarwal, Swaminathan Gurumurthy, Vasu Sharma, Michael Lewis, and Katia Sycara. Community regularization of visually-grounded dialog. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada*. IFAAMAS, 2019. 3.2

[3] Tucker Balch and Ronald C Arkin. Behavior-based formation control for multirobot teams. *IEEE transactions on robotics and automation*, 14(6):926–939, 1998. 4.1

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009. 3.4.3

[5] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Michael Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. *arXiv preprint arXiv:1810.11187*, 2018. 2.2.3

[6] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016. 1

[7] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 1, 2.2.2

[8] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017. 1

[9] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017. 2.2.3, 3.2, 4.4

[10] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, pages 7265–7275, 2018. 2.2.3

[11] Jiechuan Jiang, Chen Dun, and Zongqing Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *arXiv preprint arXiv:1810.09202*, 2018. 2.2.3

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1

[14] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018. 4.3

[15] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 4.2.1

[16] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994. 2.1.3

[17] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017. 1, 2.2.2, 4.1, 4.4.1

[18] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. 4

[19] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE, 2007. 2.2.2

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 1, 2.2

[21] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 1, 2.2.3

[22] Sasanka Nagavalli, Nilanjan Chakraborty, and Katia Sycara. Automated sequencing of swarm behaviors for supervisory control of robotic swarms. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2674–2681. IEEE, 2017. 4.1

[23] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016. 2.1.3

[24] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017. 2.2.2

[25] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018. 2.1.2

[26] Venkatesh G Rao and Dennis S Bernstein. Naive control of the double integrator. *IEEE Control Systems Magazine*, 21(5):86–97, 2001. 4.1

[27] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018. 1, 2.2.2, 4.4.1

[28] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 4.3

[29] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1): 61–80, 2009. 1

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 2.1.2, 3.4.1

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2.1.2, 2.1.2, 3.4.1

[32] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 1, 2.2

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 3.4.2, 4.4.4

[34] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016. 1, 2.2.3, 3.2, 4.4

[35] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017. 2.2.2, 4.4.1

[36] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998. 1

[37] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017. 1, 2.2.2, 4.4.1

[38] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993. 2.2.2

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 1, 3.2

[40] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jader-

berg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019. 2.2.1

[41] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. 2.2.2

[42] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 2.1.2