LEARNING MULTI-AGENT COMMUNICATION THROUGH STRUCTURED ATTENTIVE REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning communication via deep reinforcement learning has recently been shown to be an effective way to solve cooperative multi-agent tasks. However, learning *which* communicated information is beneficial for each agent's decision-making remains a challenging task. In order to address this problem, we introduce a fully differentiable framework for communication and reasoning, enabling agents to solve cooperative tasks in partially-observable environments. The framework is designed to facilitate explicit reasoning between agents, through a novel memory-based attention network that can learn selectively from its past memories. The model communicates through a series of reasoning steps that decompose each agent's intentions into learned representations that are used first to compute the relevance of communicated information, and second to extract information from memories given newly received information. By selectively interacting with new information, the model effectively learns a communication protocol directly, in an end-to-end manner. We empirically demonstrate the strength of our model in cooperative multi-agent tasks, where inter-agent communication and reasoning over prior information substantially improves performance compared to baselines.

1 INTRODUCTION

Communication is one of the fundamental building blocks for cooperation in multi-agent systems. The ability to effectively represent and communicate information valuable to a task is especially important in multi-agent reinforcement learning (MARL). Apart from learning what to communicate, it is critical that *agents learn to reason based on the information communicated to them by their teammates*. Such a capability enables agents to develop sophisticated coordination strategies that would be invaluable in application scenarios such as search-and-rescue for multi-robot systems (Li et al., 2002), swarming and flocking with adversaries (Kitano et al., 1999), multiplayer games (*e.g.*, StarCraft, (Vinyals et al., 2017), DoTA, (OpenAI, 2018)), and autonomous vehicle planning, (Petrillo et al., 2018)

Building agents that can solve complex cooperative tasks requires us to answer the question: *how do agents learn to communicate in support of intelligent cooperation?* Indeed, humans inspire this question as they exhibit highly complex collaboration strategies, via communication and reasoning, allowing them to recognize important task information through a structured reasoning process, (De Ruiter et al., 2010; Garrod et al., 2010; Fusaroli et al., 2012). Significant progress in multi-agent deep reinforcement learning (MADRL) has been made in learning effective communication (protocols), through the following methods: (i) broadcasting a vector representation of each agent's private observations to all agents (Sukhbaatar et al., 2016; Foerster et al., 2016), (ii) selective and targeted communication through the use of soft-attention networks, (Vaswani et al., 2017), that compute the importance of each agent and its information, (Jiang & Lu, 2018; Das et al., 2018), and (iii) communication through a shared memory channel (Pesce & Montana, 2019; Foerster et al., 2018), which allows agents to collectively learn and contribute information at every time instant. The architecture of (Jiang & Lu, 2018) implements communication by enabling agents to communicate intention as a learned representation of private observations, which are then integrated in the hidden state of a recurrent neural network as a form of agent memory. One downside to this approach is that as the communication is constrained in the neighborhood of each agent, communicated information does not enrich the actions of all agents, even if certain agent communications may be critical for a task. For example, if an agent from afar has covered a landmark, this information would be beneficial to another agent that has a trajectory planned towards the same landmark. In contrast, Memory Driven Multi-Agent Deep Deterministic Policy Gradient (MD-MADDPG), (Pesce & Montana, 2019), implements a shared memory state between all agents that is updated sequentially after each agent selects an action. However, the importance of each agent's update to the memory in MD-MADDPG is solely decided by its interactions with the memory channel. In addition, the sequential nature of updating the memory channel restricts the architecture's performance to 2-agent systems. Targeted Multi-Agent Communication (TarMAC), (Das et al., 2018), uses soft-attention (Vaswani et al., 2017) for the communication mechanism to infer the importance of each agent's information, however without the use of memory in the communication step.

The paradigm of using relations in agent-based reinforcement learning was proposed by (Zambaldi et al., 2018) through multi-headed dot-product attention (MHDPA) (Vaswani et al., 2017). The core idea of relational reinforcement learning (RRL) combines inductive logic programming (Lavrac & Dzeroski, 1994; Džeroski et al., 2001) and reinforcement learning to perform reasoning steps iterated over entities in the environment. Attention is a widely adopted framework in Natural Language Processing (NLP) and Visual Question Answering (VQA) tasks (Andreas et al., 2016b;a; Hudson & Manning, 2018) for computing these relations and interactions between entities. The mechanism (Vaswani et al., 2017) generates an attention distribution over the entities, or more simply a weighted value vector based on importance for the task at hand. This method has been adopted successfully in state-of-the-art results for Visual Question Answering (VQA) tasks (Andreas et al., 2016b), (Andreas et al., 2016a), and more recently (Hudson & Manning, 2018), demonstrating the robustness and generalization capacity of reasoning methods in neural networks. In the context of multi-agent cooperation, we draw inspiration from work in soft-attention (Vaswani et al., 2017) to implement a method for computing relations between agents, coupled with a memory based attention network from Compositional Attention Networks (MAC) (Hudson & Manning, 2018), yielding a framework for a memory-based communication that performs attentive reasoning over new information and past memories.

Concretely, we develop a communication architecture in MADRL by leveraging the approach of RRL and the capacity to learn from past experiences. Our architecture is guided by the belief that a structured and iterative reasoning between non-local entities should enable agents to capture higher-order relations that are necessary for complex problem-solving. To seek a balance between computational efficiency and adaptivity to variable team sizes, we exploit the soft-attention (Vaswani et al., 2017) as the base operation for selectively attending to an entity or information. To capture the information and histories of other entities, and to better equip agents to make a deliberate decision, we separate out the *attention* and *reasoning* steps. The attention unit informs the agent of which entities are most important for the current time-step, while the reasoning steps use previous memories and the information guided by the attention step to extract the shared information that is most relevant. This explicit separation in communication enables agents to not only place importance on new information from other agents, but to selectively choose information from its past memories given new information. This communication framework is learned in an end-to-end fashion, without resorting to any supervision, as a result of task-specific rewards.

Our empirical study demonstrates the effectiveness of our novel architecture to solve cooperative multi-agent tasks, with varying team sizes and environments. By leveraging the paradigm of *centralized learning and decentralized execution*, alongside communication, we demonstrate the efficacy of the learned cooperative strategies.

2 Preliminaries

2.1 Partially Observable Markov Decision Processes

We consider a team of N agents and model it as a cooperative multi-agent extension of a partially observable Markov decision process (POMDP) (Oliehoek, 2012). We characterize this POMDP by the set of state values, $\mathcal{S}$, describing all the possible configurations of the agents in the environment, control actions $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_N\}$, where each agent i performs an action $\mathcal{A}_i$, and set of observations $\{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_N\}$, where each agent i's local observation, $\mathcal{O}_i$ is not shared globally. Actions are selected through a stochastic policy $\boldsymbol{\pi}_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$ or through a deterministic policy $\boldsymbol{\mu}_{\theta_i} : \mathcal{O}_i \mapsto \mathcal{A}_i$ (Silver et al., 2014) with policy parameters θ_i, and a new state is generated by the environment according to the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathcal{S}^{'}$. At every step, the environment generates a reward, $r_i : \mathcal{S}^{'} \times \mathcal{A}_i \mapsto \mathbb{R}$, for each agent i and a new local observation $\mathbf{o}_i : \mathcal{S}^{'} \mapsto \mathcal{O}_i$. The goal is to learn a policy such that each agent maximizes the total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$ where T is the time horizon and γ is the discount factor.

2.2 Deterministic Policy Gradient Algorithms

We choose the deterministic policy gradient algorithms for all our training. In this framework, the parameters θ of the policy, $\boldsymbol{\mu}_\theta$, are updated such that the objective $J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a\sim \mu_\theta}[R(s,a)]$ and the policy gradient, (see Appendix A), is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_\theta \boldsymbol{\mu}_\theta(a|s)\nabla_a Q^{\boldsymbol{\mu}}(s,a)|_{a=\boldsymbol{\mu}_\theta(s)}] \tag{1}$$

Deep Deterministic Policy Gradient (DDPG) is an adaptation of DPG where the policy $\boldsymbol{\mu}$ and critic $Q^{\boldsymbol{\mu}}$ are approximated as neural networks. DDPG is an off-policy method, where experience replay buffers, $\mathcal{D}$, are used to sample system trajectories which are collected throughout the training process. These samples are then used to calculate gradients for the policy and critic networks to stabilize training. In addition, DDPG makes use of a target network, similar to Deep Q-Networks (DQN) (Mnih et al., 2015), such that the parameters of the primary network are updated every few steps, reducing the variance in learning.

Recent work (Lowe et al., 2017) proposes a multi-agent extension to the DDPG algorithm, so-called MADDPG, adapted through the use of centralized learning and decentralized execution. Each agent's policy is instantiated similar to DDPG, as $\boldsymbol{\mu}_{\theta_i}(a_i|\mathbf{o}_i)$ conditioned on its local observation $\mathbf{o}_i$. The major underlying difference is that the critic is centralized such that it estimates the joint action-value $\hat{Q}(\boldsymbol{x}, a_1, ..., a_N)$, where $\mathbf{x} = (\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_N)$. We operate under this scheme of centralized learning and decentralized execution of MADDPG, (Lowe et al., 2017), as the critics are not needed during the execution phase.

3 Reasoning-based Multi-Agent Communication

We introduce a communication architecture that is an adaptation of the attention mechanism of the Transformer network, (Vaswani et al., 2017), and the structured reasoning process used in the MAC Cell, (Hudson & Manning, 2018). The framework holds memories from previous time-steps separately for each agent, to be used for reasoning on new information received by communicating teammates. Through a structured reasoning, the model interacts with memories and communications received from other agents to produce a memory for the agent that contains the most valuable information for the task at hand. An agent's memory is then used to predict the action of the agent, such that policy is given by $\boldsymbol{\mu}_\theta : \mathcal{O}_i \times \mathcal{M}_i \mapsto \mathcal{A}_i$, where $\mathcal{M}_i$ is the memory of agent i.

To summarize, before any agent takes an action, the agent performs four operations via the following architectural features: (1) Thought Unit, where each agent encodes its local observations into appropriate representations for communication and action selection, (2) Question Unit, which is used to generate the importance of all information communicated to the agent, (3) Memory Unit, which controls the final message to be used for predicting actions by combining new information from other agents with its own memory, through the attention vector generated in the Question unit, and (4) Action Unit, that predicts the action. In Figure 1 we illustrate our proposed Structured Attentive Reasoning Network (SARNet).

3.1 The Thought Unit

The thought unit at each time-step t transforms an agent's private observations into three separate vector representations: query, $\mathbf{q}_i^t$, key, $\mathbf{k}_i^t$, and values, $\mathbf{v}_i^t$ for the communication units. These representations are then used to achieve structured reasoning, inspired by the self-attention mechanism from (Vaswani et al., 2017). Additionally, an encoding of the local observation, $\mathbf{e}_i^t$, is also generated for the *action* unit to complement the information generated from the communication step. Specifically, we have:

$$\mathbf{q}_i^t, \mathbf{k}_i^t, \mathbf{v}_i^t, \mathbf{e}_i^t = \varphi^{th}_{\theta_i^{th}}(\boldsymbol{o}_i^t), \qquad \mathbf{q}_i^t, \mathbf{k}_i^t \in \mathbb{R}^Q, \quad \mathbf{v}_i^t \in \mathbb{R}^V, \quad \mathbf{e}_i^t \in \mathbb{R}^e \tag{2}$$

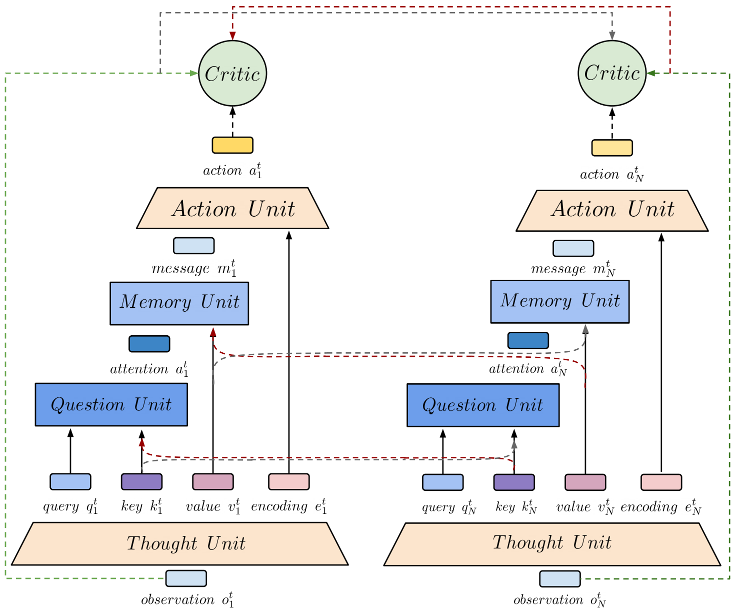


Figure 1: SARNet consists of a thought unit, question unit, memory unit and action unit. The thought unit operates over the current observation, to represent the information for the communication and action step. The question unit attends to the information from communicating agents, representing the importance of each neighboring agent's information. The memory unit, guided by the Question Unit, performs a reasoning operation to extract relevant information from communicated information, which is then integrated into memory. The Action Unit, processes the information from the memory and the observation encoding to predict an action for the current time-step.

where $\varphi^{th}_{\theta^{th}_i}$ can be characterized by a multi-layer perceptron (MLP) or any recurrent neural network (RNN) parameterized by θ^{th}_i.

The query is used by each agent i to inform the Question Unit which aspects of the communicated information are relevant to the current step.

The key and value are broadcast to all communicating agents. The key vector is used in the Question Unit to infer the relevance of the broadcasting agent to the current reasoning step, and the value vector is subsequently used to update the information into the memory of agent i.

The resulting information broadcasted by each agent i to all the cooperating agents is then:

$$c_i^t = [\ \overbrace{k_i^t}^{\text{key}} \quad \underbrace{v_i^t}_{\text{value}}\ ]. \tag{3}$$

3.2 The Question Unit

This component is designed to capture the importance of each agent in the environment, including the reasoning agent i, similar to the self-attention mechanism in (Vaswani et al., 2017). In the attention mechanism used in (Vaswani et al., 2017), the attention computes a weight for each entity through the use of the $softmax$. However, we generate the attention mechanism over all individual representations in the vector for each entity, using Eq. 5. This allows the agent to compute the importance of each individual communicated information from other agents for a particular time-step. This is performed through a soft attention-based weighted average using the query generated by agent i, and the set of keys, K, that contain the keys, $\{k_1^t, k_2^t, ..., k_N^t\}$ from all agents.

The recipient agent, i, upon receiving the set of keys, K, from all agents, computes the interaction with every agent through a Hadamard product, $\odot$, of its own query vector, q_i and all the keys, k_j, in the set K.

$$\mathbf{qh}_{ij}^t = \mathbf{q}_i^t \odot \mathbf{k_j}^t, \qquad \mathbf{k}_j^t \in K \tag{4}$$

A linear transformation, $\mathbf{W}_{iq}^{[d_q \times d_q]}$, is then applied to every interaction, qh_{ij}^t, that defines the query targeted for each communicating agent j, including self, to produce a scalar defining the weight of the particular agent. A $softmax$ operation is then used over the new scalars for each agent to generate the weights specific to each agent.

$$\mathbf{qu}_{ij}^t = \mathbf{W}_{iq}^{[d_q \times d_q]} \mathbf{qh}_{ij}^t \qquad \mathbf{qu}_{ij}^t \in \mathbb{R}^Q \tag{5}$$

$$\mathbf{a}_i^t = \text{softmax}\left[\frac{qu_{i1}^t}{\sqrt{d_q}} \; ... \; \frac{qu_{ii}^t}{\sqrt{d_q}} \; ... \; \frac{qu_{iN}^t}{\sqrt{d_q}}\right] \qquad \mathbf{a}_i^t \in \mathbb{R}^N \tag{6}$$

The use of the linear transformation in Eq. 5 allows the model to specify an importance not only for each individual agent, but more specifically it learns to assign an importance to each element in the information vector, as compared to the approach used in standard soft-attention based networks, such as Transformer (Vaswani et al., 2017), which only perform a dot-product computation between the query and keys.

3.3 The Memory Unit

The memory unit is responsible for decomposing the set of new values, V, that contain, $\{v_1^t, v_2^t, ..., v_N^t\}$, into relevant information for the current time-step. Specifically, it computes the interaction of this new knowledge with the memory aggregated from the preceding time-step. The new retrieved information, from the memory and the values, is then measured in terms of relevance based on the importance of each agent generated in the Question unit.

As a first step, an agent computes a direct interaction between the new values from other agents, $v_j \in V$, and its own memory, m_i. This step performs a relative reasoning between newly received information and the memory from the previous step. This allows the model to potentially highlight information from new communications, given information from prior memory.

$$\mathbf{mi}_{ij}^t = \mathbf{m}_i^{t-1} \odot \mathbf{v}_j^t, \qquad \forall\, \mathbf{v}_j^t \in V, \quad \mathbf{m}_{ij}^t, \mathbf{m}_i^{t-1} \in \mathbb{R}^V \tag{7}$$

The new interaction per agent j is evaluated relative to the memory, $\mathbf{mi}_{ij}^t$, and current knowledge, V, is then used to compute a new representation for the final attention stage, through a feed-forward network, $\mathbf{W}_r^{[d_v \times d_v]}$. This enables the model to reason independently on the interaction between new information and previous memory, and new information alone.

$$\mathbf{mr}_{ij}^t = \mathbf{W}_r^{[d_v \times d_v]}[\mathbf{mi}_{ij}^t + \mathbf{v}_j^t] \tag{8}$$

Finally, we aggregate the important information, $\mathbf{mr}_{ij}$, based on the weighting calculated in the Question unit, in (6). This step generates a weighted average of the new information, $\mathbf{mr}_{ij}$, gathered from the reasoning process, based on the attention values computed in (6). A linear transformation, $\mathbf{W}_m^{[d_q \times d_q]}$, is applied to the result of the reasoning operation to prepare the information for input to the action cell.

$$\mathbf{mv}_i^t = \sum_{j=1}^{N} \mathbf{a}_{ij}^t \mathbf{mr}_{ij}^t \qquad \forall\, \mathbf{a}_{ij}^t \in \mathbf{a}_j^t \tag{9}$$

$$\mathbf{m}_i^t = \mathbf{W}_m^{[d_q \times d_q]} \mathbf{mv}_i^t, \qquad \mathbf{m}_i^t \in \mathbb{R}^V \tag{10}$$

3.4 The Action Unit

The action unit, as the name implies, predicts the final action of the agent, i, based on the new memory, Eq. 10, computed from the Memory unit and an encoding, $\mathbf{e}_i^t$, Eq. 2, of its local observation, $\boldsymbol{o}_i$, from the Thought unit.

$$\mathbf{a}_i^t = \varphi_{\theta_i^a}^a(\mathbf{e}_i^t, \mathbf{m}_i^t), \qquad \mathbf{e}_i^t \in \mathbb{R}^O, \mathbf{m}_i^t \in \mathbb{R}^V \tag{11}$$

where $\varphi_{\theta_i^a}^a$ is a multi-layer perceptron (MLP) parameterised by θ_i^a.

3.5 Learning Algorithm

We incorporate the centralized learning-decentralized execution framework from (Lowe et al., 2017) to implement an actor-critic model to learn the policy parameters for each agent. We use parameter sharing across agents to ease the training process. The policy network $\boldsymbol{\mu}_{\theta_i}$ produces actions a_i, which is then evaluated by the critic $Q^{\boldsymbol{\mu}_{\theta_i}}$, which aims to minimize the loss function.

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\boldsymbol{x},a_i,r,\boldsymbol{x}'\sim\mathcal{D}}\Big[(Q^{\boldsymbol{\mu}_{\theta_i}}(\mathbf{x}, a_1, a_2, ..., a_N) - y)^2\Big] \tag{12}$$

$$y = r_i + \gamma Q^{\boldsymbol{\mu}'_{\theta_i}}(\mathbf{x}', a'_1, a'_2, ..., a'_N)$$

where $\mathbf{x}' = (\boldsymbol{o}'_1, \boldsymbol{o}'_2, \ldots, \boldsymbol{o}'_N)$, are the observations when actions $\{a_1, a_2, ..., a_N\}$ are performed, the experience replay buffer, $\mathcal{D}$, contains the tuples $(\mathbf{x}, \mathbf{x}', m_1, m_2, ..., m_N, a_1, a_2, ...a_N, r_1, r_2, ...r_N)$, and the target Q-value is defined as y. To keep the training stable, delayed updates to the target network $Q^{\boldsymbol{\mu}'_{\theta_i}}$ is implemented, such that current parameters of $Q^{\boldsymbol{\mu}_{\theta_i}}$, are only copied periodically. The goal of the loss function, $\mathcal{L}(\theta_i)$ is to minimize the expectation of the difference between the current and the target action-state function. The gradient of the resulting policy, with communication, to maximize the expectation of the rewards, $J(\theta_i) = \mathbb{E}[R_i]$, can be written as:

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_{\theta_i}) = \mathbb{E}_{\mathbf{x},a,\mathbf{m_i}\sim\mathcal{D}}\Big[\nabla_{\theta_i}\boldsymbol{\mu}_{\theta_i}(\boldsymbol{o}_i, \mathbf{m_i})\nabla_{a_i} Q^{\boldsymbol{\mu}_{\theta_i}}(\mathbf{x}, a_1, \ldots, a_N)|_{a_i=\boldsymbol{\mu}_{\theta_i}(\boldsymbol{o}_i,\mathbf{m_i})}\Big]$$

4 Experiments

We evaluate our communication architecture on OpenAI's multi-agent particle environment, (Lowe et al., 2017), a two-dimensional environment consisting of agents and landmarks with cooperative tasks. Each agent receives a private observation that includes only partial observations of the environment depending on the task. The agents act independently and collect a shared global reward for cooperative tasks. We consider different experimental scenarios where a team of agents cooperate to complete tasks against static goals, or compete against non-communicating agents. We compare our communication architecture, SARNet, to communication mechanisms of CommNet, (Sukhbaatar et al., 2016), TarMAC (Das et al., 2018) and the non-communicating policy of MADDPG, (Lowe et al., 2017).

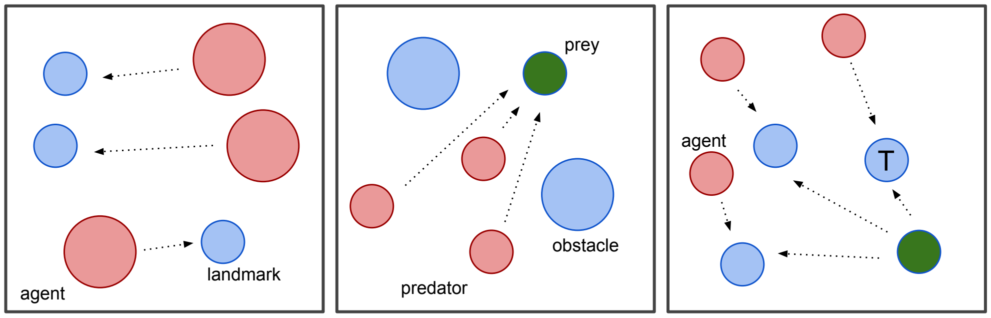


Figure 2: Illustrations of the environments in the experiments: Cooperative Navigation (left), Predator-Prey (mid), Physical Deception (right).

4.1 Partially-Observable Cooperative Navigation (POCN)

In this environment, N agents need to cooperate to reach L landmarks. Each agent observes the relative positions of the neighboring agents and landmarks. The agents are penalized if they collide with each other, and positively rewarded based on the proximity to the nearest landmark. At each time-step, the agent receives a reward of $-d$, where d is the distance to the closest landmark, and penalized a reward of -1 if a collision occurs with another agent. In this cooperative task, all agents strive to maximize a shared global reward. Performance is evaluated per episode by average reward, number of collisions, and occupied landmarks. Our model outperforms all the baselines achieving a higher reward through lower metrics of average distance to the landmark and collisions for $N = L = 3$ and $N = L = 6$ agents as shown in Table 1. We hypothesize that in an environment with more agents, the effect of retaining a memory of previous communications from other agents allows the policy to make a more informed decision. This leads to a significant reduction in collisions, and lower distance to the landmark. Our architecture outperforms TarMAC, which uses a similar implementation of soft-attention, albeit without a memory, and computing the communication for the next time-step, unlike SARNet, where the communication mechanism in time t, shapes the action, a_i^t for the current time-step. We also show the attention metrics for agent 0 at a single time-step during the execution of the tasks with $N = 6$ agents in Fig. 3a.

Table 1: Partially observable cooperative navigation. For $N = L = 3$, the agents can observe the nearest 1 agent and 2 landmarks, and for $N = L = 6$, the agents can observe, 2 agents and 3 landmarks. Number of collisions between agents, and average distance at the end of the episode are measured.

	$N = 3$		$N = 6$	
Policy	**# collisions**	**Average dist.**	**# collisions**	**Average dist.**
SARNet	**0.91**	0.61	**1.09**	**1.51**
MADDPG	1.03	0.69	1.25	2.11
TarMAC	1.01	0.58	1.13	1.66
CommNet	1.02	**0.49**	1.13	1.77

4.2 Partially Observable Predator-Prey

This task involves a slower moving team of N cooperating agents chasing M faster moving agents in an environment with L static landmarks. Each agent receives its own local observation, where it can observe the nearest prey, predator, and landmarks. Predators are rewarded by $+10$ every time they collide with a prey, and subsequently the prey is penalized -10. Since the environment is unbounded, the prey are also penalized for moving out of the environment. Predators are trained using the SARNet, TarMAC, CommNet, and MADDPG and prey are trained using DDPG. Due to the nature of dynamic intelligent agents competing with the communicating agents, the complexity of the task increases substantially. As shown in Fig. 3b, we observe that agent 0 learns to place a higher importance on agent 1's information over itself. This dynamic nature of the agent, in selecting which information is beneficial, coupled with extracting relevant information from the memory, enables our architecture to substantially outperform the baseline methods, Table 2a.

Table 2: In 2a, $N = L = 3$, $M = 1$, the agents can observe the nearest 1 predator, 1 prey and 2 landmarks, and for $N = L = 6$, $M = 2$, the agents can observe, 3 predators, 1 prey and 3 landmarks. Number of prey captures by the predators per episode is measured. For Table 2b, we measure the avg. success rate of the communicating agents N, to reach the target landmark, and the same for the adversary M. Larger values for N are desired, and lower for M.

(a) Partially observable predatory-prey

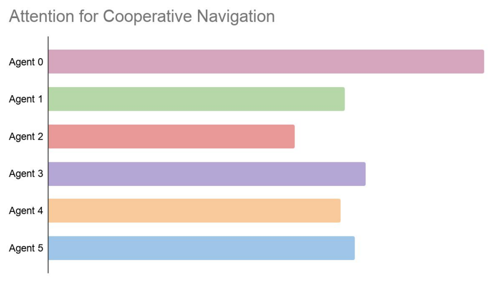
	$N = 3$	$N = 6$
Policy	**# captures**	**# captures**
SARNet	**17.8**	**24.7**
MADDPG	10.1	12.9
TarMAC	16.9	16.1
CommNet	13.8	16.8

(b) Partially observable physical deception

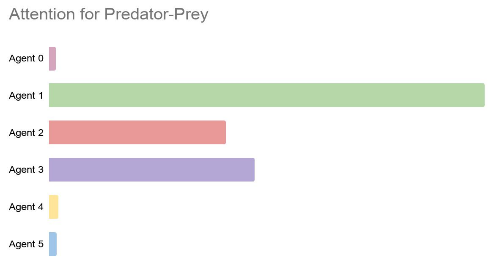
	$N = 3, M = 1$	
Policy	**# Captures 'M'**	**# Captures 'N'**
SARNet	0.58	**1.35**
MADDPG	0.59	1.17
TarMAC	0.16	0.14
CommNet	**0.14**	0.26

4.3 Partially Observable Physical Deception

A team of M adversarial agents must close in on a target landmark, without directly observing it. The landmark must be inferred from the positions of N communicating agents, whose task is to deceive the non-communicating agents. The communicating agents get positively rewarded by d and negatively rewarded by $-d_a$, where d and d_a is the distance of the communicating and adversarial agent to the target landmark, respectively. This task is highly complex and showcases how each architecture learns different behaviors. As shown in Table 2b, both TarMAC and CommNet agents choose to stay far away from the target landmark, such that the adversarial agent, follow them. In contrast, SARNet and MADDPG, learn to spread out over all the landmarks, with a higher adversarial score, but achieving an overall higher mean reward for the complete task.



(a) Attention for agent 0, at a single time-step for cooperative navigation environment for $N = L = 6$.



(b) Attention for agent 0, at a single time-step in a predator-prey environment, for $N = 6$, and $M = 2$.

Figure 3: Visualizing attention predicted by agent i over all agents during the initial stages of the task. We observe that agents learn to devalue their own information if it is more advantageous to place importance on information from other agents when reading and writing to the memory unit.

4.4 Memory Corruption

We perform additional benchmarks showcasing the importance of memory in our communication architecture. By introducing noise in the memory at each time-step, we evaluate the performance of SARNet on partially-observable cooperative navigation and predator-prey environments. We find a general worsening of the results for both tasks as demonstrated by the metrics in Table 3. However, we note that in spite of the corrupted memory, the agent's policy is robust to adversarial noise, as it learns to infer the important information from the thorough reasoning process in the communication.

Table 3: Performance metrics when a Gaussian Noise of mean 0 and variance 1 is introduced in the memory (MC-SARNet) during execution of the tasks for predator-prey and cooperative navigation.

(a) Memory in predatory-prey

	$N = 6$
Policy	**# collisions**
MC-SARNet	22.2
SARNet	24.7

(b) Memory in cooperative Navigation

	$N = 3, M = 1$	
Policy	**# Captures 'M'**	**# Captures 'N'**
MC-SARNet	1.15	1.85
SARNet	1.09	1.51

5 Conclusion

We have introduced a novel framework, SARNet, for communication in multi-agent deep RL which performs a structured attentive reasoning between agents to improve coordination skills. Through a decomposition of the representations of communication into reasoning steps, our agents exceed baseline methods in overall performance. Our experiments demonstrate key benefits of gathering insights from (1) its own memories, and (2) the internal representations of the information available to agent. The communication architecture is learned end-to-end, and is capable of computing task-relevant importance of each piece of computed information from cooperating agents. While this multi-agent communication mechanism shows promising results, we believe that we can further adapt this method to scale to a larger number of agents, through a gating mechanism to initiate communication, and decentralized learning.

REFERENCES

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*, 2016a.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016b.

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.

Jan Peter De Ruiter, Matthijs L Noordzij, Sarah Newman-Norlund, Roger Newman-Norlund, Peter Hagoort, Stephen C Levinson, and Ivan Toni. Exploring the cognitive infrastructure of communication. *Interaction Studies*, 11(1):51–77, 2010.

Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001.

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 29*. 2016.

Jakob N. Foerster, Christian A. Schröder de Witt, Gregory Farquhar, Philip H. S. Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. arXiv preprint arXiv:1810.11702, 2018.

Riccardo Fusaroli, Bahador Bahrami, Karsten Olsen, Andreas Roepstorff, Geraint Rees, Chris Frith, and Kristian Tylén. Coming to terms: quantifying the benefits of linguistic coordination. *Psychological science*, 23(8):931–939, 2012.

Simon Garrod, Nicolas Fay, Shane Rogers, Bradley Walker, and Nik Swoboda. Can iterated learning explain the emergence of graphical symbols? *Interaction Studies*, 11(1):33–50, 2010.

Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*, 2018.

Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *arXiv preprint arXiv:1805.07733*, 2018.

Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsuhi Shinjou, and Susumu Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, volume 6, pp. 739–743. IEEE, 1999.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *WLP*, pp. 146–160. Springer, 1994.

Ling Li, Alcherio Martinoli, and Yaser S Abu-Mostafa. Emergent specialization in swarm systems. In *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 261–266. Springer, 2002.

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*, pp. 6379–6390, 2017.

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.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Frans A Oliehoek. Decentralized pomdps. In *Reinforcement Learning*, pp. 471–503. Springer, 2012.

OpenAI. Openai five. 2018. URL `https://blog.openai.com/openai-five/`.

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

Alberto Petrillo, Alessandro Salvi, Stefania Santini, and Antonio Saverio Valente. Adaptive multi-agents synchronization for collaborative driving of autonomous vehicles with multiple communication delays. *Transportation research part C: emerging technologies*, 86:372–392, 2018.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

Sainbayar Sukhbaatar, arthur szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems 29*. 2016.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*. 2017.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

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

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

A Appendix

Policy Gradient Algorithms

Policy gradient (PG) methods are the popular choice for a variety of reinforcement learning (RL) tasks in the context described above. In the PG framework, the parameters θ of the policy are directly adjusted to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a\sim\boldsymbol{\pi}_\theta}[R]$, by taking steps in the direction of $\nabla_\theta J(\theta)$, where $p^{\boldsymbol{\pi}}$, is the state distribution, s is the sampled state and a is the action sampled from the stochastic policy. Through learning a value function for the state-action pair, $Q^{\boldsymbol{\pi}}(s,a)$, which estimates how good an optimal action a is for an agent in state s, the policy gradient is then written as, (Sutton et al., 2000):

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a\sim\boldsymbol{\pi}_\theta}[\nabla_\theta \log \boldsymbol{\pi}_\theta(a|s) Q^{\boldsymbol{\pi}}(s,a)], \tag{13}$$

Several variations of PG have been developed, primarily focused on techniques for estimating $Q^{\boldsymbol{\pi}}$. For example, the REINFORCE algorithm (Williams, 1992) uses a rather simplistic method of sample return calculated as a cumulative expected reward for an episode with a discount factor γ, $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. When temporal-difference learning (Sutton et al., 1998) is used, the learned function $Q^{\boldsymbol{\pi}}(s,a)$ is described as the *critic*, which leads to several different *actor-critic* algorithms (Sutton et al., 1998), (Konda & Tsitsiklis, 2000), where the actor could be a stochastic $\boldsymbol{\pi}_\theta$ or deterministic policy $\boldsymbol{\mu}_\theta$ for predicting actions.

Experiments

Hyperparameters We use batch synchronous method for off-policy gradient methods, (Nair et al., 2015; Stooke & Abbeel, 2018), for all the experiments. Each environment instance has a separate replay buffer of size 10^6. All policies are trained using the Adam optimizer with a learning rate of 5×10^{-4}, a discount factor, $\gamma = 0.96$. and $\tau = 0.001$, for the soft update of the target network. The query, key and values, share a common first layer of size 128, and subsequently are linearly projected to 32-d. Batch normalization and dropouts in the communication channel are implemented. The observation, and action units have two hidden layers of size 128 units with ReLU as the activation function. The critic is implemented as a 2-layer MLP, with 1024 and 512 units. We use a batch-size of 128, and updates are initiated after accumulating experiences for 1280 episodes. Exploration noise is implemented through Orhnstein-Uhlenbeck process with $\theta = 0.15$ and $\sigma = 0.3$. All experimental results are averaged over 3 separate runs, with different random seeds.

Baseline Hyperparameters Policies for TarMAC, CommNet and MADDPG are instantiated as an MLP, similar to SARNet. All layers in the policy network are of size 128 units, while the critic is a 2-layer network with 1024, 512 units. Both TarMAC and CommNet are implemented with 1-stage communication. For TarMAC, the query's, and key's have 16 units, and values are 32 units as described in (Das et al., 2018). All other training parameters are kept similar to the SARNet implementation.

STRUCTURED ATTENTIVE REASONING NETWORK ALGORITHM

Algorithm 1 SARNet Algorithm

```
1: Initialize actors ($\boldsymbol{\mu}_{\theta_1}, \dots, \boldsymbol{\mu}_{\theta_N}$) and critics networks ($Q_{\theta_1}, \dots, Q_{\theta_N}$)
2: Initialize actor target networks ($\boldsymbol{\mu}'_{\theta_1}, \dots, \boldsymbol{\mu}'_{\theta_N}$) and critic target networks ($Q'_{\theta_1}, \dots, Q'_{\theta_N}$)
3: Initialize replay buffer $\mathcal{D}$ for each environment instance
4: for episode = 1 to E do
5:     Initialize a random process $\mathcal{N}$ for exploration
6:     Initialize the memory $\mathbf{m}_i$ for each agent
7:     for t = 1 to max episode length do
8:         for agent $i$ = 1 to $N$ do
9:             Receive observation $\boldsymbol{o}_i$
10:            Generate query, key, value $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ Eq. (2)
11:            Receive $K, V$ from all agents
12:            Compute the question vector $\mathbf{a}_i$ Eq. (6)
13:            Process the new information $\mathbf{m}_i$ Eq. (10)
14:            Compute new observation encoding $\mathbf{e}_i$ Eq. (2)
15:            Store the new information in memory
16:            Select action $\mathbf{a_i}$ Eq. (11)
17:        end for
18:        Set $\mathbf{x} = (\boldsymbol{o}_1, \dots, \boldsymbol{o}_N)$ and $\boldsymbol{\Phi} = (\mathbf{m}_1, \dots, \mathbf{m}_N)$
19:        Execute actions $\mathbf{a} = (a_1, \dots, a_N)$, observe rewards $r$ and next observations $\mathbf{x}'$
20:        Store ($\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r$) in replay buffer $\mathcal{D}$
21:    end for
22:    for agent i = 1 to $N$ do
23:        Sample a random minibatch $\Theta$ of $Z$ samples ($\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r$) from $\mathcal{D}$
24:        Set $y = r_i + \gamma Q^{\boldsymbol{\mu}'_{\theta_i}}(\mathbf{x}', a'_1, \dots, a'_N)|_{a'_k = \boldsymbol{\mu}'_{\theta_k}(o_k, \mathbf{m_k})}$
25:        Update critic by minimizing:
26:            $$\mathcal{L}(\theta_i) = \frac{1}{Z} \sum_{(\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r) \in \Theta} (y - Q^{\boldsymbol{\mu}_{\theta_i}}(\mathbf{x}, a_1, \dots, a_N))^2$$
27:        Update actor according to the policy gradient:
               $$\nabla_{\theta_i} J \approx \frac{1}{Z} \sum_{(\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r)} \Big( \nabla_{\theta_i} \boldsymbol{\mu}_{\theta_i}(\boldsymbol{o}_i, \mathbf{m}_i) \nabla_{a_i} Q^{\boldsymbol{\mu}_{\theta_i}} (\mathbf{x}, a_1, \dots, a_i, \dots a_N)|_{a_i = \boldsymbol{\mu}_{\theta_i}(o_i, \mathbf{m}_i)} \Big)$$
28:    end for
29:    Update target networks:
               $$\theta_i^{'} = \tau \theta_i + (1 - \tau) \theta_i^{'}$$
30: end for
```