Learning to Communicate in Multi-Agent Environments

Nathan Chung
Georgia Institute of Technology
nchung32@gatech.edu

Wesley Ford
Georgia Institute of Technology
wesley.ford@gatech.edu

Mengying (Cassie) Lin
Georgia Institute of Technology
mlin365@gatech.edu

Nhi (Léona) Nguyen
Georgia Institute of Technology
nnguyen349@gatech.edu

December 18, 2025

Abstract: This project investigates the use of communication between agents in collaborative multi-agent environments. Utilizing PPO as a baseline training algorithm, we demonstrate two methods of training with communication in the multi-agent environment *PistonBall*. It is demonstrated that the use of a shared communication vector between agents used to augment the policy network input can encourage faster convergence to a solution, while further augmenting this structure with a multi-headed attention layer reduces training variance. Finally, the effect of changing the neighborhood of communication is investigated.

The video and slides can be found here and code can be found at our Github repo. Please reach out to us if you have any questions or problems.

.

Keywords: Deep Reinforcement Learning, Multi-Agent Reinforcement Learning, Communication, CS8803, Course Project

1 Introduction

Multi-Agent Reinforcement Learning (MARL) is a branch of reinforcement learning where multiple agents learn to make decisions in a shared environment. These agents' tasks can be collaborative, competitive, or mixed. With MARL, when one of the agents takes an action, they can change the global state of the environment. This means that the other agents' observation of the environment could be affected by the action of another agent and they must be able to adapt while learning and executing their own policy. However, these agents traditionally do not share information between each other and are left to generate a policy independently. Yet, humans in shared spaces often rely on different forms of communication whether it be visual cues, body language, or speech to navigate the world. For example, if we see another person with a frightened expression 100 feet away, we would be cautious to approach the area since the expression indicates danger. Our project is motivated by the benefits from human communication and we explore this idea in our project by sharing information between agents. The information that can be shared in MARL tasks can be the partial or full observations from the agents, or the actions the agents are taking as an input to the agent's observation.

Our goal is to show that the communication between agents can create richer and stronger task performance in simulation. We believe that this applies specifically well to cooperative and semi-adversarial

tasks, as agents share a common goal. For example, racing is a semi-adversarial task where all agents want to finish as high as possible, but also require some collaboration with some opponents to beat other opponents. We use Proximal Policy Optimization (PPO) as our algorithm to train our policy and we showcase 2 different methods of communication: vector encoding and multi-head attention.

2 Related Work

As the real world involves many different individuals interacting in a given environment, Multi-Agent Reinforcement Learning is an important problem to explore as it will be very likely that our robots/agents will have to work with other agents in the real world.

Researchers have also acknowledged the importance of sharing information in MARL-based tasks. A3C3 is an algorithm that is designed with a central critic that estimates a value function for all agents to use, while agents independently generate their policy function and optimize their communication network [8]. This strategy allows agents to learn what is important to communicate with other agents. A similar idea called CommNet, also developed a multi layer communication architecture that encodes the observations of the agents and shares the hidden state with the other agents, so that agents can continuously share information between each other [9]. These approaches focus on algorithm design and highlight communication directly. While proven effective, these implementations of communication cannot be easily integrated with other algorithms.

Others have approached communication through encoding. Lin et al. [3] used a single autoencoder to encode the observations of all agents and distributed the encoded message to the agents to train their policy. This method allows the autoencoder and agents to develop a 'common' language that they will use to communicate with each other. However, using autoencoders in RL can take quite a long time to train. As an autoencoder requires a lot of data being an unsupervised method, training it on an on-policy algorithm to generate data would take a very long time. Another group used mean message encoding and attention head encoding to distribute the agent observations to the others [10]. Mean message encoding takes the hidden states of each agent, and takes the average. The drawback to this approach is that it gives the same importance to each message and we cannot place any priority to them. Attention head fixes the message importance problem. Our group is approaching our problem through encoding like Lin et al. and Vanneste et al. This gives our communication approach the potential to be easily explored with other existing reinforcement learning algorithms in the future.

While algorithm design and encoding are some of the main approaches in MARL communication, other interesting approaches should be noted. Graph Neural Networks has been shown to work with communication between multiple real drones for reforestation where the location of the drones relative to one another dictates what agents can share information between each other. [7]. Another group explored a strategy where they leave it to the agents to discover and learn the significance of these communication channels rather than having the models expect a communication channel provided to them [4].

3 Methods

Our project investigates the impact of communication strategies on multi-agent performance and coordination challenges of agents working in shared spaces. The goal is to assess how communication mechanisms can improve task performance compared to non-communicating baselines.

Our algorithm, PPO is a popular policy gradient-based method developed by OpenAI and features a clipped objective function that prevents large policy updates [6]. PPO is the baseline algorithm for policy training due to its simplicity, stability, and effectiveness in reinforcement learning tasks. This algorithm choice lets us focus more on exploring the communication selections. We also chose PPO as it has been successfully used with competitive Multi-Agent Racing where agent states are observable to every agent [5]. A proven algorithm applied to MARL lets us focus on communication.

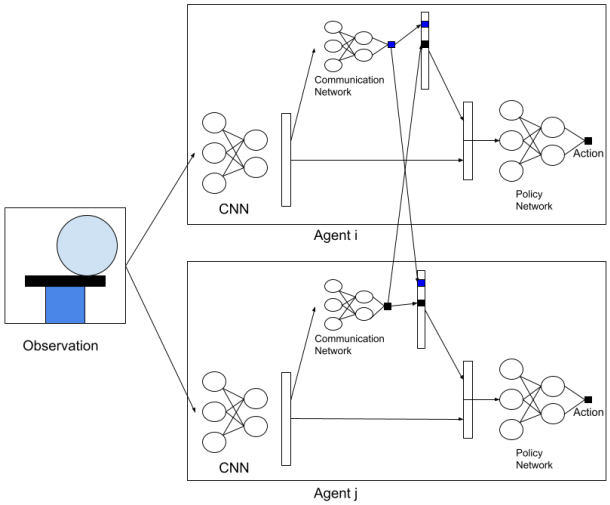

Figure 1: Structure of the Communication Vector Algorithm. Agents generate and share communication values with each other, which are then used in the policy to generate an action.

3.1 PPO with Multi-Agent Communication Vector

Our first approach to communication is an attempt to have agents learn to communicate with other agents based on their local observations. Each Agent uses a *Communication* network to produce a *communication scalar*, c_j, based on the agent's observation. Each agent's *communication scalar* is then concatenated into a vector, C, which is propagated to each agent. This vector is then used as an input into the agents policy and value networks, along with the hidden features of the observation, the result of using a Convolutional Neural Network to transform the 2D visual observations into a latent representation. In order to make training computationally tractable for our resources, a single policy is trained for all agents using local observations from all agents in the environment. As actions from agents in this environment are spatially correlated with nearby agents, it may be important for specific agents to prioritize communication signals from nearby agents over agents further away in the environment. To enable this while still training a single policy, each input to the communication, policy, and value networks are also conditioned with the id, I_i, of the specific agent the observation came from. Observations, communication vectors, actions and rewards are then stored in a replay buffer, which is used to update the agents policy via PPO. This algorithm is summarized in **Algorithm 1**. The general structure and flow of the algorithm is show in Figure 1.

3.2 PPO with Multi-Head Attention Communication

To improve the scalability and expressiveness of communication among agents, we extend the communication mechanism using a *multi-head attention* architecture. Unlike the communication vector method, which aggregates communication scalars into a single vector, this approach uses multiple attention heads to directly produce compressed representations by attending to relevant signals from other agents. Each attention head processes information independently, enabling the agent to focus on different aspects of incoming communication simultaneously.

Each agent A_i processes its local observation s_i using a neural network to extract hidden features h_i. The agent then concatenates these features with its unique ID, producing a communication input vector:

$$c_i \leftarrow \text{Concat}(h_i, \text{id}_i)$$

Algorithm 1 PPO with Multi-Agent Communication Vector

procedure PPO-MULTI-AGENT
  Initialize policy, value and communication networks θ, ϕ, ψ for each agent A_i.
  Initialize communication vector c_i for all agents.
  for each training iteration **do**
    for each environment step **do**
      for each agent A_i in $\{A_1, \ldots, A_N\}$ **do**
        Collect local observation s_i.
        Compute communication scalar $c_i = \pi_\psi(s_i, I_i)$,
        where I_i is a conditioning factor, and is the agent's unique id.
        Compute communication vector by aggregating neighbors:

$$C \leftarrow \text{Aggregate}(\{c_j\}_{j \in \mathcal{N}(i)})$$

        Form augmented observation: $\tilde{s}_i = [s_i, C, I_i]$, conditioned on agent id: I_i
        Select action $a_i \sim \pi_\theta(a_i|\tilde{s}_i)$.
      end for
      Execute joint action $a = \{a_1, \ldots, a_N\}$ and observe rewards.
    end for
    Update θ, ϕ and ψ using PPO.
  end for
end procedure

To aggregate communication signals, each agent applies *multi-head attention* over the inputs from its neighboring agents $\mathcal{N}(i)$. The attention mechanism computes the aggregated communication context using the standard scaled dot-product attention:

$$c_i = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)W^O$$

where each attention head is computed as:

$$\text{head}_h = \text{Softmax}\left(\frac{Q_i^h K^{h\top}}{\sqrt{d_k}}\right) V^h$$

Here, Q_i^h, K^h, and V^h are the query, key, and value representations generated by learned linear projections of the input features for head h, and d_k is the dimensionality of the key vectors.

The augmented observation $\tilde{s}_i = [s_i, c_i]$ is passed to the agent's policy and value networks, parameterized by θ and ϕ, respectively. This enables agents to adaptively focus on relevant communication while discarding irrelevant or noisy information from other agents.

The training process follows the PPO algorithm with shared policy and value function parameters across agents. Experience tuples containing observations, actions, communication features, and rewards are stored in a replay buffer for policy updates. The full approach is detailed in **Algorithm 2**.

4 Experimental Results

4.1 Environment Setup

We chose Pistonball and MultiCarRacing as the two environments where we can explore the effect of communication in multi-agent problems. Pistonball is a cooperative physics-based game where 20 stationary pistons (our agents) work together to push the ball from one side of the wall to the other [1]. PistonBall is "solved" when all agents can work together seamlessly to move the ball from one side to the other. This corresponds to a reward of approximately 60. MultiCarRacing is a semi-adversarial multi-agent version of CarRacing, a 2-dimensional environment that has a simple track layout where each car

Algorithm 2 PPO with Multi-Head Attention Communication

procedure PPO-MULTI-AGENT
  for each training iteration **do**
    for each environment step **do**
      for each agent $A_i \in \{A_1, \ldots, A_N\}$ **do**
        Collect local observation s_i;
        Extract hidden features: $h_i \leftarrow \text{Network}(s_i)$
        Concatenate hidden features with agent ID:

$$c_i \leftarrow \text{comm\_input}_i \leftarrow \text{Concat}(h_i, \text{id}_i)$$

        Compute communication vector using multi-head attention:

$$c_i \leftarrow \text{Attention}(\{c_j\}_{j \in \mathcal{N}(i)})$$

        Form augmented observation: $\tilde{s}_i = [s_i, c_i]$
        Select action: $a_i \sim \pi_\theta(a_i | \tilde{s}_i)$
      end for
      Execute joint action $a = \{a_1, \ldots, a_N\}$; observe rewards
    end for
    Optimize policy θ and value function ϕ using PPO
  end for
end procedure

(agent) can control its steering, gas, and brake [2]. It should be noted that our group did not choose to move forward with the MultiCarRacing environment as we had some issues with depreciated packages from a repository we were basing our project off of. We were able to solve some of these issues but we were then concerned with completing the training given the timeline of the project and the subsequent compute resources required to generate a decent policy.

4.2 Results

4.2.1 Communication Enhanced Policies

We first started to compare the results with the baseline performance to see how improved it is in a collaborative multi-agent environment. Our team started with the Comm-Vector method.

Figure 2 shows the Episode Rewards, Value Loss, and Policy Loss for the PistonBall environment trained using the communication-vector (Comm-Vector) method (blue), described in **Algorithm 1**, the Multi-Headed Attention method (orange) described in **Algorithm 2**, and a baseline comparison implementation (green) where communication is not used. The structure and number of parameters were not modified for the Baseline agents. Instead, Baseline agents were always provided null-tokens as their communication vector. Agents that utilized either communication method managed to solve the environment within 500 epochs, while there is little improvement to the baseline agent. Although Comm-Attn learns faster, it has a higher variance in policy and value loss. This means that the communication method is having a harder time learning, which is reasonable since it has more complex input.

Comparing the two communication policies together, we observe in Figure 2 that Comm-Vec seems to prioritize aggressive learning more than Comm-Attn. This results in Comm-Vec higher variability in both value and policy losses but achieves better peak performance in returns. On the other hand, Comm-Attn trades off slightly lower returns for greater stability and consistency in the learning process, making it potentially more reliable.

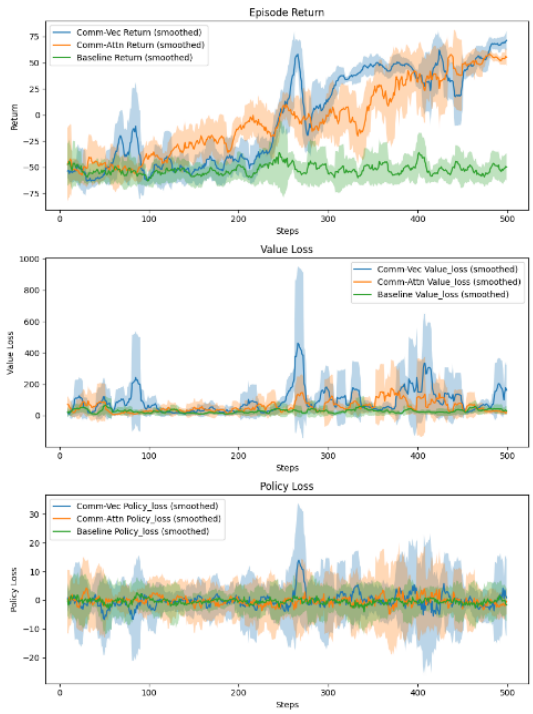

Figure 2: Rewards, Value Loss and Policy Loss plots based on epochs between baseline and Multi-head Attention method

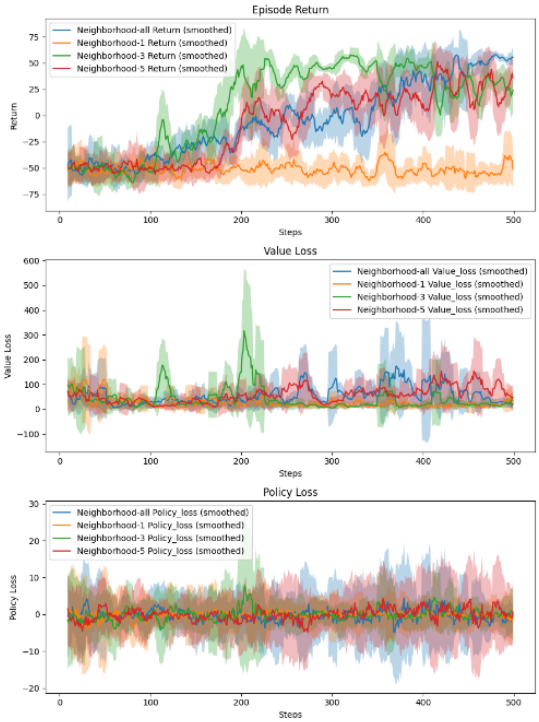

Figure 3: Results for different neighborhood ranges. The best performance is achieved with a neighborhood size of 3–5 agents, while too much or too little information might degrade performance.

4.2.2 Effect of Neighborhood Range

To evaluate the impact of the neighborhood range on agent performance, we compared the episode returns for different neighborhood configurations, as shown in Figure 3. The ranges tested include using information from all agents, one neighboring agent, three neighboring agents, and five neighboring agents.

The results indicate that agents perform best when their communication is limited to a range of 3–5 neighbors. When agents attempt to communicate with all other agents, their performance is significantly reduced. This is likely due to information overload, where irrelevant or redundant signals make it difficult for agents to focus on actionable insights. Conversely, limiting communication to only one neighboring agent results in underloading, as agents lack sufficient context to make informed decisions.

The performance plateau observed with a neighborhood size of 3–5 suggests that there is a sweet spot for balancing information richness and relevance. This aligns with the intuition that local context is often sufficient for coordination, as actions of nearby agents are more likely to directly affect an agent's environment. Larger neighborhoods add diminishing value while increasing computational and communication overhead.

4.2.3 Attention Visualization

To better understand the behavior of our attention-based communication methods, we analyzed the attention weights generated by the multi-head attention mechanism. Figure 4 shows the average attention weights as a heatmap, where each row corresponds to a querying agent and each column represents the attended agents. Darker areas indicate higher attention weights.

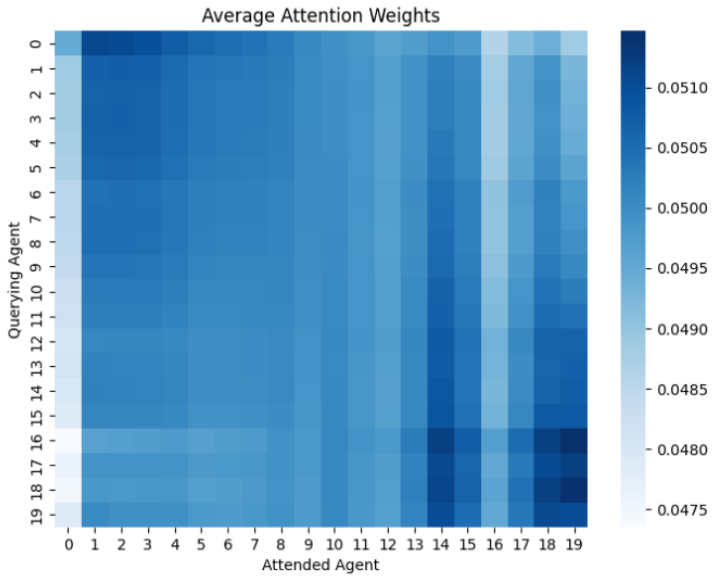

Figure 4: Average Attention Weights heat map

Agents tend to assign higher attention to themselves or their immediate neighbors, as seen in the darker regions near the diagonal. This behavior is intuitive, as an agent's own state and the states of nearby agents are typically the most relevant for decision-making. For example, in the PistonBall environment, pistons adjacent to one another often coordinate their actions, such as pushing the ball together, to achieve the shared objective.

Interestingly, certain querying agents, particularly those on the right side (e.g., agents 17–19), exhibit significantly higher attention to specific attended agents. This is likely because all agents on the left need to attend to their states to make meaningful decisions as the ball is moving leftward. Additionally, the dominance of attention weights along the start and ends of the diagonal reflects the spatial organization of agents, with edge agents prioritizing themselves and their closest neighbors due to their limited interactions with others.

These visualizations provide valuable insights into how agents utilize attention to prioritize information. By focusing on their own state and nearby neighbors, agents effectively balance computational efficiency and task relevance.

5 Discussion and Analysis

Our approach to agent communication demonstrated notable benefits in a collaborative multi-agent environment, such as PistonBall. Across our experiments, communication-enabled agents consistently outperformed baselines, finding a policy that successfully solved the environment. In contrast, baseline methods were unable to achieve similar results within the same number of epochs. This suggests that inter-agent communication can play a critical role in achieving coordination and efficiency in multi-agent reinforcement learning.

One of the strengths of our approach lies in its easy integration with reinforcement learning algorithms. The primary modifications were limited to introducing a communication network, adjusting the inputs to the policy and value networks. These relatively lightweight changes highlight the scalability and adaptability of the method to other multi-agent systems.

However, a key limitation of our approach is the lack of interpretability regarding *what* information is being communicated between agents. Since the communication consists of feature vectors, the exact content of these signals is unclear and could potentially be arbitrary. Investigating the nature of these sig-

nals—whether they encode explicit spatial, temporal, or task-specific information—could provide valuable insights and help refine communication protocols for better efficiency and effectiveness.

Additionally, comparing our approach to agents with access to more complete state-space observations could provide more valuable insights. For instance, agents with a global view of the environment or a compressed representation of the entire state might achieve better performance than our communication-based method. Future studies could evaluate how our approach stacks up against these state-aware baselines, offering a clearer understanding of its relative strengths and limitations.

Another avenue for future work is applying our method to more diverse and challenging multi-agent environments, such as semi-adversarial tasks like multi-agent racing. Although we initially considered this domain, the computational cost of training such agents exceeded our available resources. These environments present interesting questions, such as whether agents might learn to ”lie” or mislead each other or develop more complex coordinated strategies.

Finally, our findings underscore the importance of selecting the appropriate neighborhood range for communication. As discussed in Section 4.2.2, agents performed best when limited to communicating with a small set of nearby neighbors (e.g., 3–5). Overloading agents with information from all agents or restricting them to a single neighbor negatively impacted performance. Future work could investigate adaptive mechanisms to dynamically adjust the neighborhood size based on task complexity or environmental changes, enabling agents to balance information richness and relevance effectively.

In summary, our approach demonstrates the potential of communication-based strategies in multi-agent systems, while also identifying areas for further exploration and improvement, including interpretability, comparisons with global state-aware methods, and applications in more complex environments.

6 Conclusion

In conclusion, our study has shown the effectiveness of implementing communication in enhancing learning in a collaborative muti-agent environment. Our experiments demonstrated that communication-integrated methods are able to solve multi-agent problems that baseline approaches could not within the same number of epochs. This emphasizes the importance of communication consideration in achieving good coordination and performance. We also validate the importance of selecting communication ranges to maintain relevance and avoid overloading information.

Both simple communication methods, such as aggregated communication vectors, and more advanced approaches, like Multi-head attention, were sufficient to solve the Pistonball Environment. These results show that even simple communication strategies can make a big difference, while more complicated methods offer more stability and consistency.

Moving forward, there is a lot of potential to explore how these techniques perform in more complex environments, including cooperative and semi-adversarial environments. These scenarios can investigate deeper insights like how agents learn to communicate, work together, or deceive one another to succeed.

Acknowledgments

We would like to thank Dr. Animesh Garg and the CS 8803 Instructional team for a great semester and teaching us the foundation to be able to produce a project like this one.

References

[1] F. Foundation. Pistonball - pettingzoo documentation. URL https://pettingzoo.farama.org/environments/butterfly/pistonball/.

[2] igilitschenski. Multi-car racing gym environment, 2020. URL https://github.com/igilitschenski/multi_car_racing.

[3] T. Lin, M. Huh, C. Stauffer, S.-N. Lim, and P. Isola. Learning to ground multi-agent communication with autoencoders, 2021. URL https://arxiv.org/abs/2110.15349.

[4] Y. L. Lo, C. S. de Witt, S. Sokota, J. N. Foerster, and S. Whiteson. Cheap talk discovery and utilization in multi-agent reinforcement learning, 2023. URL https://arxiv.org/abs/2303.10733.

[5] R. Sander. Emergent autonomous racing via multi-agent proximal policy optimization. Technical report, 05 2020.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[7] P. D. Siedler. Dynamic collaborative multi-agent reinforcement learning communication for autonomous drone reforestation, 2022. URL https://arxiv.org/abs/2211.15414.

[8] D. Simões, N. Lau, and L. Paulo Reis. Multi-agent actor centralized-critic with communication. *Neurocomputing*, 390:40–56, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2020.01.079. URL https://www.sciencedirect.com/science/article/pii/S0925231220301314.

[9] S. Sukhbaatar, A. Szlam, and R. Fergus. Learning multiagent communication with backpropagation. NIPS'16, page 2252–2260, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[10] A. Vanneste, T. Somers, S. Vanneste, K. Mets, T. D. Schepper, S. Mercelis, and P. Hellinckx. Scalability of message encoding techniques for continuous communication learned with multi-agent reinforcement learning, 2023. URL https://arxiv.org/abs/2308.04844.