IJCNN 2019. International Joint Conference on Neural Networks. Budapest, Hungary. 14-19 July 2019

Multi-Agent Deep Reinforcement Learning with Emergent Communication

David Simões*†‡, Nuno Lau*† and Luís Paulo Reis‡§
david.simoes@ua.pt, nunolau@ua.pt, lpreis@fe.up.pt
*DETI/UA - Electronics, Telecommunications and Informatics Department, University of Aveiro, Portugal
†IEETA - Institute of Electronics and Informatics Engineering of Aveiro, University of Aveiro, Portugal
‡LIACC - Artificial Intelligence and Computer Science Lab, Oporto, Portugal
§DEI/FEUP - Informatics Engineering Department, Faculty of Engineering of the University of Porto, Portugal

***Abstract*—When compared with their single-agent counterpart, multi-agent systems have an additional set of challenges for reinforcement learning algorithms, including increased complexity, non-stationary environments, credit assignment, partial observability, and achieving coordination. Deep reinforcement learning has been shown to achieve successful policies through implicit coordination, but does not handle partial-observability. This paper describes a deep reinforcement learning algorithm, based on multi-agent actor-critic, that simultaneously learns action policies for each agent, and communication protocols that compensate for partial-observability and help enforce coordination. We also research the effects of noisy communication, where messages can be late, lost, noisy, or jumbled, and how that affects the learned policies. We show how agents are able to learn both high-level policies and complex communication protocols for several different partially-observable environments. We also show how our proposal outperforms other state-of-the-art algorithms that don't take advantage of communication, even with noisy communication channels.**

***Index Terms*—multi-agent systems, neural networks, noisy communication, reinforcement learning, distributed deep learning**

I. Introduction

Communication is one of many different methods used in Multi-Agent Systems (MAS) to achieve coordination. They are dependent on the inherent communication constraints of the MAS, but are general and flexible methods, and allow agents to share low- and high-level information [1]. The contents of transmitted information fall under two main categories: informational (also known as low-level), where world state knowledge, beliefs, useful events and opportunities are shared [2], [3]; and propositional (also known as high-level), where intentions and goals are shared [4], [5].

The transmission of local information compensates for the partial observability of the environment, and helps reduce the complexity of a challenge. While some proposals feature hand-developed communication protocols [6], [7], others demonstrate that communication can also been learned *tabula rasa* [8], [9].

We propose a deep-learning framework for MAS, where a general reinforcement-learning algorithm is used for agents to learn policies and communication protocols simultaneously. Our algorithm, which we call Asynchronous Advantage Actor-Critic with Communication (A3C2), supports distributed execution of agents, has a distributed implementation, handles complex partially-observable domains, and is robust to noisy communication channels. Based on asynchronous actor-critic techniques, we demonstrate how coordination is achieved in multiple varied domains, by simultaneously learning policies and communication protocols.

The remainder of this paper is structured as follows. Section II describes the state of the art with respect to reinforcement learning algorithms that learn and use communication protocols alongside the learned policies. Section III describes our framework and algorithm, including our methodology to handle noisy communications, and Section IV shows results obtained in multiple MAS environments. Finally, Section V draws conclusions and lists future work directions.

This work is supported by: Portuguese Foundation for Science and Technology (FCT) under grant PD/BD/113963/2015; IEETA (UID/CEC/00127/2019); and LIACC (PEst-UID/CEC/00027/2019).

II. Related Work

Multiple recent proposals have focused on learning communications while learning agent policies. Some have strict assumptions and do not support distributed execution, while others focus on learning communication based in a specific set of symbols, or communication with discretized messages.

The Differentiable Inter-Agent Learning (DIAL) algorithm [10] has shown how messages can be used in a neural network to help optimize a policy. Through an end-to-end differentiable communication channel, agents learn to send messages which act as inputs for the policy networks. Gradients are pushed through this network and the communication channels in order to optimize the message networks. The authors assume perfect communication and discrete messages between agents, and test their proposal on 2-player short guessing games.

The CommNet [8], on the other hand, learns continuous communication protocols between agents. With a single network, multiple communication transmissions are performed at each time-step, and agents receive a jumbled sum of all messages sent by other agents. The algorithm outputs a joint-action, thus disallowing the distributed execution of the agents, and limiting its scalability to large numbers. The authors also assume perfect communication between agents.

978-1-7281-2009-6/$31.00 paper N-19388.pdf

The Multi-Agent Bidirectionally Coordinated Network (BiCNet) [11] has agents organized in a hierarchical order, and each agent shares RNN [12] states with its neighbors. Based on Minimax Q-Learning [13] and Joint-Action Learners [14], the policy network takes as input a shared observation of all agents, once again requiring centralized execution of agents. It is unclear what are the constraints of this sharing methodology and its robustness when communication channels can fail. RNN are also harder to train that simpler feed-forward networks [15].

Mordatch et al. [16] present an algorithm where agents learn communication based on a set of discrete symbols, based on a Gumbel distribution and a joint-reward function. Recurrent memory modules are also used, which allows agents to maintain a stream of symbols with a specific meaning, while learning fully cooperative policies.

Some authors also show how communication can arise in a mix of multi-agent reinforcement learning frameworks and supervised learning techniques. By training agents to maximize a goal, and interspersing the training with supervised learning, Lewis et al. [17] demonstrate agents that learn natural language protocols. Using dialogue rollouts, the models plan ahead in bargaining tasks, and fake interest to take advantage of high-value goals. The Multi-Step, Multi-Agent Neural Network (MSMANN) algorithm [18] uses supervised learning for decentralized agents to learn to imitate a centralized strategy. Agents learn action and communication policies simultaneously during centralized training. Authors leave a reinforcement-based approach for future work. Das et al. [19] propose an algorithm for a one-on-one cooperative game. Using Hierarchical Recurrent Encoder-Decoder neural networks to model policies, and the REINFORCE algorithm [20] for learning a communication policy, agents learn to communicate using a pre-determined vocabulary consisting of natural-language symbols. Eventually, one of the agents guesses what image the remaining agent was shown. Lazaridou et al. [21] also propose a communication learning algorithm for agents to use in order to identify images. In their work, communication is simply the action space with which policies are learned to complete tasks.

Our proposal focuses on continuous communication protocols, based on generic message passing, which supports both distributed execution, as well as a variable number of agents during execution. We do not require any additional information from the environment, other than the one individually supplied to each agent.

III. Asynchronous Advantage Actor-Critic with Communication

The simplest multi-agent adaptation of reinforcement learning algorithms is to apply them in a multi-agent environment, having each agent learning independently, as shown in the Independent Q-Learners algorithm [14]. Theoretical convergence guarantees are lost, due to the non-stationarity of the environment, but the method is versatile and very popular [22]. Another popular approach is the Joint-Action Learners [14], where a central entity decides the joint-action for all agents based on the joint observation of the team. This approach is highly restrictive, as it does not support decentralized execution.

Both these techniques are applicable to actor-critic methods, where each agent has its own actor and critic, from its own state-action history. Independent learning is straightforward, but the lack of information sharing at training time makes it difficult for agents to learn coordinated strategies that depend on interactions between multiple agents, or for an individual agent to estimate the contribution of its actions to the team's reward. In order to stimulate coordination, we share information through communication, by having agents learn to communicate crucial information in order to improve each other's policies. In order to use back-propagation techniques to optimize the communication protocols, an end-to-end differentiable model is required, and transmitted messages must adhere to this requirement. Despite making the model and training much more complex, this technique allows distributed execution of agents and compensates for the partial observability of the environment.

We now describe our proposed algorithm, Asynchronous Advantage Actor-Critic with Communication (A3C2). It is a multi-agent extension of Asynchronous Advantage Actor-Critic (A3C) [23], where communication is simultaneously learned. The algorithm is distributed and keeps global networks for each agent, as can be seen in Figure 1. Workers then asynchronously and periodically copy these networks to local memory, and update the global ones with mini-batches. Agents keep actor, critic, and communication networks during the learning phase, as can be seen in Figure 2, and require only the actor and communication networks for distributed execution. A3C is a specific case of A3C2, with the number of agents $J = 1$ and no communication.

We formally describe A3C2 in Algorithm 1. Workers copy the global networks into local memory and each handle an entire environment and set of agents for that environment. A worker samples each agent's observation, and computes an action and outgoing message. The environment may be partially observable, and each agent's observation may be a local partial observation of the environment's current state. Actions are executed on the environment, an action-state reward is obtained for each agent, and a new set of observations is sampled. After enough steps, or when a terminal state is reached, the loss of local networks is computed, those gradients are applied to the global networks, and these are then copied back into local memory. This process is repeated until convergence has been found.

An agent j optimizes its own actor network θ_a^j, by repeatedly taking observations s_t^j of the environment at time-step t, which are concatenated with the messages rc_t^j received from other agents on that same time-step. All agents compute an action a_t^j based on the policy $\pi(a_t^j|s_t^j, rc_t^j, \theta_a^j)$ given by their actor network, which is executed to start a new cycle. Every t_{max} time-steps, or when agents reach a terminal state, a mini-batch of state, action, and reward tuples is used to calculate

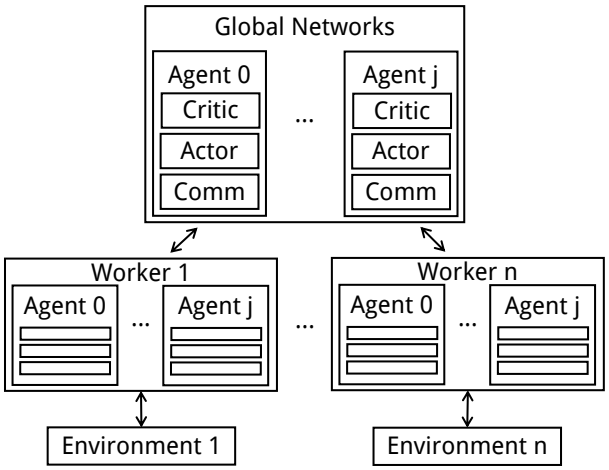


Fig. 1: The algorithm architecture, using n separate workers. Each worker keeps local copies of each agent's three networks, and interacts with its own environment and its separate set of j agents. As samples are collected in mini-batches, workers calculate gradients based on their local networks, asynchronously update the global networks, and update their copies.

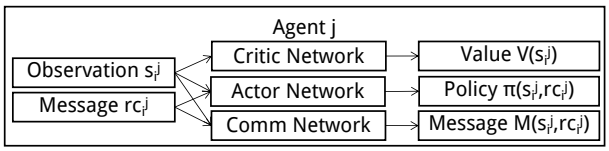


Fig. 2: The architecture of an agent j at time-step i, which sampled observation s_i^j and received messages rc_i^j. Each agent has three separate networks: a policy (or actor) network, which outputs an action probability $\pi(s_i^j, rc_i^j)$ (from which a_i^j is sampled); a communication network, which outputs an outgoing message sc_i^j; and a value network, which outputs a value estimation.

gradients, based on the error $R - V(S_i^j, \theta_v^j)$ between the actual returns and the critic's expectation, and the logarithm of the policy network's output strategies $\log \pi(a_i^j | s_i^j, rc_t^j, \vartheta_a^j)$. An entropy factor $H(\pi(a_i^j | s_i^j, rc_i^j, \vartheta_a^j))$ is also used, to discourage premature convergence.

An agent j also optimizes its own critic network θ_v^j, using its observations as input. If a terminal state has not been reached by the end of the mini-batch, an expected value $V(S_t^j, \theta_v^j)$ for the next state S_t^j is estimated with the critic network. Gradients are calculated with the squared error $(R - V(S_i^j, \theta_v^j))^2$ between the actual returns and the critic's expectation.

Finally, an agent j also concurrently optimizes its own communication network θ_c^j. This computes n-channel messages, where each channel has a continuous value based on the activation function (if any) of the communication network's output layer. For example, a 2-node output layer using binary activations computes 2-bit messages.

Each environment can define communication constraints, ranging from size to range properties. Additional properties, problem-dependent, can be specified, and may define

Require: Global shared learning rate η, discount factor γ, entropy weight β, number of agents J, network weights θ_a^j, network weights θ_v^j, network weights θ_c^j, batch size t_{max}, maximum iterations T_{max}, and default message value rc_{initial}
Require: Local network weights ϑ_a^j, network weights ϑ_v^j, network weights ϑ_c^j, and step counter t

$t \leftarrow 0$
$rc_0^j \leftarrow rc_{\text{initial}}$ for all agents j
for iteration T = 0, T_{max} **do**
  Reset gradients $d\theta_a^j \leftarrow 0$, $d\theta_v^j \leftarrow 0$, and $d\theta_c^j \leftarrow 0$ for all agents j
  Synchronize $\vartheta_a^j \leftarrow \theta_a^j$, $\vartheta_v^j \leftarrow \theta_v^j$, $\vartheta_c^j \leftarrow \theta_c^j$ for all agents j
  $t_{start} \leftarrow t$
  Sample observation s_t^j for all agents j
  repeat
    for agent $j = 1, J$ **do**
      Calculate message sc_t^j to send with $sc_t^j \leftarrow M(s_t^j, rc_t^j, \vartheta_c^j)$
      Sample action a_t^j according to policy $\pi(a_t^j | s_t^j, rc_t^j, \vartheta_a^j)$
      Map sent communication sc_t^j into received communication rc_{t+1}, and build communication map m_t
    end for
    Take action a_t^j for all agents j
    Sample reward r_t^j and new observation s_{t+1}^j for all agents j
    $t \leftarrow t + 1$
  until terminal s_t^j for all agents j or $t - t_{start} = t_{max}$
  for agent $j = 1, J$ **do**
    $R^j = \begin{cases} 0 \text{ for terminal state} s_t^j \\ V(s_t^j, \vartheta_v^j) \text{ otherwise} \end{cases}$
    $L_c \leftarrow 0$
    for step $i = t - 1, t_{start}$ **do**
      $R \leftarrow r_i^j + \gamma R$
      Value loss $L_{v_i}^j \leftarrow (R - V(s_i^j, \vartheta_v^j))^2$
      Actor loss $L_{a_i}^j \leftarrow \log \pi(a_i^j | s_i^j, rc_i^j, \vartheta_a^j)(R - V(s_i^j, \vartheta_v^j)) - \beta * H(\pi(a_i^j | s_i^j, rc_i^j, \vartheta_a^j))$
    end for
    for step $i = t, t_{start} + 1$ **do**
      Received communication loss $L_{rc_i}^j \leftarrow \frac{\partial L_{a_i}^j}{\partial rc_i^j}$
    end for
  end for
  Map received communication loss $L_{rc_{i+1}}$ into sent communication loss L_{sc_i} using communication map m_i for all agents
  for agent $j = 1, J$ **do**
    for step $i = t - 1, t_{start}$ **do**
      Accumulate gradients $d\theta_c^j \leftarrow d\theta_c^j + \frac{\partial L_{sc_i}^j}{\partial \vartheta_c^j}$
      Accumulate gradients $d\theta_a^j \leftarrow d\theta_a^j + \frac{\partial L_{a_i}^j}{\partial \vartheta_a^j}$
      Accumulate gradients $d\theta_v^j \leftarrow d\theta_v^j + \frac{\partial L_{v_i}^j}{\partial \vartheta_v^j}$
    end for
  end for
  Update network weights $\theta_a^j \leftarrow \theta_a^j + \eta d\theta_a^j$, $\theta_v^j \leftarrow \theta_v^j + \eta d\theta_v^j$, and $\theta_c^j \leftarrow \theta_c^j + \eta d\theta_c^j$ for all agents j
end for

Algorithm 1: Pseudo-code for a worker thread running Asynchronous Advantage Actor-Critic with Communication (A3C2).

whether learned networks support dynamic numbers of agents during execution. Communication can be categorized based on reliability (whether messages always reach their targets, and with which delay), connectivity (whether agents send messages to all other agents, spatially close agents, or specific agents), parallelism (whether equal or unique messages are sent to all other agents, and whether messages are distinctly received from others or jumbled together). Sending distinct messages to all other agents or receiving separate messages does not allow dynamic team-sizes during execution, as the network architectures become dependent on the team-size. These properties are captured within a communication map, which describes what messages were received by each agent, who sent them, and at what time-step. A standard value rc_{initial}

for a non-received message is used as the actor network input for both the initial turn and for when messages are lost. If agents are allowed to send messages to themselves, they form a recurrent network and can essentially create a memory channel.

Every time-step t, an agent j determines a message sc_t^j to send. At time-step $t+1$, that message is received as message rc_{t+1}^j by other agents, based on the communication properties. Every $t_{\max}$ time-steps, or when agents reach a terminal state, the actor network's loss L_a^j is used to calculate its gradients $L_{rc_i}^j \leftarrow \frac{\partial L_{a_i}^j}{\partial rc_i^j}$ with respect to the received messages. The communication map is then used to map these gradients as the loss L_{sc_i} of the messages sent by other agents in the previous cycle, which is then minimized to optimize the communication network. An example can be seen in Figure 3, where three agents use broadcast communication.

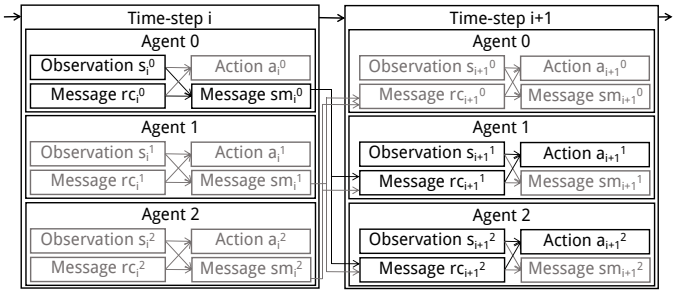


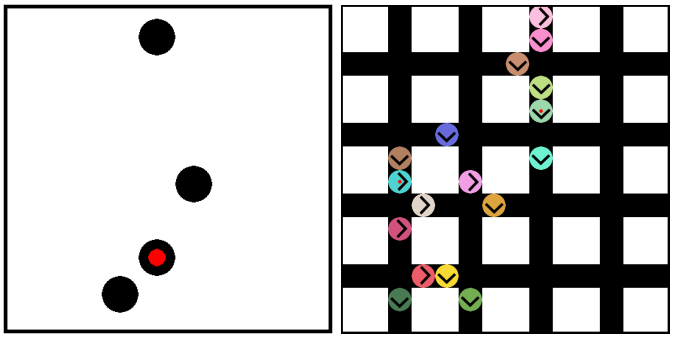
Fig. 3: Diagram of how broadcast communication with three agents is performed across two time-steps (arrow direction), and of how gradients are propagated backwards (emphasized lines). Agent 0 sends messages to Agents 1 and 2, which calculate the gradients of their policy error with respect to that message. Those gradients are pushed as the error of the originally sent message into Agent 0, who uses them to optimize its Communication network.

The intuition behind this is that an agent j optimizes the messages sent by other agents to improve its own policy. By doing this for all agents, they learn to output information that is useful for others, thus learning coordination and optimizing each other's policies. The error of a sent message can be summed, averaged, or otherwise combined from the gradients of that message received by multiple agents. Summing the error may lead to very large network updates, while averaging them leads to safer but slower updates.

Homogeneous agents may share the same actor and communication networks, and agents with the same reward function may share the same critic network, for efficiency.

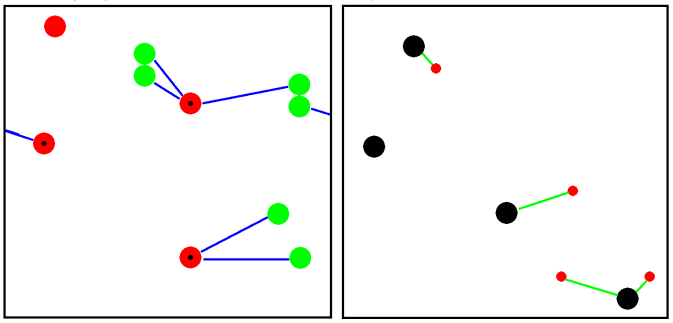
IV. Results

We test our proposal in multiple multi-agent environments where communication is crucial to overcome the partial observability of the environment. We test against A3C (a specific case of A3C2 with no communication), and against a Joint-Action ensemble learner (using A3C as well), which gathers the observations of all agents and outputs and joint-action for the entire team. This Joint-Action learner could not learn any successful policy in any of our environments within the same amount of iterations as A3C2. We test a Hidden Reward challenge, a Traffic Intersection simulator, a Pursuit game, and a Navigation task, all described in the following sections, and shown in Figure 4. The Hidden Reward game focuses on classic exploration, and agents need to learn how to efficiently explore the environment and alert team members when the target is found. The Traffic Intersection is a close-horizon game, with large amounts of agents, where agents need to learn to adhere to rules and overcome multiple indistinguishable intersections. The Pursuit game is a classical benchmark where agents need to explore and coordinate in order to capture the prey, and the Navigation task focuses on goal assignment, and agents learn how best to distribute themselves in order to complete the task. We publish a video of learned policies on our repository, linked below.



(a) Hidden Reward environment. Agents (black) explore the map and know when they have reached the reward zone (red). They must then broadcast that location to remaining agents.

(b) Traffic Intersection environment. Agents (colored) must cross intersections without hitting. They are given a desired direction (black arrow) and are penalized if they collide (red marker).



(c) Pursuit environment. Predators (red) must chase, surround and capture the prey (green), which are hard-coded to run from the closest predator (shown by the blue line).

(d) Navigation environment. Agents (black) must cover all the beacons (red), and are rewarded by how close any agent is to each beacon (shown by the green line).

Fig. 4: Examples of tested multi-agent environments.

We also test the effects of noise in the communication channels against a baseline where communication is unhindered. We focus on three major types of noise:

- Loss - Covering messages that are lost, sent messages

have a chance P_{loss} of not being delivered. This also covers delays in messages, and we harshly assumed a delayed message to be lost.
- Noise - Covering external interference in received messages, as these are continuous-valued vectors. Gaussian noise $\mathcal{N}(0, V_{\text{noise}})$ is added to these values.
- Jumble - Covering internal interference in messages, received messages have a chance P_{jumble} of having been mixed with others. Instead of receiving the original message, that message is instead a sum of all received messages, and there is no indication to the agent whether the message has been jumbled or not.

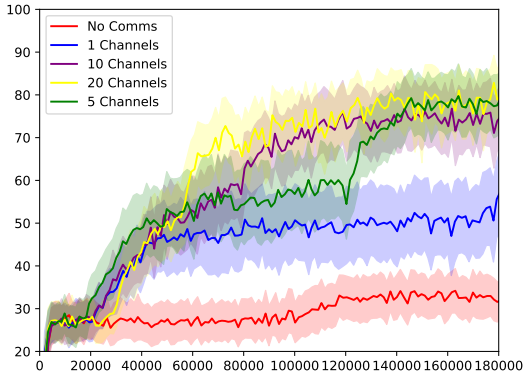
We test each individual effect, as well as all three simultaneously (shown as "All" in the following sections).

Unless otherwise stated, we used a learning rate $\eta = 10^{-4}$, a future reward discount factor $\gamma = 0.9$, an entropy weight $\beta = 0.01$, $J = 4$ agents, and $N = 3$ concurrent worker threads. We tested several network architectures, with less and more layers, and different amounts of nodes per layer, and focused on the simplest we found that yielded positive results. The actor, critic, and communication networks had two fully connected hidden layers of $40x$ and $20x$ nodes activated with a ReLU function, where x is a problem-dependent complexity measure. This was the simplest architecture we found that consistently converged to proper policies, and allowed us increase the speed of our tests. The communication network had a single fully connected hidden layer of $20x$ nodes activated with a hyperbolic tangent function, with a non-received message rc_{initial} default value being all zeros. All networks' initial random weights were computed with the Xavier initializer [24] with default parameters, and were optimized with the Adam optimizer [25] with default parameters. The chosen learning rate η was also the highest and fastest we found that consistently allowed the networks to converge. The future reward discount factor γ depends on the importance of future rewards, with 0 leading to policies that focus on short-term rewards, and 1 focusing on long-term expected rewards. We used $P_{\text{loss}} = 0.5$, $V_{\text{noise}} = 0.5$, and $P_{\text{jumble}} = 0.5$ to simulate noise in communications.

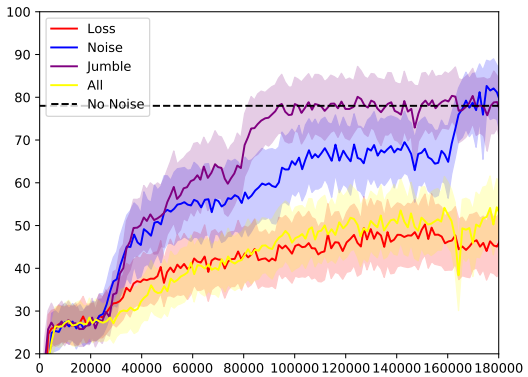
A. Hidden Reward

Inspired by robotics competitions [26], the Hidden Reward challenge is a multi-agent partially-observable domain where agents explore a map until they find a hidden reward zone. Within a short time-limit, agents should find this zone, and more points are earned with more agents having found it. Rewards are individual, since agents that haven't found the reward zone get no points, as opposed to agents on it, and collisions are disabled. Agents can only move in several directions, sense whether they are on the reward zone and the coordinates of each agent in the team, and broadcast messages to other agents. Optimal agents spread out through the map (exploring it in a coordinated manner) and warn others about the reward zone's position when it is found.

We tested a 4-agent team with a time-limit that does not allow a single agent to fully explore the entire map, and an $x = 1$ complexity factor for the network architecture. This forces agents to coordinately explore and alert others when the reward zone is found. For noise analysis, we used 20-channel communication. We can see in Figure 5(a) shows the policy evolution across training episodes of the team, using multiple communication properties. We can see that, without communication, the average reward stagnates around 30, since agents cannot share the reward zone's position, and thus resort to random exploration to find it. However, with a single communication channel, a better policy can immediately be found. This continuous channel helps reduce the area of exploration, and allows the policy to greatly improve. With at least two channels, the optimal policy is found, and as we increase the amount of channels up to twenty, the speed at which the policy is found also increases. Figure 5(b) shows how communication can be robust to both noisy interference and jumbled messages, as they allow some non-negligible form of coordination. Message loss has the greatest impact on the performance of the team, and only allows a slightly better policy than one without communication to be learned. We believe a larger learning time can help compensate for this.

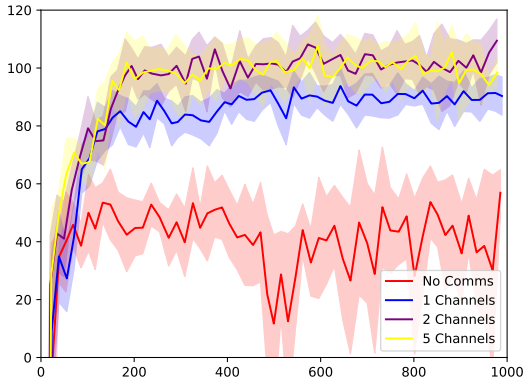


(a) Communication channel comparison.

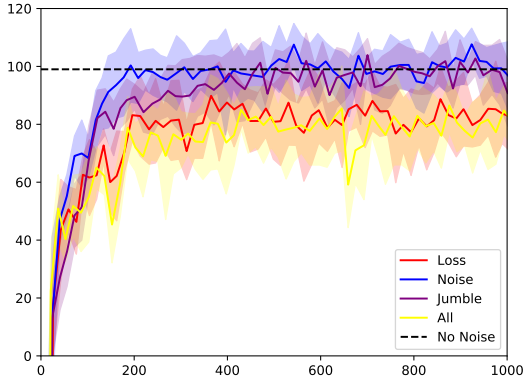


(b) Communication noise comparison.

Fig. 5: Results of different communication channels for the Hidden Reward environment. The plots represent the average reward and standard deviation obtained by agents, over training episodes.



(a) Communication channel comparison.



(b) Communication noise comparison.

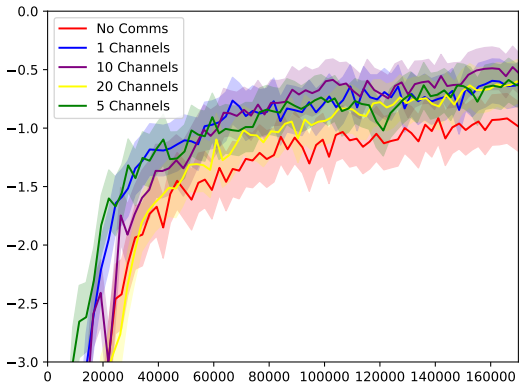
Fig. 6: Results of different communication channels for the Traffic Intersection simulator. The plots represent the average reward and standard deviation obtained by agents, over training episodes.

B. Traffic Intersection

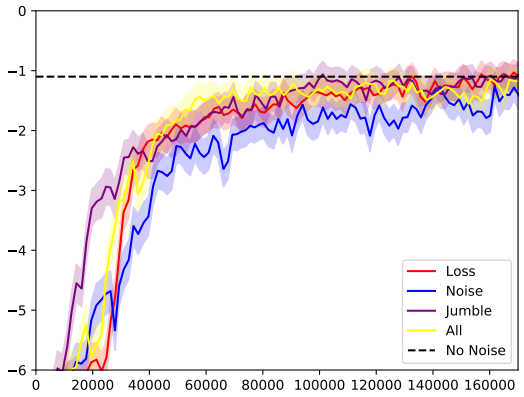
The Traffic Intersection simulator is a multi-agent partially-observable domain where multiple road intersections are crossed by cars coming from north or west and going south or east. Agents know which direction they intend to go, and see nearby cars. There are small penalties for stalling, and large penalties for colliding (larger if car did not have priority). In a collision, one of the agents is randomly allowed to move forward, while the other remains in the same position. The amount of cars on the map at any given moment is variable, and agent communication is spatially constrained. Optimal agents inform their desire to turn (or lack thereof) to other cars at intersections, and based on the information, decide which car should move, and which one should wait.

We tested a 40-agent team with 6 by 6 road intersections, with new cars showing up with a frequency of 0.5 cars/cycle, and an $x = 1$ complexity factor. Such a large amount of agents forces them to avoid waiting until no other cars are at the intersection, as that would lead to car jams. Because intersections are indistinguishable and lead to an unstable value estimation, we use a discount factor $\gamma = 0.01$. For noise analysis, we used 5-channel communication. We can see in Figure 6(a) that how communication is crucial to achieve a decent policy, where traffic flows easily and without collisions. Without communication, no suitable policy is found, since agents cannot communicate whether they have to turn or not, which leads to agents always trying to cross the intersection, and randomly colliding. Multiple communication channels are required to achieve an optimal policy (agents either learn to indicate that they have to turn, or that they have to move forward, depending on the randomly initialized weights). Figure 6(b) shows how learned policies can handle all types of noise. Message loss most affects the performance of the team, as losing messages at intersections forces agents to wait for that turn if they do not have priority. This learned behavior is necessary for to collision avoidance, but even with all noise types enabled, cooperation is still achieved in the simulation and results are better than policies with no communication.

C. Pursuit



(a) Communication channel comparison.



(b) Communication noise comparison.

Fig. 7: Results of different communication channels for the Pursuit game. The plots represent the average reward and standard deviation obtained by agents, over training episodes.

The Pursuit game, or Predator/Prey game, is a multi-agent partially-observable domain and a classical MAS benchmark, where a team of predators must capture a team of prey. Prey have a fixed policy, complete map vision, and run from the closest predator. Predators have local partial observations

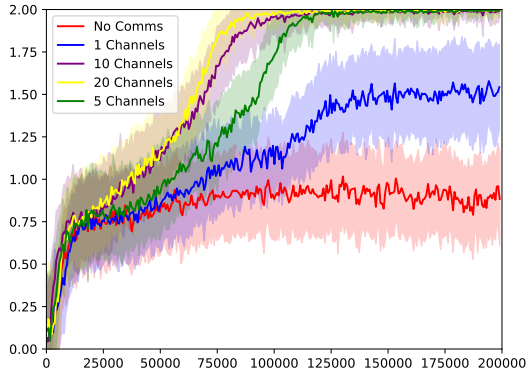
of the environment, being only able to perceive their own coordinates and a small area around themselves. Collisions are penalized and lead agents to random placements, while messages are broadcast to all other predators. Optimal predators explore the map without overlapping until a prey is found, alert other predators, and converge on it until it's captured.

We tested a 4-agent team on a map where each predator can see less than 10% of the complete map, and all prey must be captured within 100 time-steps, and an $x = 2$ complexity factor. Shared rewards are used when prey are captured, and penalties are used when collisions occur. Because episodes are longer, and policies harder to learn, we used a future reward discount factor $\gamma = 0.95$, and $N = 12$ concurrent worker threads. For noise analysis, we used 10-channel communication and a bigger map, to emphasize the effects of noise. We can see in Figure 7(a) that, without communication, an almost optimal policy is eventually found, as a good exploration tactic can compensate for the lack of communication. Despite this, analysis of the policy's behavior and of the average time taken to complete a game shows many cycles are wasted while a single predator chases a prey, until another predator randomly crosses its path. As communication channels increase, a better policy can be found, since searching for prey is no longer random, and there is also now a communication protocol for coordinating the capture. Figure 7(b) shows how multiple agents can learn robust communication protocols, barely hindered by multiple amounts of noise, and still successfully able to complete the task.

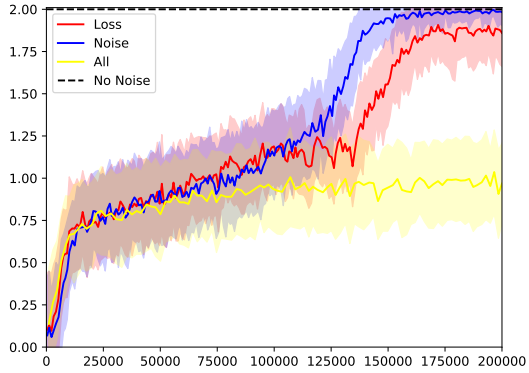
D. Navigation

Inspired by swarm challenges [27], the Navigation tasks consists on multiple agents coordinating to cover all the beacons available on the map. Contrary to the Hidden Reward environment, agents now know the locations of the beacons, but not the positions of the remainder of the team. The task is considerer successful when all beacons are covered, and agents can communicate with all other team members. Optimal agents will broadcast their own position and perform coordinated task assignment, by deciding which agent covers which beacon and minimizing the time taken for all beacons to be covered.

We tested a 2-agent team on a map with the same amount of beacons, and an $x = 1$ complexity factor. When both beacons are covered or the time limit has elapsed, agents received a normalized shared reward, based on the minimum distance of each beacon to any agent. For noise analysis, we used 20-channel communication. We can see in Figure 8(a) that, without communication, no suitable policy is found, since agents cannot share their position, and thus resort to pure chance to cover all the beacons. Agents try to cover both beacons simultaneously and end up failing the task. However, communication allows agents to coordinate with one another, and while a single channel does not allow enough information to be conveyed, multiple channels lead to optimal policies. Figure 8(b) shows how message loss or noise slow-down the learning process and increase the variance of the team's performance. Since only two agent exist, the jumbled noise type is negligible and not shown. With both noise types engaged, the team does not learn how to complete the task within the amount of episodes shown, possibly due to P_{loss} leading to less communication samples to learn from.



(a) Communication channel comparison.



(b) Communication noise comparison.

Fig. 8: Results of different communication channels for the Navigation environment. The plots represent the average reward and standard deviation obtained by agents, over training episodes.

V. Conclusion

Deep reinforcement learning is a general technique for learning complex policies in single-agent environments. We propose the Asynchronous Advantage Actor-Critic with Communication (A3C2) algorithm, which is based on distributed actor-critic deep learning techniques, but targets multi-agent environments where communication is necessary to achieve coordination and successfully complete tasks. The algorithm handles varying number of agents during distributed execution, heterogeneous reward functions, imperfect communication channels, and partially-observable domains.

We formally describe the algorithm and show how noisy communication affect its learning process. We report results in multiple multi-agent tasks, namely traffic simulators with tens of agents, Pursuit and Hidden Reward games, and swarm-inspired Navigation tasks, all partially-observable domains, complex and requiring communication. As expected, noise hinders the communication and overall performance

of the team, but agents can still learn successful policies and communication protocols *tabula rasa*. While the meaning of messages shared between agents may not be easily inferred from external human observers, agents compensate for the partial observability of the environment, alert other team members of new information, perform task assignment, optimize exploration patterns, and learn to coordinate, all without any additional information from the environment. We have published our source-code and environments at https://github.com/bluemoon93/A3C2.

Future research can focus on multiple fields. From a software approach, a coordination protocol that allows agents and worker threads to run on different machines will further increase the scalability of our system. Other advantage estimators, like Generalized Advantage Estimation (GAE) [28], can be used to increase the speed and reliability of the learning process. The critic network can be enhanced with more inputs, such as agent actions, or received communications, as well as with centralized information, like the concatenation of each agent's local partial observation. Memory channels can also be exploited by either using neural frameworks like LSTM or RNN, or by allowing agents to send messages to themselves, which creates a recurrent network. Additional environments, network architectures, algorithm parameters, team sizes, or noise sources will also demonstrate the robustness of our algorithm.

REFERENCES

[1] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[2] Nuno Lau and Luis Paulo Reis. Fc portugal-high-level coordination methodologies in soccer robotics. In *Robotic Soccer*. InTech, 2007.

[3] Prasanna Velagapudi, Oleg Prokopyev, Paul Scerri, and Katia Sycara. A token-based approach to sharing beliefs in a large multiagent team. In *Optimization and Cooperative Control Strategies*, pages 417–429. Springer, 2009.

[4] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[5] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[6] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM, 1994.

[7] Mihai Barbuceanu and Mark S Fox. Cool: A language for describing coordination in multi agent systems. In *ICMAS*, pages 17–24. Citeseer, 1995.

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

[9] David B DAmbrosio, Skyler Goodell, Joel Lehman, Sebastian Risi, and Kenneth O Stanley. Multirobot behavior synchronization through direct neural network communication. In *International Conference on Intelligent Robotics and Applications*, pages 603–614. Springer, 2012.

[10] 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.

[11] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *CoRR*, abs/1703.10069, 2017.

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[13] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, pages 157–163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[14] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[15] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

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

[17] Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning of negotiation dialogues. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2443–2453. Association for Computational Linguistics, 2017.

[18] Qiyang Li, Xintong Du, Yizhou Huang, Quinlan Sykora, and Angela P. Schoellig. Learning of coordination policies for robotic swarms. *CoRR*, abs/1709.06620, 2017.

[19] Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2951–2960, 2017.

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

[21] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. *Proceedings of the International Conference on Learning Representations*, 2017.

[22] 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.

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[24] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations, San Diego*, 2015.

[26] David Simões, Rui Brás, Nuno Lau, and Artur Pereira. A coordinated team of agents to solve mazes. In *Robot 2015: Second Iberian Robotics Conference*, pages 381–392. Springer, 2016.

[27] Martin Saska, Jan Chudoba, Libor Precil, Justin Thomas, Giuseppe Loianno, Adam Tresnak, Vojtech Vonasek, and Vijay Kumar. Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 584–595. IEEE, 2014.

[28] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference for Learning Representations*, 2016.