Cooperation Learning in Time-Varying Multi-Agent Networks

Vasanth Reddy Baddam[1], Almuatazbellah Boker[2], Hoda Eldardiry[1]
[1]Department of Computer Science
[2]Department of Electrical and Computer Engineering
Virginia Tech
vasanth2608@vt.edu, boker@vt.edu, hdardiry@vt.edu

Abstract

We propose a Multi-Agent Systems coordination framework for complex and dynamic environments, where agents' neighbors vary over time. We consider an agent-agent communication network, where we represent time-varying interactions as weighted network edges, and propose a heat kernel mechanism to compute those time-varying edge weights. We represent our network as a graph and build a heat diffusion kernel using the graph Laplacian. The key idea of our proposed approach is to model time-varying edge weights using the heat diffusion approach. Our proposed approach Cooperation Learner in MultiAgent Networks (CooLMAN) has a number of features: (1) it captures information flow in a dynamic environment using temporal indexing, (2) agents can achieve optimal policy and stability by the system-enabled timed interaction and coordination, thereby (3) providing trained weights that can be deployed to larger swarms in a scalable manner. We demonstrate the effectiveness of our proposed method using a cooperation task, where our model significantly outperforms state-of-the-art method.

Introduction

Reinforcement Learning (RL) is an emerging field in which learning happens through trial and error. RL is especially advantageous in real-time applications when an agent is expected to perform tasks in hazardous environments such as extreme weather conditions, high altitudes, or uncertain conditions (e.g, in outer space). In some environments, only a single agent is modeled and trained to perform the designed task. However, in many other cases, there is a need for more than one agent to cooperate with other agents to accomplish a task.

Multi-Agent Systems (MAS) is a field that focuses on control, collaboration between robots, and a wide range of problems where agents need to cooperate to achieve global tasks and sometimes compete. The remarkable characteristic of MAS is that it enables robots to work together to reach a common goal. Robots can have similar or various tasks depending on their role (local and/or global), design objectives, and environmental conditions. MAS can solve complex tasks that individual robots cannot. This is due to several key advantages of MAS including more robustness, fault tolerance, and low-cost operation compared to single robots. In MAS, agents typically need to mutually achieve an optimal policy in cooperative settings like traffic control (Wei et al. 2019), multirobot control (Cortés and Egerstedt 2017), smart grid network (Mocanu et al. 2016), multiplayer games (Jaderberg et al. 2019), and construction tasks (Petersen, Nagpal, and Werfel 2011), (Sartoretti et al. 2019). This gives rise to a set of MAS challenges including non-stationarity, communication, task allocation, and scalability.

Jiang (Jiang and Lu 2018) proposed Graph Convolutional Reinforcement Learning (DGN), in which MAS is modelled as a graph. where nodes represent the observation encoding of agents and edges represent the dynamic communication channel between agents through which they can interact and exchange the information between the each other. Thus sharing the information would let the agents perceive the area of the environment that was not in their vision and could overcome the non-stationarity issue. DGN uses attention mechanism as the convolutional kernel to extract the representation between the agents. Additionally, to mitigate the inconsistency in learning the cooperative policy occurring from changing neighbors, they temporally regularized the attention weights (representation weights). They empirically were able to outperform the existing methods and show the consistent cooperation between the agents.

Inspired by the work of (Jiang et al. 2019), we propose a mechanism for explicit parameter sharing across agents in a dynamic environment. Our proposed parameter sharing idea is based on the well-known heat diffusion mechanism (Unsworth and Duarte 1979). In particular, we use heat diffusion to represent the flow of exchange of information. Unlike the convolutional kernel-based approach proposed by Jiang et. al (Jiang et al. 2019), heat kernel captures the gradient flow of information and provides stability when the neighboring agents changes constantly. Moreover, we use multiple channels with heat kernel in parallel to capture the different variations of information and concatenating them together to capture the information enriched encoder. The model parameters are then trained end-to-end and updated based on Proximal Policy Optimization. We refer to our algorithm as *CooLMAN*, which is an abbreviation for *Cooperation Learner in Multi Agent Networks*. We show the efficacy of our approach in the context of battle environment in which two competing groups try to replace each other.

Through this work, we show empirically that CooLMAN outperforms (Jiang et al. 2019) and DQN (Mnih et al. 2013) in a dynamic cooperative environment. More specifically, we show that by following our approach all the agents cooperatively achieve a relatively large mean reward when compared to other methods. Furthermore, though we train our model on a smaller network of agents, through our experimental results we show that CooLMAN is able to scale to relatively larger network with the considerate average returns.

Related work

Multi-Agent Reinforcement Learning (MARL) is one of the fields that is gaining momentum in recent years. At first, they used to train each agent individually assuming that all other agents are part of the environment (Tan 1993). However, this process creates instability and inconsistent policy since the environment is non-stationary and it would eventually fail to scale for more agents. This is the main reason for us to focus on the dynamic (time-varying) environment. In this section, we discuss different architectures relevant to the MARL domain.

In Decentralized learning, each agent acts independently without any explicit information from other agents towards the common goal. Previous work proposed Hysteric Q-learning, which uses an extension of tabular - Q learning (Watkins and Dayan 1992), and performs tasks without any need for additional communication assuming that it is fully observable (Matignon, Laurent, and Le Fort-Piat 2007). More recent work has focused on partial observability by using the Decentralized Hysteric Deep Recurrent Q-Network (Dec-HDRQN) model which surmounts the non-stationarity (Omidshafiei et al. 2017). Both of these approaches suffer from a number of limitations. The first is the convergence and then the scalability, as it does not go beyond two agents. Dec-HDRQN (Omidshafiei et al. 2017), achieves cooperation but is still limited as there is no communication setup between the agents.

The work in (Lowe et al. 2017) employs the actor-critic algorithm in which a centralized critic is learned by using the joint set of states and actions of all agents. During execution, only an actor is used for individual agents that choose the policy for the observed individual state. The work in (Foerster et al. 2017) is similar to (Lowe et al. 2017) as it employs a centralized critic and an individual actor. It addresses the multiagent credit assignment by using a counterfactual baseline that marginalises out a single agent's action while keeping the other agents' actions fixed. Value decomposition in (Sunehag et al. 2017) used centralized critic to learn the value function during training and deploying the actor during execution, decomposes central state action value function into a sum of individual agent functions. However, it limits the complexity of the action value function and, additionally, it does not use the whole state information. QMIX (Rashid et al. 2018) overcomes this limitation by eliminating the necessity of additive decomposition of the centralised critic. This architecture enables scalability as the trained weights can be reused on any number of agents but does not consider inter-agent communication.

In all the previously mentioned work, there is no communication setup between agents. Instead, it only managed to achieve co-operative policy during training. TarMac (Das et al. 2019) uses attention mechanism to enable the inter-agent communication, where an agent learns what kind of messages and to whom it should be addressed. This model shows that communication between agents improves the co-operation policy. DGN (Jiang et al. 2019) follows the same approach but with limited communication between agents. More precisely, DGN restricts communication to only a certain number of agents using the distance metric, and thereby reducing communication cost. Moreover, DGN introduced regularisation kernel that helps in stabilizing the long term co-operative policy. However, all the above work do not consider time-varying edges in the dynamic multi-agent networks. Our work bridges this gap by modelling the time-varying edges through the function of heat diffusion.

Problem Formulation

We consider a partially observable, fully cooperative task, where the MAS is described by a Markov Decision Process (MDP). A Markov Decision Process G is defined by the tuple $M_G = \langle S, A, P, R, O, N, \gamma \rangle$, where the environment has N agents receiving the local observations $(o_1, o_2, \cdots o_n) \in O$ which has the partial information of the global state $s \in S$.

For each time step, each agent chooses an action, $(a_1, a_2, ..., a_N) \in A$ following the transition probability function $P : S \times a_1 \times a_2 \times \cdots \times a_N \rightarrow [0, 1]$, for which each agent i, $i \in (1, 2, ...N)$, receives an individual reward, $r_i \in R$ and transits into the next state $o_i^{'}$. $\gamma \in [0, 1]$ is the discount factor which prioritize rewards in the immediate time steps. Accordingly, conditioning on the partial observation and the transition probability, each agent learns a policy $\pi_{\Phi_i} : o_i \times a_i \rightarrow [0, 1]$. where Φ_i are the parameters of the policy network.

The goal is that each agent learns a policy that maximizes their expected discounted returns, which is given by

$$J(\Phi_i) = \mathbb{E}_{<o_i, a_i, r_i, o_i^{'}> \sim \mathcal{D}} \left[\sum_{t=0}^{\infty} \gamma^t * r_i(o_i, a_i, o_i^{'}) \right] \quad (1)$$

where $\mathcal{D}$ is the experience buffer used to store the experience tuples, $< o_i, a_i, r_i, o_i^{'} > .. \mathbb{E}$ represents the expected sum of the samples. Ultimately, the agents need to cooperate together to solve a global objective, which is based on the composition of the local objectives. The agents will share information with neighbors while optimizing their own local objective (1).

Background

Policy Gradient for Single Agent The core idea of policy gradients methods is to tune the parameters Φ of policy network in the direction of the gradient of the objective function $J(\Phi)$ using the stochastic gradient descent, minimizing the objective function $J(\Phi)$ and it is given as:

$$J(\Phi) = \mathbb{E}_{s \sim \mathcal{D}, a \sim \boldsymbol{\pi}_\Phi} \left[\log \pi_\Phi(a \mid s) \bar{A} \right], \quad (2)$$

where π_Φ is the policy and $\bar{A}$ is the advantage function, which could be the expected return as given in (1). The objective

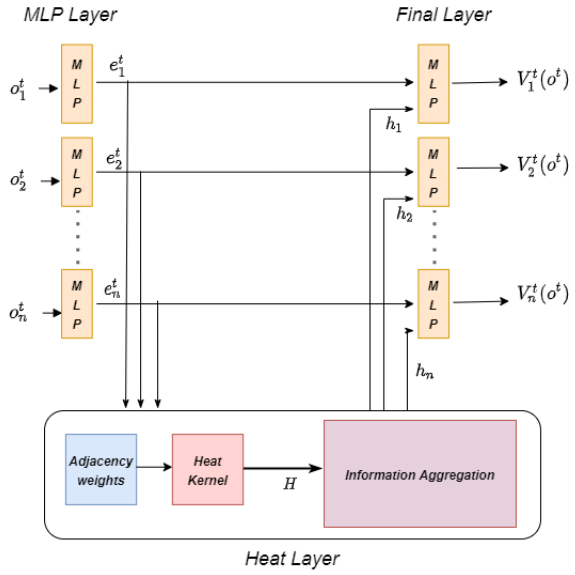


Figure 1: Illustrates the main architecture of the proposed heat diffused critic model, which perceives the local observations of all the agents and gives out the state values for each agent

function differs in how we choose to estimate the advantage function. The gradient for equation (2) is given by:

$$\nabla_\Phi J(\Phi) = \mathbb{E}_{s\sim\mathcal{D},a\sim\boldsymbol{\pi}_\theta}\left[\nabla_\Phi \log \pi_\Phi(a \mid s)\bar{A}\right]. \quad (3)$$

The main drawback of the Policy Gradient method is that it needs to perform multiple number of policy step updates on the same batch of experiences. This would lead to destructively larger policy updates (Schulman et al. 2015a).

Proximity Policy Optimization This method overcomes the above problem by updating the policies by taking a large step updates while keeping the measure of closeness of new policy to old policy as the objective. The main idea here is to clip the objective in between certain values, inhibiting the overflow of the measure of the closeness.
The objective function with the clipping contribution is given by (Schulman et al. 2017):

$$J^{CLIP}(\Phi) = \mathbb{E}\left[min(r(\Phi) * \bar{A}, clip(r(\Phi), 1-\epsilon, 1+\epsilon))\right], \quad (4)$$

where $r(\Phi)$ is the probability ratio between the new policy and old policy and it is given as $r(\Phi) = \pi_{\Phi_{new}}(a \mid s)/\pi_{\Phi_{old}}(a \mid s)$ and ϵ is the clipping parameter. min and $clip$ are the functions to find the minimum and to clip the values, respectively. Equation (4) is used to optimize the policy network parameters.

In the next section, we are going to extend the Proximity Policy Optimization to Multi Agent Systems by: 1. Utilizing the policy update as given in equation (4). 2. presenting the novel state value network i.e. Heat Diffused Critic, to compute the advantage function, which is used in equation (4).

Proposed Approach

Cooperative Learner in Multi Agent Network(CooLMAN)

In extending the PPO to Multi Agent Systems, we utilize the heat diffused critic networks and actors during the training phase and deploy the actors in a decentralized manner. To adapt this, we use two separate networks for each agent; one for critic V_θ to estimate the value of the state and another network π_Φ for actor to predict the policy. where, θ are the parameters of the critic network.

Heat Diffused Critic In this work, we represent the MAS network as a graph network, in which observation features are accommodated as the nodes and the parameter sharing channels as the edges between the nodes. Taking this into account, we model the parameter sharing between the critic networks of other agents as a heat function. We employ the graph's heat kernel to mimic a heat transfer diffusion process over time, which is subsequently used by the critic to appropriately weight the signals received by neighboring critic nodes. Since we allowed to share the parameters, our learning paradigm would also help in overcoming the non-stationarity issue and also further helps in learning the stable policy (Omidshafiei et al. 2017). Figure 1 demonstrates the main components of our critic networks. The proposed critic framework method involves the following components:

Multi Layer Perception (MLP) Layer: Local observation o_i of an agent i is fed into the MLP to extract the hidden state e_i, which further can be used to capture the important information from the neighbors as well as passing the hidden state to the final layer.

Each critic perceives the observation o_i of an agent and the important information from the neighboring agents as an input and outputs the value function $V_i(o)$.

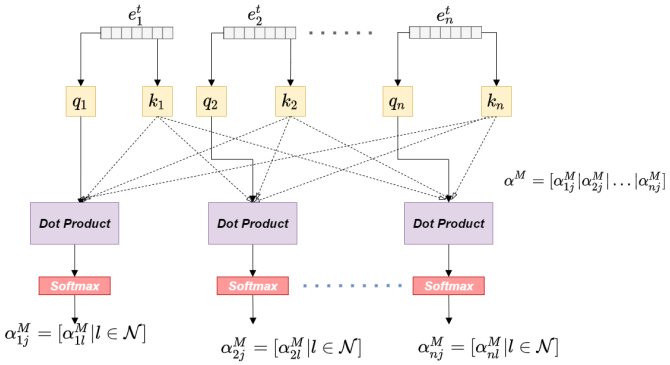
$$V_i(o) = F_i(b_i(o_i), h_i) \quad (5)$$

where, F_i and b_i are linear functions. h_i is the aggregated information from the i^{th} agent neighboring agents. It should be noted that MLP can also be replaced by Convolutional Neural Network or Recurrent Neural Network based on the observation input. The following layer contains the mechanism of how to find the aggregated information h_i from the neighboring agents.

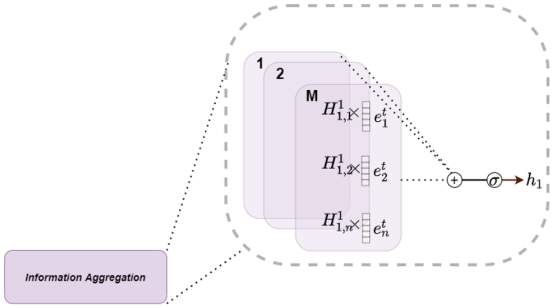
Heat Layer: We represent the critic networks as the nodes in the time-varying graph. Each node embodies the observation feature which is an input to the critic. The edge across the nodes serves as a communication link between agents through which they could share the information. Considering a weighted graph with n nodes, $\mathcal{G}^t = (V, E^t)$, where $v_i \in V$ is the node of the graph and $e^t_{ij} \in E^t$ is the edge connecting the node i and node j at a time t. The Laplacian matrix $\mathcal{L}^t$ of the graph network $\mathcal{G}^t$ at time t is given by:

$$\mathcal{L}^t = D^t - A^t, \quad (6)$$

where, D^t is the diagonal matrix at a time t and it is given by $D^t = \sum_{v_j \in V} A^t(v_i, v_j)$ and A^t is the adjacency matrix,



(a) Adjacency matrix elements



(b) Information aggregation

Figure 2: Left (a) illustrates the dot-product mechanism outputs the adjacency matrix elements and the same would be followed for M communication channels. Right (b) illustrates the step Information aggregation which emulates the information from the agents neighborhood and concatenating the m different variations of information to produce the final feature vector.

whose elements are given by

$$A_{ij}^t = \begin{cases} \alpha_{ij} & \text{if } (v_i, v_j) \in E^t \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Recall that each critic has a hidden state embedding e_i^t at a time t that correlates with the observation state o_i^t and the whole graph network has a set of hidden state features $(e_1^t, e_2^t, .e_3^t, ..., e_n^t)$ as the features of nodes. We find the encoding of each edge using dot-product mechanism (Vaswani et al. 2017), which contains the information of agents connecting that edge. This can be done as follows

$$e_{ij}^t = (q_i^t W_e) \cdot (k_j^t W_c)^T, \tag{8}$$

where W_e and W_c are trainable weights. where, q_i and k_i are the copies of the embedding state e_i.

We next find the elements of the adjacency matrix by passing the encoding of edges through the softmax function so that the adjacency matrix is row stochastic. This can be done as follows

$$\alpha_{ij} = \frac{\exp(\frac{e_{ij}}{d})}{\sum_{k \in \mathcal{N}_i} \exp(\frac{e_{ik}}{d})}, \tag{9}$$

where $\mathcal{N}_i$ is the neighborhood set of agent i and d is the dimension of hidden state. This is illustrated in Figure 2a.

Note: In this paper, we mostly use the normalized Laplacian matrix $\hat{\mathcal{L}}^t$, as it is the fundamental term of the heat flow equation and it is given by $\hat{\mathcal{L}} = D^{-\frac{1}{2}}\, \mathcal{L}\, D^{-\frac{1}{2}}$, ignoring time t for simplicity. Normalized Laplacian matrix is then represented in its eigenspectrum form $\hat{\mathcal{L}}^t = \Phi^t \Lambda^t {\Phi^t}^T$, where $\Lambda^t = diag(\lambda_1^t, \lambda_2^t, \lambda_3^t, ..., \lambda_n^t)$ is the diagonal matrix with ordered eigenvalues of $\hat{\mathcal{L}}^t$ as elements and $\Phi^t = (\phi_1^t | \phi_2^t | \phi_3^t | .... | \phi_n^t)$ is the matrix with ordered eigenvectors as columns. $\hat{\mathcal{L}}^t$ is a positive semidefinite matrix.

Heat Kernel: Heat kernel gives us the amount of information flowing from one node to another and how information propagates across the network. Heat kernel matrix is the fundamental solution of the heat flow equation that is associated with the normalized Laplacian matrix and it is given as

$$\frac{\partial H^t}{\partial t} = -\hat{\mathcal{L}}^t H^t \tag{10}$$

where H^t and $\hat{\mathcal{L}}^t$ are the heat kernel and normalized Laplacian matrices at a time t, respectively. The solution to the equation is found by solving the first-order differential equation and it is given by:

$$H^t = e^{-\hat{\mathcal{L}^t} t} \tag{11}$$

We can also find the Heat Kernel by exponentiating the Laplacian eigenspectrum,

$$H^t = \Phi^t \exp(-\Lambda^t t) {\Phi^t}^T \tag{12}$$

where the matrix Φ^t is previously defined and is the Jordan form of $\mathcal{L}^t$. More specifically, the information flow across two nodes (Xiao, Wilson, and Hancock 2005), v_i and v_j at a time t can be written as

$$H_{v_i,v_j}^t = \sum_{l=1}^{n} \exp^{-\lambda_i^t t}\, \phi_l^t(v_i) \phi_l^t(v_j) \tag{13}$$

where $\phi_l^t(v_i)$ is the eigenvector of ith node.

Information Aggregation: To obtain the information from the neighboring agents for a specific target agent, we project the respective heat flow element onto the hidden state feature of its neighbors as follows

$$f_i(e_j^t) = H_{v_i,v_j}^t \left(e_j^t W_q\right) \tag{14}$$

W_q is the trainable weights and H_{v_i,v_j}^t is defined in (13). Notice that $f_i(e_j^t)$ represents the information flow from an agent j to the target agent i at a time t. We then concatenate the information from all neighboring agents to the target agent and pass it through the dense layer to obtain the updated feature encoding of the target agent. This can be done as follows:

$$h_{ai}^t = \sigma\left(W_p \cdot \sum_{j \in \mathcal{N}_i} f_i(e_j^t) + b_e\right) \tag{15}$$

where W_p and b_e are trainable weights and bias and $f_i(h_j^t)$ is defined in (14). e_j^t is the feature embedding from previous layer of agent j at a time t.

Through performing different linear projections in parallel, we obtain different variants of information. We then concatenate all the different output encodings and then pass them through the dense layer to obtain the final feature encoding of the target agent. This can be written as follows:

$$h_i = \sigma(g_i^1 \oplus g_i^2 \oplus \dots g_i^m), \tag{16}$$

where

$$g_i^m = \sum_{j \in \mathcal{N}_i} f_i^m(h_j), \tag{17}$$

$f_i^m(e_j) = H_{v_i,v_j}^m \left(e_j W_q^m\right)$ and W_q^m are network weight parameters for m heads, m is the number of communication channels, $\mathcal{N}_i$ is the neighborhood set of an agent i and the symbol $\oplus$ denotes the concatenation operator. Notice that t is omitted to make the presentation simple. This procedure is illustrated in Figure 2b.

Final Layer: Final layer, as shown in Figure 1, is fit with the value function (18). It is fed with the hidden state e_i and aggregated state information h_i as the input. This process is represented as:

$$V_i(o_i) = G_i(e_i, h_i), \tag{18}$$

where G_i is a linear function. We then estimate the state value function for each critic, outputs the state value for each agent. Introducing the heat kernel helps us capture the dynamic nature of the graph and establish the cooperative stability between the agents over the long term.

Critic Training In setting up learning the critic networks, we use a training critic network V_{θ_i} and a target critic network V_{φ_i}. Initially, the target network has the same weights as the training network. The target network acts as the desired stationary value. It is used to inhibit the running state values and its weights are updated frequently for a given number of time steps using the soft update (Mnih et al. 2013).

All the critics are updated together using the joint loss function given as:

$$J^{critic}(\theta) = \sum_{i=1}^{\mathcal{N}} \mathbb{E}_{<o_t, r, o_{t+1}> \sim \mathcal{D}} \left[(y_i^{td} - V_{\theta_i}(o_t))^2 \right], \tag{19}$$

where y_i^{td} is the target temporal difference.

We use the joint loss function across all the agents so that we accommodate the common gradients for the shared parameters in the heat layer. The target temporal difference y_i^{td} is given as:

$$y_i^{td} = A_i^{GAE} + V_{\varphi_i}(o_{t+1}), \tag{20}$$

where A^{GAE} is the Generalized Advantage Estimate (Schulman et al. 2015b) and it is given as:

$$A_t^{GAE} = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{21}$$

where $\delta_t = r_t + \gamma V_{\varphi_i}(o_{t+1}) - V_{\varphi_i}(o_t)$, T is the number of time steps and $\gamma \in [0, 1]$ is the discount factor.

Algorithm 1 CooLMAN

Initialize training state value networks $(V_{\theta_1}, V_{\theta_2} \dots, V_{\theta_N})$ and target state value $(V_{\varphi_1}, V_{\varphi_2} \dots, V_{\varphi_N})$ networks for N agents
Initialize actor networks $(\pi_{\phi_1}, \pi_{\phi_2}, \dots, \pi_{\phi_N})$ for N agents
while episode $<$ max episodes **do**
  reset environment, steps (t) = 0
  while steps $<$ max steps **do**
    Get observations $(o_1^t, o_2^t \dots o_N^t)$
    Sample actions $(a_1^t, \dots a_N^t)$ from observations using **Prediction**
    Get reward $(r_1^t, \dots r_N^t)$ and transits to next state
    Next observations $(o_1^{t+1}, \dots o_N^{t+1})$ correlates with state
    store tuple $< o_i^t, a_i^t, r_i^t, o_i^{t+1} >$ for N agents
    steps += 1
    Recall the stored batch of experiences $< o^t, a^t, r^t, o^{t+1} >$
    Estimate Generalizes Advantage Estimate
    Compute joint critic loss using Equation 19 and update critics
    for 1 to P epochs **do**
      Sample mini-batch of experiences
      Update actor networks using the loss Equation 22
    end for
    Update the target critic networks using the soft update $\theta^{\Gamma} = \beta\theta + (1-\beta)\theta^{\Gamma}$
  end while
end while

Actor Each actor network π_{Φ_i}takes the local observation o_i of an agent as the input. We allow the actors to learn in a conventional PPO way using the loss function (4) in an independent manner. Accordingly, using the advantage function given in equation (21), we define the loss for each actor π_{Φ_i} as:

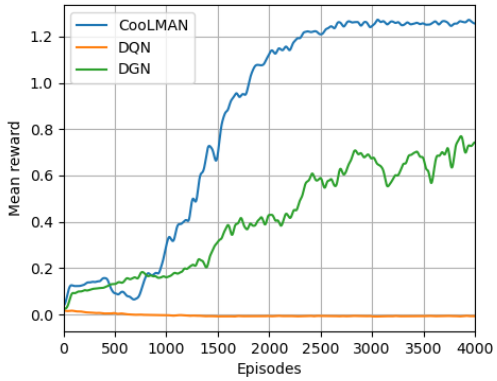
$$J_i^{CLIP}(\Phi_i) = \mathbb{E}\left[min(r(\Phi_i)A_i^{GAE}, clip(r(\Phi_i), 1-\epsilon, 1+\epsilon)) \right]. \tag{22}$$

Parameter sharing After finding the heat kernel matrix H for each agent i through (12), we limit the communication as described below:

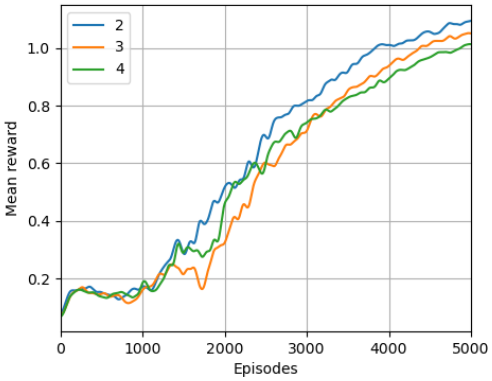
$$H_{v_i,v_j} = \begin{cases} H_{v_i,v_j} & \text{if } H_{v_i,v_j} > p \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

where $p = \text{argsort}([H_{v_i,v_j} | j \in \mathcal{N}])[K]$ and, K is the number of neighbors of each agent. In essence, through (23), we limit the communication neighborhood of each agent to K agents. This neighborhood size is selected through trial and error in the training phase.

To summarize, the CooLMAN algorithm is given in Algorithm 1 and 2.



(a) Mean reward for different models



(b) Mean reward for different number of neighbors

Figure 3: (a): This figure illustrates the plot of mean reward over 4000 episodes for three models. CooLMAN, DGN and DQN are fighting against 'enemy army', which are pre-trained using DQN model. It is the reason why DQN in the plot performs worse as it is competing against itself. (b) Performance comparison of CooLMAN when varying number of neighbors around 8 agents during training.

Experiments

To verify the results of our work, we consider a battle scenario. In this environment, there are two competing multi-agent systems, each with a given number of agents, that need to compete and eliminate each other. Agents in each group should cooperate by forming mini-groups within the whole MAS system as a strategy. Both groups have N agents and each agent is given the same set of abilities. We define the environment as a $n \times n$ grid and the view range and attacking range of each agent by six and two nearby units, respectively. Each agent corresponds to one unit in the grid. No agent is superior to other agents. An agent can only be removed if it is hit six times, and the agents need to coordinate with each other and hit the opposing agents persistently. During training, as soon as an agent is eliminated, it is replaced with another new agent with the same memory as the replaced agent. During the evaluation, we consider a scenario in which agents in the 'adversarial group' are pre-trained using a DQN model (Tampuu et al. 2017), which provides the skills to beat an amateur group of agents. On the other side, agents that are trained by our CooLMAN algorithm need to cooperate with others and have to develop strategies to win against the opposing group. More information regarding the reward and observation of each agent is given in Table 2. To demonstrate our strategy, we use a cooperative battle environment from *MAgent* platform[1](Zheng et al. 2018). In the environment Battle, we chose to evaluate CooLMAN against graph convolutional reinforcement learning (DGN) (Jiang et al. 2019) and DQN (Tampuu et al. 2017). This is because of different reasons. First, the DGN algorithm has similar features to CooLMAN since it uses the attention mechanism to achieve the communication between agents with temporal regularization to achieve long-term cooperative policy. However, it does not adequately address the dynamic nature of the MAS network. The DQN, on the other hand, is a basic algorithm used to train the adversarial group and is based on a decentralized architecture with no interaction between agents. We trained all the three models with hyperparameter settings as given in Table 3 over 4000 episodes and unfolding each episode over 500 time-steps to calculate the mean reward[2] for 3 different models. Figure 3a shows the plot of the mean reward of the episodes. We strict our measure of success in this scenario by maximizing the mean reward attained by each algorithm. As it can be seen from the figure, the best performance is achieved by our CooLMAN algorithm, converging to higher reward than DGN and DQN. It can be noticed that DGN outperforms both CooLMAN and DQN during the first few episodes but then fails to take off with a higher reward as it is unable to maintain the same cooperation between the agents. It can be deduced that DQN converges to a policy that results in lower reward as there is no such communication between agents, since agents are working individually to achieve their local policies. We further evaluated the trained models through three experiments; (i) we evaluated the models on a test round by running each model for 50 test episodes with each episode over 500-time steps using the same setting of the environment as given in Table 2 and the results can be seen in Table 4. As can be noticed, similar to the training phase, CooLMAN performs better than DGN and DQN during the testing with higher mean reward. (ii) We then test our model for scalability to see how well CooLMAN performs if we increase the number of agents. We trained CooLMAN on two different settings; one using 8 agents and another using 14 agents. After the training is done on two different scenarios, we deployed the updated weights to larger swarms of up to 100 agents. As can be seen from Table 1, the mean reward does not drop much as we increase

[1]Environment can be downloaded at: https://github.com/geek-ai/MAgent or can be installed using pip.

[2]Mean reward = $\frac{\text{Total reward}}{\text{Number of agents * Number of time steps}}$

Table 1: Mean reward for the different number of agents during testing. $\mathcal{N}$ is the size of the neighborhood set of the agents. Map size is the metric given for $n \times n$ grid. Mean reward for agents 50 and 100 during testing using the weights trained on 8 agents are omitted as the number of agents does not fit in the given map size.

			During testing Number of agents(Mean reward)				
Trained on	Map Size	$\mathcal{N}$	8	14	20	50	100
8 Agents($\mathcal{N}=2$)	12	2	1.21	1.03	1.10	-	-
		3	1.08	1.06	1.12	-	-
		4	1.07	1.08	1.14	-	-
14 Agents($\mathcal{N}=4$)	21	2	-	0.95	0.90	0.83	0.74
		3	-	0.93	0.87	0.82	0.73
		4	-	0.95	0.84	0.80	0.74

Table 2: Setting of Battle Environment

Name	Value
State Space	(13, 13, 7)
Action Space	Discrete(21)
Reward	Step reward: 0.0 Kill reward: 5 dead penalty: -0.01 attack penalty:-0.02
Agents	8
Enemies	8
Neighborhood Size	2

Table 3: Hyperpaarameters for CooLMAN

Hyperparameter	Value
β	0.01
discount factor(γ)	0.96
epsilon(ϵ)	0.6
Replay Buffer	Batch Size: 128
Optimizer	Adam(lr= $0.9e$-5)
Communication channels	3
CNN Layers	3
Encoder units	(512, 128)

the number of agents. (iii) We further tested our model on various neighborhood sets. During training, we varied the number of neighboring agents and plotted the mean reward against the number of episodes as can be seen in Figure 3b. It is observed that the mean reward is higher for the model having the neighboring set of 2 agents rather than 3 and 4. It is worth noting that, in (Jiang et al. 2019), DGN outperformed the baseline models (Das et al. 2019), (Sukhbaatar, Szlam, and Fergus 2016), (Yang et al. 2018), (Jiang and Lu 2018). This makes the results reported in this work significant as we are showing empirically that CooLMAN performs better than DGN.

Table 4: Statistics for different models. Hit is the number of agents that the model removed its opposite agents and Expired is the number of that respective agents got removed

Stats	CooLMAN	DGN	DQN
Hit	412	289	4
Expired	2	3	381
mean reward	1.14	0.76	-0.015

Conclusion

By modelling time-varying edges as the function of heat diffusion, we enable dynamic communication channel between agents. Thus our model fits the dynamic nature of the complex time-varying multi-agent systems. Besides, linear projections in parallel helped in obtaining the information flow across agents in different variations. On the whole, our model converged to a consistent policy and achieved long-term stability. Empirically, we showed that our model(CooLMAN) out-performed the DGN model and DQN in a dynamic environment scenario. As a future work, it is interesting to see how CooLMAN would preform in more challenging environments, for example, where heterogeneous agents face obstructions in their path.

References

Cortés, J., and Egerstedt, M. 2017. Coordinated control of multi-robot systems: A survey. *SICE Journal of Control, Measurement, and System Integration* 10(6):495–503.

Das, A.; Gervet, T.; Romoff, J.; Batra, D.; Parikh, D.; Rabbat, M.; and Pineau, J. 2019. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, 1538–1546.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2017. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*.

Jaderberg, M.; Czarnecki, W. M.; Dunning, I.; Marris, L.; Lever, G.; Castaneda, A. G.; Beattie, C.; Rabinowitz, N. C.; Morcos, A. S.; Ruderman, A.; et al. 2019. Human-level performance in 3d multi-player games with population-based reinforcement learning. *Science* 364(6443):859–865.

Jiang, J., and Lu, Z. 2018. Learning attentional communication for

multi-agent cooperation. *Advances in Neural Information Processing Systems* 31:7254–7264.

Jiang, J.; Dun, C.; Huang, T.; and Lu, Z. 2019. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*.

Lowe, R.; Wu, Y. I.; Tamar, A.; Harb, J.; Abbeel, O. P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, 6379–6390.

Matignon, L.; Laurent, G. J.; and Le Fort-Piat, N. 2007. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 64–69. IEEE.

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

Mocanu, E.; Nguyen, P. H.; Kling, W. L.; and Gibescu, M. 2016. Unsupervised energy prediction in a smart grid context using reinforcement cross-building transfer learning. *Energy and Buildings* 116:646–655.

Omidshafiei, S.; Pazis, J.; Amato, C.; How, J. P.; and Vian, J. 2017. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*, 2681–2690.

Petersen, K. H.; Nagpal, R.; and Werfel, J. K. 2011. Termes: An autonomous robotic system for three-dimensional collective construction. *Robotics: science and systems VII*.

Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304.

Sartoretti, G.; Wu, Y.; Paivine, W.; Kumar, T. S.; Koenig, S.; and Choset, H. 2019. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Distributed Autonomous Robotic Systems*. Springer. 35–49.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015a. Trust region policy optimization. In *International conference on machine learning*, 1889–1897. PMLR.

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

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

Sukhbaatar, S.; Szlam, A.; and Fergus, R. 2016. Learning multiagent communication with backpropagation. In *NIPS*.

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

Tampuu, A.; Matiisen, T.; Kodelja, D.; Kuzovkin, I.; Korjus, K.; Aru, J.; Aru, J.; and Vicente, R. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PloS one* 12(4):e0172395.

Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, 330–337.

Unsworth, J., and Duarte, F. 1979. Heat diffusion in a solid sphere and fourier theory: an elementary practical example. *American Journal of Physics* 47(11):981–983.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Wei, H.; Xu, N.; Zhang, H.; Zheng, G.; Zang, X.; Chen, C.; Zhang, W.; Zhu, Y.; Xu, K.; and Li, Z. 2019. Colight: Learning network-level cooperation for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 1913–1922.

Xiao, B.; Wilson, R. C.; and Hancock, E. R. 2005. Characterising graphs using the heat kernel. In *Proc. BMVC*.

Yang, Y.; Luo, R.; Li, M.; Zhou, M.; Zhang, W.; and Wang, J. 2018. Mean field multi-agent reinforcement learning. In *35th International Conference on Machine Learning, ICML 2018*, volume 80, 5571–5580. PMLR.

Zheng, L.; Yang, J.; Cai, H.; Zhang, W.; Wang, J.; and Yu, Y. 2018. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 8222–8223. AAAI.