Cooperation Learning in Time-Varying Multi-Agent Networks

December 31, 2022

Introduction, Main Challenges

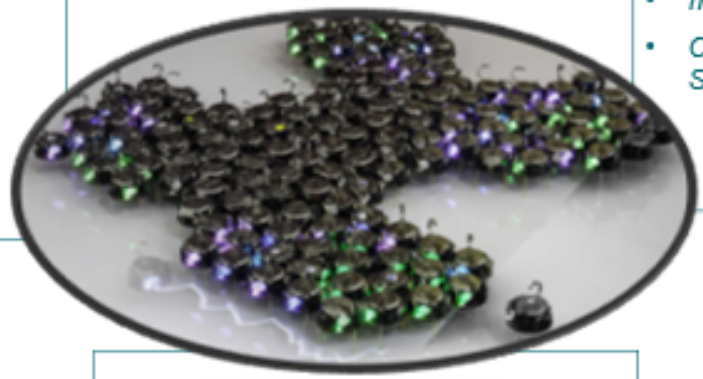
Introduction

Extension to that of the Single Agent Learning

- *Complex tasks*
- *Optimal time*

Multi Agent Systems

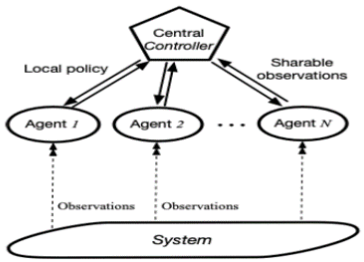
- *Independent Systems*
- *Cooperative, Competitive and Mixed System*
  - *Communicating*
  - *Non-Communicating*

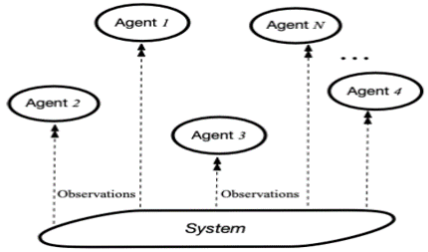


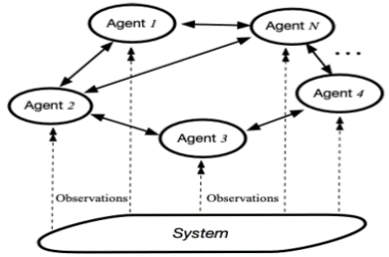
- *Non-stationarity*
- *Scalability*
- *Coordination– Communication*
- *Multiagent Credit Assignment*

Challenges

Framework setting







- 1. Centralized setting: Central Controller aggregates the information – Central Critic
  - Joint states/Global states. Joint Actions. Decentralized actors during execution.
- 2. Decentralized setting: No central controller
  - Independent agents. No explicit communications
- 3. Networked: Decentralized setting with Networked Agents
  - Agents are connected via communication channel. Local states with information from its neighboring agents

Background

- Consider a fully cooperative environment. Defined by the tuple

$$M_G = \langle S, A, P, R, O, N, \gamma \rangle$$

  - Global state : $s \in S$
  - Action Space: $(A_1, A2, ..., A_N) \in A$
  - Transition probability: $P : S \times A_1 \times A_2 \times \cdots \times A_N \to [0, 1]$
  - Reward function: $R_i : S \times A_1 \times A_2 \times \cdots \times A_N \to \mathcal{R}$
  - Discount factor : γ
- Specifically considering a partial observation environment setting
  - Partial observation of the agent – partial state of the global state O_i
  - Each agent learns their individual policy: $\pi_i : O_i \times A_1 \to [0, 1]$

Q-learning

- Given the local state of an agent s_i, we get the output values for each action.

$$\hat{q}(s_i^t) = \hat{h}_i^t W_p + b_p$$

- Agent then selects the action which has the highest value.

$$a_i^t = argmax(\hat{q}(s_i^t))$$

- For each action it takes, it receives a reward r_i^t, which acts as a feedback which would help the agent to steer in a right direction.
- This process iterates over the certain time steps and then all the parameters gets updated.
- For each iteration/time-step, we store the raw local state (s_i^t), action (a_i^t), reward (r_i^t) and the next local observation (s_i^{t+1}) in a tuple.

- During updation step, we find the loss.

$$\mathcal{L}(\theta) = \frac{1}{t}\sum_{t}\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{q}(s_i^t, a_i^t; \theta))$$

where, $y_i = r_i^t + \gamma q(s_i^{t+1}, a_i^{t+1}; \theta')$
- γ is the discount rate. y_i is the target network. θ and θ' are the parameters/weights of training network and target network respectively.
- We then update the target network parameters using gradient descent after we find the gradients.

$$\theta'_n = \theta'_o - \beta\Delta\mathcal{L}(\theta)$$

Jiang - 2020

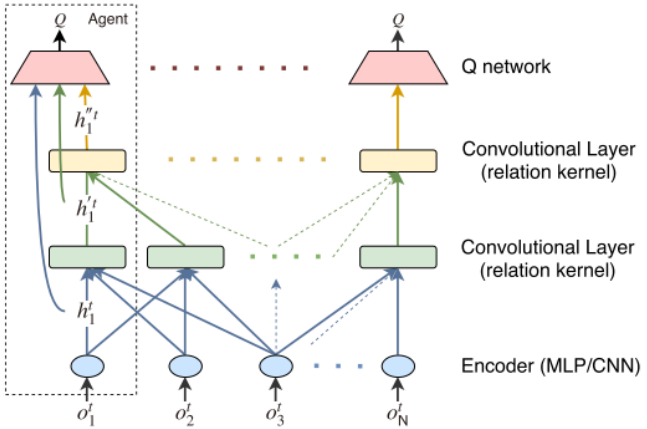


Figure: GCRL

- Uses relational kernel – Attention mechanism to capture the information from other agents
- Each hop / level has more high level information.
- Restricts to it's neighborhood.
- Uses KL divergence for the attention weights as the regularization
- How to represent this network or parameter sharing in a different way that could extract much more important information in time-varying network?

Graphs

- Given a set of agents $A = \{A_1, A_2, ...A_n\}$, each agent is considered as a node or a vertices of graph $\mathcal{G} = (\mathrm{V}, \mathrm{E})$.
- Graph is weighted and undirected, where all the edges are bidirectional.
- The edge between the nodes (v_i, v_j), $e_{ij} \in \mathrm{E} \subseteq V \times V$ is regarded as the communication between the agents.
- Adjacent Matrix of the graph is given as A.
- Diagonal degree matrix is given by $D(v_i, v_j) = \sum_{v_j \in V} A(v_i, v_j)$

Heat Kernel Layer

- Heat kernel gives the flow of information across the edges from one node to another node with time.
- This feature can be used to exploit the temporal feature of dynamic graph.

Heat flow is given by H_{ij}^{t}, where t is the time instant and the flow is given between agent i and agent j.

- Laplacian matrix is then given by:

$$\mathcal{L}^t = D^t - A^t$$

- Where, D^t is the diagonal degree matrix and A^t is the adjacency matrix at a time t.
- **Heat flow** equation across the graph is given by:

$$\frac{\partial H_t}{\partial t} = -\hat{\mathcal{L}}^t H_t$$

- H_t is the heat kernel and $\hat{\mathcal{L}}^t$ is the normalised laplacian matrix.

Laplacian matrix

- Normalised laplacian matrix is obtained by:

$$\hat{\mathcal{L}}^t = D^{-\frac{1}{2}} \mathcal{L}^t D^{\frac{1}{2}}$$

- It can be represented in its eigen decompositon form using eigen values and eigenvectors of $\mathcal{L}$

$$\hat{\mathcal{L}} = \phi \Lambda \phi^T$$

where, ϕ is the is the matrix with the ordered eigenvectors as columns and Λ is the diagonal matrix with the ordered eigenvalues as elements.

$$\Lambda = \text{diag}\left(\lambda_1, \lambda_2 \cdots \lambda_v\right)$$

$$\phi = \left(\phi_1 \left|\phi_2\right| \ldots \phi_v\right)$$

Information flow

- Eigen decomposition laplacian matirx at a time t :

$$\hat{\mathcal{L}}^t = \phi^t \Lambda^t {\phi^t}^T$$

- Solving heat flow equation gives:

$$\frac{\partial H_t}{\partial t} = -\hat{\mathcal{L}}^t H_t$$

$$H_t = \phi^t \exp[-t\Lambda^t] {\phi^t}^T$$

- For any two nodes $(v_i, v_j) \in V$, the amount of heat flow $H_t(v_i, v_j)$ is given by:

$$h_t(v_i, v_j) = \sum_{i=1}^{V} \exp^{-\lambda_i^t t} \phi_i^t(v_i)\phi_i^t(v_j)$$

Adjacency matrix

- Let's get back to the laplacian matrix:

$$\mathcal{L}^t = D^t - A^t$$

- Weights of adjacent matrix are unknown here.
- To learn how much the information is flowing between nodes, we project one node observation embedding onto another agent node.
- Given graph has n nodes, let's see the edge link set up for agent i.
- The edge weight from agent j to agent i is given by:

$$e_{ij} = (h_i w_i) \dot{(} h_j w_j)^T$$

- Where, w_i and w_j are weight of the networks. Similarly, we will find edge weight for all the nodes that associated with agent i.

- we normalise all the edge weights to retrieve general edge values and another reason is to sum up the all the row values to 1.

$$\alpha_{ij} = \frac{exp(e_{ij})s}{\sum_{j \in \mathcal{N}} exp(ij)}$$

- Where, N is the set of neighboring agents of agent i. Thus, we found the first row of adjacency matrix. Similarly, we find the edge weights for all the agents.

Temporal neighborhood information

- To model the overall influence of flow of information from the neighboring nodes to the agent, individual information flow of all agents summed up together.

$$\hat{h}_i^t = \sigma(W_s \sum_{\mathcal{N}} H_{ij}^t \dot{(} h_j W_c) + b_d)$$

Where, H_{ij}^t is the flow of information across node i and j at a time t.

- $\hat{h}_i^t$ accumulates the important information from it's neighboring agents at a time t.

Architecture Schema

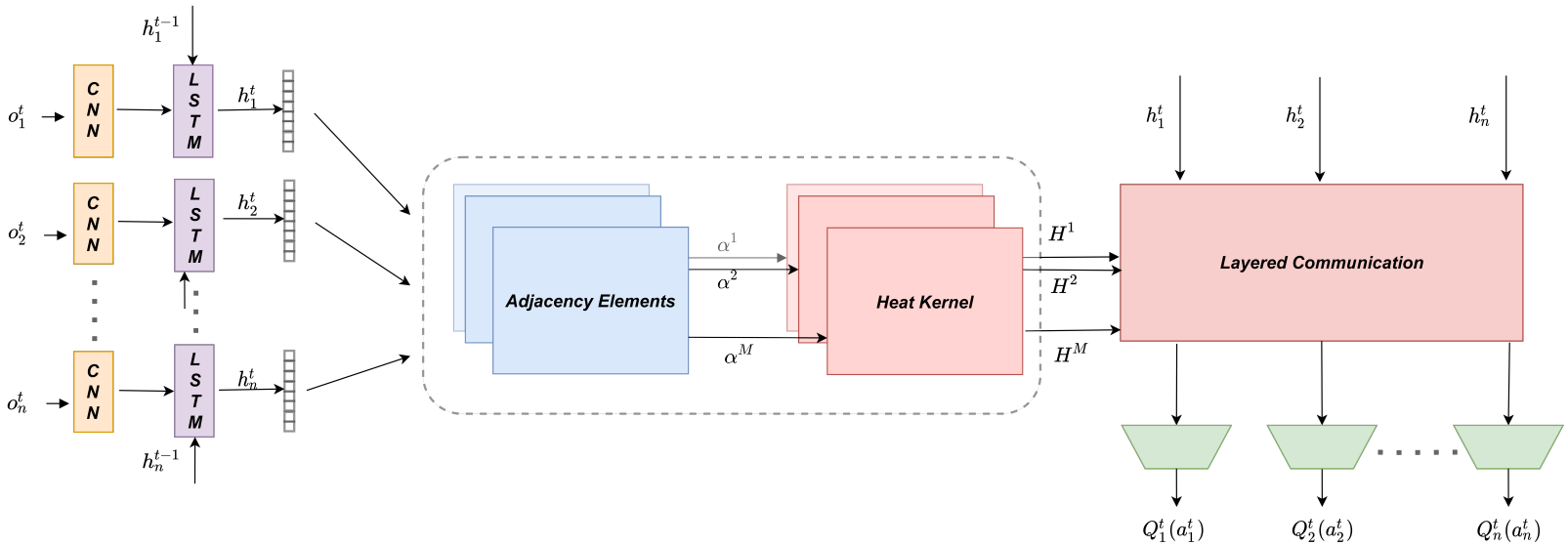


Figure: illustrates is the main architecture of the proposed model, which perceives the local observations of all the agents and gives out the best actions they need to make. It has two important components.

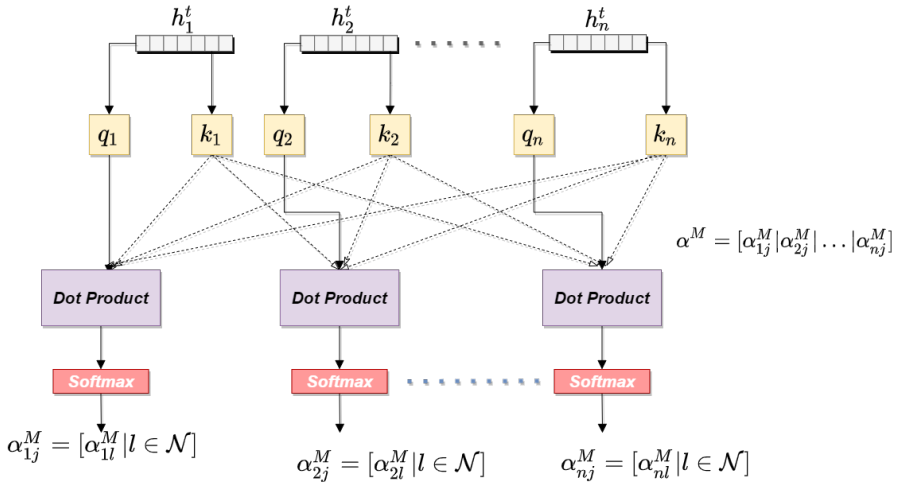


Figure: illustrates the dot-product mechanism outputs the adjacency matrix elements and the same would be followed for M communication channels

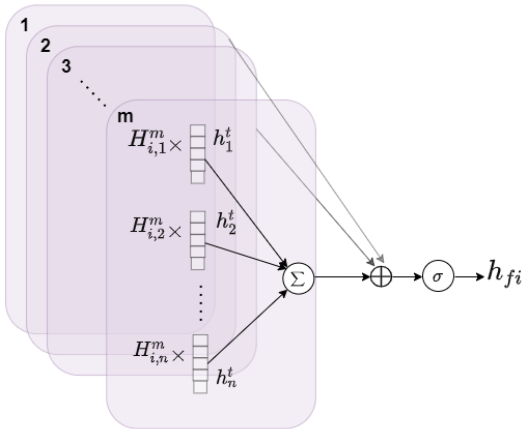


Figure: illustrates the step Information aggregation which emulates the information from the agents neighborhood and concatenating the m different