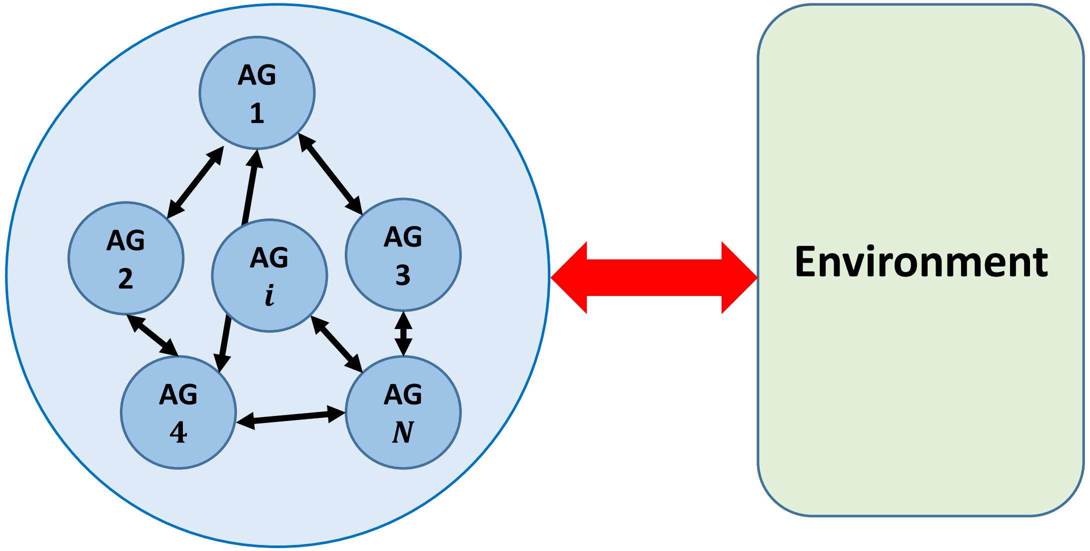
Lecture 11. Deep Multi-Agent Reinforcement Learning Centralized Training and Decentralized Execution Approach

Motivation

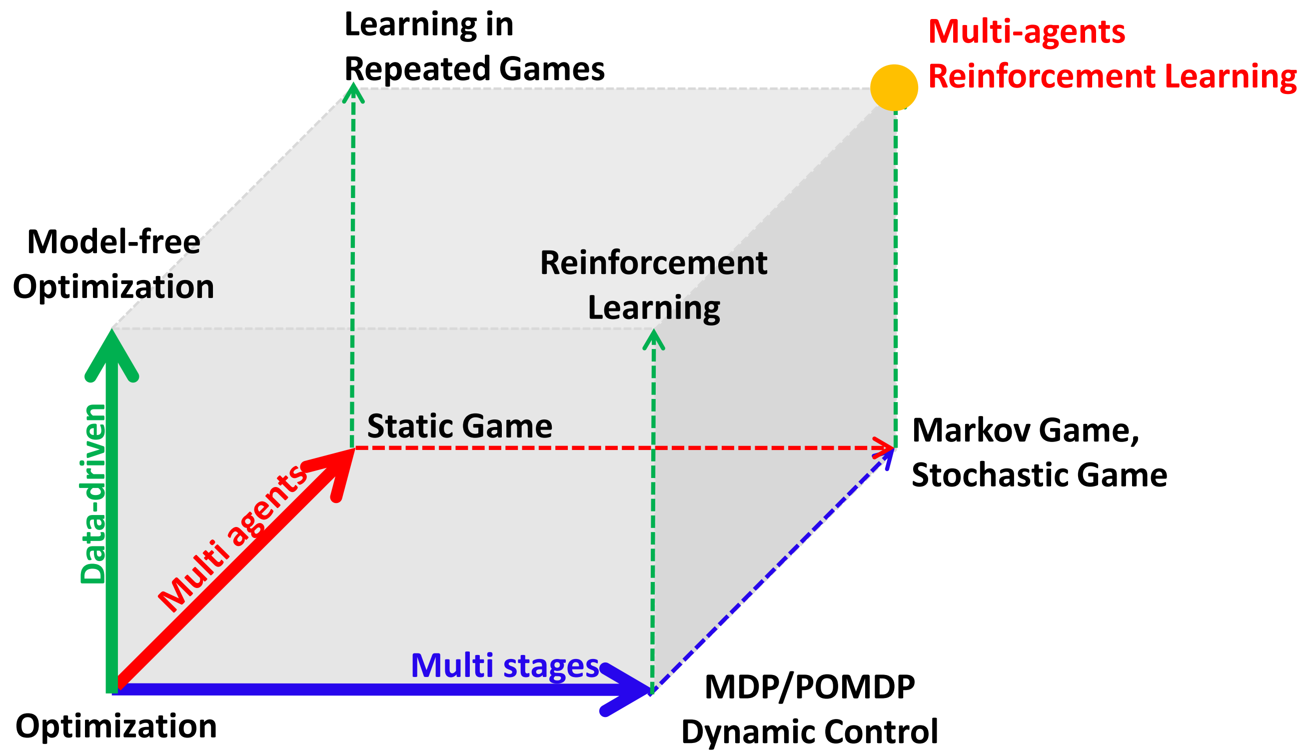


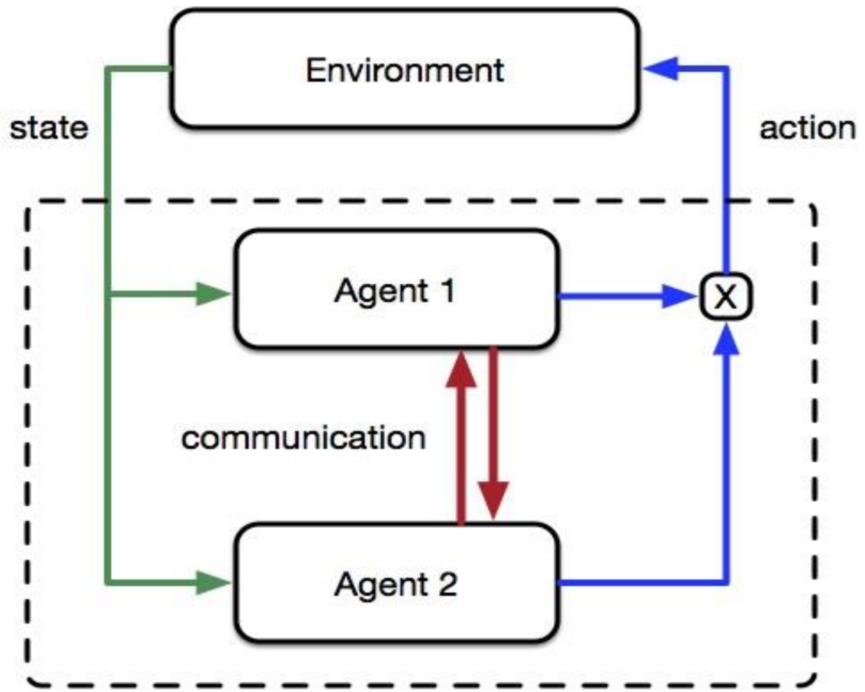
Joint control policy approximated as a decentralized control policy:

$$\pi(s, a_1, \dots, a_i, \dots, a_n) \approx \prod_{i=1}^{N} \pi_i(s, a_i) \approx \prod_{i=1}^{N} \pi_i(o_i, a_i)$$

- As a system becomes larger and more complex, it become more difficult to understand and control the target systems
- A methodology has been developed to independently model the agents that make up the entire system, to efficiently model the entire system considering their interaction, and to derive distributed control strategies
  - Decentralized MDP, Decentralized-POMDP, Decentralized Cooperative Control, Team Game, Cooperative Game..
  - Analytical solutions to these problems are limited to only special cases
- Recent advanced in Deep learning and Reinforcement learning approach can open up new solution approaches

Motivation





- Multi Agent Reinforcement Learning (MARL) aims to derive a decentralized policy while considering the interactions among agents (cooperative, competitive, Nash, etc.)
- We mainly aim to derive ***a decentralized decision making policy for each agent*** (e.g., intersection, zones, OHT) that can lead ***a global performance of the whole system*** (AMHS)

Motivation

Equilibrium concept: **Team; Cooperative**; Nash; Zero-sum; Stackelberg; Correlated

	Single Agent	Multi Agent
Static	Static Optimization	Static Game
Dynamic	Dynamic Optimization	**Dynamic Game**

Action space

Time space

Model free	Finite	Infinite
Discrete	Multi-Agent Value-based RL	**Multi-Agent Policy-based RL**
Continuous		

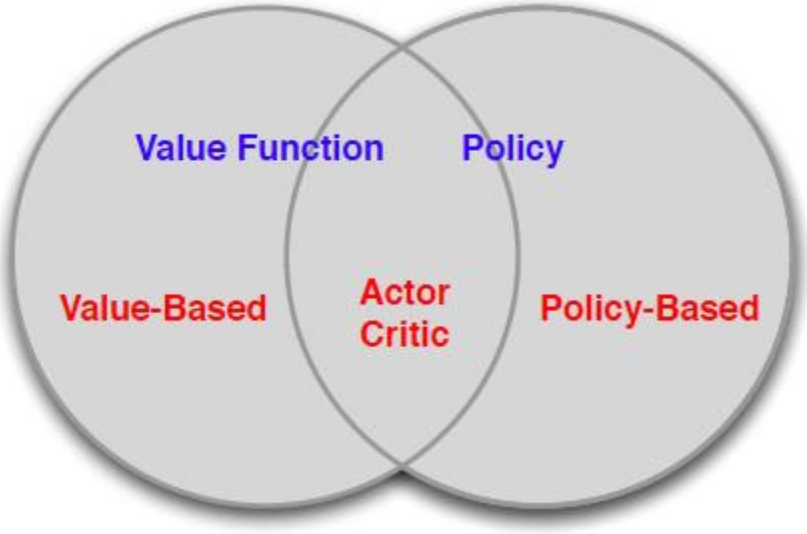
Actor-Critic algorithm is used	**Team Game**	**Cooperative Game**	**Non-Cooperative Game**	**Non-Cooperative Game**	**Non-Cooperative Game**	**Stackelberg Game**
Equilibrium Concept	**Optimum Policy**	**Optimum Policy**	**Nash Equilibrium Policy**	**Correlated Equilibrium Policy**	**Mean-Field Nash Equilibrium Policy**	**Nash-Equilibrium Policy or Mean-Field Nash**
Reward function	Common reward $R^i(s,a) = R(s,a)\ \forall i$	Independent $R^i(s,a), i = 1, \dots, n$	Independent $R^i(s,a), i = 1, \dots, n$	Independent $R^i(s,a), i = 1, \dots, n$	Independent $R^i(s,a), i = 1, \dots, n$	Independent $R^i(s,a), i = 1, \dots, n$
Q function	Common central Q $Q(s,a) = \sum_{t=0}^{T} \gamma^t r_t$	Common central Q $Q(s,a) = \sum_{t=0}^{T} \sum_{i=1}^{N} \gamma^t r_t^i$	individual Q^i $Q^i(s,a) = \sum_{t=0}^{T} \gamma^t r_t^i,$ $i = 1, \dots, n$	individual Q $Q^i(s,a) = \sum_{t=0}^{T} \gamma^t r_t^i,$ $i = 1, \dots, n$	Mean field Q $Q^i(s, a^i, \bar{a}) = \sum_{t=0}^{T} \gamma^t r_t^i,$ $i = 1, \dots, n$	individual Q $Q^i(s,a) = \sum_{t=0}^{T} \gamma^t r_t^i,$ $i = 1, \dots, n$
Agents	Moderate N	Moderate N	Moderate N	Moderate N	Large N	1 to N or 1 to large N
Policy $\pi(s,a) = \prod_{i=1}^{N} \pi^i(s,a^i)$	Decentralized independent policy $\pi^i(o^i, a^i), o^i = h^i(s)$	Decentralized independent policy $\pi^i(s, a^i)$	Decentralized independent policy $\pi^i(o^i, a^i), o^i = h^i(s)$	Decentralized independent policy $\pi^i(s, a^i)$	Decentralized independent policy $\pi^i(s, a^i)$	Decentralized independent policy $\pi^i(s, a^i)$
Consensus mechanism (how the mutual interaction is modeled?)	Each agent learn central $Q(s,a)$ using its own local reward and train $\pi^i(o^i, a^i)$	Each agent Learn central $Q(s,a)$ using its own local reward by sharing local parameters for $Q(s,a)$ and independently train $\pi^i(o^i, a^i)$	Learn other player's policy and use that to estimate $Q^i(s,a) = Q^i(s, \pi^1(s), \dots, \pi^N(s))$	Use collective gradient (coordinated gradient)	Model agent's state distribution and consider agents collective behavior using averaged actions	
Target Application	Decentralized control for large system with single goal: (OHT, wind Farm)	Imposing coordination among agents with its individual goals	Derive the equilibrium policy under incomplete information on reward functions (ESS, EV)	Impose implicit coordination under non-cooperative setting (ESS, EV)	Modelling collective behavior of a large number of agents and find the best response (OHT, wind Farm, EV, ESS)	Optimal hierarchical decision making (Smart grid, Marketing)
Representative Paper	Counterfactual MAPG (Oxford, 2017)	Fully-Decentralized MARL (UIUC, 2018)	MADDPG (OpenAI, 2017)	Correlated-DDPG	Mean Field MARL (UCL, 2018)	Open-loop Stackelberg Learning (VT, 2017)

Recap: Actor Critic Reinforcement Learning

- **Value-based RL** and Policy-based RL
- Approximate value or action-value function using parameter θ

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- Incremental Methods
  - Monte Carlo control, TD(λ), SARSA…
- Batch Methods
  - LSTD, DQN..
- Implicit policy from value function
  - e.g. ϵ-greedy



Recap: Actor Critic Reinforcement Learning

- Value-based RL and **Policy-based RL**
- Use parametrized policy $\pi_\theta(s,a) = \mathbb{P}[a|s;\theta]$
- Policy gradient
  - State $s \sim d^{\pi_\theta}$, one-step reward $r = \mathcal{R}_{s,a}$

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \pi_\theta(s,a)\, \mathcal{R}_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)\, \mathcal{R}_{s,a}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, r]$$

- Equivalent form

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, v_t]$$ Monte-Carlo

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q^w(s,a)]$$ Q function

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, A^w(s,a)]$$ Advantage function

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta]$$ TD(0)

Recap: Actor Critic Reinforcement Learning

- Actor-critic maintain two sets of parameters

  Critic Updates action-value function parameter w $\quad Q^w(s,a) \approx Q^{\pi_\theta}(s,a)$

  Actor Updates policy parameters θ, in direction suggested by critic $\quad \pi_\theta(s,a) = \mathbb{P}[a|s;\theta]$

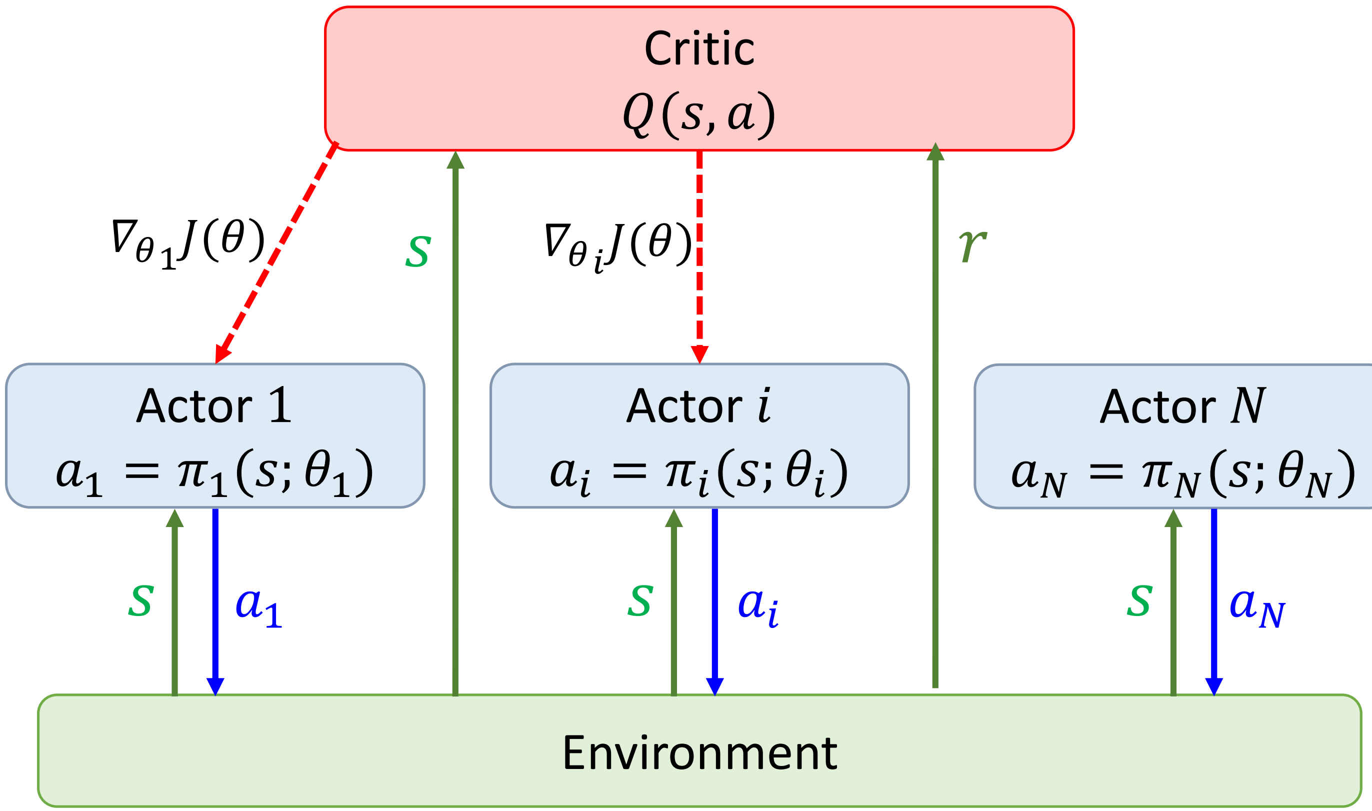
- Bias in actor-critic algorithm
  - To reduce variance, subtract a baseline function $B(s)$ from the policy gradient

$$
\begin{aligned}
\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, B(s)] &= \textstyle\sum_{s\in S} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) B(s) \\
&= \textstyle\sum_{s\in S} d^{\pi_\theta}(s)\, B(s) \nabla_\theta \sum_a \pi_\theta(s,a) \\
&= 0 \qquad (\because \textstyle\sum_a \pi_\theta(s,a) = 1)
\end{aligned}
$$

  - Rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s,a)$

$$
\begin{aligned}
A^w(s,a) &= Q^w(s,a) - V^w(s) \\
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, A^w(s,a)]
\end{aligned}
$$

Solving Team & Cooperative Game using Actor Critic Reinforcement Learning



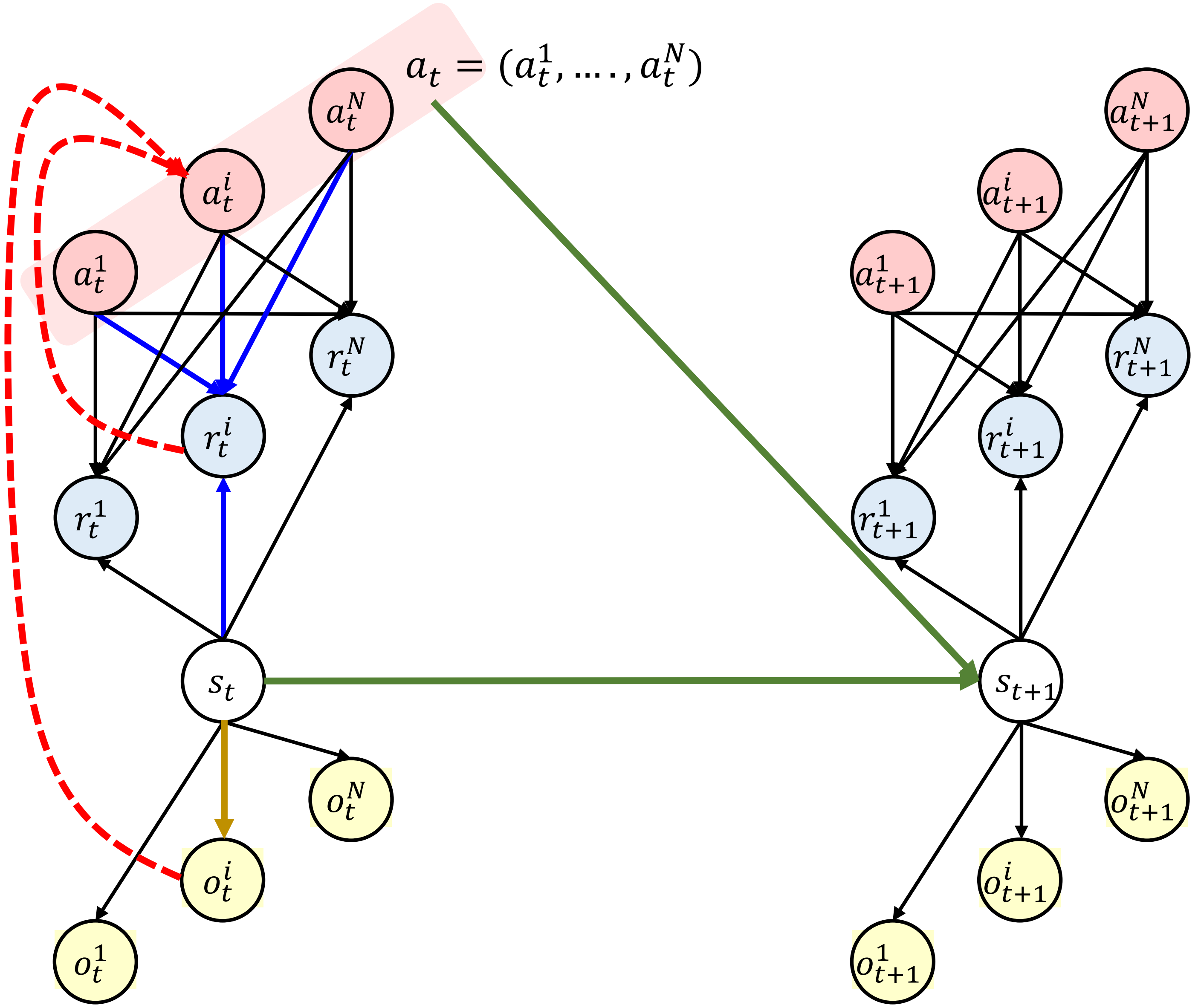
Main Principle:

Centralized Training (using high fidelity simulator)

distributed and decentralized execution

- Centralized Critic $Q(s,a)$: Capture the interactions among agents in terms of achieving a global objective
- Localized Policy $\pi_i(s,a_i)$: Determine the local action in a decentralized manner

Stochastic Game



- ***Transition:***

  $P(s_{t+1} | a_t^1, \dots, a_t^N, s_t)$

- ***Reward:***

  $r_t^i(s_t, a_t^1, \dots, a_t^N)$

- ***Observation:***

  $o_t^i = h^i(s_t)$

- ***Decentralized Policy:***

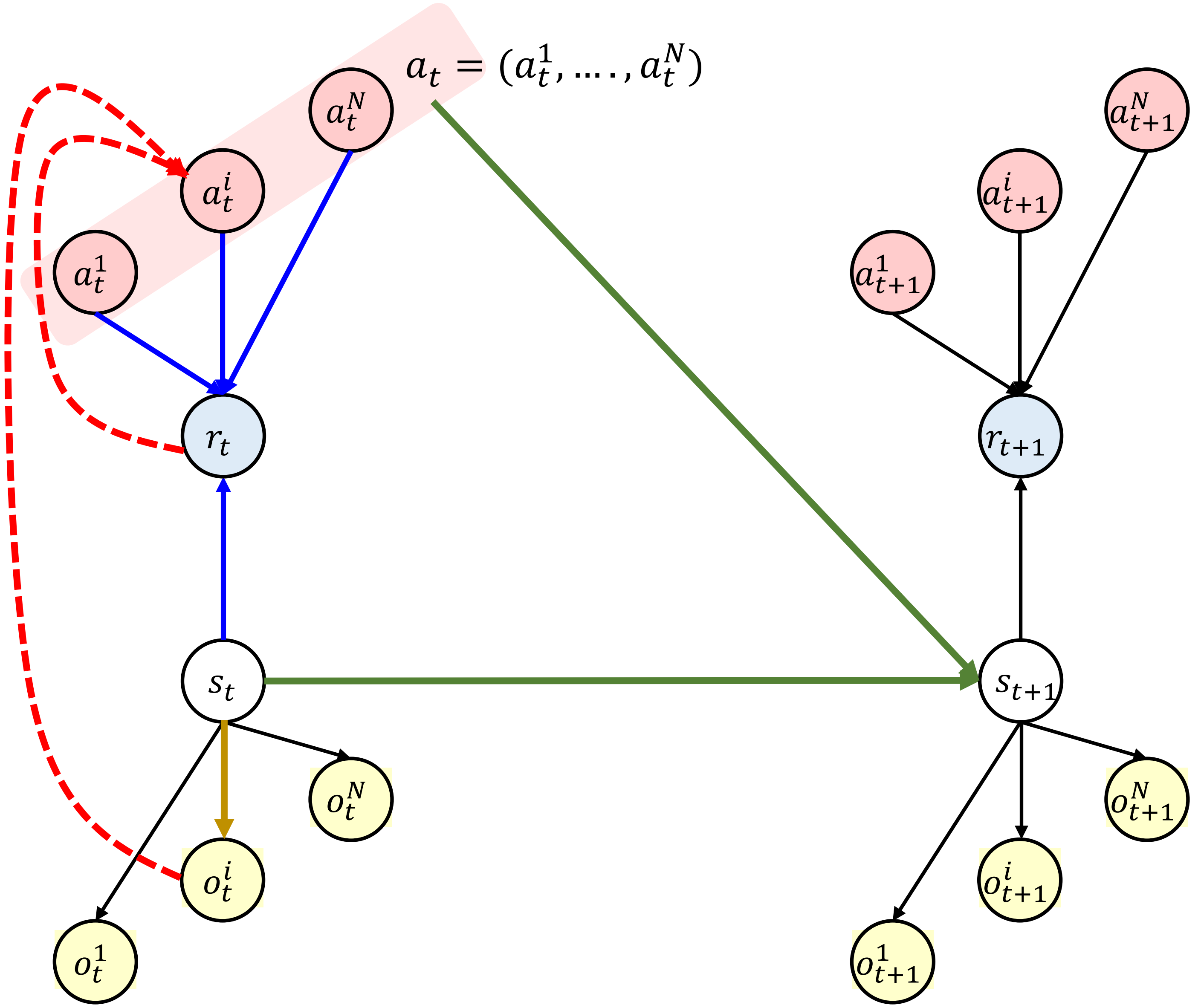
$$\pi(s_t, a_t^1, \dots, a_t^N) \approx \prod_{i=1}^{N} \pi_i(s_t, a_t^i)$$

$$\approx \prod_{i=1}^{N} \pi_i(o_t^i, a_t^i)$$

- ***Objective of agent i:***

$$\max_{\pi_i} E\left[\sum_{t=1}^{T} r_t^i(s_t, a_t^1, \dots, a_t^N)\right]$$

Team Game



- ***Transition:***

  $$P(s_{t+1} | a_t^1, \dots, a_t^N, s_t)$$

- ***Reward:***

  $$r_t(s_t, a_t^1, \dots, a_t^N)$$

- ***Observation:***

  $$o_t^i = h^i(s_t)$$

- ***Decentralized Policy:***

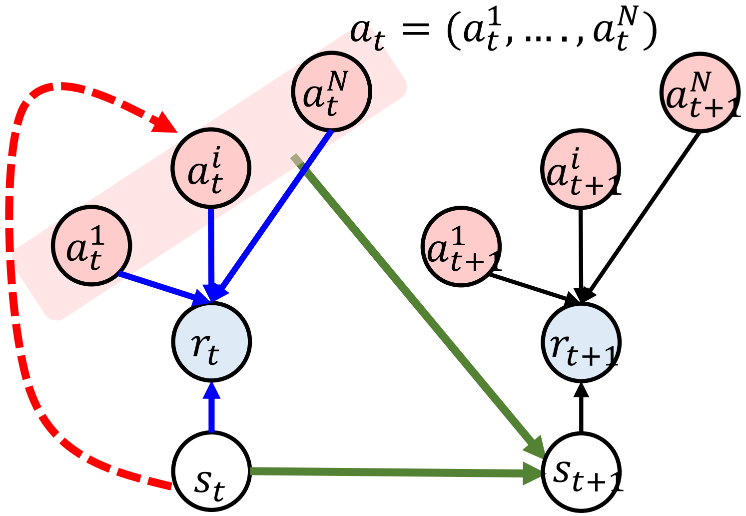
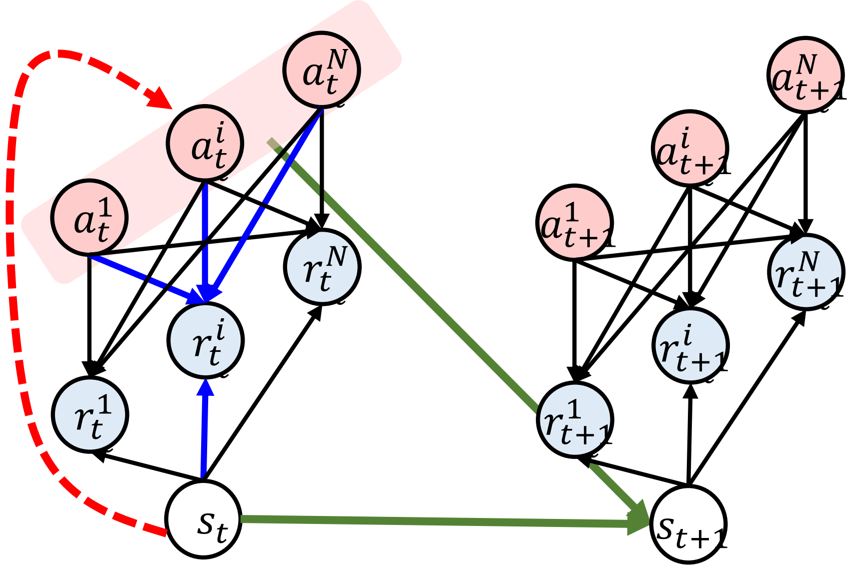
$$\pi(s_t, a_t^1, \dots, a_t^N) \approx \prod_{i=1}^{N} \pi_i(s_t, a_t^i)$$

$$\approx \prod_{i=1}^{N} \pi_i(o_t^i, a_t^i)$$

- ***Objective of agent i:***

$$\max_{\pi_i} E\left[\sum_{t=1}^{T} r_t(s_t, a_t^1, \dots, a_t^N)\right]$$

Team Game & Cooperative Game

	Team Game	Cooperative Game
Reward	Each agent shares common reward, $r(\boldsymbol{a}, s)$ 	Each agents have independent reward, $r^i(\boldsymbol{a}, s)$ 
Objective	$\max_{\theta} \mathbb{E}_{\pi_\theta}[r(s, \boldsymbol{a})]$	$\max_{\theta} \mathbb{E}_{\pi_\theta}\left[\frac{1}{N}\sum_{i \in N} r^i(s, \boldsymbol{a})\right]$
Representative Paper	COunterfactual MAPG (Oxford, 2017)	Fully-Decentralized MARL (UIUC, 2018)

MARL approach to Team Game

		Training	
		Centralized Training	Decentralized Training
Execution	Centralized Execution	MDP	
	Decentralized Execution	Dec-(PO)MDP	?

MARL approach to Team Game

		Training	
		Centralized Training	**Decentralized Training**
Execution	Centralized Execution	MDP	
	Decentralized Execution	Dec-(PO)MDP	?

- **Centralized Training** vs **Decentralized Training**
  - In centralized training, we assume a central learner who can access all the global information $s = (s_1, \ldots, s_N)$ and $a = (a_1, \ldots, a_N)$ for modeling mutual interactions among agents
  - In decentralized training, each agent has a limited information about the global state and the joint action

MARL approach to Team Game

		Training	
		Centralized Training	Decentralized Training
Execution	**Centralized Execution**	MDP	
	Decentralized Execution	Dec-(PO)MDP	?

- Centralized Training vs Decentralized Training
  - In centralized training, we assume a central learner who can access all the global information $s = (s_1, \dots, s_N)$ and $a = (a_1, \dots, a_N)$ for modeling mutual interactions among agents
  - In decentralized training, each agent has a limited information about the global state and the joint action

- **Centralized Exestuation** vs **Decentralized Execution**
  - In centralized exestuation, a joint action $a = (a_1, \dots, a_N)$ is selected by a central agent
  - In centralized exestuation, each agent selects individual action to compose a joint action:

$$\pi(s, a) \approx \prod_{i=1}^{N} \pi_i(\text{Information of agent } i, a_i)$$

MARL approach to Team Game

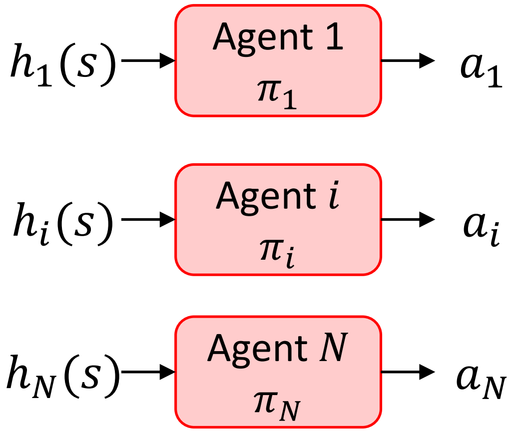
		Training	
		Centralized Training	Decentralized Training
Execution	Centralized Execution	MDP	
	Decentralized Execution	Dec-(PO)MDP (CTDE)	?

$s = (s_1, \dots, s_N)$
$a = (a_1, \dots, a_N)$
$r = f(s, a)$

$Q(s, a)$

$\pi_1, \dots, \pi_i, \dots, \pi_N$

Centralized Training

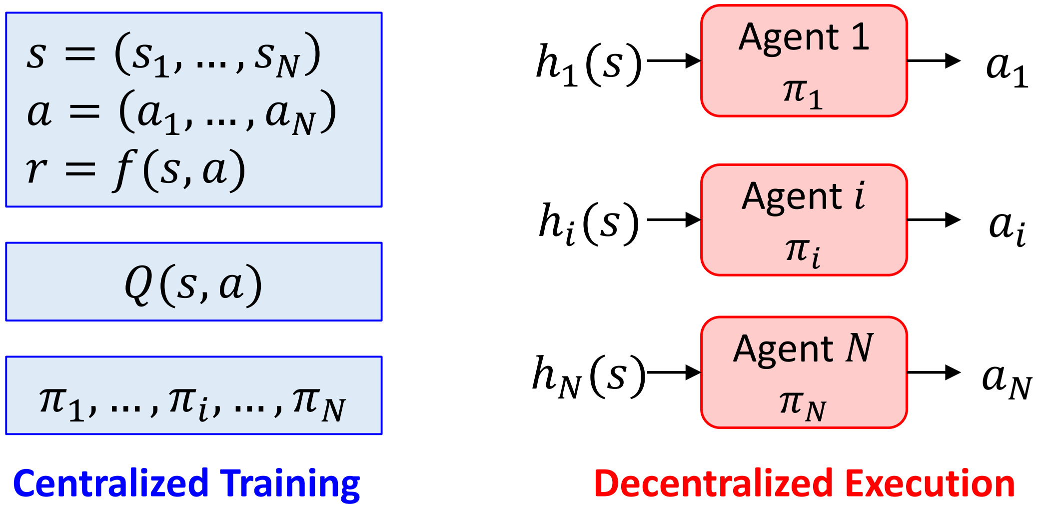


Decentralized Execution

- **Centralized Training** and **Decentralized Execution** (CTDE) is a widely adopted to overcome the non-stationarity problem when training multi-agent systems (Oliehoek et al., 2008; Kraemer & Banerjee, 2016; Jorge et al., 2016; Foerster et al., 2018)
- CTDE enables to leverage the observations of each agent, as well as their actions, to better model the interactions among agents during training.
- Depending on the information structure $h_i(s)$ of each agent has in execution, there are various approaches in CTDE
  - Local observations: each agent can access only to its local (state) observation
  - Global observations: each agent can access to the global state (observation)

CTDE framework for Team Game

		Training
		Centralized Training
Execution	Centralized Execution	MDP
	Decentralized Execution	Dec-(PO)MDP (CTDE)



- **Depending on the information that each agent can use when making a decision (information structure)**, CTDE has different approaches.
  - Local observation: $h_i(s) = s_i (\text{or } o_i)$: (when there is partial observability and/or communication constraints)
  - Global observation: $h_i(s) = s (\text{or } o)$: (global observation or communication is allowed)

Information Structure in CTDE framework

Local observation: $h_i(s) = s_i(\text{or } o_i)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s_i, a_i)$$

Do not know other agent's state and action

$$\approx \prod_{i=1}^{N} \pi_i(s_i, a_i; \theta_i)$$

:Function approximation

Global observation: $h_i(s) = s(\text{or } o)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s, a_i)$$

Do not know other agent's action

$$\approx \prod_{i=1}^{N} \pi_i(s, a_i; \theta_i)$$

:Function approx.:

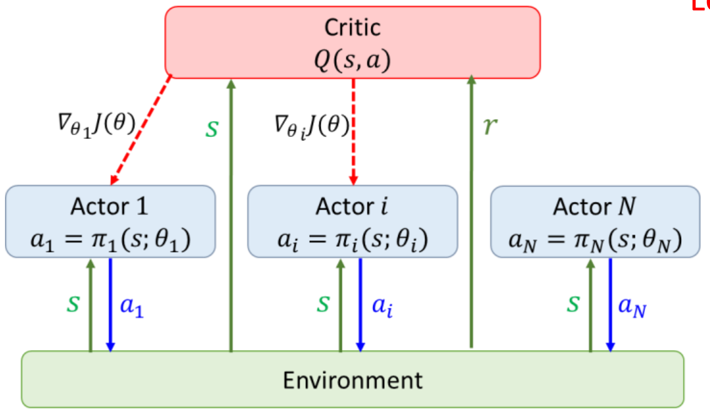
→**difficult** to process s (high dim)

$$\approx \prod_{i=1}^{N} \pi_i(h_i = GNN(s; \phi_i), a_i; \theta_i)$$

(State representation with inductive biases)

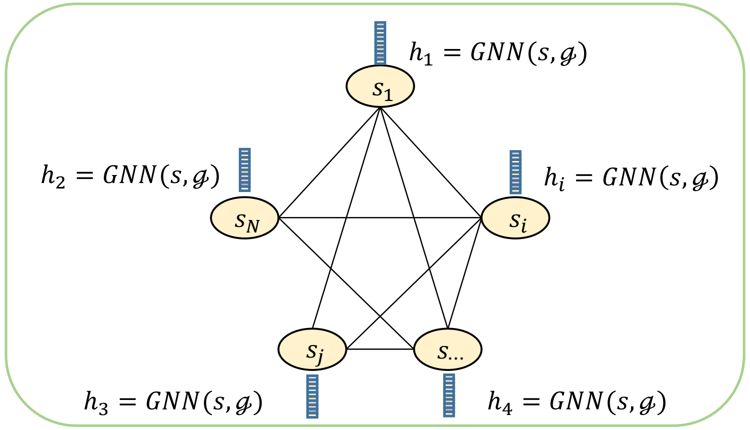
Consensus through central $Q(s, a; \phi)$ modeling

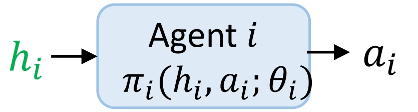
Learning to cooperate



Consensus through Communication & GNN

Learning to communicate





Information Structure in CTDE framework (homogeneous agents)

Local observation: $h_i(s) = s_i (\text{or } o_i)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s_i, a_i)$$

Do not know other agent's state and action

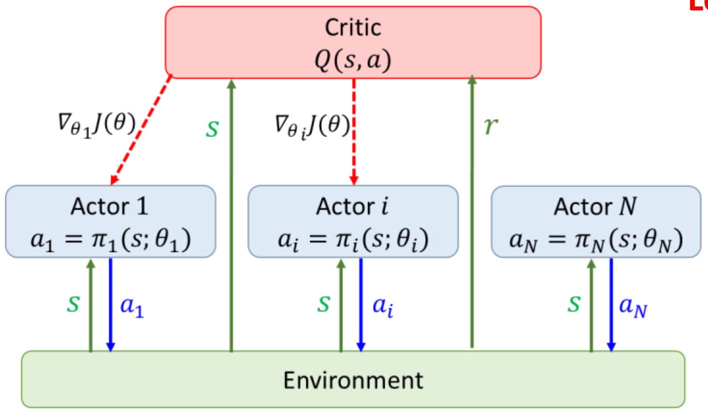
$$\approx \prod_{i=1}^{N} \pi_i(s_i, a_i; \theta_i)$$:Function approximation

$$= \prod_{i=1}^{N} \pi(s_i, a_i; \theta)$$

Parameter sharing among **homogeneous** agents

Consensus through central $Q(s, a; \phi)$ modeling

Learning to cooperate



Global observation: $h_i(s) = s (\text{or } o)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s, a_i)$$

Do not know other agent's action

$$\approx \prod_{i=1}^{N} \pi_i(s, a_i; \theta_i)$$:Function approx.:

→**difficult** to process s (high dim)

$$\approx \prod_{i=1}^{N} \pi_i(h_i = GNN(s; \phi_i), a_i; \theta_i)$$

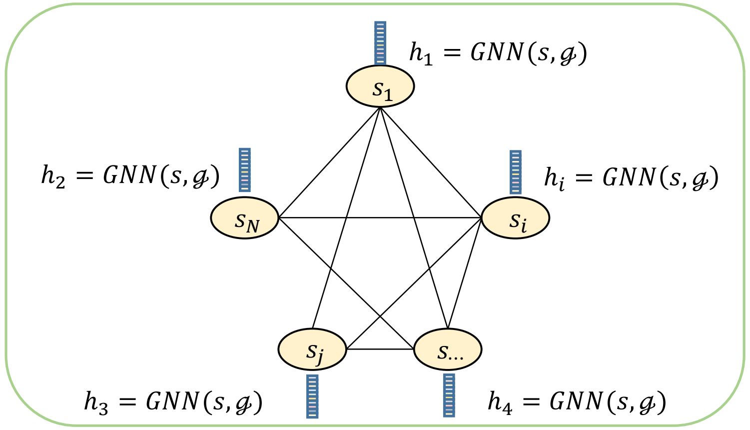
(State representation with inductive biases)

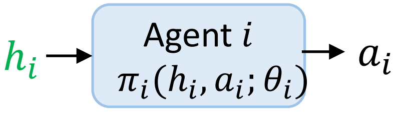
$$\approx \prod_{i=1}^{N} \pi(h_i = GNN(s; \phi), a_i; \theta)$$

Parameter sharing among **homogeneous** agents

Consensus through Communication & GNN

Learning to communicate





Information Structure in CTDE framework (groups of homogeneous agents)

Local observation: $h_i(s) = s_i (\text{or } o_i)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s_i, a_i)$$

Do not know other agent's state and action

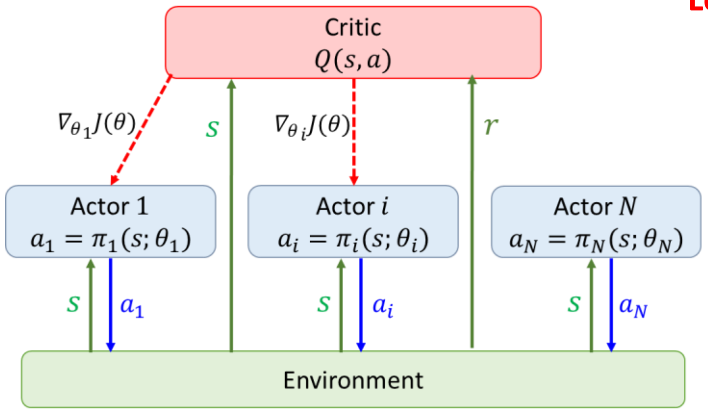
$$\approx \prod_{i=1}^{N} \pi_i(s_i, a_i; \theta_i)$$:Function approximation

$$= \prod_{i=1}^{N} \pi(s_i, a_i; \theta_{g(i)}) \quad g(i) \in \{1, \ldots, M\}$$

Parameter sharing among **a group of agents**

Consensus through central $Q(s,a;\phi)$ modeling

Learning to cooperate



Global observation: $h_i(s) = s (\text{or } o)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s, a_i)$$

Do not know other agent's action

$$\approx \prod_{i=1}^{N} \pi_i(s, a_i; \theta_i)$$:Function approx.:

→**difficult** to process s (high dim)

$$\approx \prod_{i=1}^{N} \pi_i(h_i = GNN(s;\phi_i), a_i; \theta_i)$$

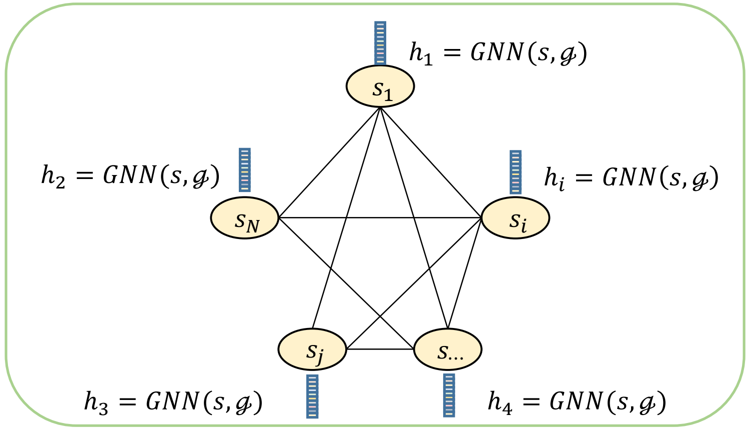
(State representation with inductive biases)

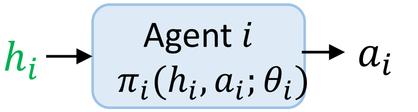
$$\approx \prod_{i=1}^{N} \pi(h_i = GNN(s;\phi), a_i; \theta_{g(i)}) \quad g(i) \in \{1, \ldots, M\}$$

Parameter sharing among **a group of agents**

Consensus through Communication & GNN

Learning to communicate





Consensus through central Critic modeling
(learning to cooperate)

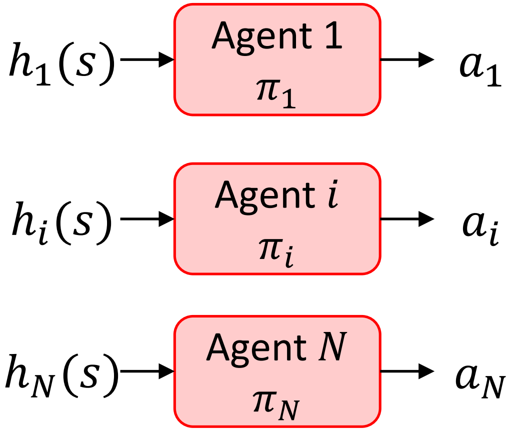
Challenges in “learning to cooperate approach”

$s = (s_1, \dots, s_N)$
$a = (a_1, \dots, a_N)$
$r = f(s, a)$

$Q(s, a)$

$\pi_1, \dots, \pi_i, \dots, \pi_N$

Centralized Training



Decentralized Execution

$h_i(s) = o_i$

Usually local observation is assumed

Two challenging questions:

- How to represent the action-value function that capturing the interactions among agents
  - Require the global state and joint action with the associated global reward
  - Difficult to learn when the dimension of state is large
  - Difficult to learn when the dimension of action (the number of agents) is large

- How to extract decentralized policies that allow each agent to select only an individual action based on an **individual observation**
  - Not obvious!

Factorized or Decomposed Approaches to Represent Central Q

Individual Action Value Function
(IQL)

- Cannot explicitly represent interactions between the agents and may not converge, as each agent's learning is confounded by the learning and exploration of others
- (Tan, 1993)

Factorized or Decomposed Approaches
(with structural assumptions)

- Learn a fully ***centralized but factorized*** state-action value function Q(s, a)
- Apply an inductive biases (structural assumptions) to factorize or decompose the Q(s, a)
- Simple example would be fully decomposed one:

$$Q(s, a) = Q_1(s_1, a_1) + \cdots + Q_N(s_N, a_N)$$

- Value Decomposition Network (VDN) (Sunehag et al., 2019)
- Monotonic Value Function Factorization (QMIX) (Rashid et al., 2018)

Fully Centralized State Action Value Function
+
Decentralized Policy
(Actor Critic Framework)

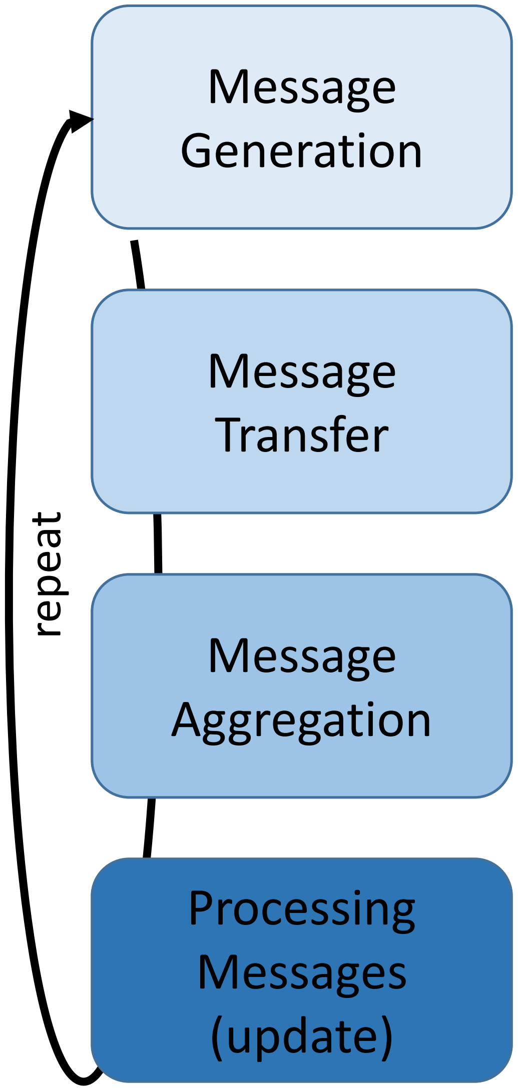
- Learn a fully centralized state-action value function Q(s, a) and then use it to guide the optimization of decentralized policies in an actor-critic framework
- Counterfactual multi-agent (COMA) policy gradients (Foerster 3t al., 2018)
- (Gupta et al., 2017)
- These requires on-policy learning, which can be **sample-inefficient**
- **Not scalable** to learn central Q(s, a) with more than a handful of agents

Consensus through Communication
(learning to communicate)

Why Communication is Required?

- Communication is a fundamental aspect of intelligence, enabling agents to behave as a group, rather than a collection of individuals
- It is vital for performing complex tasks in real-wolrld environments whre each actor has limited capabilities and/or visibility of the world
- Practical examples include
  - Elevator control
  - Sensor network
  - Robot soccer
- In any partially observed environment, the communication between agents is vital to coordinate the behavior of each individual

Categorization of Communication



How to generate a message

- Use only sender's information: :message = f(sender's info)
- Use both sender and receiver information: :message = f(sender's info, receiver's info)
- Use sender, receiver, and communication path info :message = f(sender, receiver, edge)

How to transfer a generated message

- Communication path is not specified (transfer message over the Air)
- Communication path is specified using a network or graph
- Communication path is learned through attending mechanism

How to aggregate messages from multiple senders

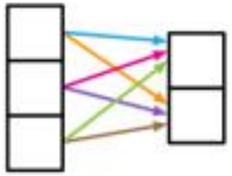
- Concatenate
- Average-pooling, Max pooling
- With varying importance (Attention)

How to process information (node embedding)

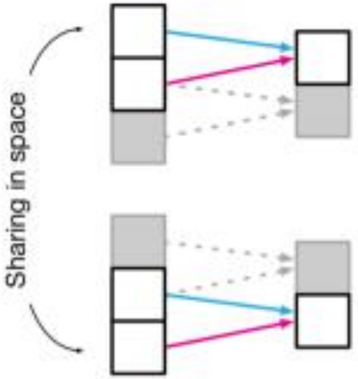
- Combined with local information (hidden vector) with the received and aggregated messages
- In case historical trajectories (observations, actions) are important, use RNN type
- Use the updated hidden embedding to conduct some task

Graph Neural Networks

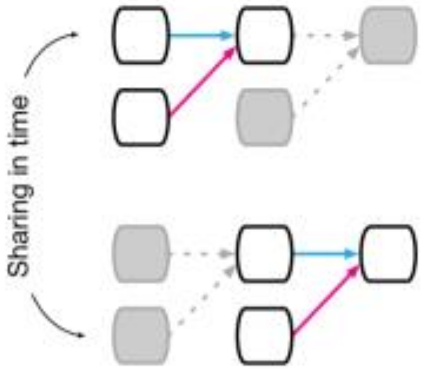
- **Combinatorial generalization** is an ability for constructing new inferences, predictions and behaviors from known building blocks. (AI to achieve a human-like abilities)
- **Relational inductive biases** within deep learning architectures can facilitate learning about entities, relations, and the rules for composing them
- We need models with explicit representations of entities and relations, and learning algorithms which find rules for computing their interactions, as well as ways of grounding them in data.
  - ➢ **Thus, we need Graph**



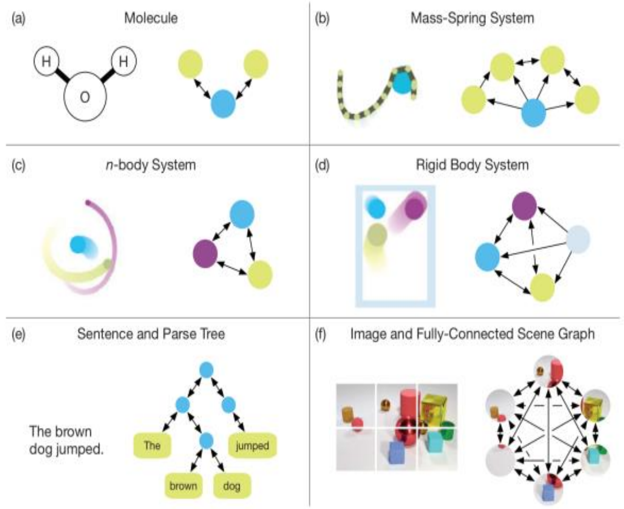
Fully connected



Convolutional

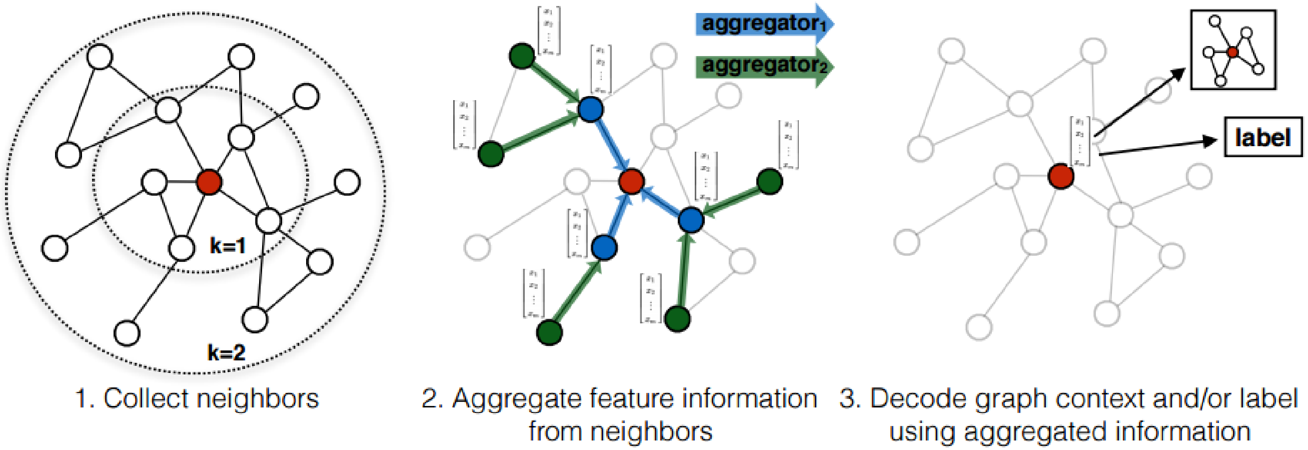


Recurrent



Graph

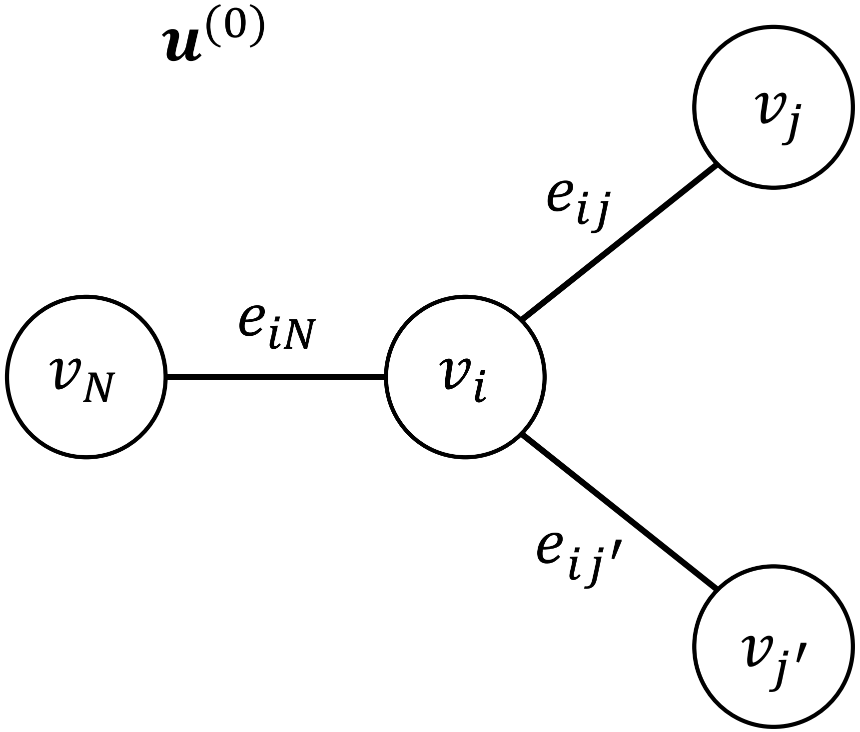
Graph Neural Networks



(Figure: example showing how GNN process and propagate data among nodes and edges)

- Graph The Neural Network (GNN) accepts a graph $G = (\boldsymbol{u},\ V,\ E)$ as an input
  - ✓ $\boldsymbol{u}$ is global attribute
  - ✓ V is a set of nodes (each represented by attributes)
  - ✓ E is a set of edges (also represented by attributes)
- Graph Neural Network (GNN) outputs a graph or vector as predictions
- GNN learns how to propagate and process the data among neighboring nodes and edges using neural network
- Because it learns the relationships among nodes, it can applied to any size of graph with different nodes (generalization)

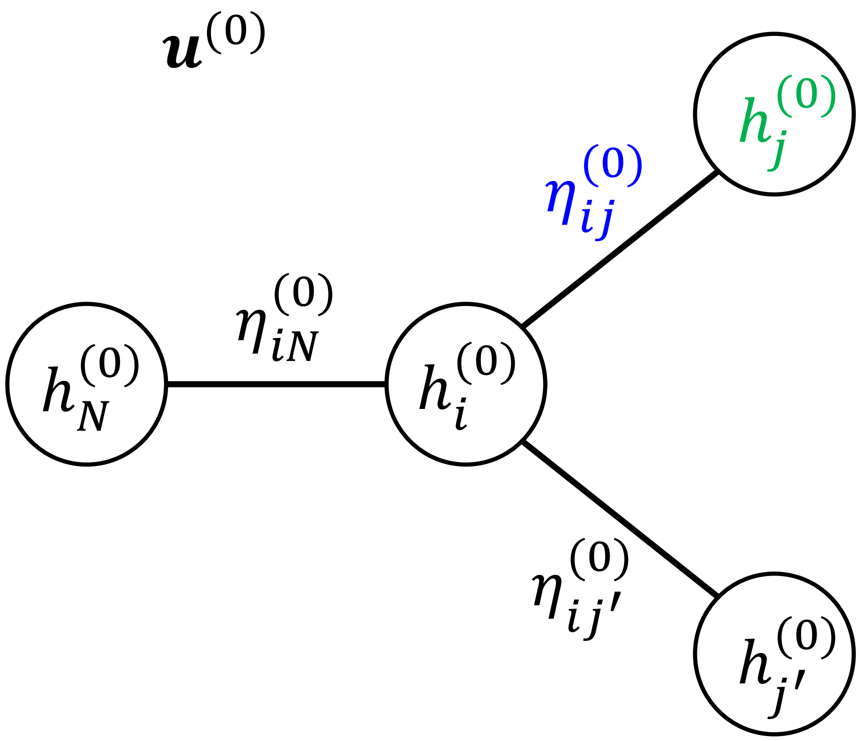
Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$

- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

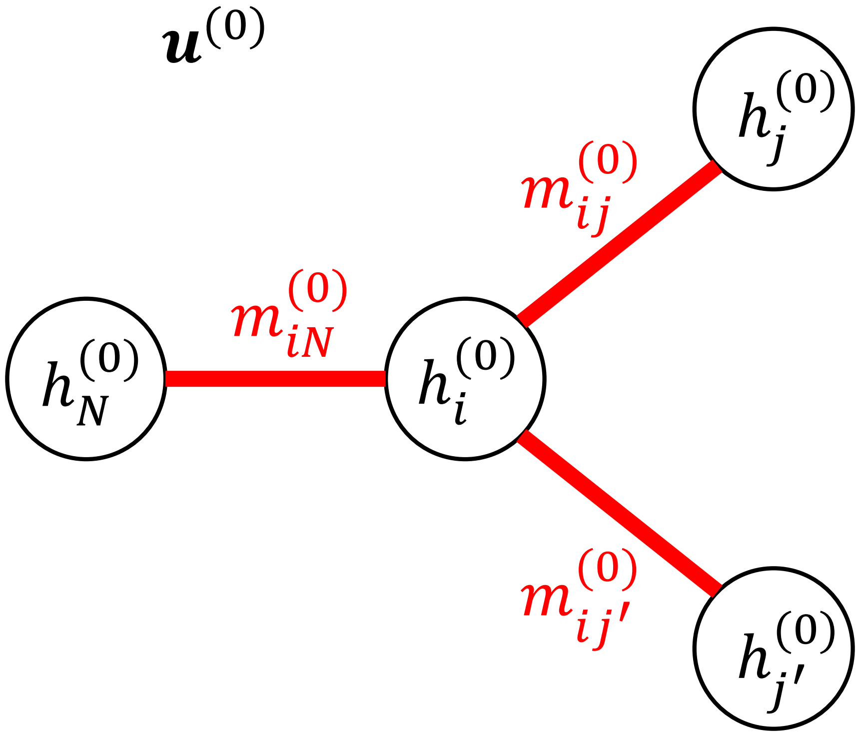
Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

- **Node Initialization & Edge Initialization**
  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i
  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair
- Message Computation
  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair
- Aggregate Message
  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$
- Node & Update
  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$
- Edge Update
  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$
- Graph Feature Update
  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

- **Node Initialization & Edge Initialization**

  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i

  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair

- **Message Computation**

  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair

- **Aggregate Message**

  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$

- **Node & Update**

  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$

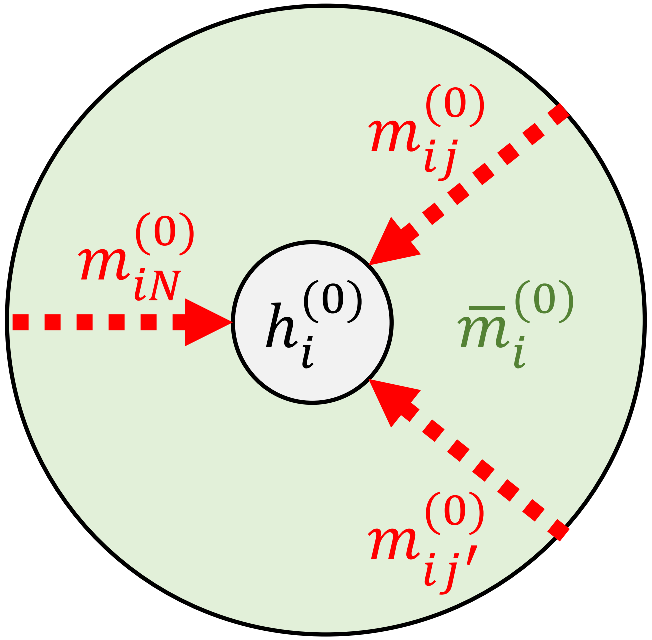
- **Edge Update**

  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$

- **Graph Feature Update**

  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$

- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

- **Node Initialization & Edge Initialization**

  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i

  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair

- **Message Computation**

  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair

- **Aggregate Message**

  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$

- **Node & Update**

  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$

- **Edge Update**

  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$

- **Graph Feature Update**

  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

Communication by Graph Neural Networks

$$h_i^{(1)} \leftarrow \bar{m}_i^{(0)} + h_i^{(0)}$$

- **Node Initialization & Edge Initialization**

  $$h_i^{(0)} = f(v_i; \theta_0^N) \text{ for all } i$$
  $$\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right) \text{ for all } (i,j) \text{ pair}$$

- **Message Computation**

  $$m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right) \text{ for all } (i,j) \text{ pair}$$

- **Aggregate Message**

  $$\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$$

- **Node & Update**

  $$h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$$

- **Edge Update**

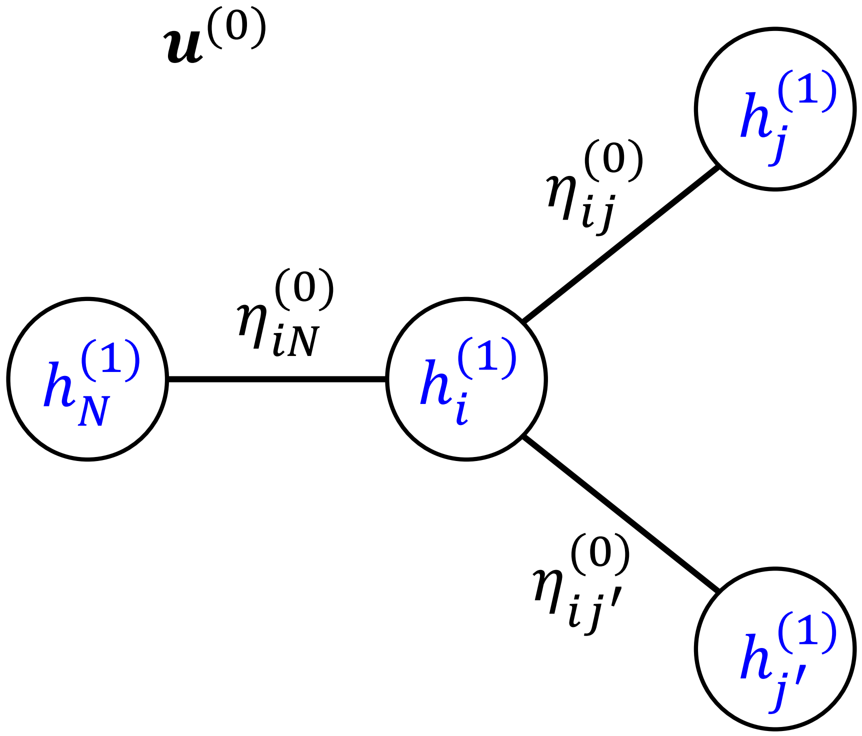
  $$\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$$

- **Graph Feature Update**

  $$\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$$

Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

- **Node Initialization & Edge Initialization**

  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i

  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair

- **Message Computation**

  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair

- **Aggregate Message**

  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$

- **Node & Update**

  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$ **for all *i***

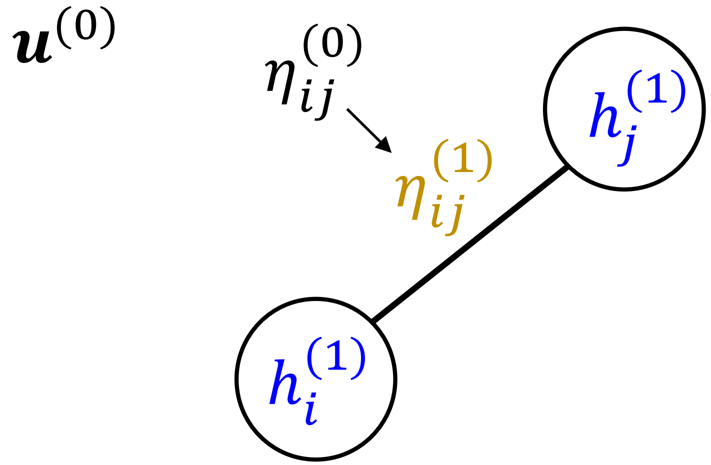
- **Edge Update**

  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$

- **Graph Feature Update**

  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

Communication by Graph Neural Networks



- **Node Initialization & Edge Initialization**

  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i

  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair

- **Message Computation**

  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair

- **Aggregate Message**

  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$

- **Node & Update**

  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$ **for all $\boldsymbol{i}$**

- **Edge Update**

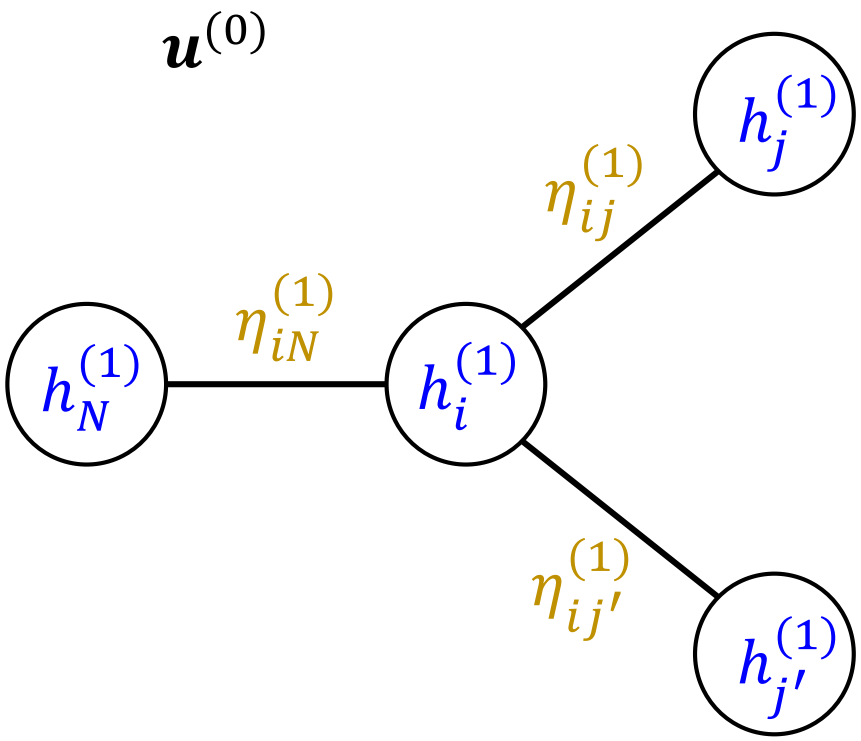
  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$

- **Graph Feature Update**

  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

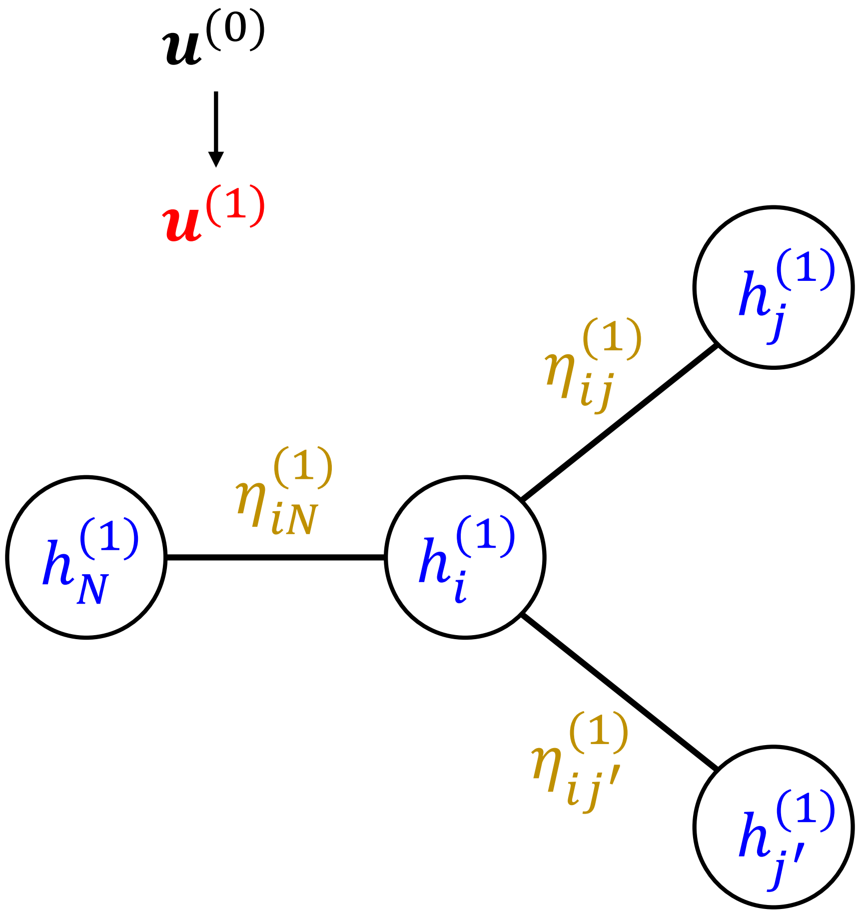
Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

- **Node Initialization & Edge Initialization**
  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i
  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair
- **Message Computation**
  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair
- **Aggregate Message**
  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$
- **Node & Update**
  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$ for all i
- **Edge Update**
  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$ **for all i**
- **Graph Feature Update**
  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

Communication by Graph Neural Networks



Input a graph, defined as tuple $G = (\boldsymbol{u},\ V,\ E)$
- $\boldsymbol{u}$ is global attribute
- $V = \{v_1, \dots, v_N\}$ is a set of nodes with v_i node features
- $E = \{e_{12}, \dots, e_{iN}\}$ is a set of edges with e_{ij} edge features

- **Node Initialization & Edge Initialization**

  $h_i^{(0)} = f(v_i; \theta_0^N)$ for all i

  $\eta_{ij}^{(0)} = f\left(e_{ij}; \theta_0^E\right)$ for all (i, j) pair

- **Message Computation**

  $m_{ij}^{(0)} = g\left(h_j^{(0)}, h_i^{(0)}, \eta_{ij}^{(0)}; \theta^m\right)$ for all (i, j) pair

- **Aggregate Message**

  $\bar{m}_i^{(0)} = \sum_{j \neq i} m_{ij}^{(0)}$

- **Node & Update**

  $h_i^{(1)} = g\left(h_i^{(0)}, \bar{m}_i^{(0)}; \theta_1^u\right)$ for all i

- **Edge Update**

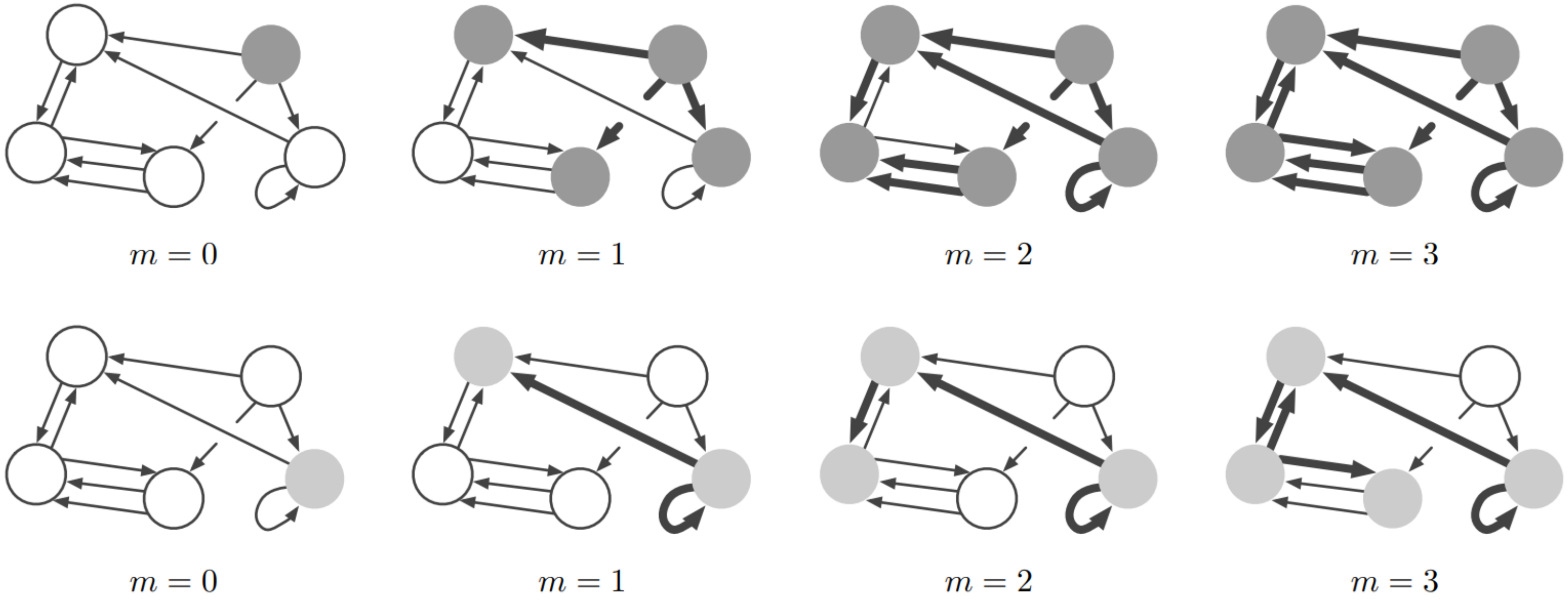
  $\eta_{ij}^{(1)} = g\left(\eta_{ij}^{(0)}, h_i^{(1)}, h_j^{(1)}; \theta_2^u\right)$ for all i

- **Graph Feature Update**

  $\boldsymbol{u}^{(1)} = g\left(\boldsymbol{u}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{\eta}^{(1)}; \theta_3^u\right)$

  where $\boldsymbol{h}^{(1)} = \left\{\boldsymbol{h}_1^{(1)}, \dots, \boldsymbol{h}_N^{(1)}\right\}$ $\boldsymbol{\eta}^{(1)} = \left\{\boldsymbol{\eta}_1^{(1)}, \dots, \boldsymbol{\eta}_N^{(1)}\right\}$

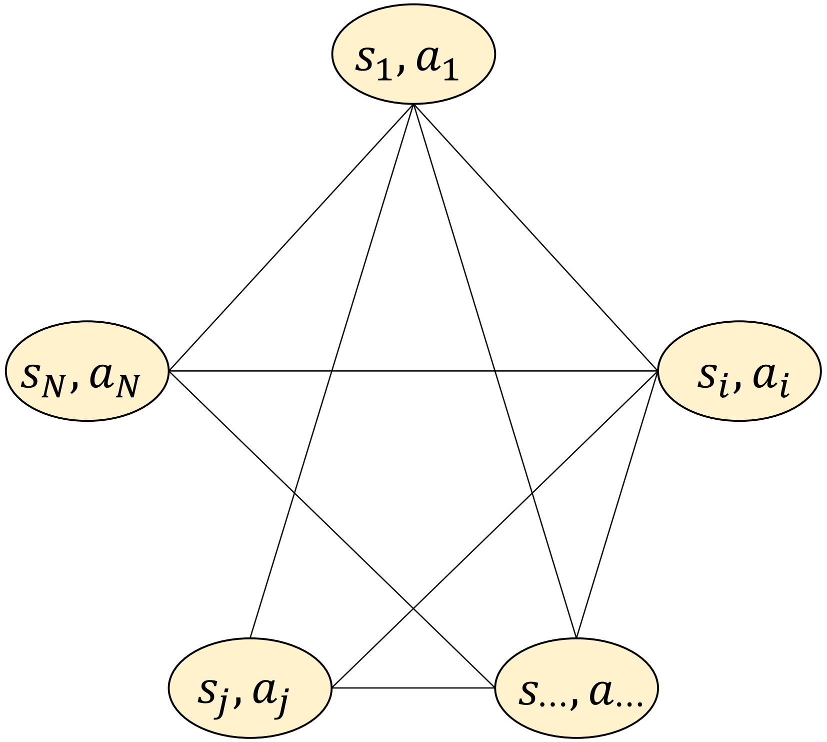
Communication by Graph Neural Networks



- As the iteration repeats, the information from the far nodes can be propagated into a target node
- The propagated and processed data at a target node then capture the information about all nodes

Figure from: Inductive Biases

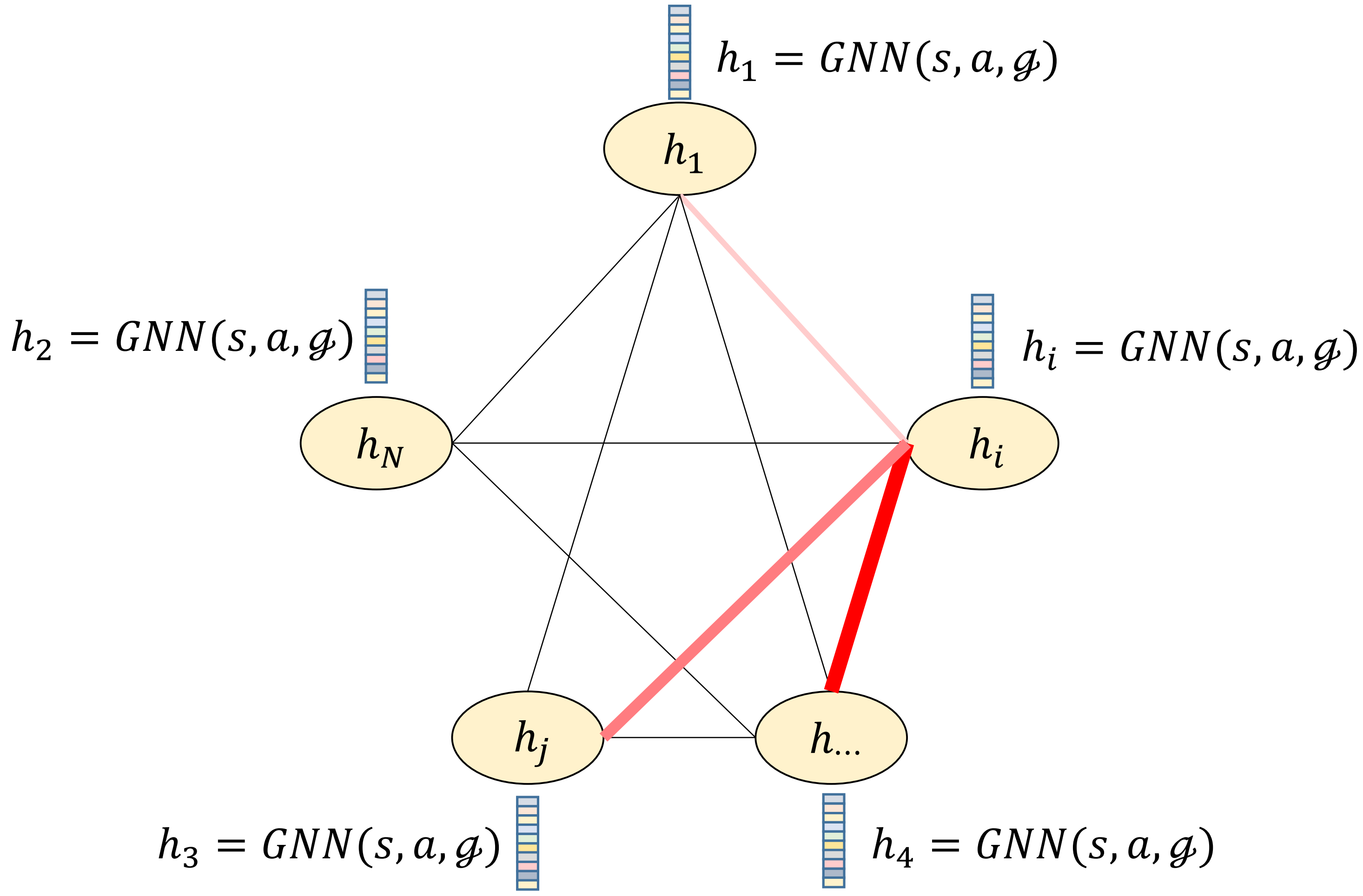
Central Critic approach + Communication Approach



Objective:

Find the decentralized policy $\pi_\theta(s, a) \approx \prod_{i=1}^{N} \pi_{\theta_i}(s_i, a_i)$

Central Critic approach + Communication Approach



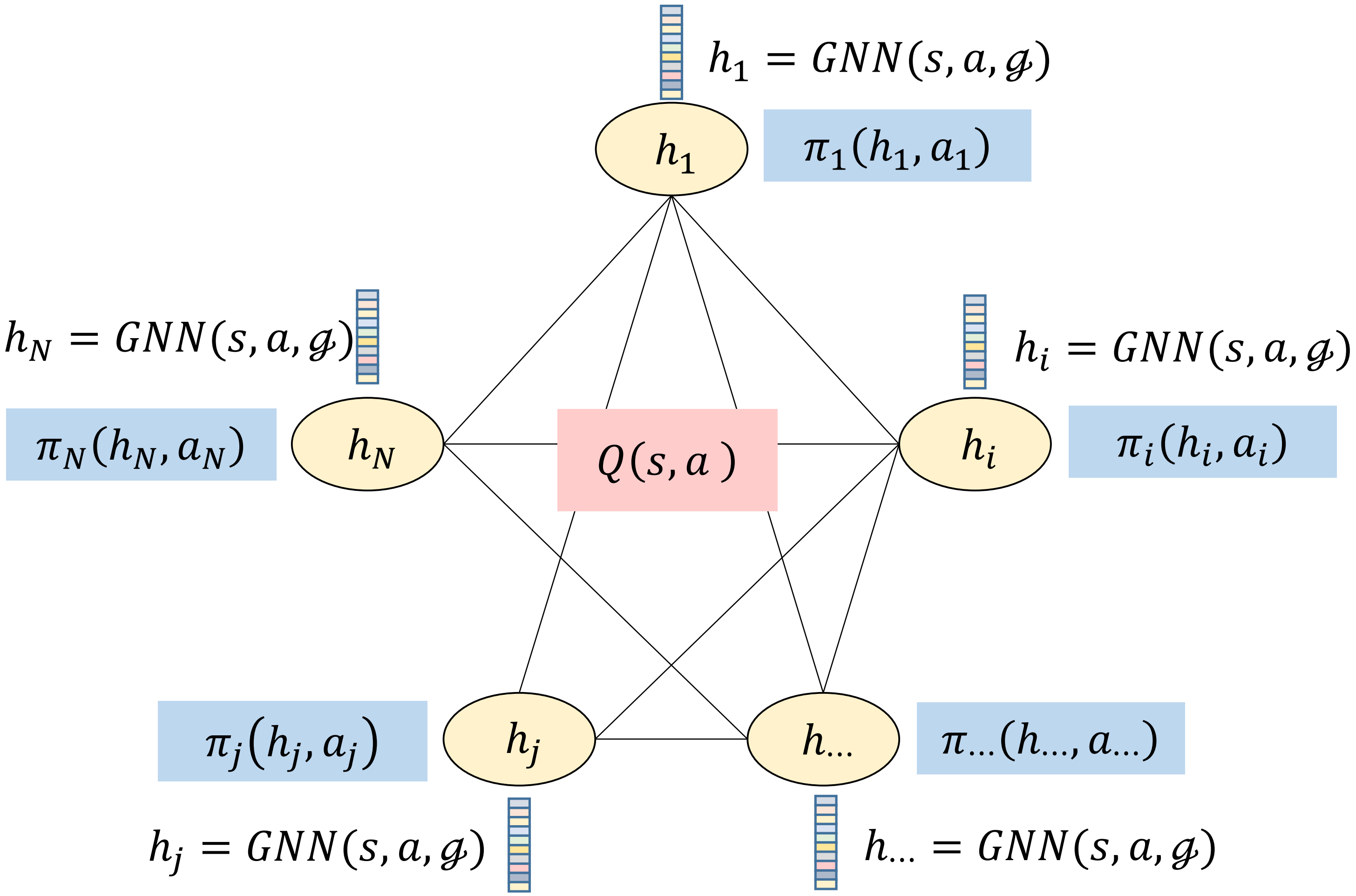
Message computation

- $\bar{h}_i = \sum_{j \in \mathcal{N}(i)} m(h_i, h_j)$ *or*
- $\bar{h}_i = \sum_{j \in \mathcal{N}(i)} a_{ij} m(h_j)$ *or*
- $h_i = \sum_{j \in \mathcal{N}(i)} m(h_j)$

Node update

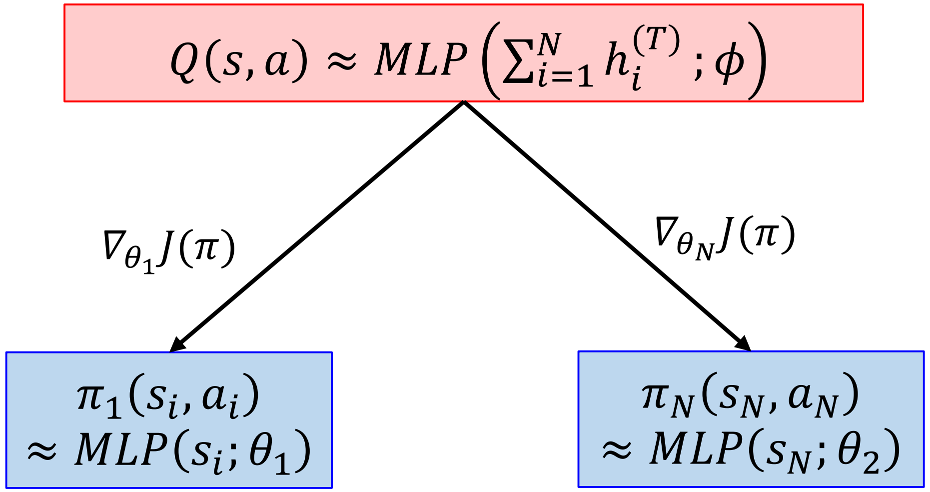
- $h_i' = g(h_i, \bar{h}_i)$

Central Critic approach + Communication Approach



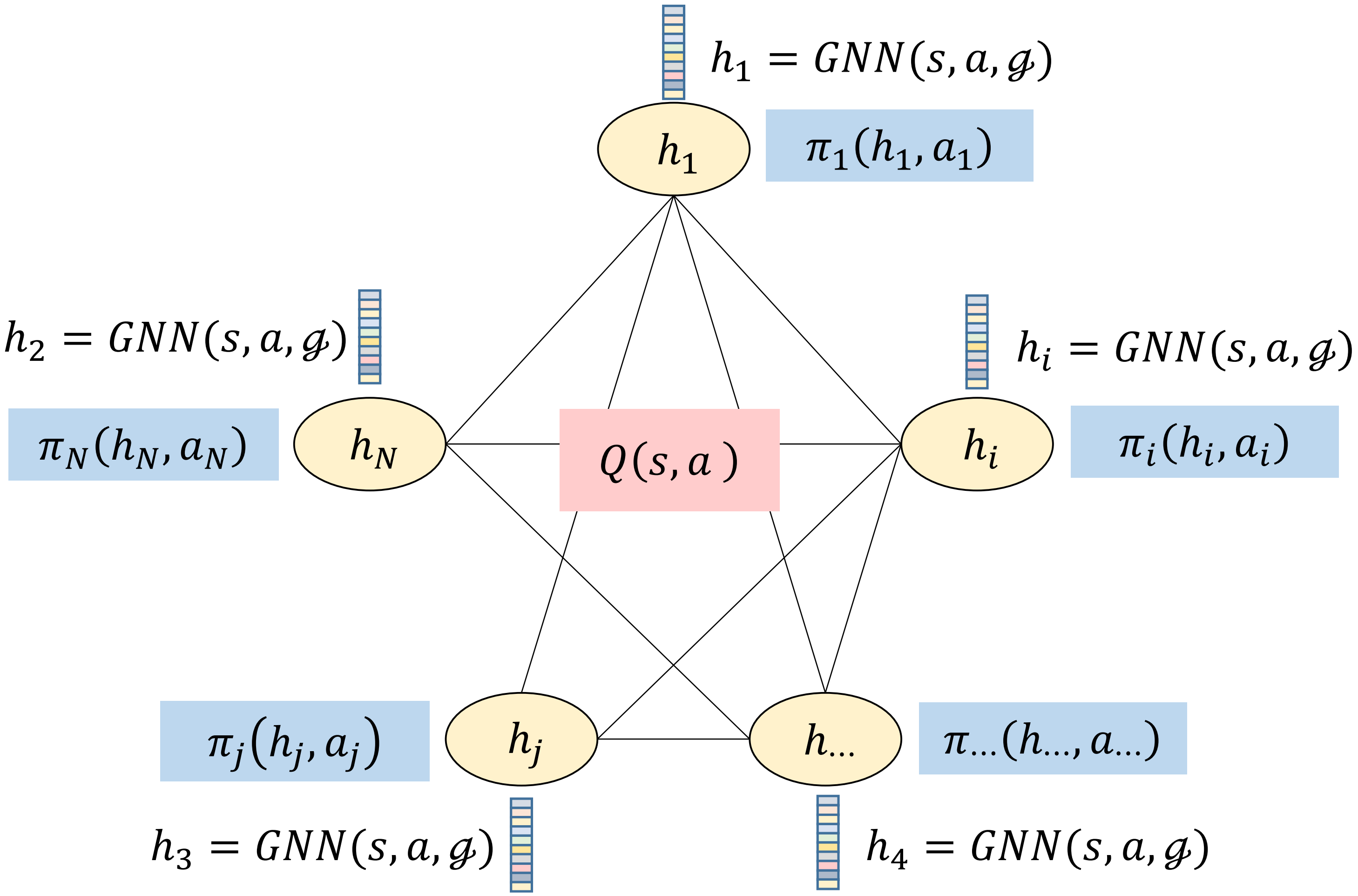
GNN for approximating global Q

Centralized Critic:



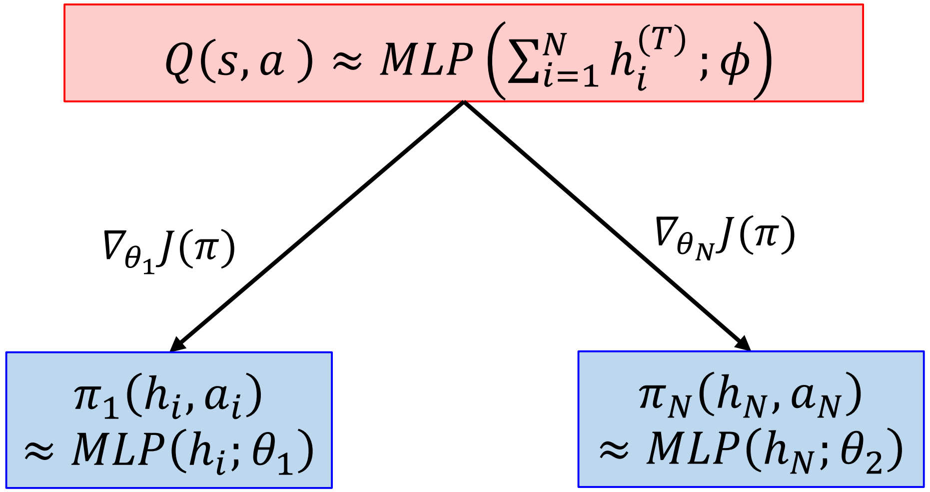
Decentralized actor

Central Critic approach + Communication Approach



GNN for approximating global Q

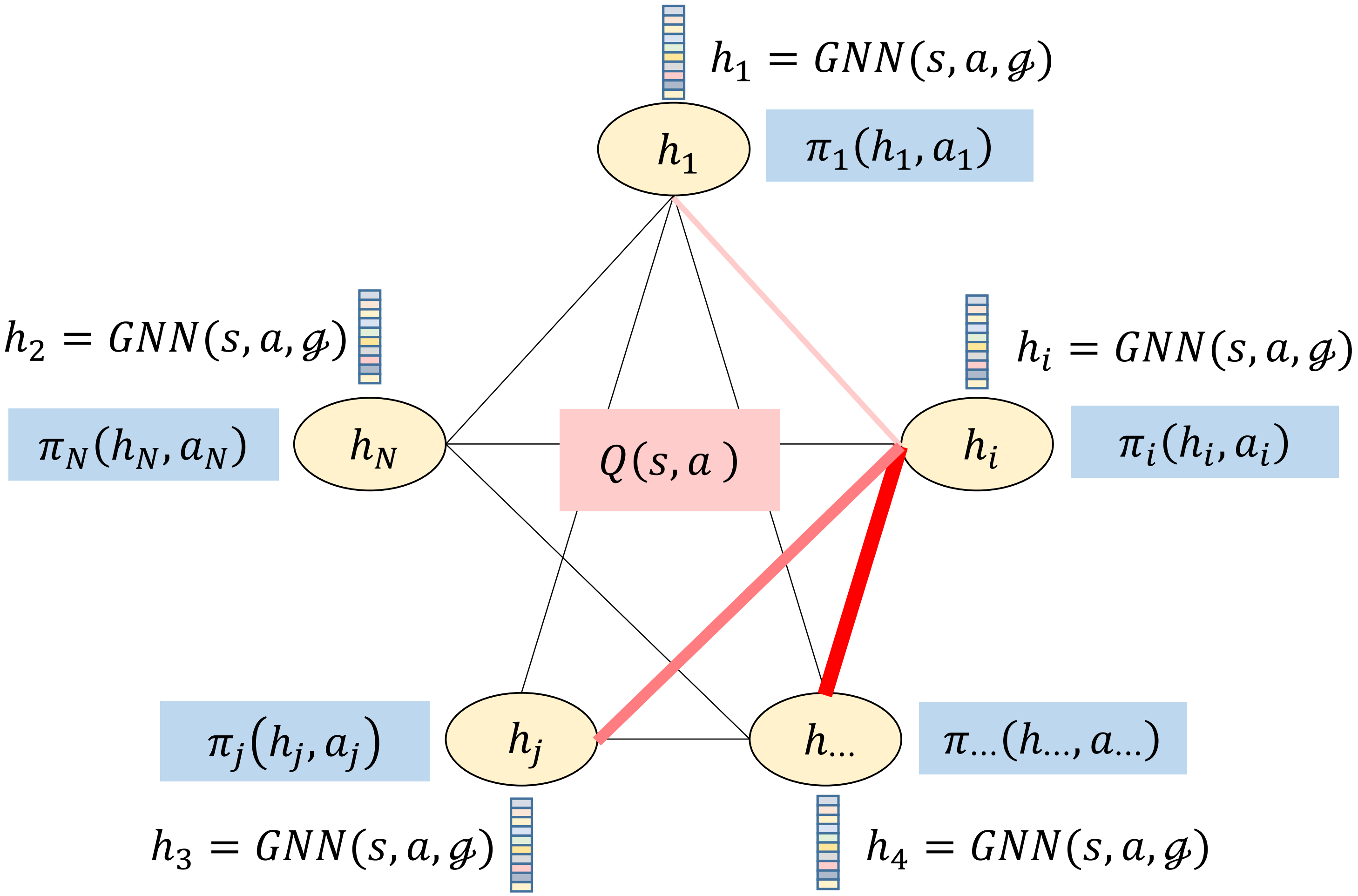
Centralized Critic:



Decentralized actor

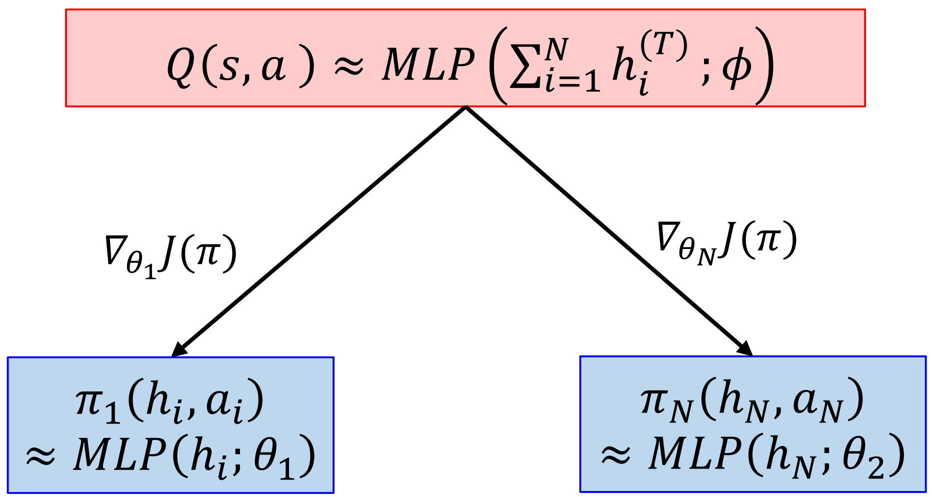
GNN for approximating Individual Policy

Central Critic approach + Communication Approach



GNN for approximating global Q

Centralized Critic:



Decentralized actor

GNN for approximating Individual Policy

Summary

- Widely adopted to overcome the non-stationarity problem when training multi-agent systems

- CTDE enables to leverage the observations of each agent, as well as their actions, to better estimate the action-value functions during training.

- As the policy of each agent only depends on its own private observations during training, the agents are able to decide which actions to take in a decentralized manner during execution