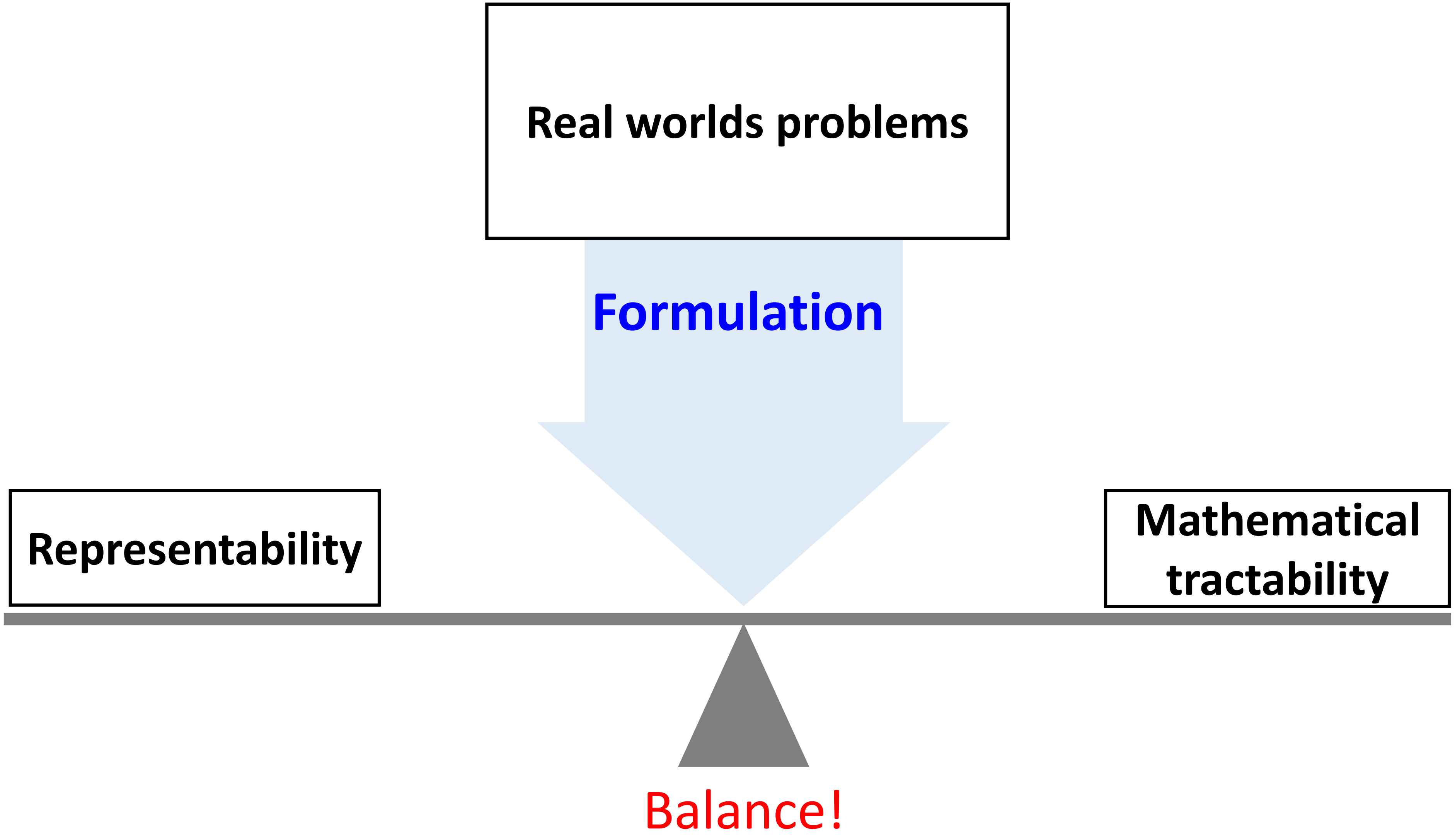
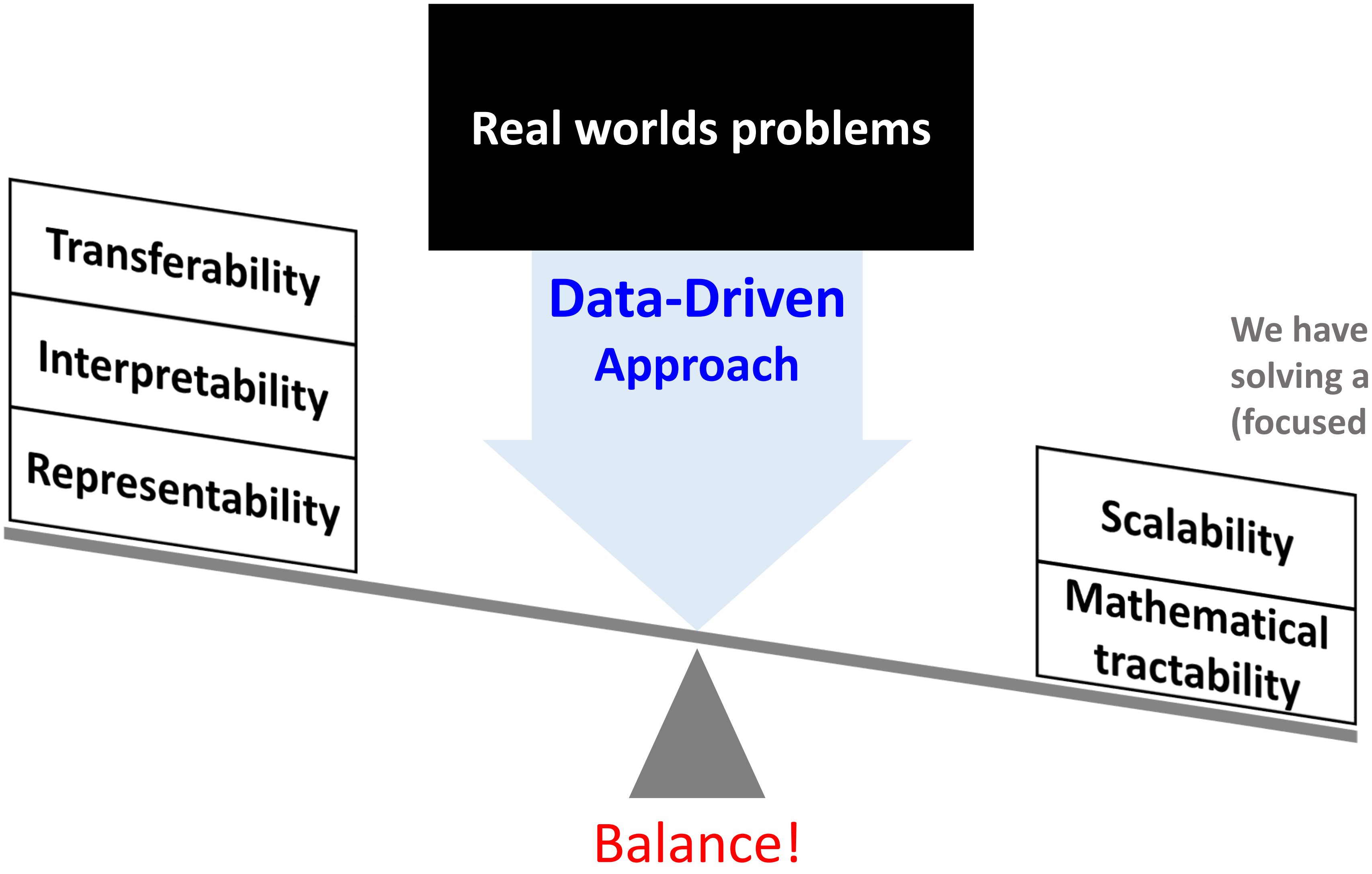
Lecture 11. Deep Multi-Agent Reinforcement Learning Centralized Training and Decentralized Execution Approach

Data-Driven Approach to Solve Problems

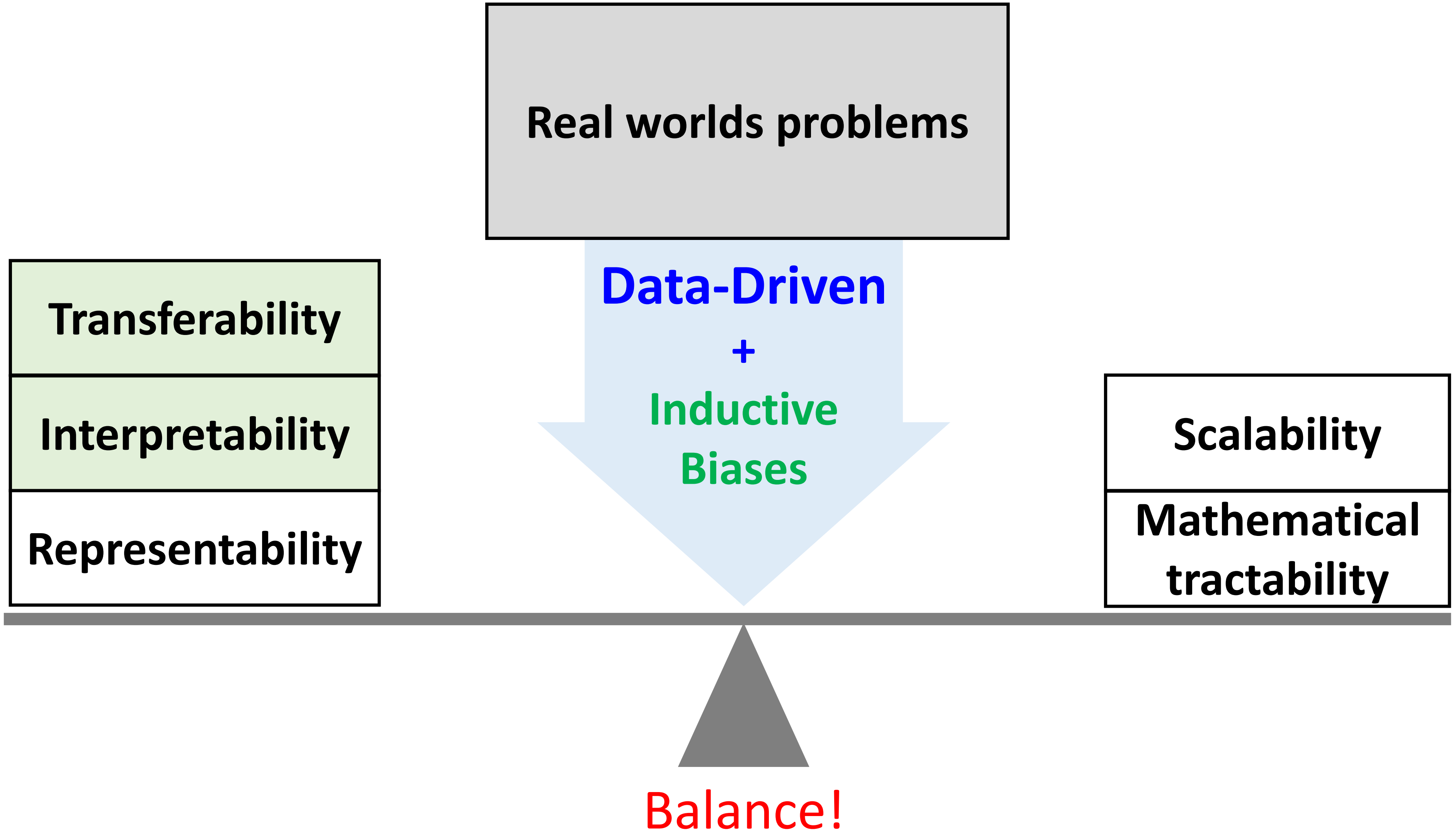


Data-Driven Approach to Solve Problems

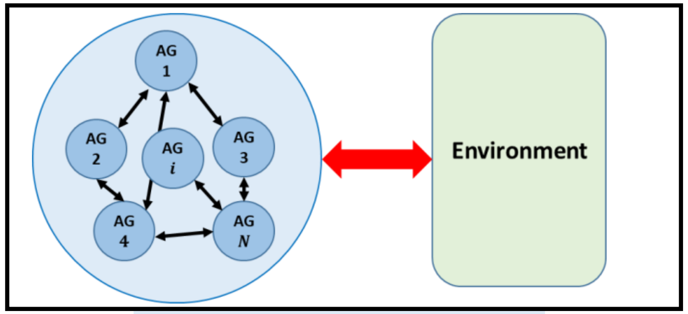


We have mainly focused on solving a target problem (focused on performance)

Data-Driven Approach to Solve Problmes



Data-Driven Approach to Solve Problmes



Transferability

Interpretability

Representability

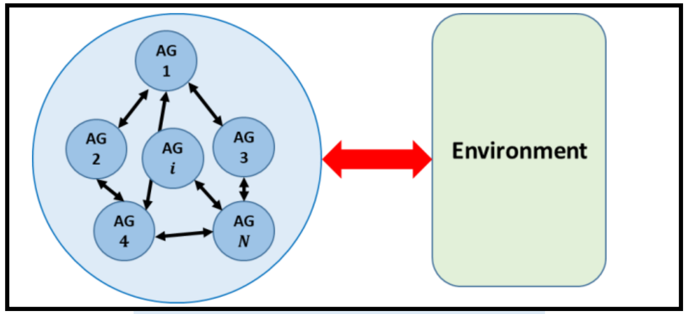
Data-Driven
+
Relative
Inductive
Biases

Scalability

Mathematical tractability

Balance!

Data-Driven Approach to Solve Problmes



MARL
+
Graph Neural Network

Transferability

Interpretability

Representability

Scalability

Mathematical tractability

Balance!

Deep learning with inductive bias

Physical system
(real world, observation, etc.)

Design feature (representation)
Design mappings with learnable parameters (model)

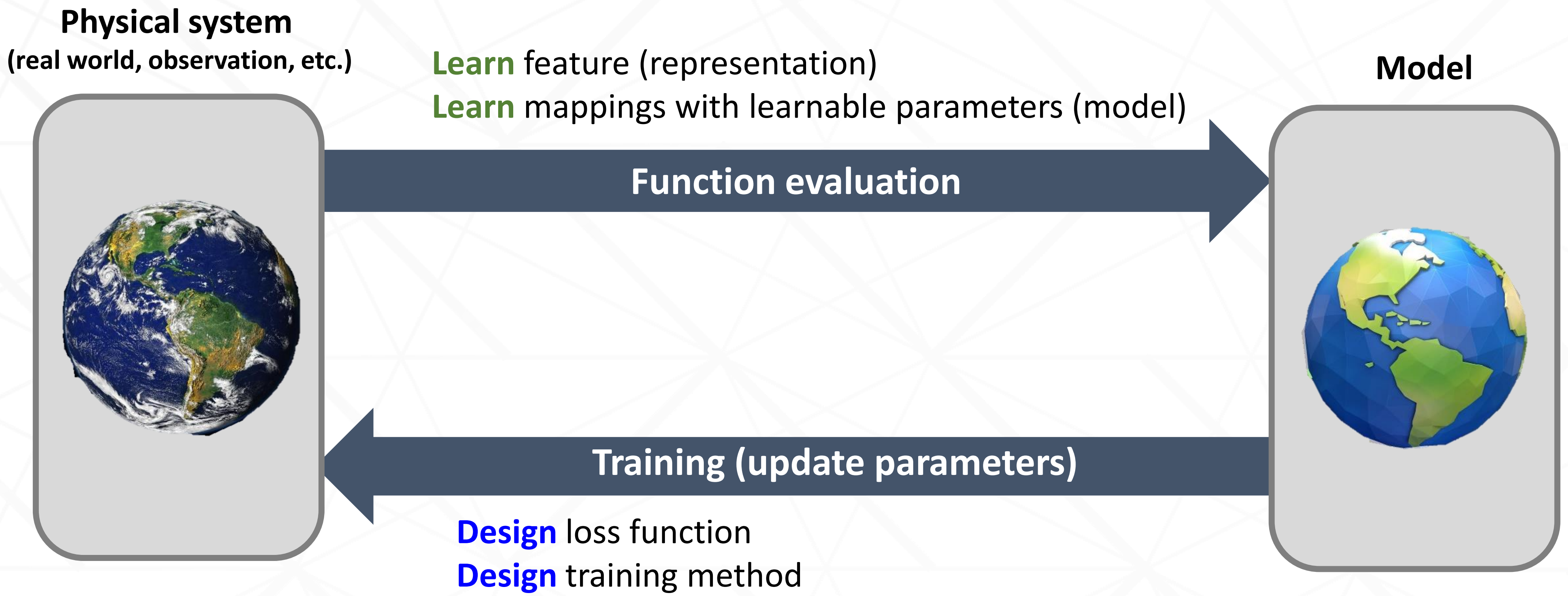
Function evaluation

Model

Training (update parameters)

Design loss function
Design training method

Deep learning with inductive bias



Deep learning with inductive bias

Physical system
(real world, observation, etc.)

Learn feature (representation)
Learn mappings with learnable parameters (model)

Function evaluation

employing prior knowledge (often called as inductive bias)
on feature and mapping learning process

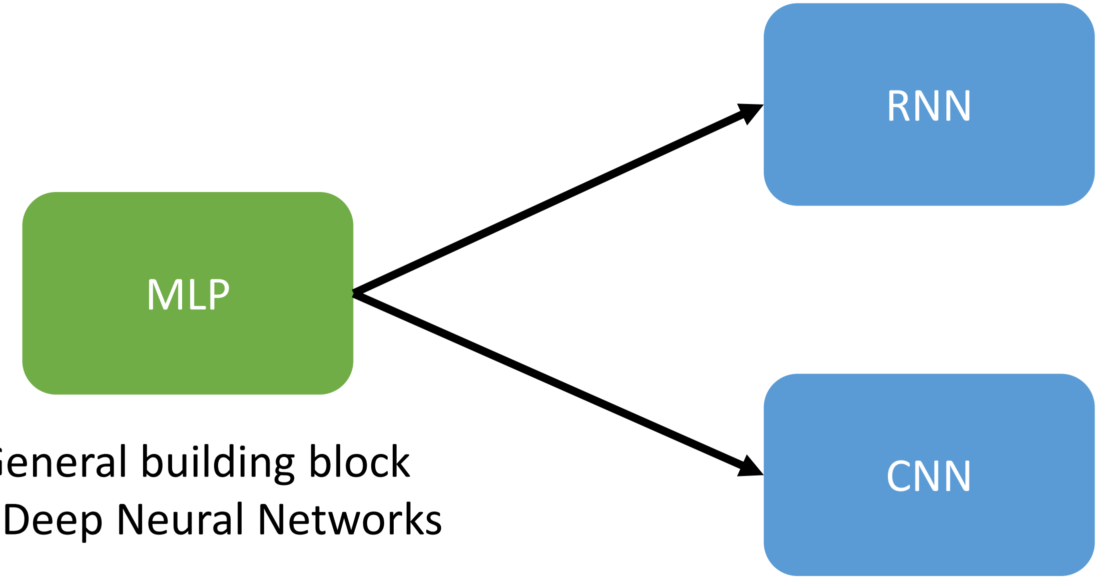
Training (update parameters)

Design loss function
Design training method

Model

Inductive biases on network architecture

Impose 'temporal structure' on data



$$x_{t+1} = f_\theta(x_t, u_t)$$

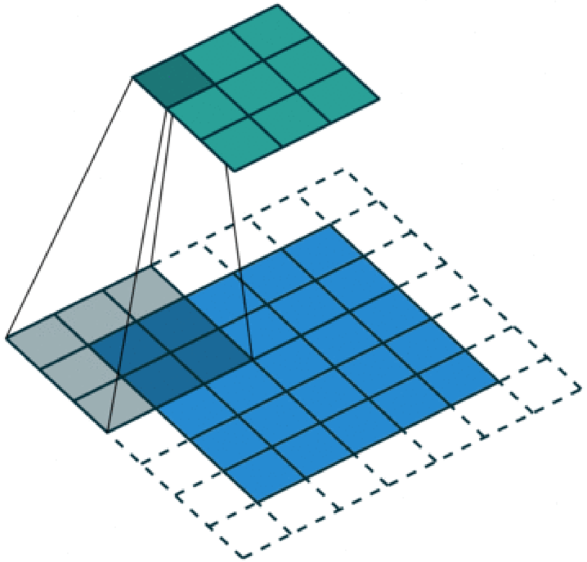
General building block
in Deep Neural Networks

$$x_{i,j} = g_\theta\left(x_{p,q}\right)_{(p,q)\in\mathcal{N}(i,j)}$$

Impose 'spatial structure' on data

$$\begin{cases}\text{CNN}\\ \text{RNN}\end{cases}\text{shares functions}\begin{cases}\textbf{Conv filter}\\ \textbf{RNN cell}\end{cases}\text{to learn interaction patterns among}\begin{cases}\textbf{Nearby pixels}\\ \textbf{time stamped data}\end{cases}$$

Inductive biases on network architecture



Convolution operation presumes that

‘Nearby pixels are somewhat related’.

Since we ***share*** the convolution filters

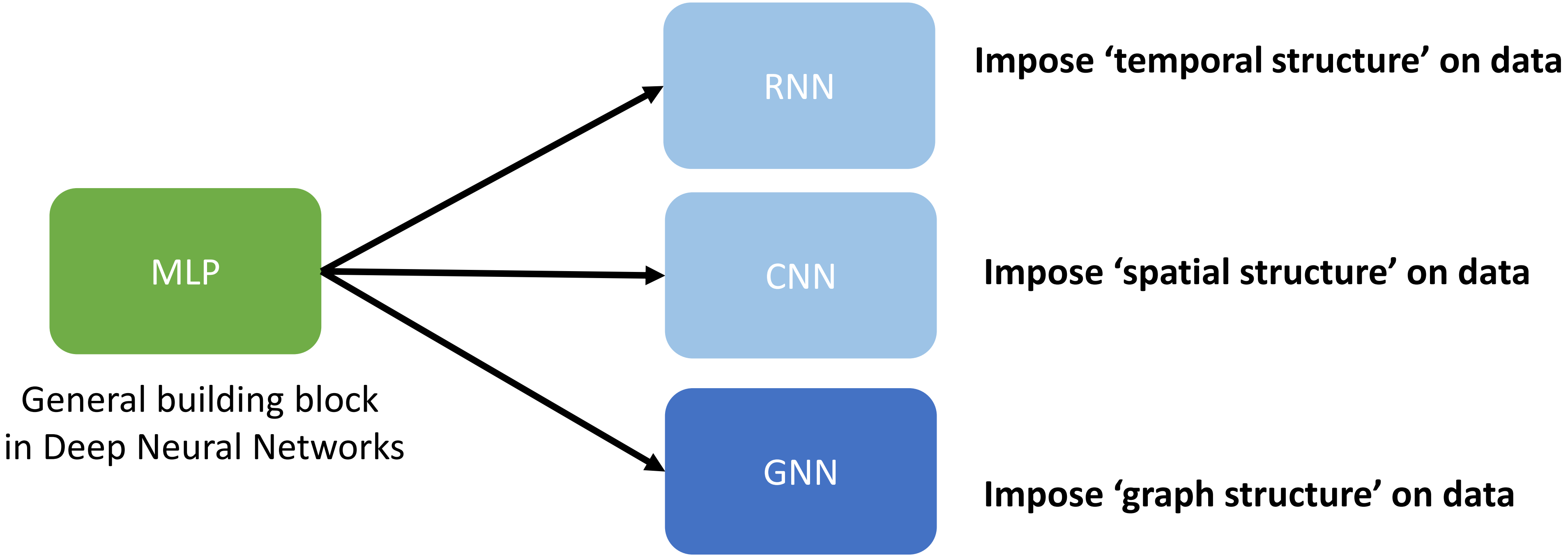
RNNs presumes that

‘Nearby inputs are somewhat related’.

Since we ***share*** the RNN blocks.

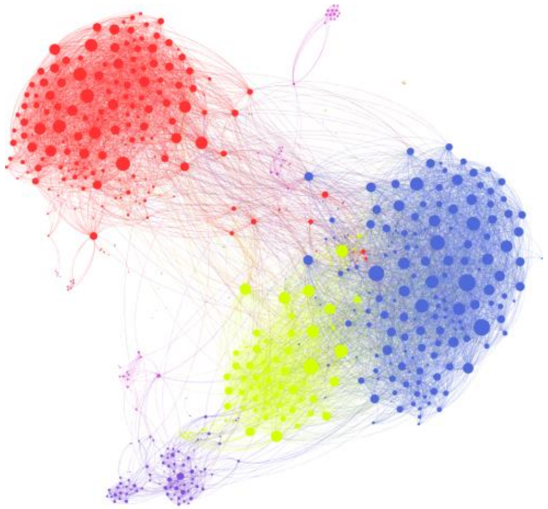
Figure source <Left: https://github.com/vdumoulin/conv_arithmetic>, <Right: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9>

Graph Neural networks – A general way for imposing inductive bias

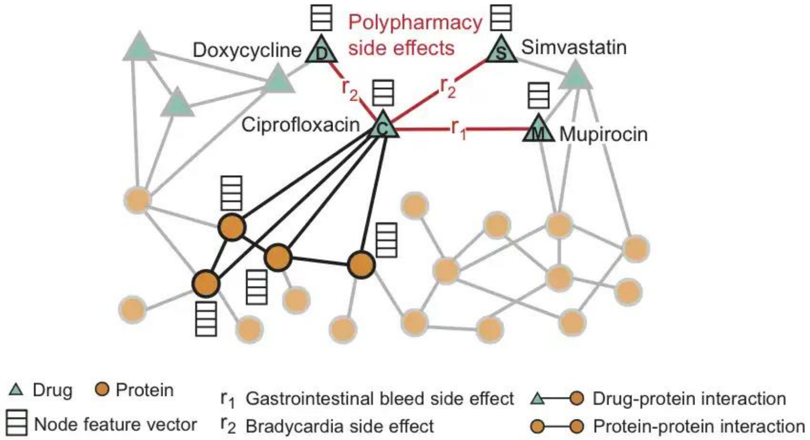


- GNN learns **pairwise interactions among nodes (or edges)** as we did for CNN and RNN.
- The learned interactions (usually represented as outputs of NN) can be used to perform inference tasks on differently composed graphs. e.g.) More/less nodes or edges than training cases.

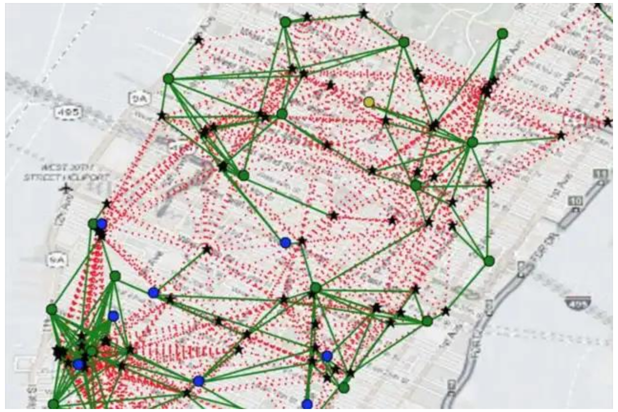
Graph Neural Network



<GNN for Social network analysis>



<GNN for modelling polypharmacy>

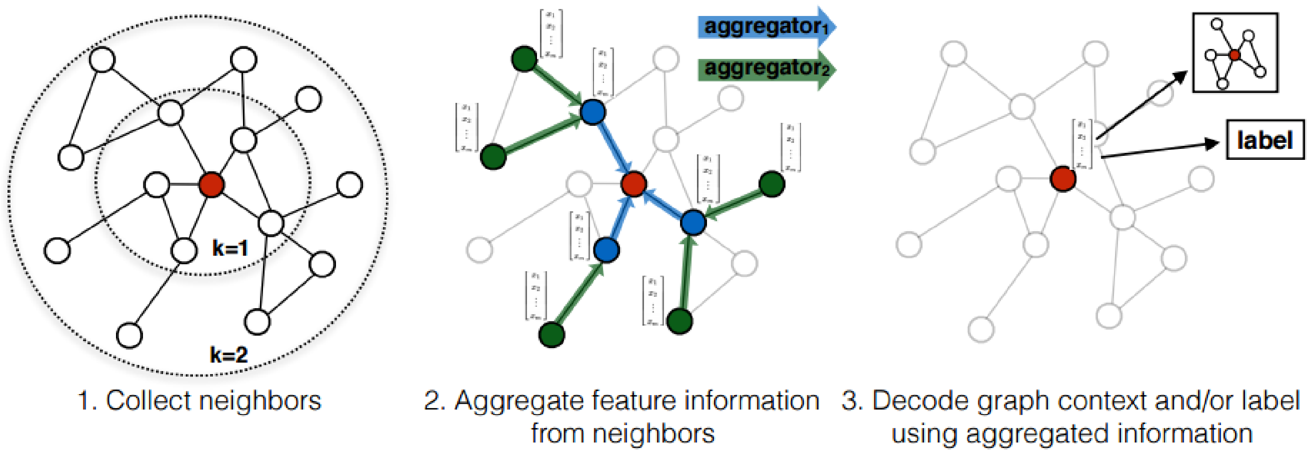


<GNN for modelling traffic system>

- Graph Neural Network (GNN) is a powerful tool for processing graph represented data.
- GNN employ sub-neural networks for processing information (Generalization)
- GNN computations employ the optimized batch computation for reducing computation time (Scalability)

Ref <https://griffsgraphs.wordpress.com/2012/07/02/a-facebook-network/> , <http://snap.stanford.edu/decagon/>, <http://news.mit.edu/2016/how-ride-sharing-can-improve-traffic-save-money-and-help-environment-0104>

Graph Neural Network + Classification/Regression

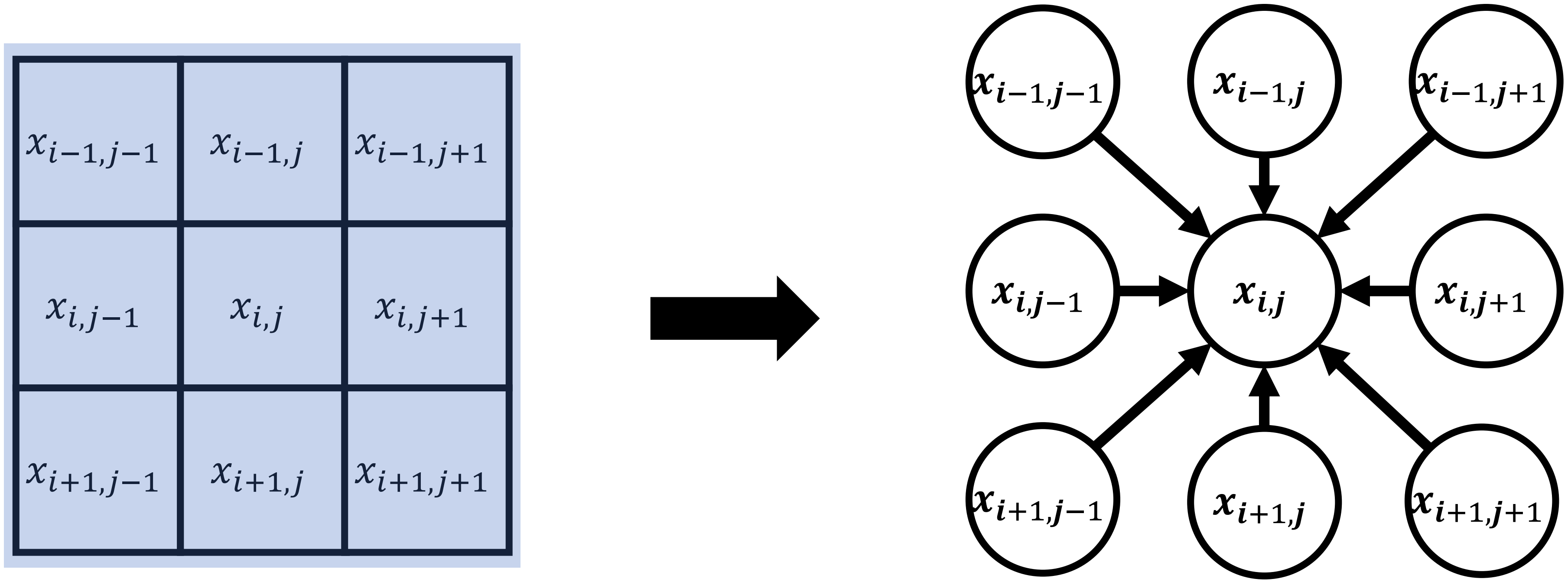


(Figure: example showing how GNN process and propagate data among nodes and edges)

- Graph The Neural Network (GNN) accepts a graph $G = (u,\ V,\ E)$ as an input
  - ✓ 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 NN
- Because it learns the relationships among nodes, it can applied to any size of graph with different nodes =

Ref:"Representation Learning on Graphs: Methods and Applications", Hamilton et al., Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

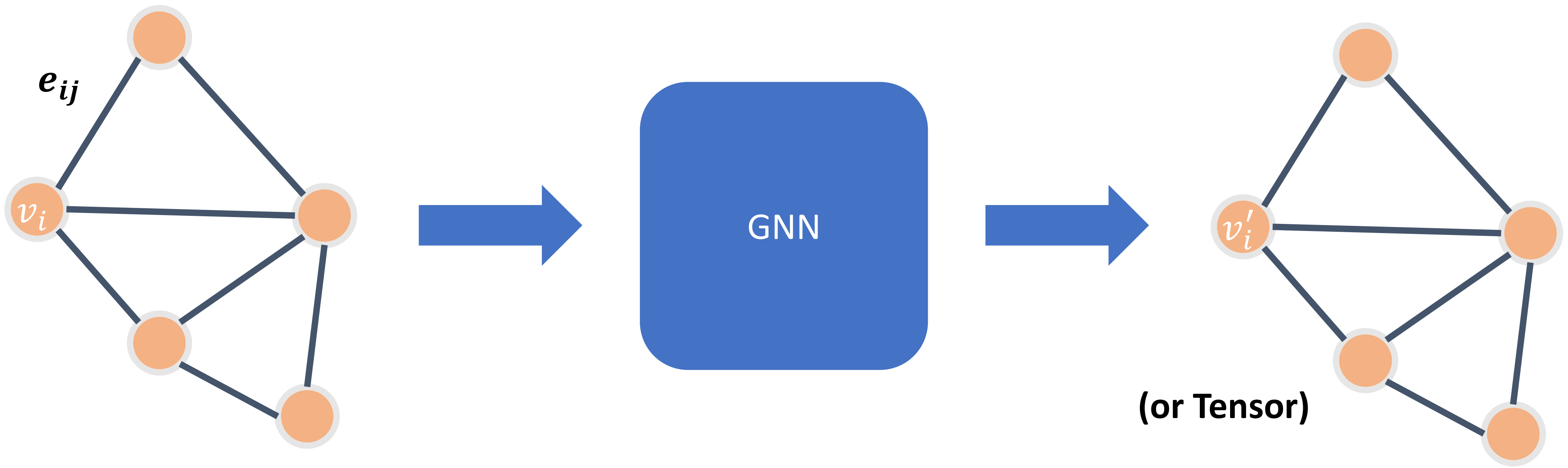
CNN as a special case of GNN



Convolution filter presume that the nearby pixels are correlated **in the same manner**.
Convolution filter presume the input image as **an graph** that nearby pixels are fully connected.

Graph Neural Networks (GNN)

Graph Neural Network (GNN) is a neural network that takes **graph(s)** as inputs



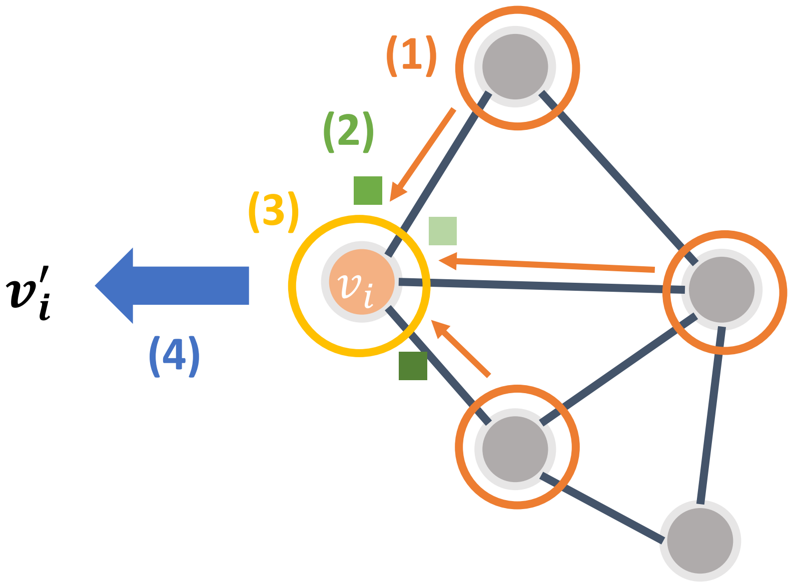
Input graph $\mathcal{g} =< \mathcal{V}, \mathcal{E} >$

Updated graph $\mathcal{g}' =< \mathcal{V}', \mathcal{E}' >$

Almost every GNN defines $\begin{cases} \textbf{(1) update rule for a node/edge} \\ \textbf{(2) apply the rule on entire (partial) nodes/edges} \end{cases}$ to get updated features

Typical Node update procedure in GNNs

(1) Generate message
(2) Weighting messages
(3) Aggregate message
(4) Update node feature

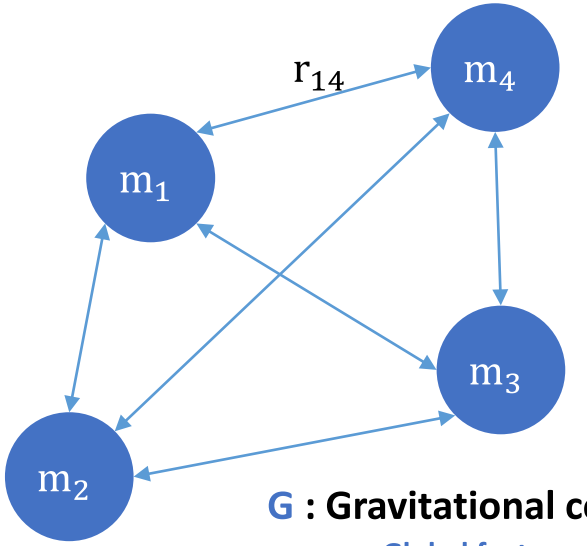


$$v_i' = f_\theta\left(v_i, aggreate\left(w_{j\to i}(v_i, v_j) * g_\theta(v_i, v_j)\right)_{j\in\mathcal{N}_i}\right)$$

Neighborhood set

- v_i : Node i feature
- f_θ, g_θ : Differentiable function with parameter θ
- $w_{j\to i}$: Weight coefficient from node $\boldsymbol{j}$ to node $\boldsymbol{i}$
- $aggreate(\cdot)$: Aggregator functions which satisfies ‘permutation invariants’. e.g.) Mean, Sum, Max …

Iterative methods on "__" and GNN update rule



G : Gravitational constant

<Global feature>

$$v_i' = f_\theta\left(v_i, aggreate\left(w_{j\to i}(v_i, v_j) * g_\theta(v_i, v_j)\right)_{j\in\mathcal{N}_i}\right)$$

GNN update routine

Objective : Find stationary state of bodies.

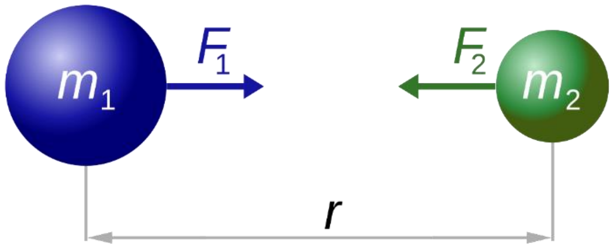
Repeat until converge

Step 1 : compute all pairwise forces $F_{ij} = G\frac{m_i m_j}{r_{ij}^2}$

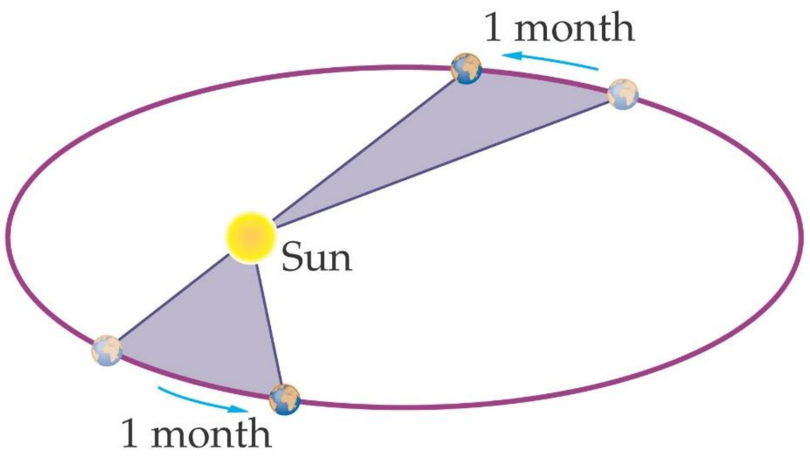
Step 2 : aggregate (sum) all forces $F_i = \sum_{j\in\mathcal{N}_i} F_{ij}$; (net force)

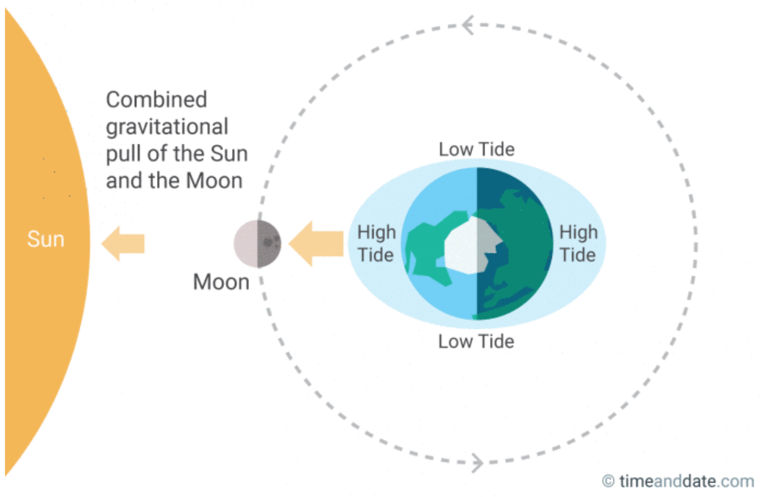
Step 3 : Adjust position of balls based on current position and net force

Relative Inductive Biases Example



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$

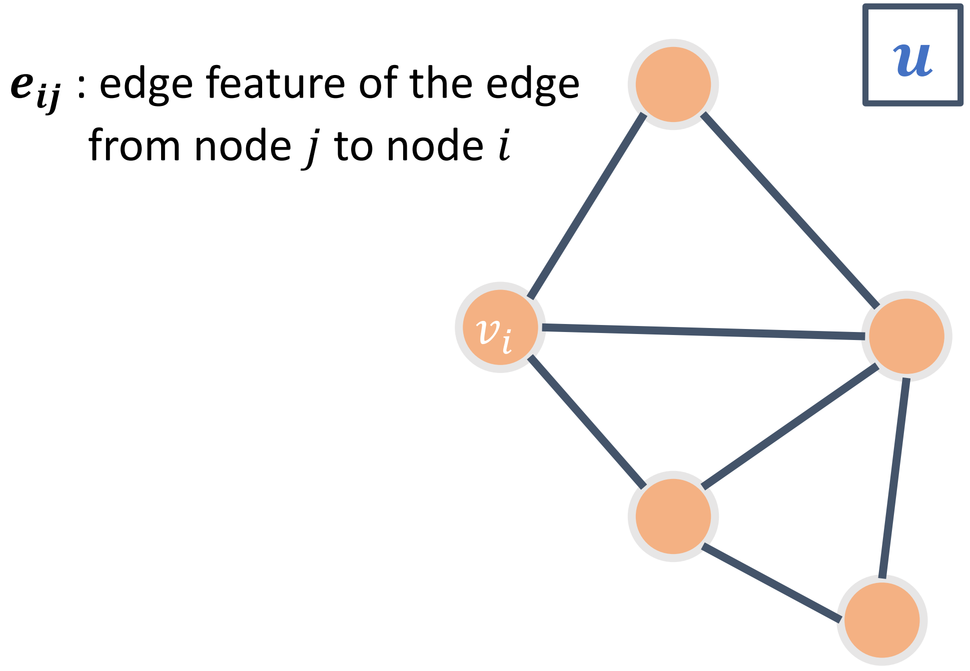




Relational inductive biases among objects

Combinatorial generalization

Graph with global feature



$\mathcal{g} = < \mathcal{V}, \mathcal{E}, u >$: A directed graph with **the global feature u**

$\mathcal{V}$ is the set of node features, $\mathcal{E}$ is the set of edge features

The global feature indicates 'graph-level' common information

Graph Network (GN) block[4]: a generalized update rule

GN Block

GN block maintains **3** differentiable, also **trainable, modules**:

- Edge updater $f_e(\cdot)$
- Node updater $f_n(\cdot)$
- Global updater $f_g(\cdot)$

Usually, MLPs

GN block also maintains **3 aggregating modules**:

- Edge aggregator for node update $\rho^{e\to v}(\cdot)$
- Edge aggregator for global feature update $\rho^{e\to u}(\cdot)$
- Node aggregator for global feature update $\rho^{v\to u}(\cdot)$

- **Edge update**

$$\boldsymbol{e}'_{ij} = \boldsymbol{f}_{\boldsymbol{e}}(\boldsymbol{e}_{ij}, \boldsymbol{v}_i, \boldsymbol{v}_j, \boldsymbol{u})$$

- **Node update**

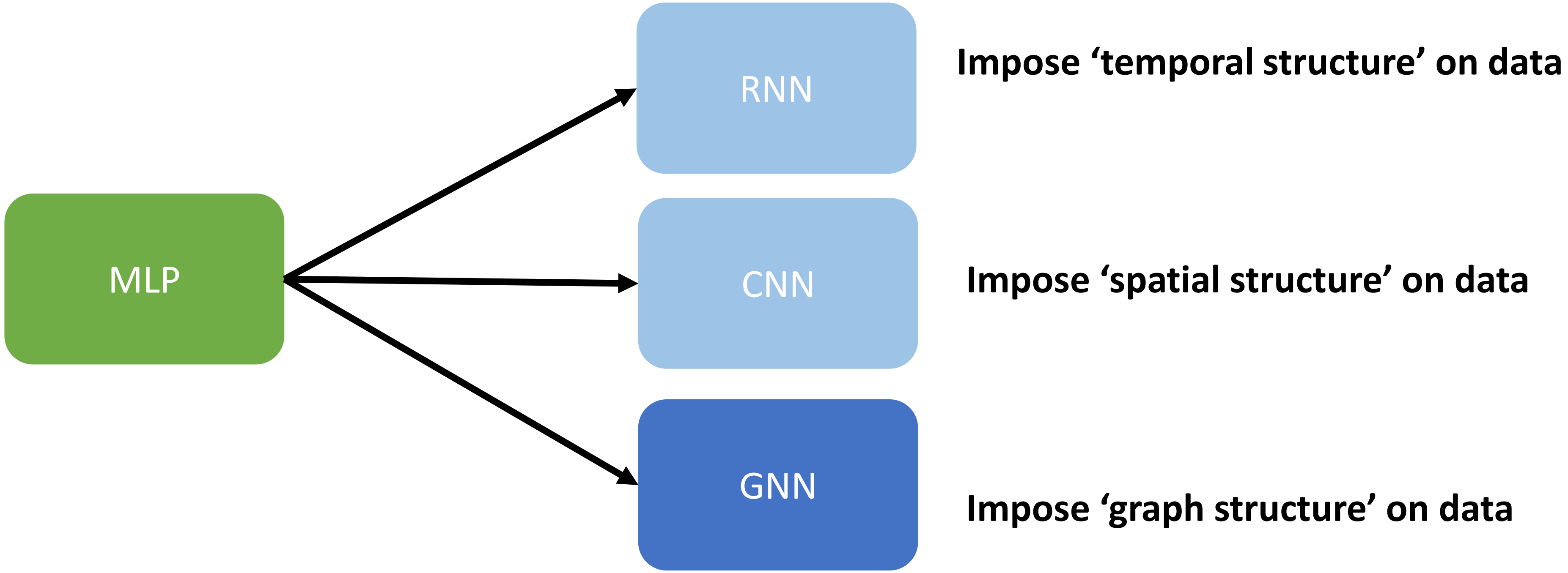
$$\boldsymbol{v}'_i = \boldsymbol{f}_{\boldsymbol{n}}\left(\boldsymbol{v}_i, \boldsymbol{u}, \boldsymbol{\rho}^{\boldsymbol{e}\to\boldsymbol{v}}\left(\left\{\boldsymbol{e}'_{ij}\right\}_{j\in\mathcal{N}_i}\right)\right)$$

- **Global feature update**

$$\boldsymbol{u}' = \boldsymbol{f}_{\boldsymbol{g}}\left(\boldsymbol{u}, \rho^{e\to u}(\mathcal{E}'), \rho^{v\to u}(\mathcal{V}')\right)$$

4. Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." arXiv preprint arXiv:1806.01261(2018).

Back to the main point

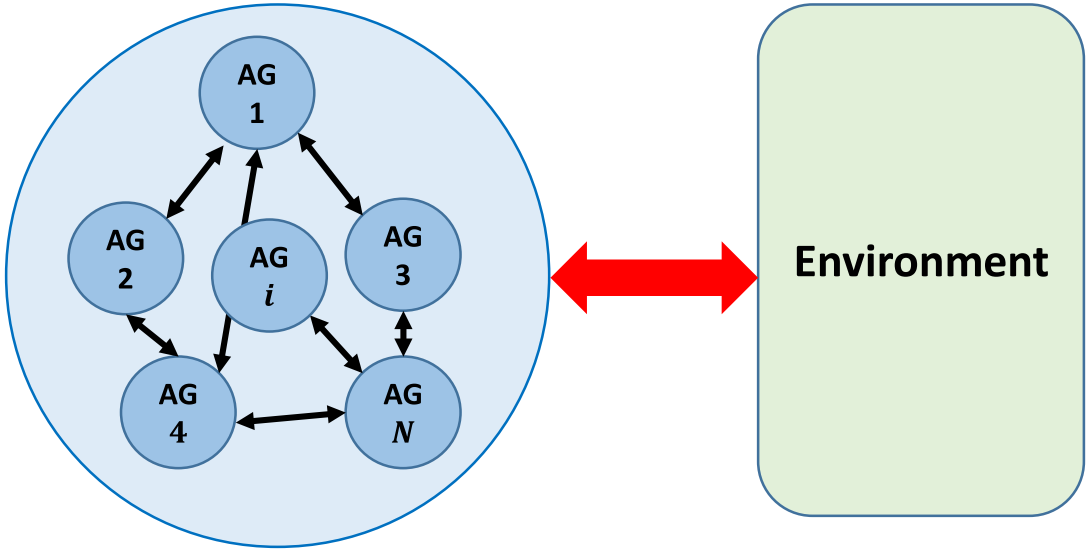


GNN, the one comprises of especially GN blocks, maybe suitable for modeling physical systems

- with "object /relation – centric" graph representation of system in supervised learning fashion.
- without prior knowledge on physics.

Why ignore prior knowledge?

Motivation for Multi-Agent Reinforcement Learning

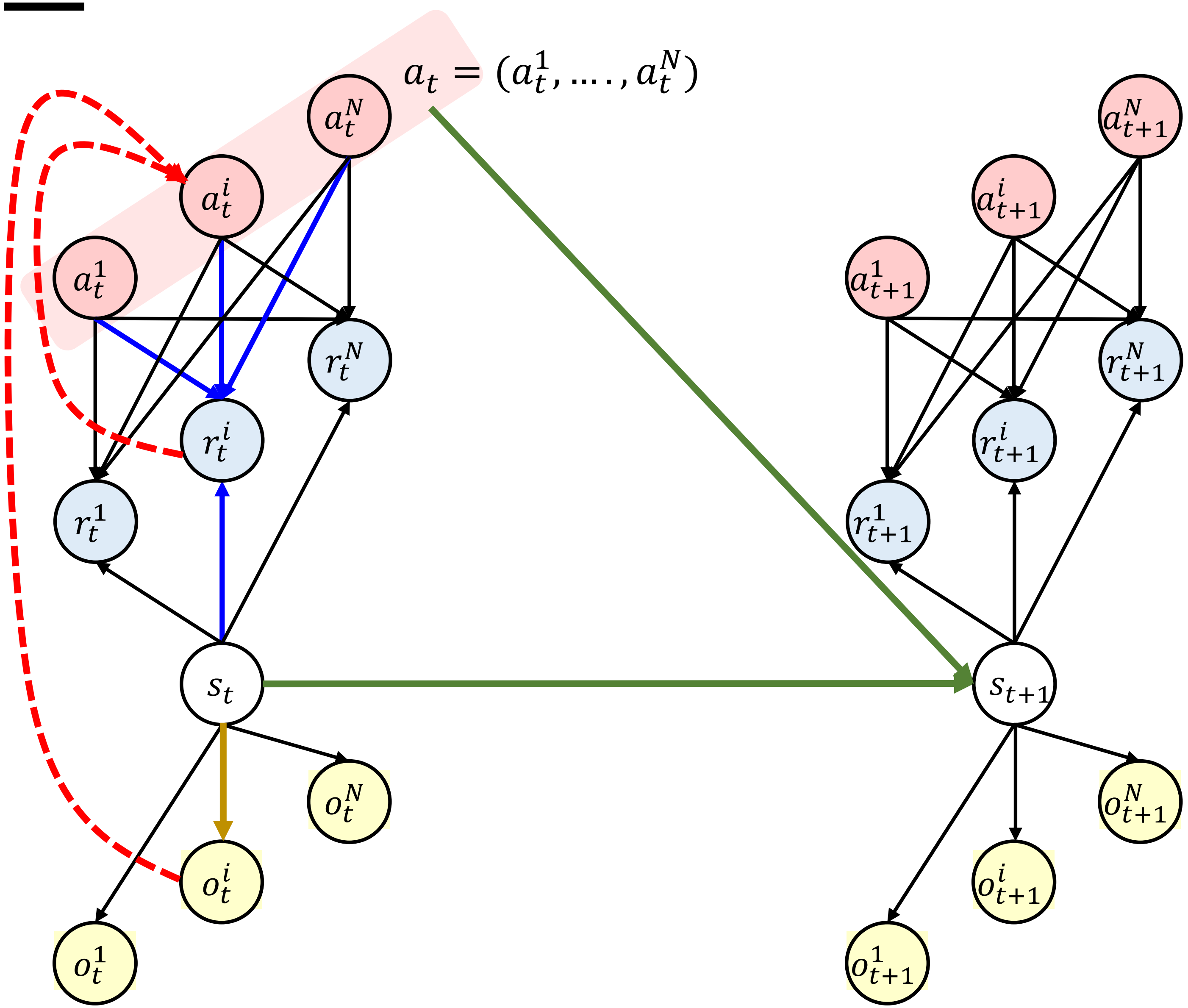


- As a system becomes larger and more complex, it become more difficult to understand and control

- Many methods to model and control multi-agent system is impractical for real world problem
  - Decentralized MDP, Decentralized-POMDP, Decentralized Control, Team Game, Cooperative Game..
  - Analytical solutions to these problems are limited to only special cases

- Recent advances in Deep Learning and Reinforcement Learning approach can open up new solution approaches

Stochastic Game



Goal: Each agent derive its policy to maximize $\sum_{t=1}^{T} \gamma^t r_i(s_t, a_t^i, a_t^{-i})$

- ***Transition:***
  $P(s_{t+1} | a_t^1, \dots, a_t^N, s_t)$
- ***Reward:***
  $r_i(s_t, a_t^1, \dots, a_t^N)$
- ***Observation:***
  $o_t^i = h^i(s_t)$
- ***Decentralized Policy:***
  $$\pi(s_t, a_t^1, \dots, a_t^N) \approx \prod_{i=1}^{N} \pi_i(s_t, a_t^i)$$
  $$\approx \prod_{i=1}^{N} \pi_i(o_t^i, a_t^i)$$

Stochastic Game

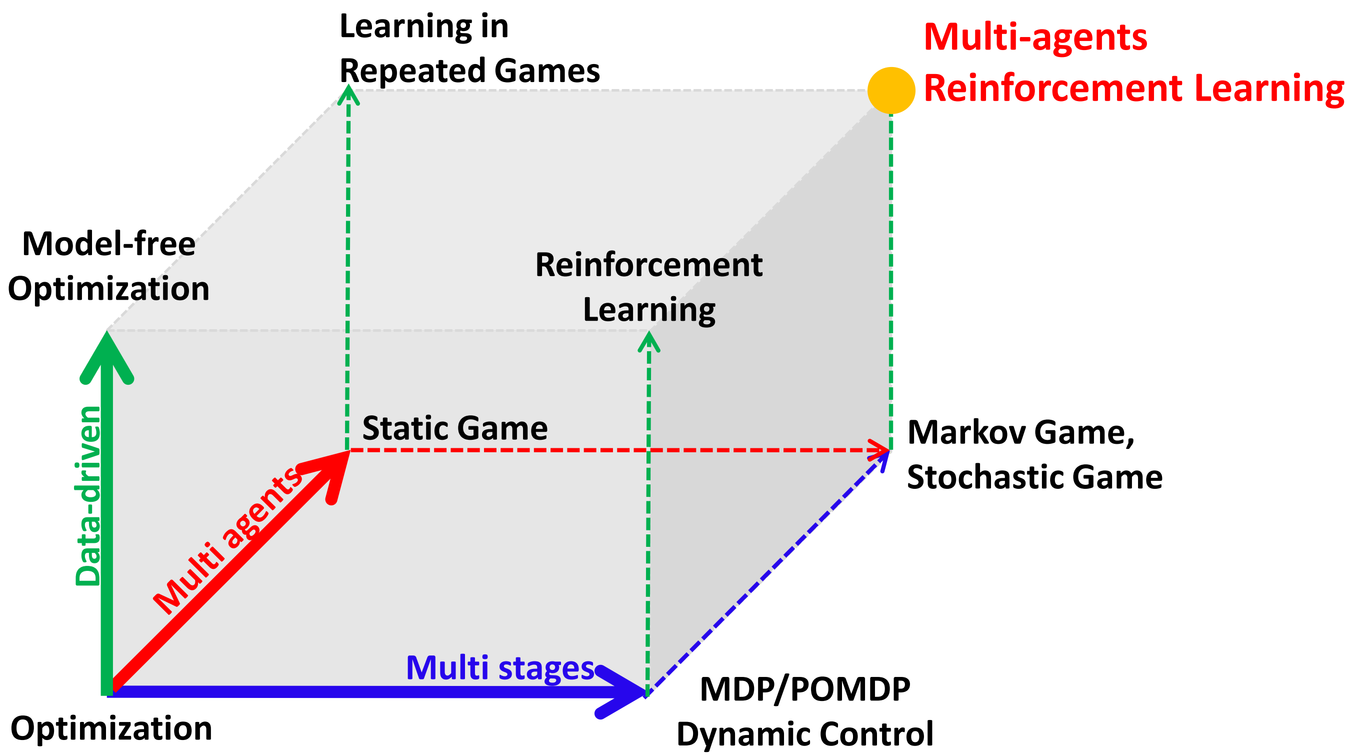
Equilibrium concept:

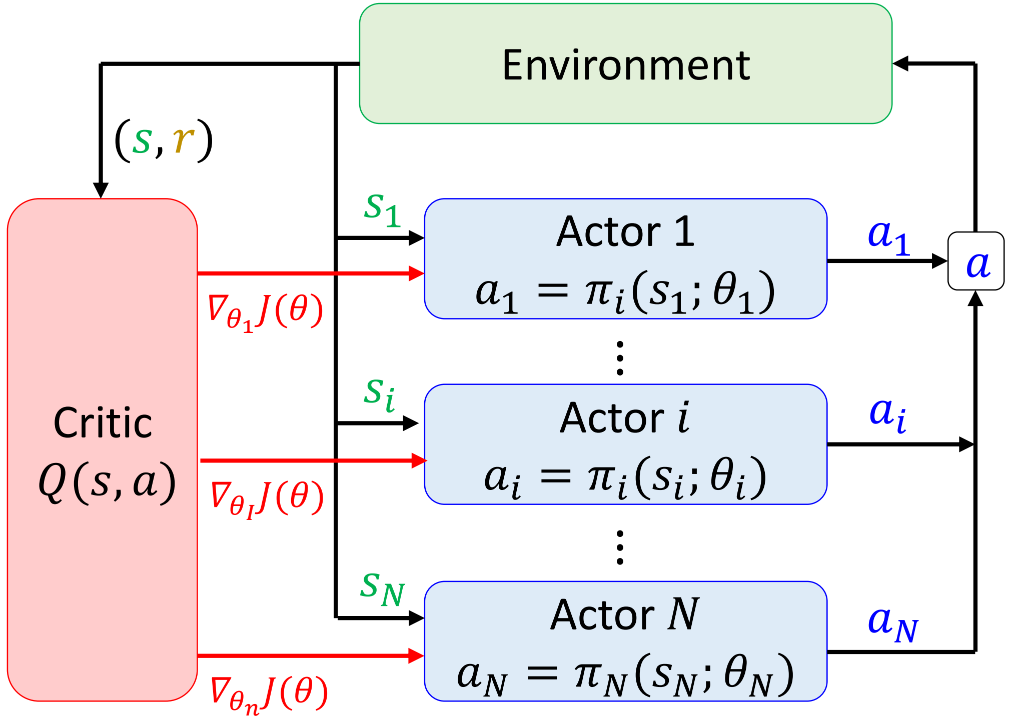
		Cooperative (Team)	Nash	Zero-sum (robust)	Stackelberg	Correlated
Information structure	Open-loop (perfect state)	Open-loop Nash-Strategy	Open-loop Nash-Strategy	Open-loop Zero-sum Strategy	…	…
	Feedback (perfect state)	**Feedback Cooperative Strategy**	Feedback Nash-Strategy	Feedback Zero-sum Strategy	…	…
	⋮	…	…	…	…	…

- We need to specify ***information structure***
  - ✓ Open-loop vs. close-loop (feedback)
  - ✓ Perfect vs. imperfect
- We need to ***equilibrium concept***
  - ✓ Cooperative (team), Nash, Zero-sum, Stackelberg, Correlated,…

Equilibrium concept + information structure → solution method

Multi-Agent Reinforcement Learning





- 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*** that can lead ***a global performance of the whole system*** (Team Game or Cooperative Game)

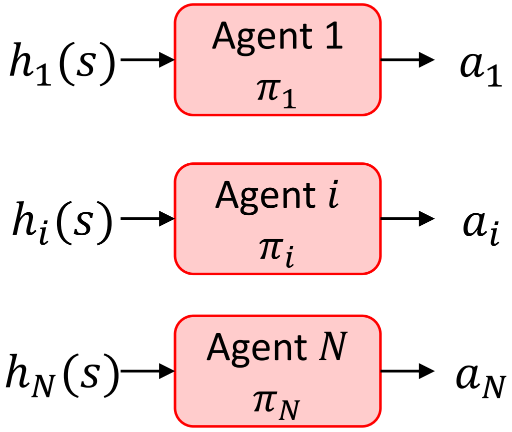
Training Principle for MARL

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

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

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

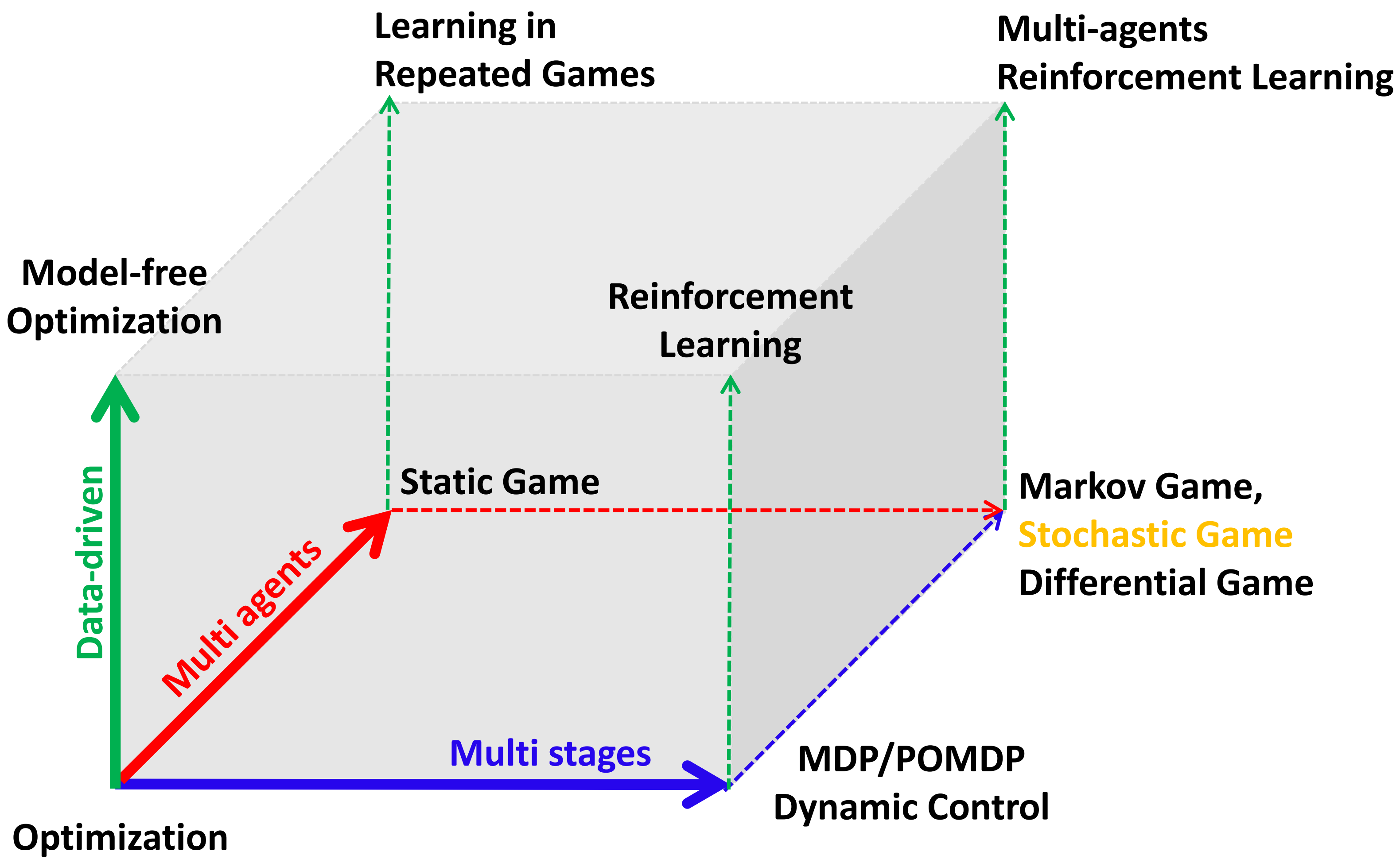
Centralized Training



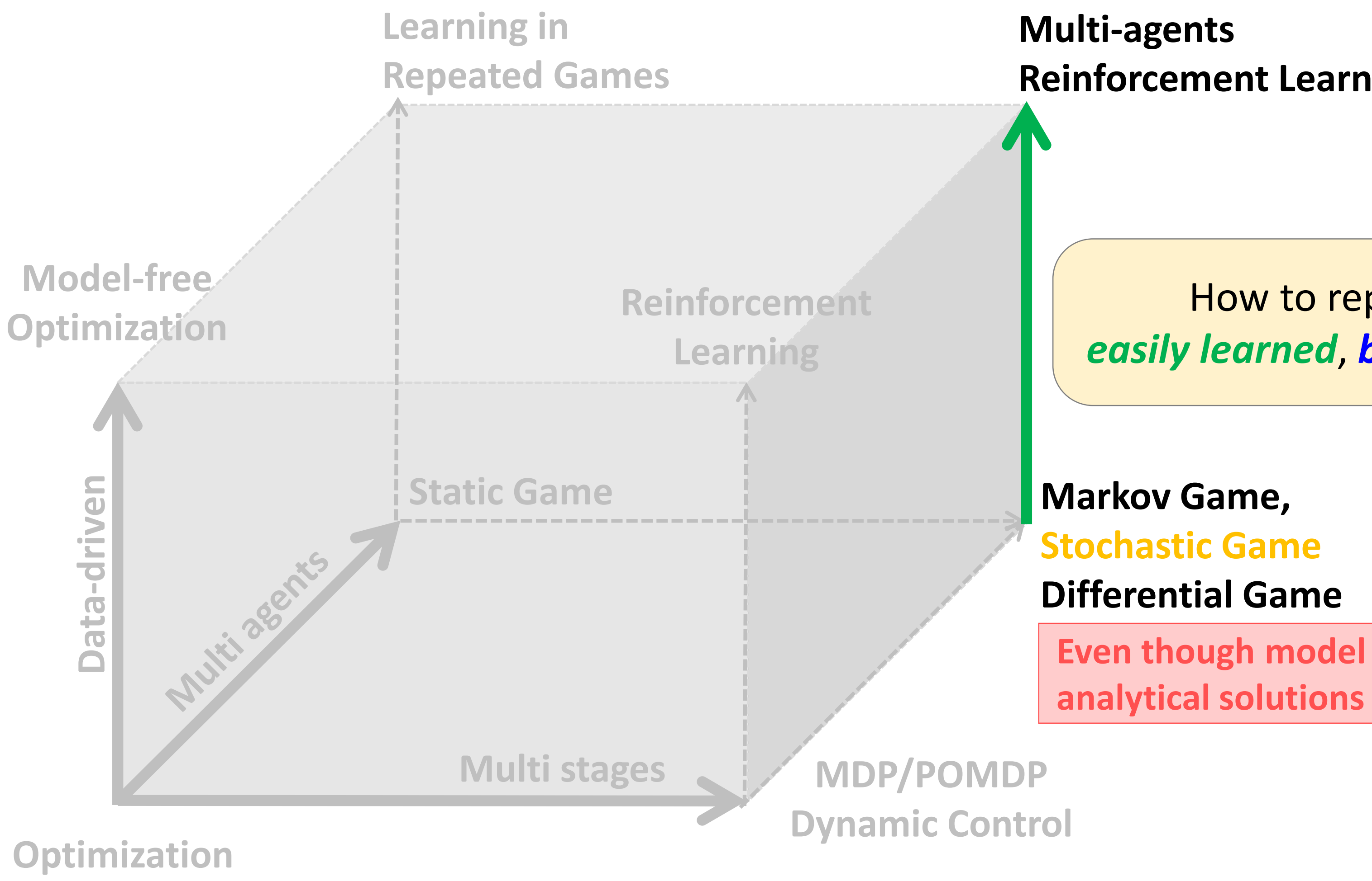
Decentralized Execution

- **Centralized Training** and **Decentralized Execution** (CTDE) is a widely adopted to overcome the non-stationarity problem when training multi-agent systems
- 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, 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)

Multi-Agent Reinforcement Learning



Multi-Agent Reinforcement Learning

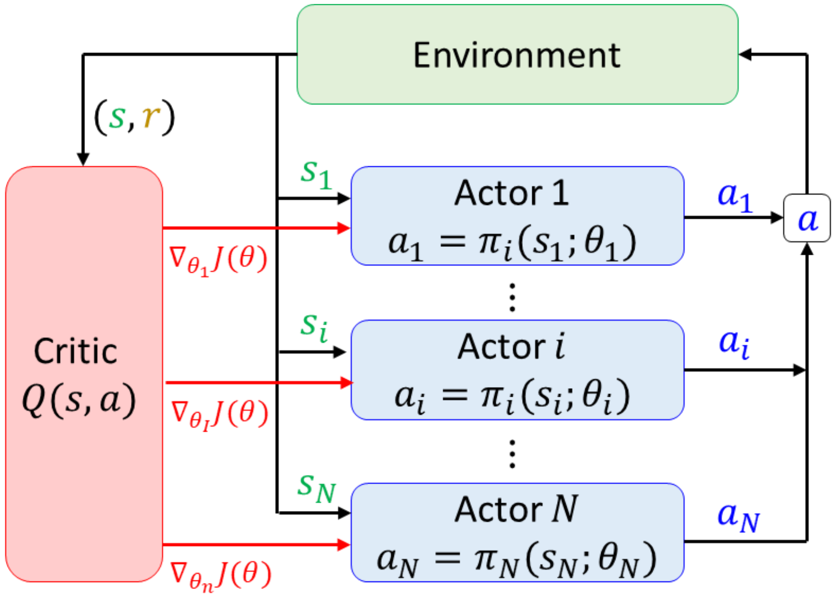


(Control/OR + Game Theory + AI)

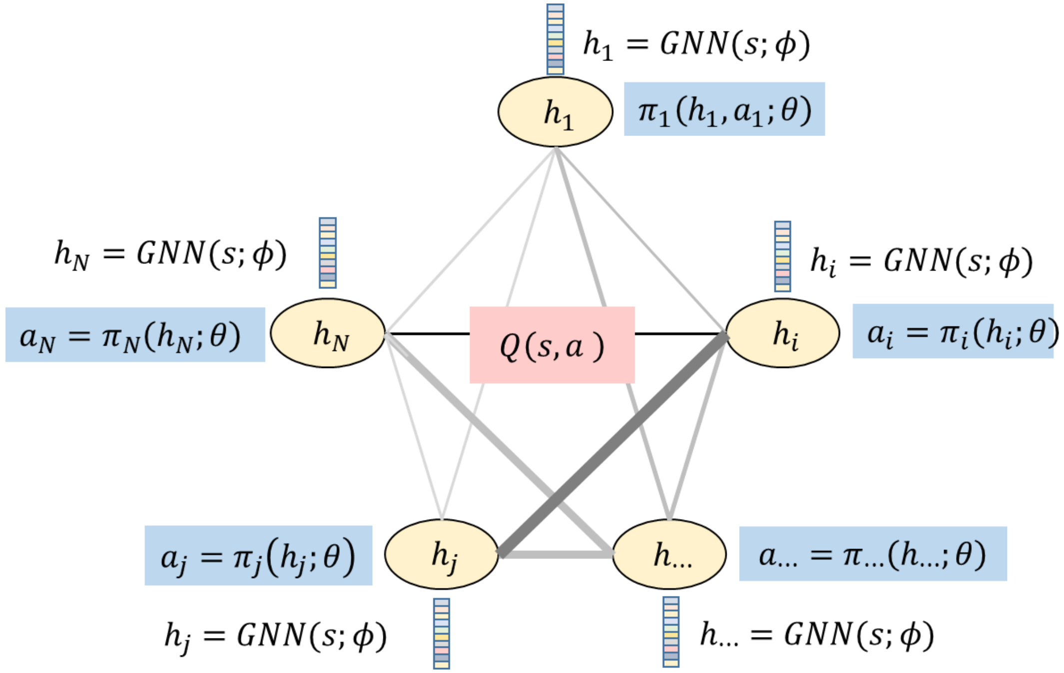
How to represent a target problem so that it can be *easily learned*, *better generalized*, and *intuitively interpreted*?

Even though model is known,
analytical solutions are only available to specific cases

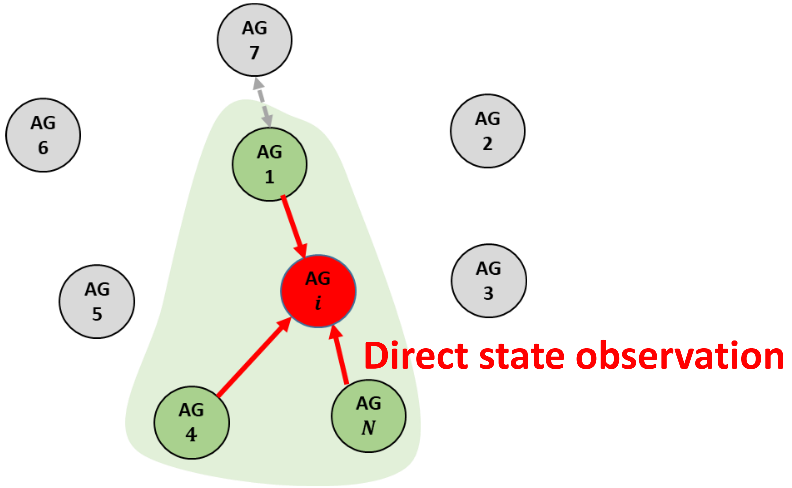
MARL + Graph Neural Network (GNN)

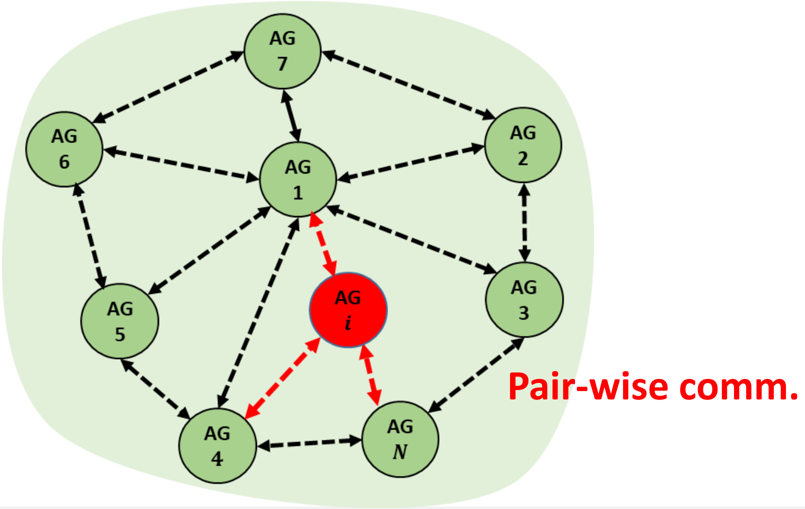


Apply relative inductive biases using graph



State localization depending on information available





Learning to *cooperate* (only local observation is allowed)

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

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

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

Parameter sharing among **a group of agents**

$$= \prod_{i=1}^{N} \pi(h_i = GNN(o_i; \phi), a_i; \theta_{g(i)})$$

(State representation with inductive biases)

Learning to *communicate* (more than local observation)

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

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

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

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

(State representation with inductive biases)

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

Parameter sharing among **a group of agents**

Learning to Cooperate Approach
(Decentralized Control)

Multiple-Actor-Attention-Critic (MAAC)

Multiple-Actor-Attention-Critic (MAAC)

- Multi-agent DDPG (MADDPG) extends DDPG to an environment where multiple agents are coordinating to complete tasks with only local information.

- In the viewpoint of one agent, the environment is non-stationary as policies of other agents are quickly upgraded and remain unknown.
  - MADDPG is an actor-critic model redesigned particularly for handling such a non-stationary environment and interactions between agents.

- The problem can be formalized in the multi-agent version of MDP, also known as Markov games

$\mathcal{N}$: set of agents
$\mathcal{S}$: set of joint states
$\mathcal{A}_i$: set of possible action for agent i; $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$ set of joint action
$\mathcal{O}_i$: set of observation for agent i; $\mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$: set of joint observation
$\mathcal{T}: \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \longmapsto \mathcal{S}$ transition function
$\pi_{\theta_i}: \mathcal{O}_i \times \mathcal{A}_i \longmapsto [0,1]$ stochastic policy of agent i
$\mu_{\theta_i}: \mathcal{O}_i \longmapsto \mathcal{A}_i$ deterministic policy of agent i

Multiple-Actor-Attention-Critic (MAAC)

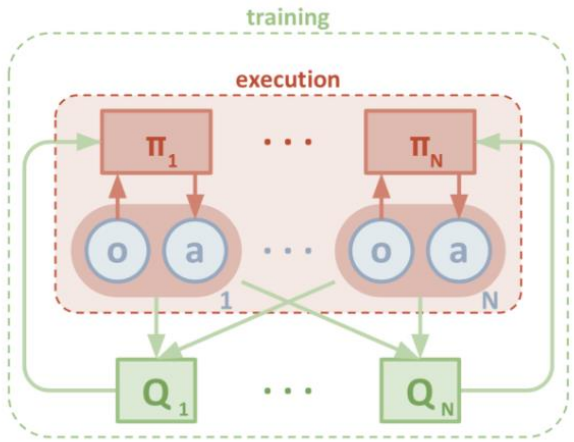
- The critic in MADDPG learns a centralized action-value function for agent i

$$Q_i^{\vec{\mu}}(\vec{o}, a_1, \dots, a_N)$$

  - ✓ $\vec{\mu} = \mu_1, \dots, \mu_N$:joint policies
  - ✓ $\vec{o} = o_1, \dots, o_N$:joint observations
  - ✓ $\vec{a} = a_1, \dots, a_N$:joint actions

- Each $Q_i^{\vec{\mu}}$ is **learned separately** for $i = 1, \dots, N$ and therefore **multiple agents can have arbitrary reward structures**, including conflicting rewards in a competitive setting.

- Meanwhile multiple actors, one for each agent, are exploring and upgrading the policy parameters θ_i on their own.



Multiple-Actor-Attention-Critic (MAAC)

- **Actor update:**

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{(\vec{o},a)\sim D}\left[\nabla_{a_i} Q_i^{\vec{\mu}}(\vec{o}, a_1, \ldots, a_N) \nabla_{\theta_i} \mu_{\theta_i}(o_i)\Big|_{a_i=\mu_{\theta_i}(o_i)}\right]$$

  ✓ D is the memory buffer for experience reply, containing multiple episode samples $(\vec{o}, a_1, \ldots, a_N, r_1, \ldots, r_N, \vec{o}')$: given current observation $\vec{o}$, agents take joint actions $a_1, \ldots, a_N$ and get rewards $r_1, \ldots, r_N$, leading to the new observation $\vec{o}'$

- **Critic update:**

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\vec{o},a_1,\ldots,a_N,r_1,\ldots,r_N,\vec{o}')\sim D}\left[\left(Q_i^{\vec{\mu}}(\vec{o}, a_1, \ldots, a_N) - y\right)^2\right]$$

$$\text{where } y = r_i + \gamma\, Q_i^{\vec{\mu}'}(\vec{o}', a_1', \ldots, a_N')\Big|_{a_i'=\mu_{\theta_i}'(o_i)}$$

  ✓ $\vec{\mu}'$ are the target policies with delayed softly-updated parameters.
  ✓ For each agent need to have $\vec{\mu} = \mu_1, \ldots, \mu_N$ to compute the joint actions at the next observation $\vec{o}'$
  ✓ Each agent learns other agents policy its own during the learning process.
  ✓ Using the approximated policies, MADDPG still can learn efficiently although the inferred policies might not be accurate.

Multiple-Actor-Attention-Critic (MAAC)

- To mitigate the high variance triggered by the interaction between competing or collaborating agents in the environment, MADDPG proposed one more element - policy ensembles:

  ✓ Train K policies for one agent;
  ✓ Pick a random policy for episode rollouts;
  ✓ Take an ensemble of these K policies to do gradient update.

- In summary, MADDPG added three additional ingredients on top of DDPG to make it adapt to the multi-agent environment:

  ✓ Centralized critic + decentralized actors;
  ✓ Actors are able to use estimated policies of other agents for learning;
  ✓ Policy ensembling is good for reducing variance.

Multiple-Actor-Attention-Critic (MAAC)

Attention mechanism: each agent queries the other agents for information about their observations and actions, and estimate individual value function

- Calculate **centralized critic**, $Q_i^{\psi}(o,a)$ for all i

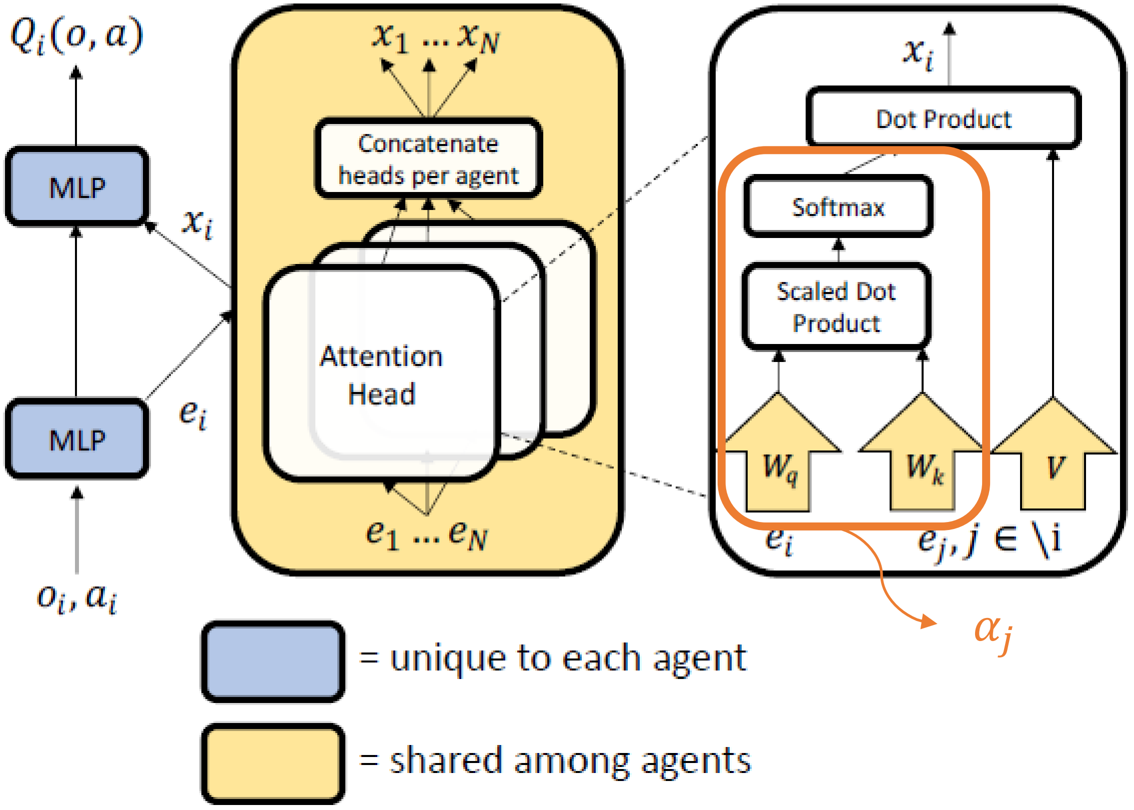
$$Q_i^{\psi}(o,a) = f_i(e_i, x_i)$$

  - Embedding $e_i = g_i(o_i, a_i)$
  - Contribution from other agents x_i,

$$x_i = \sum_{j \neq i} \alpha_j v_j = \sum_{i \neq j} \alpha_j h\left(V e_j\right)$$
$$\alpha_j \propto \exp\left(e_j^T W_k^T W_q e_i\right)$$

  - $\left(W_k, W_q, V\right)$ shared among agents
  - W_q transforms e_i into "query", W_k transforms e_j into "key"

- Use multiple attention head[1]:
  - Concatenate multiple attention head to ensemble learning



Multiple-Actor-Attention-Critic (MAAC)

Attentive centralized critics (CT)

- All critics are updated together to minimize a joint regression loss, due to the parameter sharing
- Q_i receives <u>observations and actions for all agents</u>

$$\mathcal{L}_Q(\psi) = \sum_{i=1}^{N} \mathbb{E}_{(o,a,r,o')\sim D}\left[(Q_i^{\psi}(o,a) - y_i)^2\right], \text{where}$$

$$y_i = r_i + \gamma \mathbb{E}_{a'\sim\pi_{\bar{\theta}}(o')}\left[Q_i^{\bar{\psi}}(o',a') - \alpha\ \log(\pi_{\bar{\theta}_i}(a_i'|o_i'))\right]$$

  - $\bar{\psi}, \bar{\theta}$ are parameters of target critics and policies respectively

Updating Individual actor (DE)

$$\nabla_{\theta_i} J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_{\theta_i}\log(\pi_{\theta_i}(a_i|o_i))(\alpha\ \log(\pi_{\theta_i}(a_i|o_i)) - Q_i^{\psi}(o,a) + b(o,a_{\setminus i}))\right]$$

- Multi-agent advantage function

$$A_i(o,a) = Q_i^{\psi}(o,a) - b(o,a_{\setminus i})), \text{where}$$

$$b(o,a_{\setminus i})) = \mathbb{E}_{a_i\sim\pi_i(o_i)}\left[Q_i^{\psi}(o,(a_i,a_{\setminus i}))\right]$$

$$\mathbb{E}_{a_i\sim\pi_i(o_i)}\left[Q_i^{\psi}(o,(a_i,a_{\setminus i}))\right] = \sum_{a_i'\in A_i} \pi(a_i'|o_i) Q_i(o,(a_i',a_{\setminus i}))$$

Multiple-Actor-Attention-Critic (MAAC)

Table 1: Comparison of various methods for multi-agent RL

	Base Algorithm	Attention	Centralized Critic(s)	Number of Critics	Multi-task Learning of Critics	Multi-Agent Advantage
MAAC (ours)	SAC	✓	✓	N	✓	✓
MAAC (Uniform) (ours)	SAC	uniform	✓	N	✓	✓
COMA*	Actor-Critic (On-Policy)		✓	1		✓
MADDPG†	DDPG		✓	N		
COMA+SAC	SAC		✓	1		✓
MADDPG+SAC	SAC		✓	N		✓
DDPG‡	DDPG			N	N/A	N/A

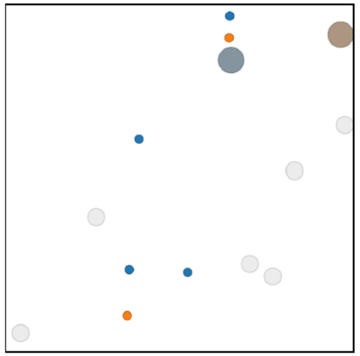
Centralized Critic(s): each agent's estimate of Q_i takes the actions and observations of the other agents into account. *Number of Critics*: number of separate networks used for predicting Q_i for all N agents. *Multi-task Learning of Critics*: all agents' estimates of Q_i share information in intermediate layers, benefiting from multi-task learning. *Multi-Agent Advantage*: cf. Sec 3.2 for details. * (Foerster et al., 2017a), † (Lowe et al., 2017), ‡ (Lillicrap et al., 2015)

- To learn discrete action space for MADDPG and DDPG, use Gumbel-Softmax reparametrizon (Jang et al., 2016) + soft actor critic
- Uniform MAAC: $\alpha_j = \frac{1}{N+1}$ for all j

Multiple-Actor-Attention-Critic (MAAC)

Impact of Rewards and Required Attention

- Information relevant to reward can dynamically change during an episode



(a) Cooperative Treasure Collection. The small grey agents are "hunters" who collect the colored treasure, and deposit them with the correctly colored large "bank" agents.

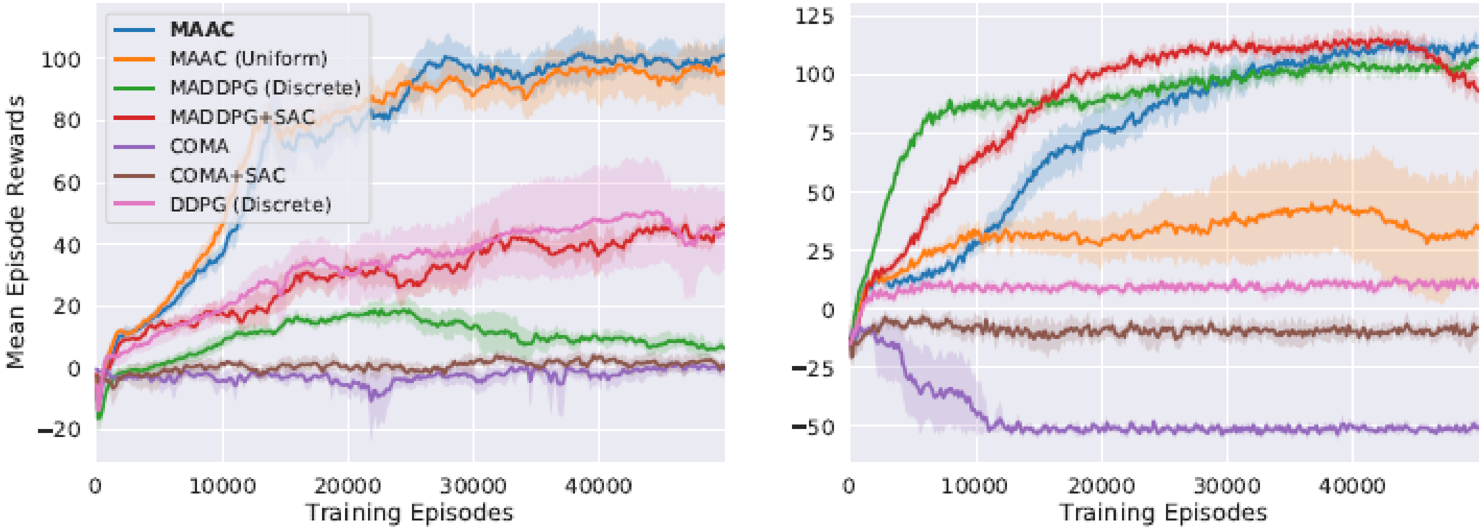
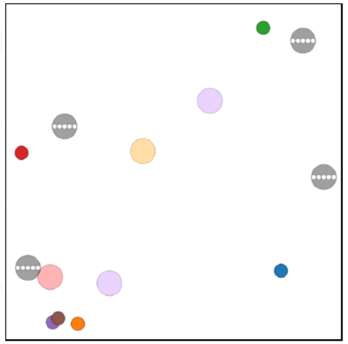


Figure 3: (Left) Average Rewards on Cooperative Treasure Collection. (Right) Average Rewards on Rover-Tower. Our model (MAAC) is competitive in both environments. Error bars are a 95% confidence interval across 6 runs.



(b) Rover-Tower. Each grey "Tower" is paired with a "Rover" and a destination (color of rover corresponds to its destination). Their goal is to communicate with the "Rover" such that it moves toward the destination.

- Uniform attention is competitive with CTC, but not in RT
- COMA uses a single centralized network for predicting Q-values for all agents. Thus, COMA perform only environments with global rewards and agents with similar action spaces.
- MADDPG performs low in CTC due to environment's relatively large observation space for all agents

Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

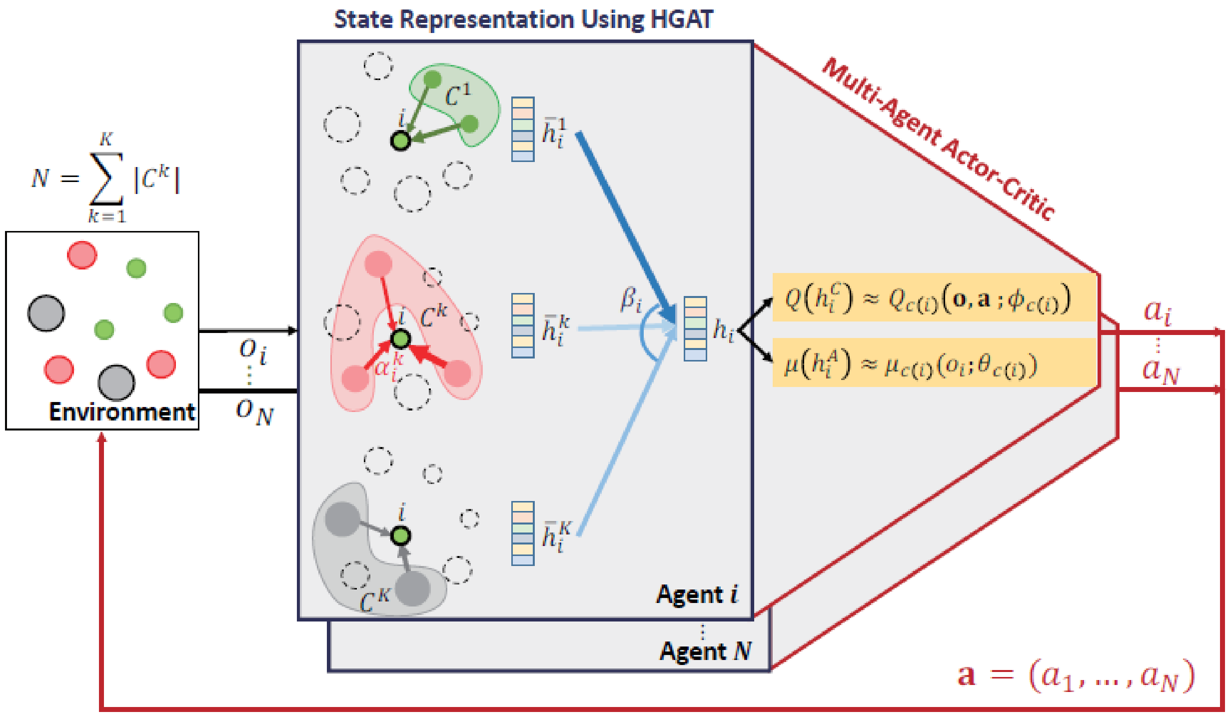
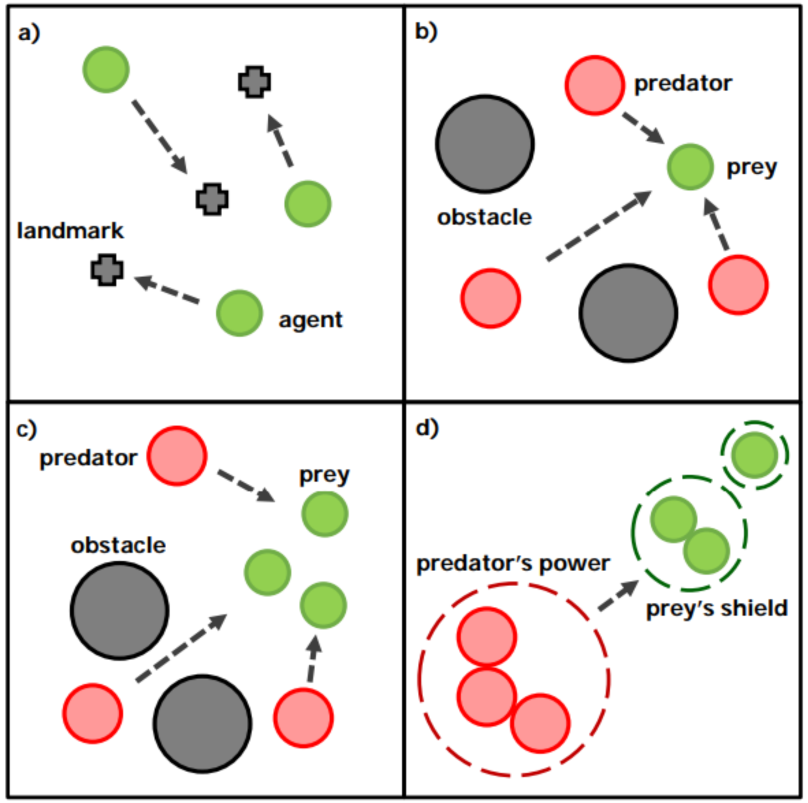


Figure 1: Overview of HAMA



Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

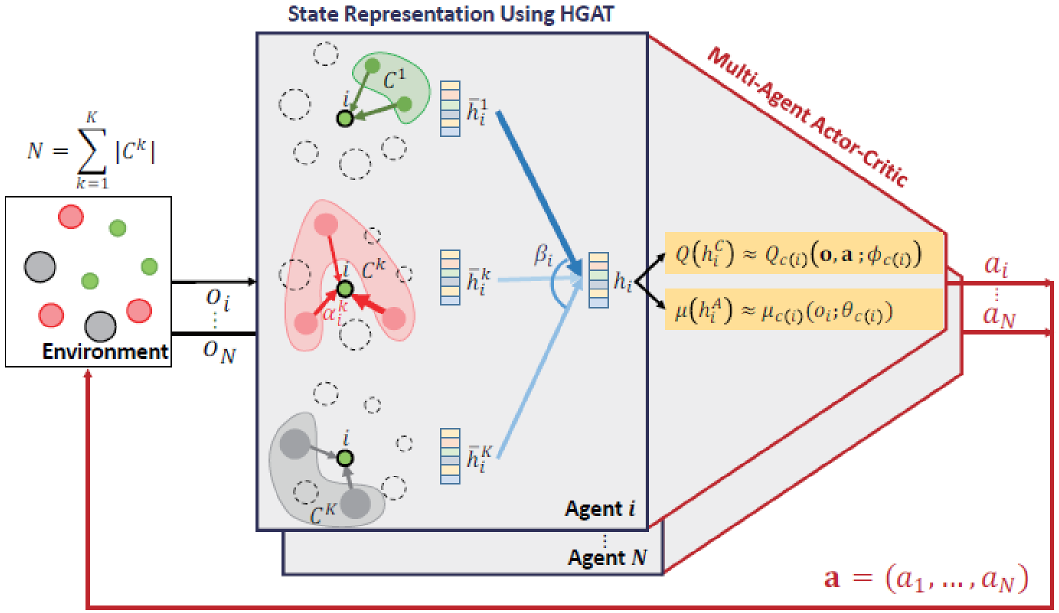


Figure 1: Overview of HAMA

Agent Clustering. The first step in representation learning is to cluster all the agents into distinct groups C^k using prior knowledge or data. For pure cooperative tasks, all the agents can be categorized into a single group. If the target task involves competition between two groups, we can cluster the agents into two groups. In addition, we can cluster into a group the agents that do not execute any actions but participate in the game (i.e., terrain components or obstacles). In this study, we assume that the agents can be easily clustered into K groups using prior knowledge on the agents, which implies that HAMA utilizes enhanced relative inductive biases regarding the group relationships.

Node-Embedding Using GAT in Each Cluster. Agent i has the local observation $o_i = \{s_j | \, j \in V(i)\}$ where s_j is the local state of agent j, and $V(i)$ specifies the visual range of agent i. The visual range can be specified depending on environment settings so that agent i can observe the agents within a certain distance. Thus, our agent can observe nearby agents as a partial observation. Agent i computes the different node-embedding vectors $\bar{h}_i^k$ for different groups $k = 1, ..., K$ to summarize the individual relationships between agent i and agents from different groups. To compute $\bar{h}_i^k$, agent i first computes embedding $h_{ij}^k = f_M^k(s_i, s_j; w_M^k)$ between itself and agents in $j \in C^k \cap V(i)$ and computes the aggregated embedding $\bar{h}_i^k = \sum_{j \in C^k \cap V(i)} \alpha_{ij}^k h_{ij}^k$. The inter-agent attention weight α_{ij}^k quantifies the importance of the embedding h_{ij}^k from agent j to agent i. The inter-agent attention weight is computed as softmax $\alpha_{i,\cdot}^k \propto \exp(e_{i,\cdot}^k)$ where $e_{ij}^k = f_\alpha^k(s_i, s_j; w_\alpha^k)$. The attention can be

Hierarchical State Representation Using Multi-Graph Attention. This step aggregates the group-level node-embedding vectors $\bar{h}_i^1, ..., \bar{h}_i^K$ of agent i for the information-condensed and contextualized state representation of agent i as $h_i = \sum_{k=1}^K \beta_i^k \bar{h}_i^k$ while considering the relationships between agent i and the groups of other agents. The inter-group attention weight β_i^k guides which group agent i should focus more on to achieve its objective. For example, if β_i^k is large for the same group which agent i belongs to, it implies that agent i focuses on cooperating with the agents in the same group. Otherwise, agent i would focus more on competing with agents from different groups. The inter-group attention weight is computed as softmax $\beta_i = (\beta_i^1, ..., \beta_i^K) \propto \exp(q_i)$ where $q_i = [q_i^1, \dots, q_i^K] = f_\beta([\bar{h}_i^1, \dots, \bar{h}_i^K]; w_\beta)$. The hierarchical state representation is particularly useful when considering mixed cooperative-competitive games where each agent or group possesses their own objectives, which will be empirically shown by various experimental results in this study. The embedding and attention functions in this study comprise a two-layered MLP with 256 units and ReLUs.

Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

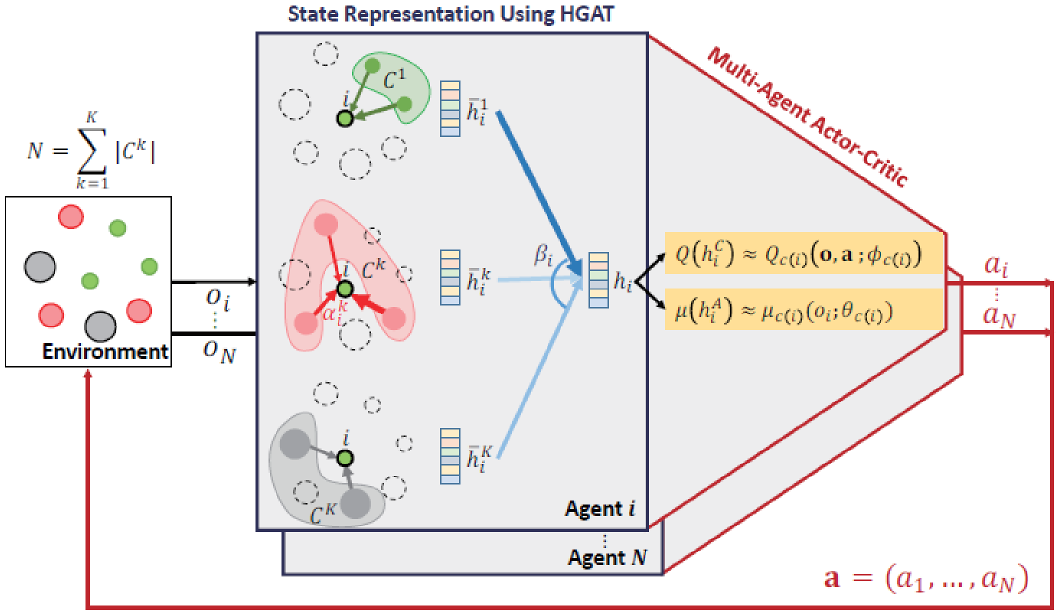


Figure 1: Overview of HAMA

The proposed method uses the embedding vectors h_i^C and h_i^A of agent i to compute, respectively, the individual action-value $Q_{c(i)}(\mathbf{o}, \mathbf{a}) \approx Q_{c(i)}(h_i^C; \phi_{c(i)})$ and determine the action $a_i = \mu_{c(i)}(o_i) \approx \mu_{c(i)}(h_i^A; \theta_{c(i)})$, where $c(i)$ is the group to which agent i belongs. Note that the embedding vectors h_i^C and h_i^A are computed separately using two different HAGTs; computing h_i^C requires a joint action $\mathbf{a}$ in the training phase under CTDE. Additionally, agents in the same group share the actor and critic networks for generalization.

The training of HAMA is similar to that of MADDPG. The shared critic Q_k for agent i in group k is trained to minimize the loss $\mathcal{L}$:

$$\mathcal{L}(\phi_k) = \mathbb{E}_{\mathbf{o},\mathbf{a},r_i,\mathbf{o}'\sim\mathcal{D}}[(Q_k^\mu(\mathbf{o}, \mathbf{a}; \phi_k) - y_i)^2], y_i = r_i + \gamma Q_k^{\mu'}(\mathbf{o}', \mathbf{a}'; \phi_k')|_{a_i'=\mu'(o_i';\theta')}$$

where $Q^{\mu'}$ and μ' are, respectively, the target critic and actor networks for stable learning with delayed parameters, which is called *soft target* [13]. In CTDE framework, the joint observation and action are assumed to be available for training. The shared actor μ_k for agent i in group k is then trained using gradient ascent algorithm with the gradient computed as:

$$\nabla_{\theta_k}\mathcal{J}(\theta_k) = \mathbb{E}_{\mathbf{o},\mathbf{a}\sim\mathcal{D}}[\nabla_{\theta_k}\mu_k(o_i;\theta_k)\nabla_{a_i}Q_k^\mu(\mathbf{o}, \mathbf{a}; \phi_k)|_{a_i=\mu_k(o_i;\theta_k)}]$$

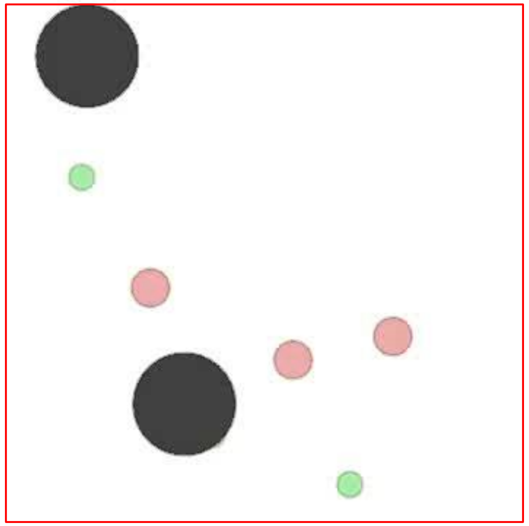
where a_i is the action of agent i in $\mathbf{a}$. During the training, the joint observation $\mathbf{o}$ and joint action $\mathbf{a}$ are used, whereas during the execution, only the learned policy $\mu_k(o_i; \theta_k) \approx \mu_{c(i)}(h_i^A; \theta_{c(i)})$ is used with the embedding vector h_i^A computed using only local observation o_i of agent i.

Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

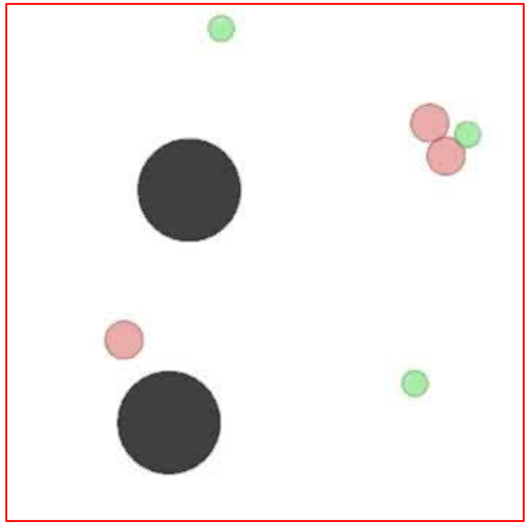
Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

Heuristic vs MADDPG
1 agent independently tries to catches the nearest one



HAMA vs MADDPG

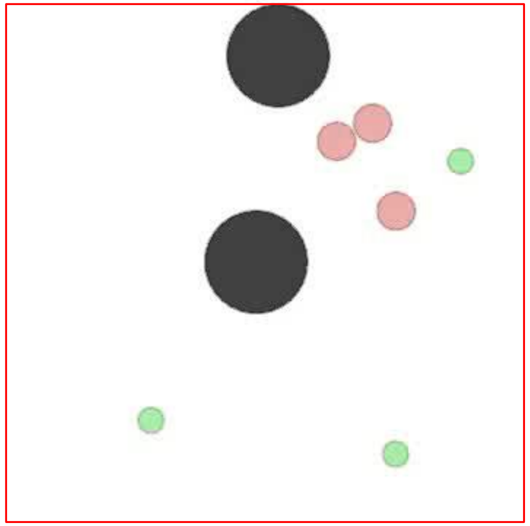


Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

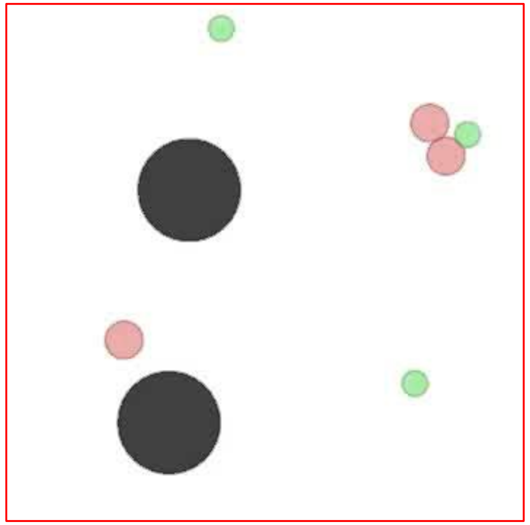
Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

Heuristic vs MADDPG

3 agent together tries to catches the nearest one

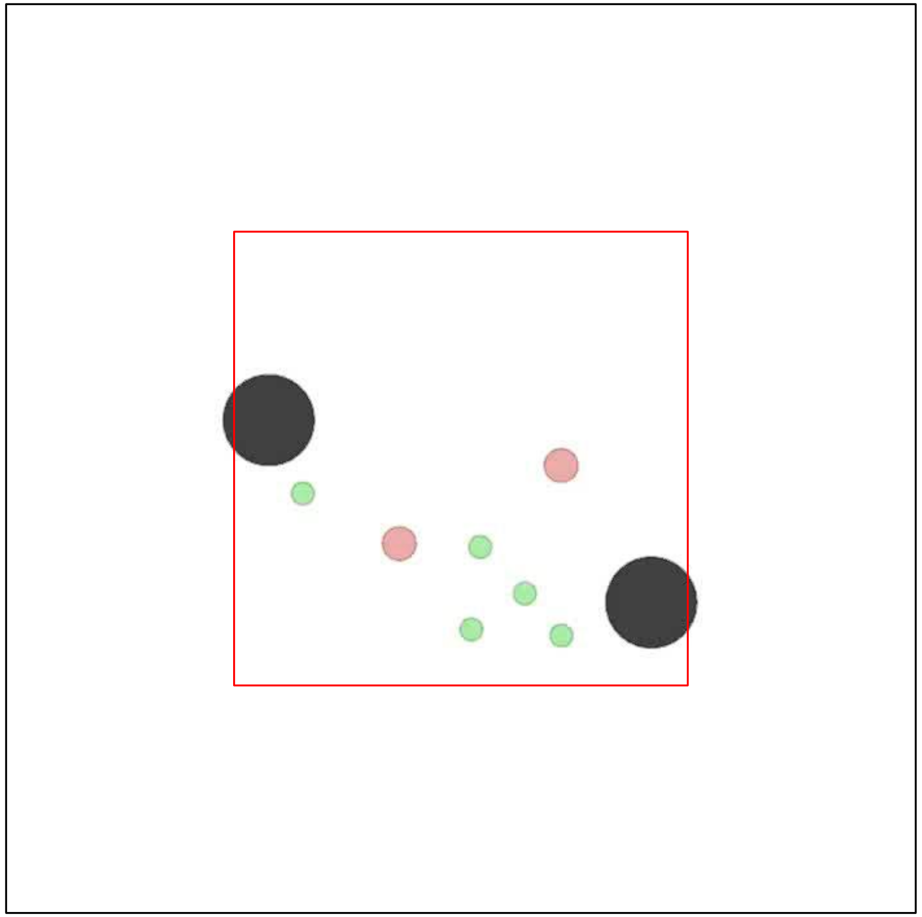


HAMA vs MADDPG

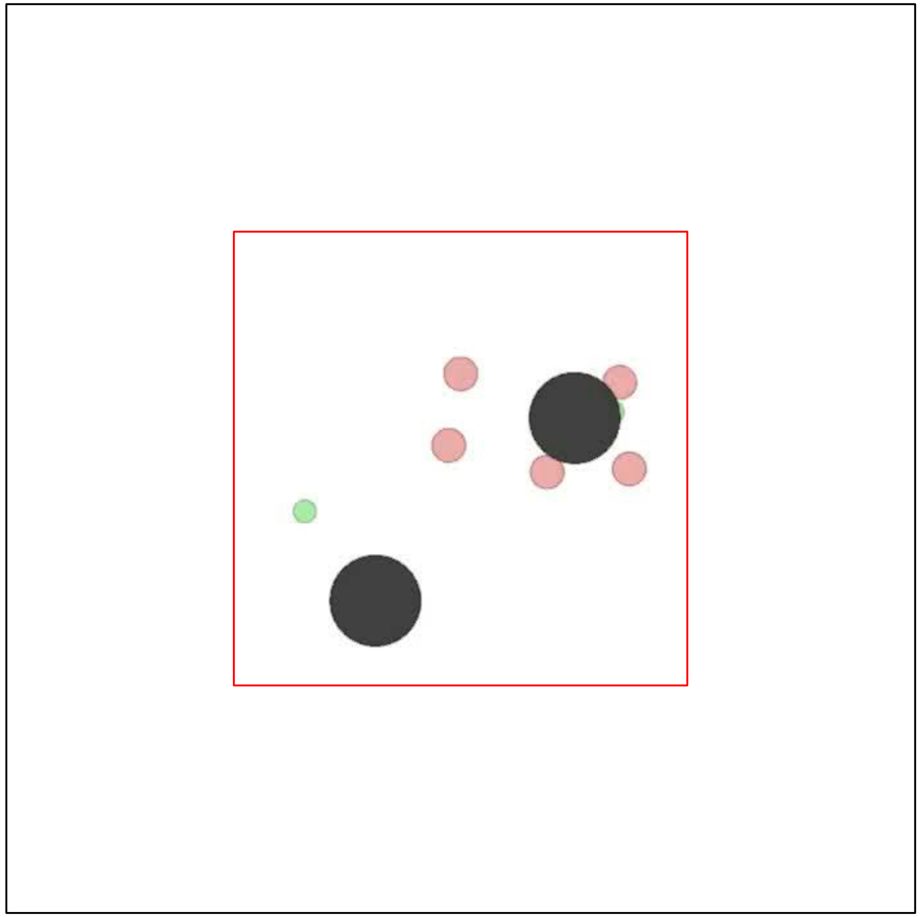


Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)



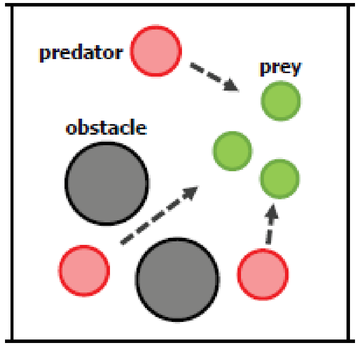
2:5

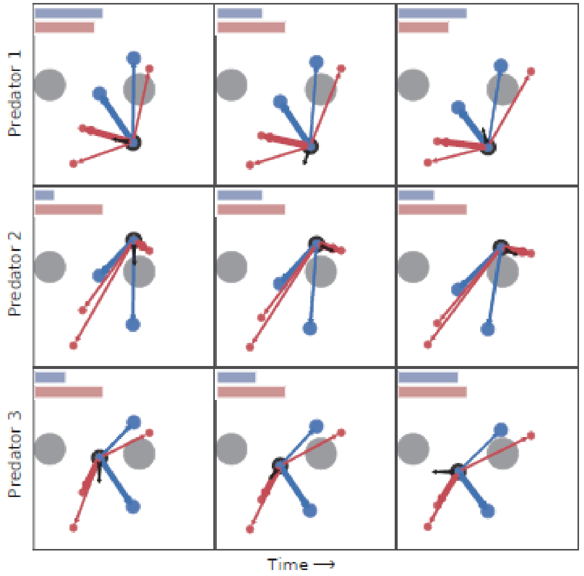


5:2

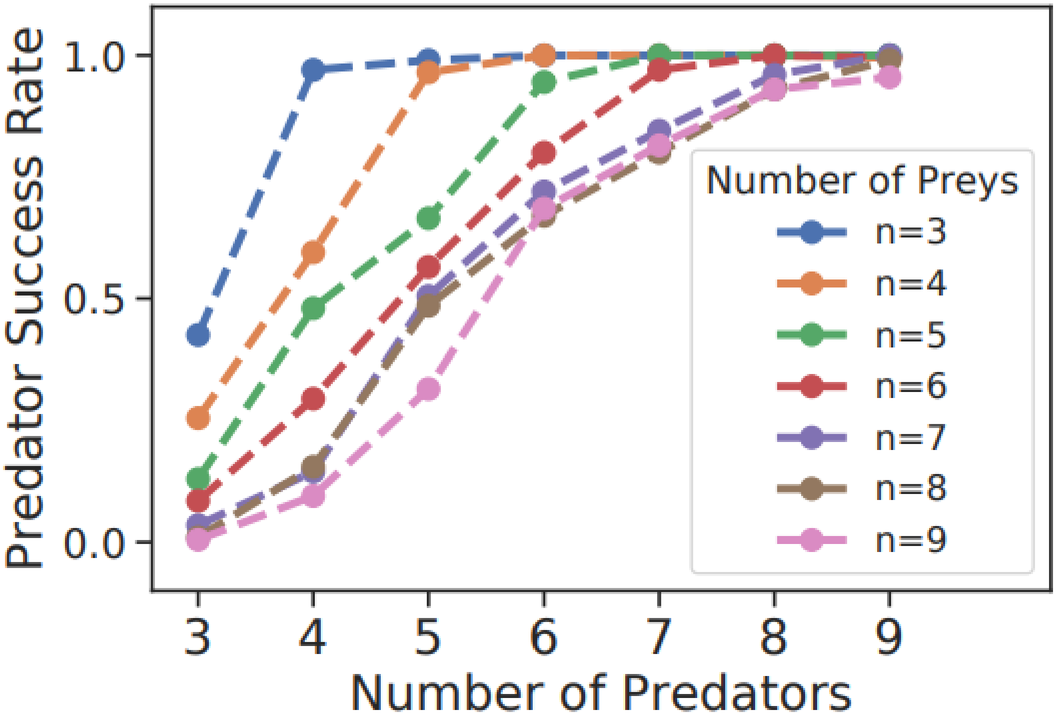
Multi-Agent Actor-Critic with Hierarchical Graph Attention Network

Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)





		prey ($n = 3$)		
		MADDPG	ATOC	HAMA
predator ($n = 3$)	Heuristic1	0.35	0.88	0.005
	Heuristic2	0.72	0.85	0.01
	MADDPG	1.18	1.24	0.02
	ATOC	0.50	0.16	0.02
	HAMA	**6.33**	**5.32**	**1.19**



Heechang Ryu, Hayong Shin, Jinkyoo Park (AAAI 2020)

Factorized Approach

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

Value-Decomposition Networks For Cooperative Multi-Agent Learning (VDN)

- Independent DQN-style agents
- Value-decomposition

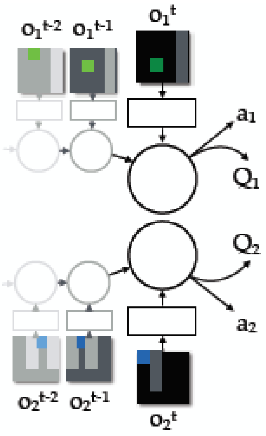


Figure 1: Independent agents architecture showing how local observations enter the networks of two agents over time (three steps shown), pass through the low-level linear layer to the recurrent layer, and then a dueling layer produces individual Q-values.

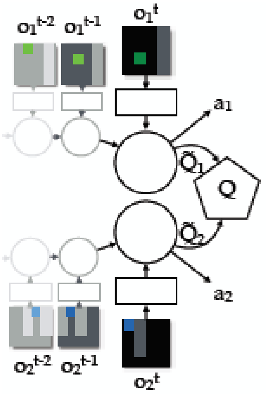


Figure 2: Value-decomposition individual architecture showing how local observations enter the networks of two agents over time (three steps shown), pass through the low-level linear layer to the recurrent layer, and then a dueling layer produces individual "values" that are summed to a joint Q-function for training, while actions are produced independently from the individual outputs.

- Main assumption : the joint action-value function for the system can be additively decomposed into value functions across agents,

$$Q((h^1, h^2, ..., h^d), (a^1, a^2, ..., a^d)) \approx \sum_{i=1}^{d} \tilde{Q}_i(h^i, a^i)$$

Monotonic Value Function Factorization for Deep Multi-Agent Reinforcement Learning (QMIX)

- The full factorization of VDN is not necessary in order to be able to extract decentralized policies that are fully consistent with their centralized counterpart.
- The value function class representable with QMIX includes any value function that can be factored into a non-linear monotonic combination of the agents' individual value functions in the fully observable setting.
- Idea :

$$\operatorname*{argmax}_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix} \qquad \frac{\partial Q_{tot}}{\partial Q_a} \geq 0,\ \forall a \in A.$$

- Architecture :

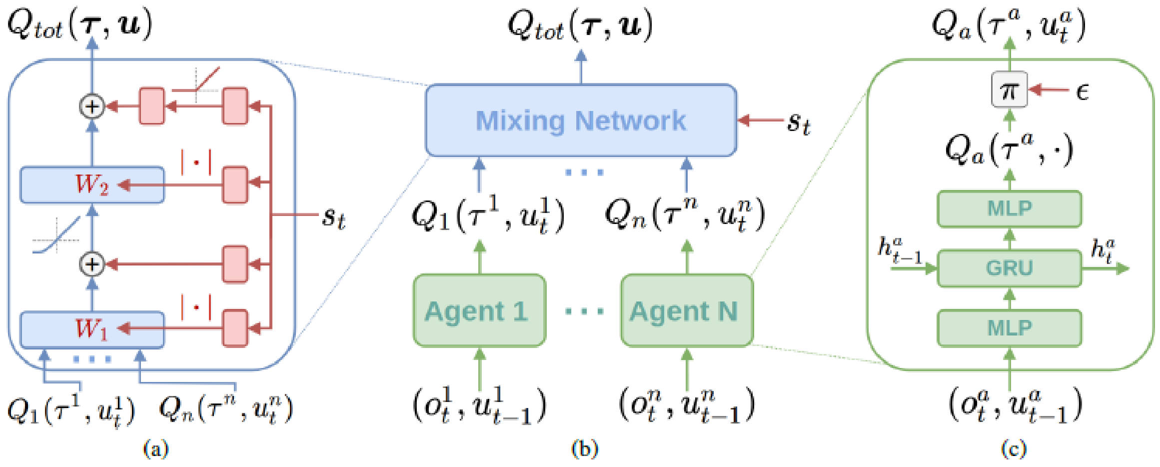


Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. International Conference of Machine Learning (ICML) 2018

Learning to Communicate
(Distributed Control)

ATOC

- Each ActorNet output thought or action
  - ActorNet (I) gets thought or intention
  - ActorNet (II) gets action from thought

- Use Attention Unit to decide whether the agent i be an initiator (binary classifier)

- Initiator forms communication groups with neighbors at most m collaborators
  - Choose un-selected agents first, then selected agents, other initiators last

- Communication channel is a LSTM, integrate internal state of agents within a group
  - Agents selected by more than one group enable group communication
  - Communication is fully determined (when and how long) by the attention unit initially

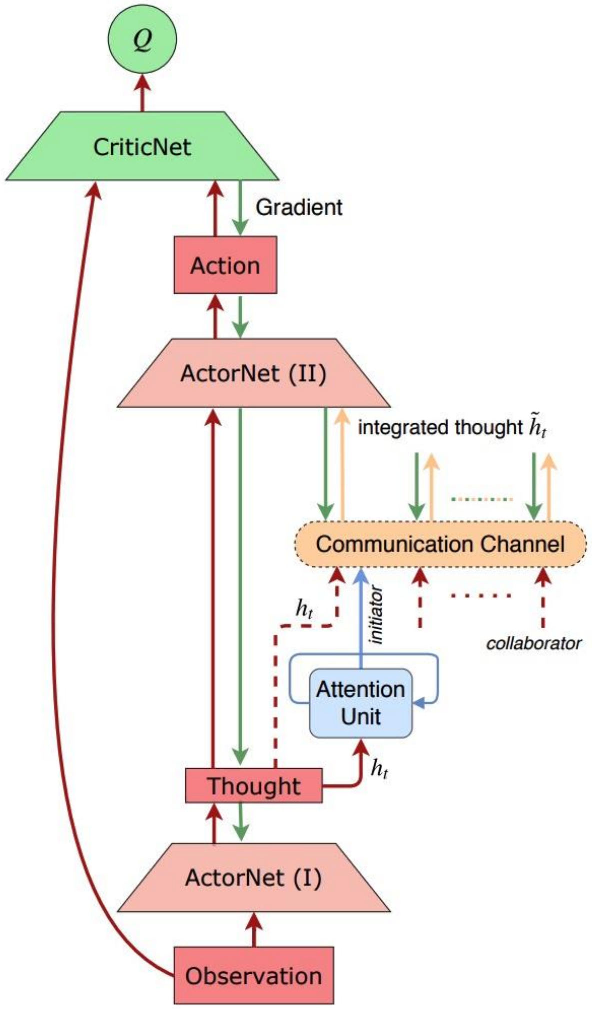


Figure 1: ATOC architecture.

ATOC

- Policy network takes local observation, extracts a hidden layer as *thought*, $h_t^i = \mu_I(o_t^i; \theta^\mu)$ which encodes both local observation and action intentions.

- Every attention unit takes thought h_t^i as input and determines whether communication is needed for cooperation.

- If needed, agent called *initiator*, selects other agents, *collaborators*, in its field to form a communication group.

- Communication channel connects each agent of the communication group, takes as input the though of each agent and outputs the integrated thought that guides agents to generate coordinated actions.

- Integrated thought $\tilde{h}_t^i$ is merged with h_t^i and fed into the rest of the policy network.

- Policy network outputs the action $a_t^i = \mu_{II}(h_t^i, \tilde{h}_t^i; \theta^\mu)$

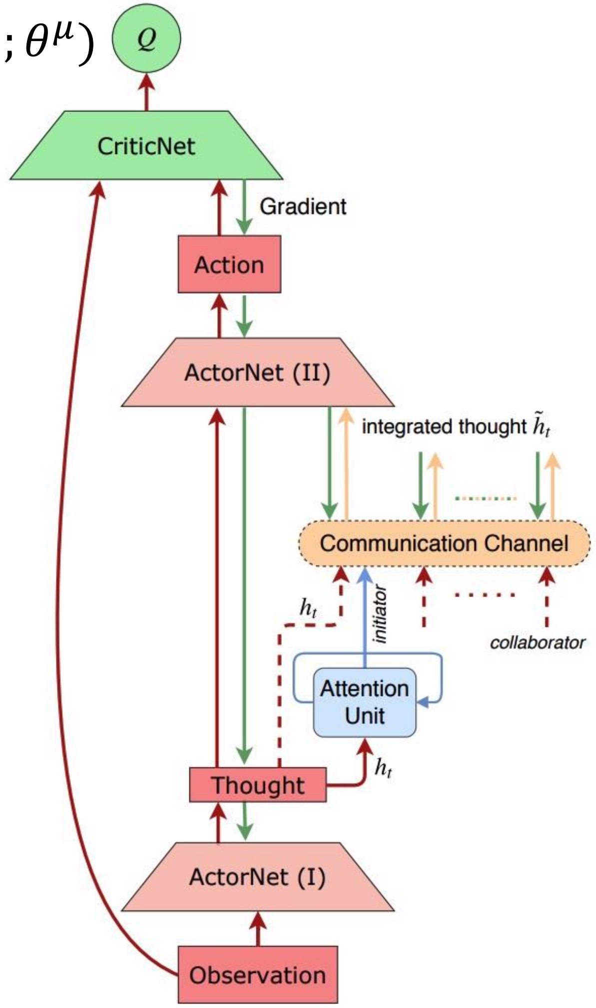


Figure 1: ATOC architecture.

ATOC

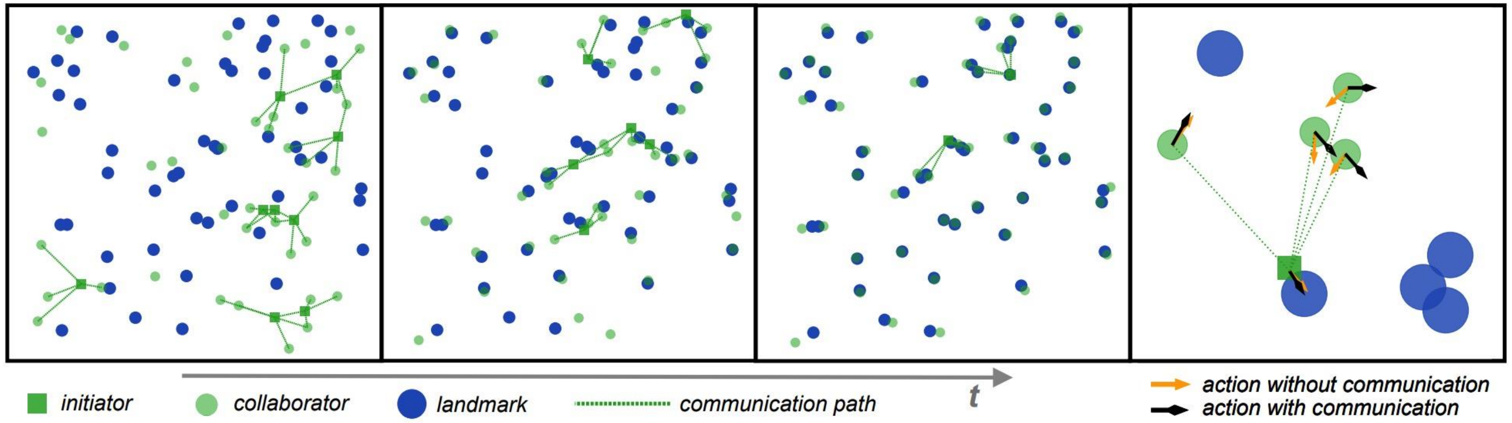


Figure 4: Visualizations of communications among ATOC agents on cooperative navigation. The rightmost figure illustrates actions taken by a group of agents with and without communication.

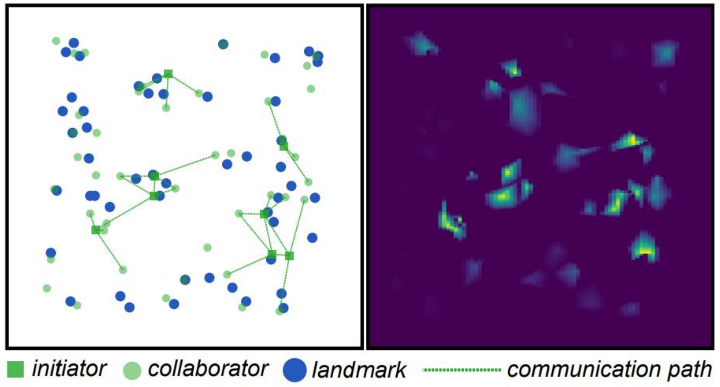


Figure 5: Heatmap of attention corresponding to communication among ATOC agents in cooperative navigation.