Northeastern University
Khoury College of
Computer Sciences

A Short Introduction to Cooperative Multi-Agent Reinforcement Learning

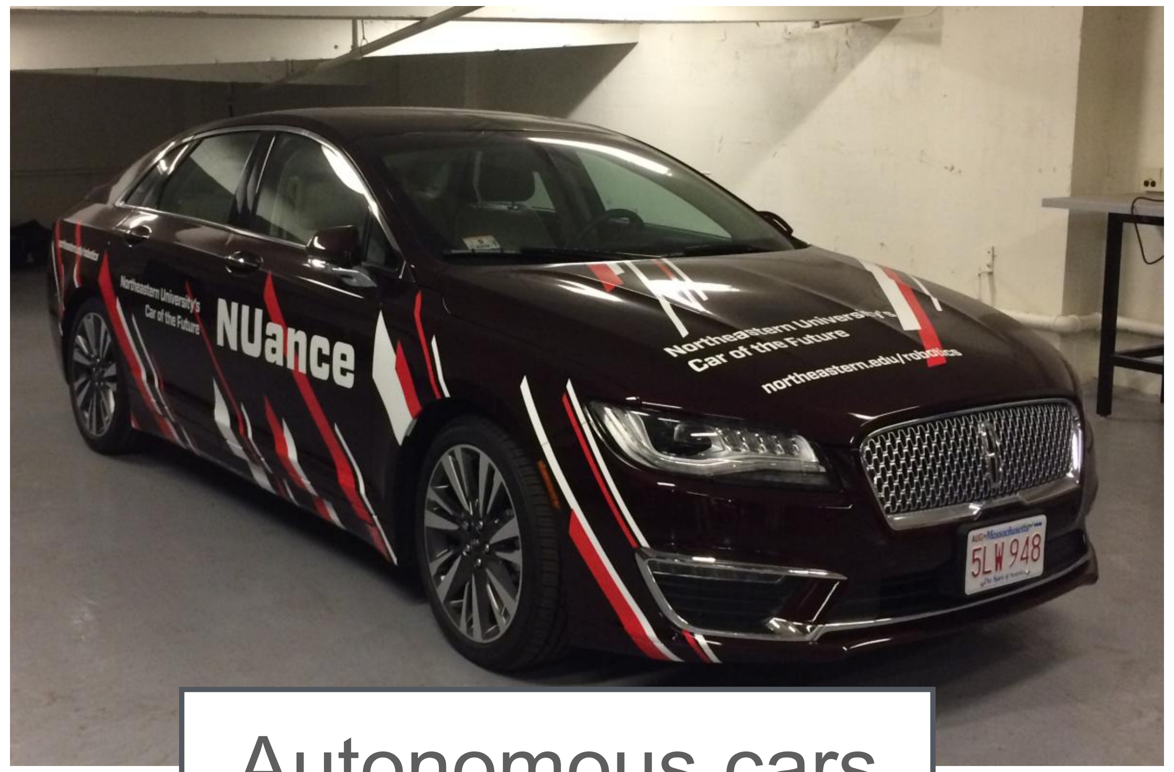
Chris Amato

Multi-agent systems are (going to be) everywhere

Drone delivery

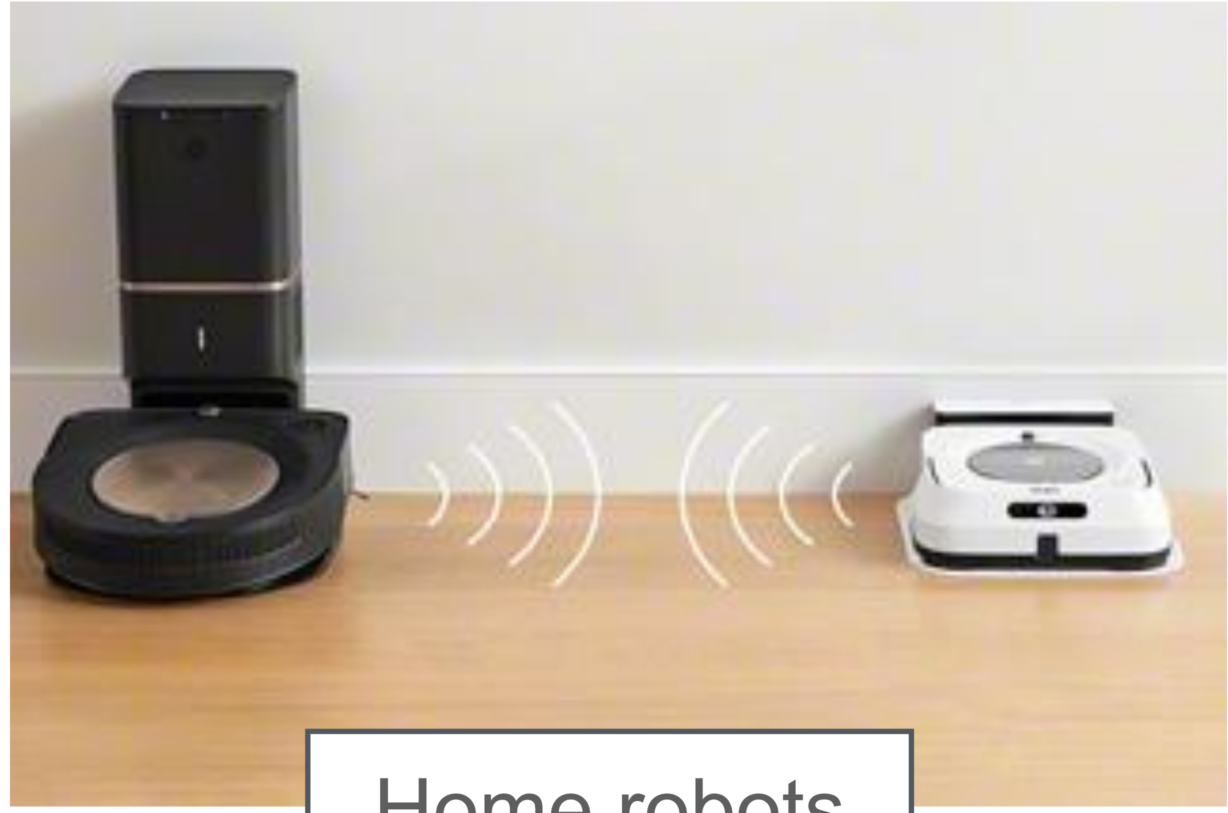
Amazon

LLM agents



Autonomous cars

Northeastern

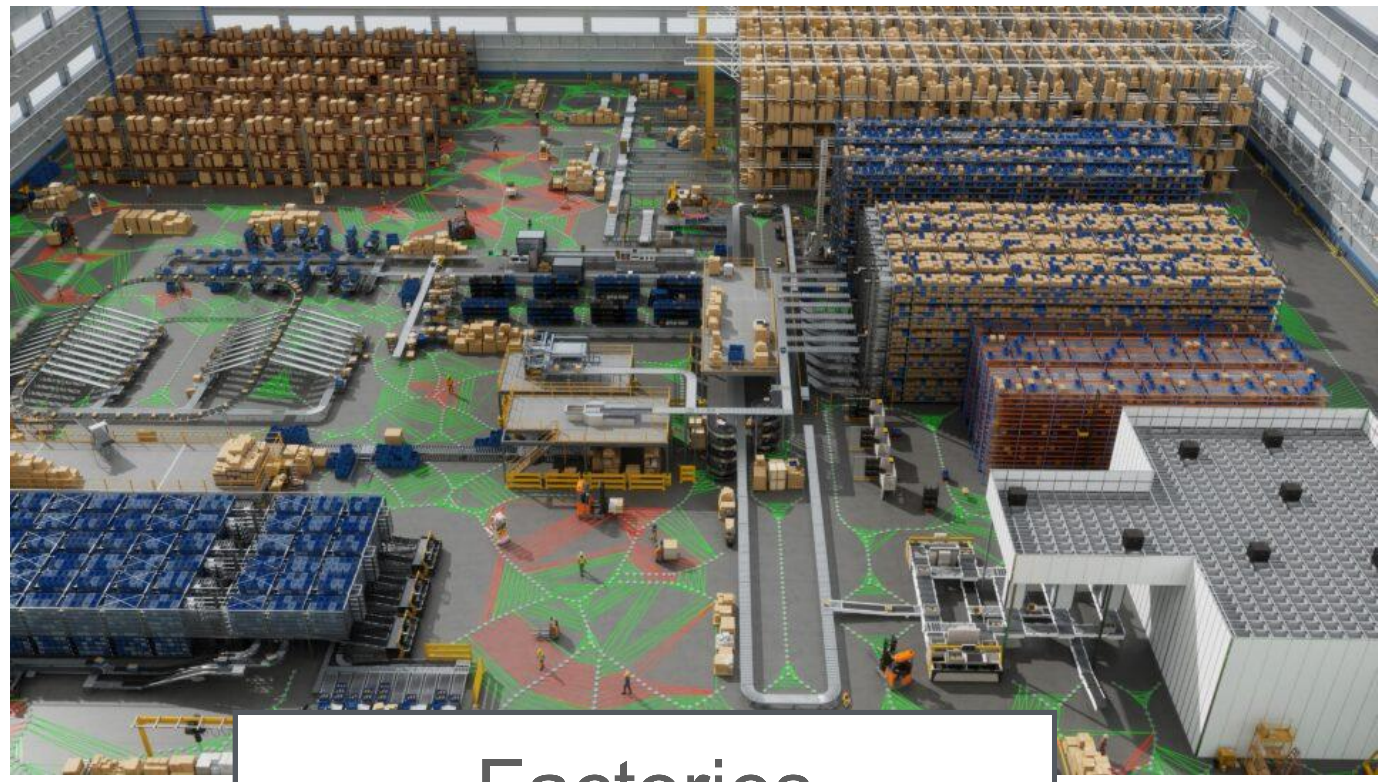


Home robots

iRobot

Warehouse robots

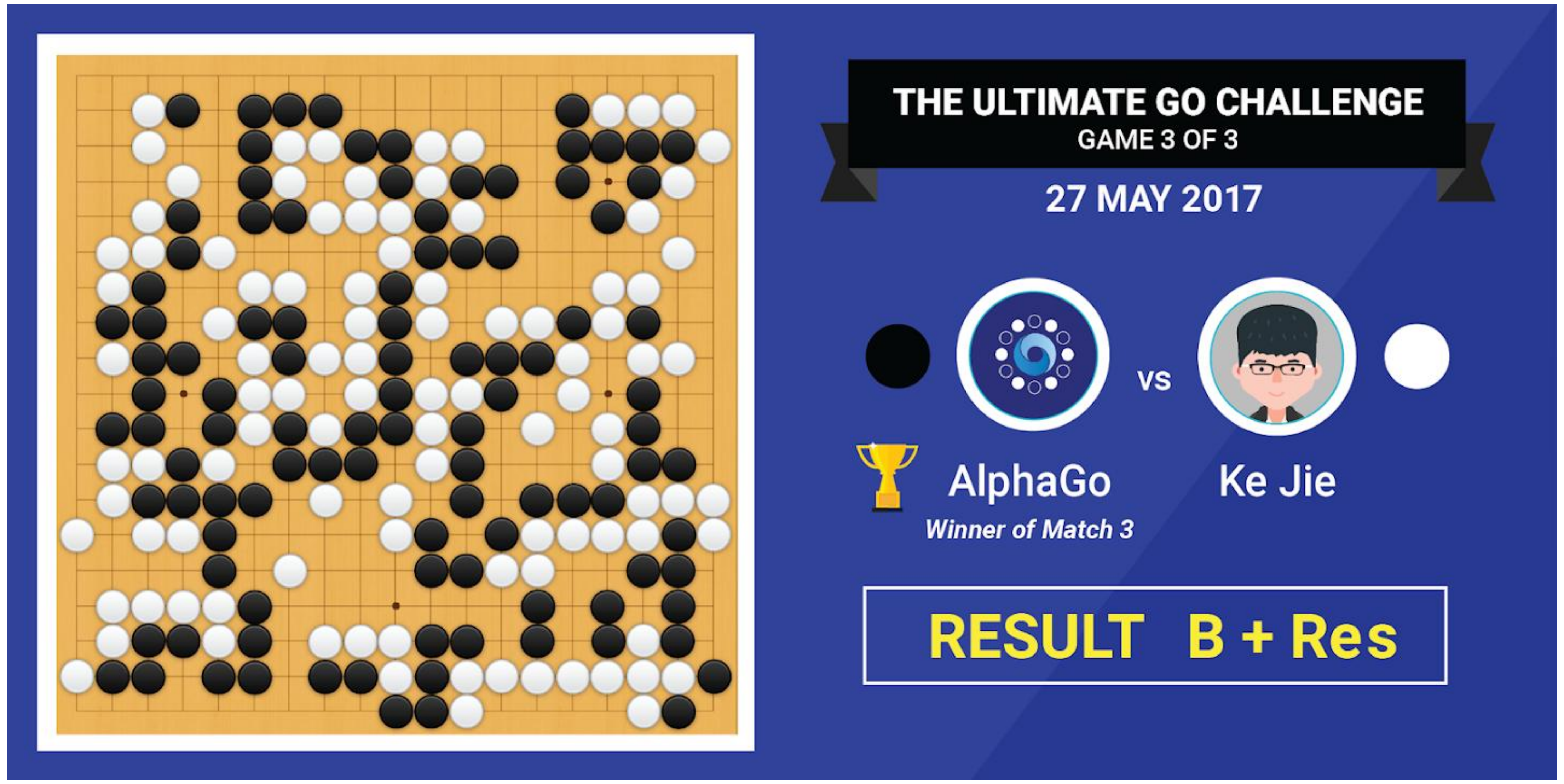
Amazon



Factories

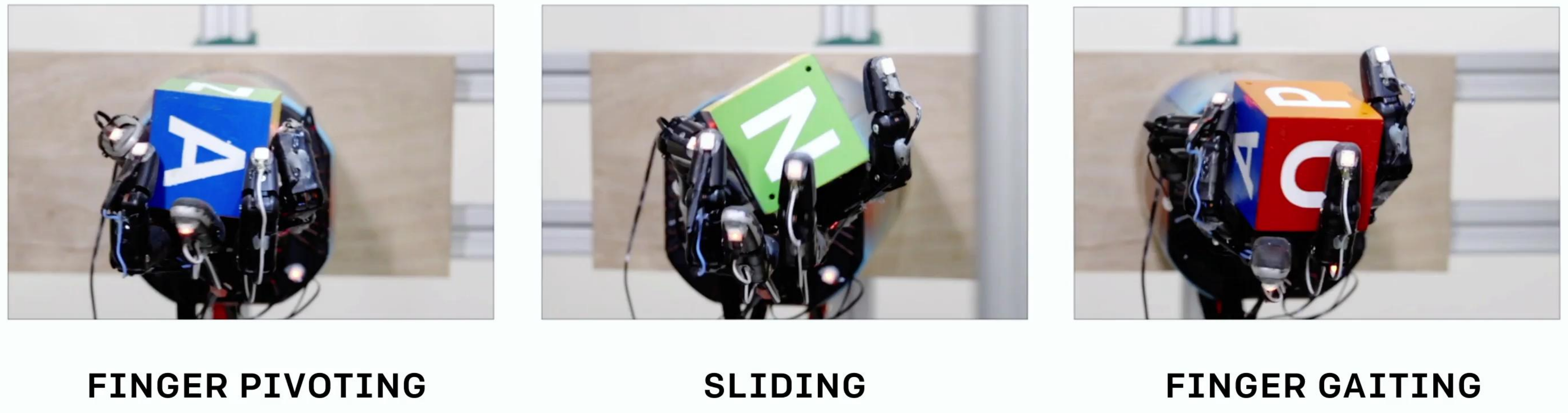
Accenture/nvidia

Reinforcement learning has a number of successes



AlphaGo (Google DeepMind)

Atari (Google DeepMind)



Robot manipulation (OpenAI)

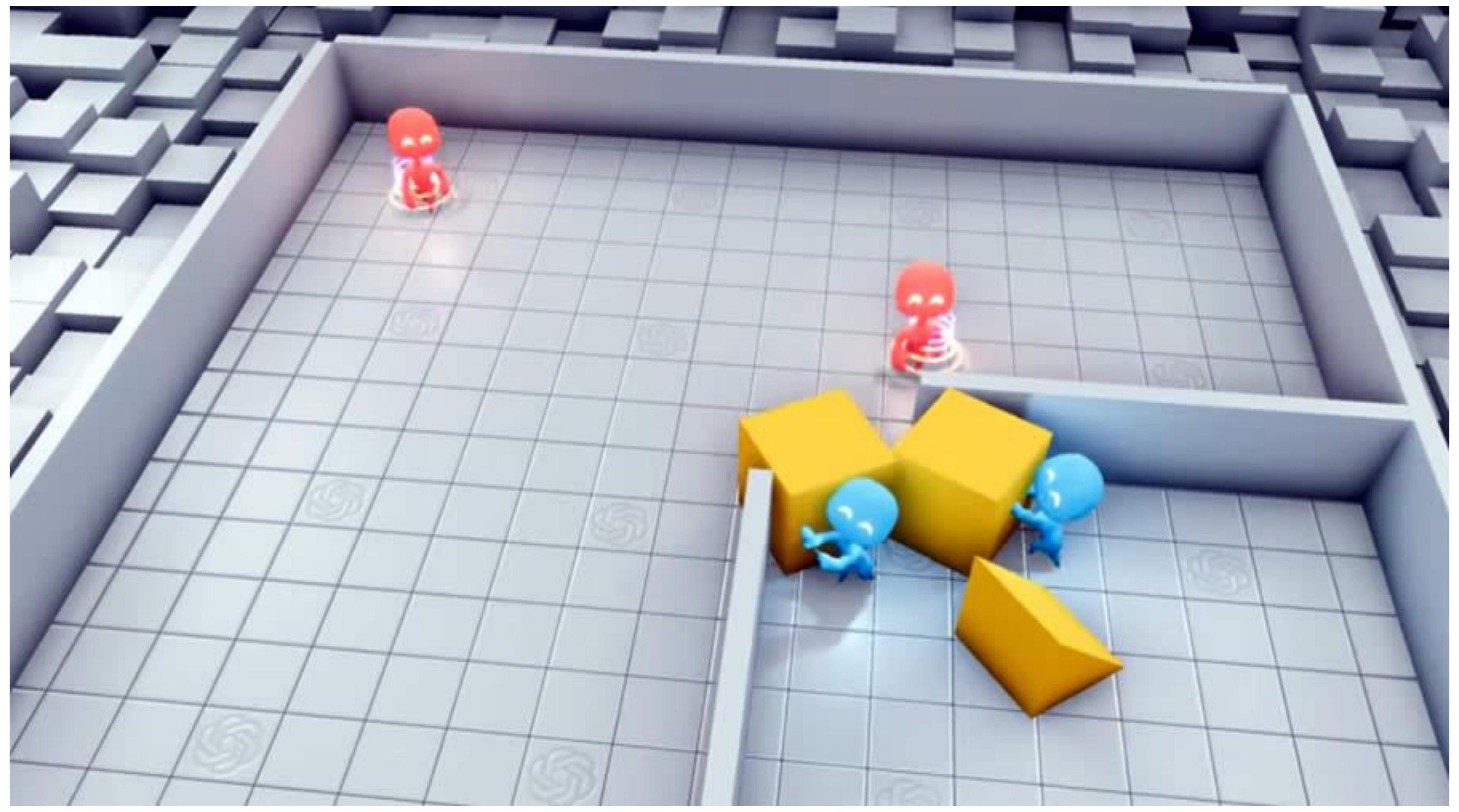
ChatGPT (OpenAI)

Multi-agent RL has had some successes

Project Malmo (Microsoft Research)

Humanoid Soccer (Google DeepMind)

OpenAI Five Dota 2 (OpenAI)

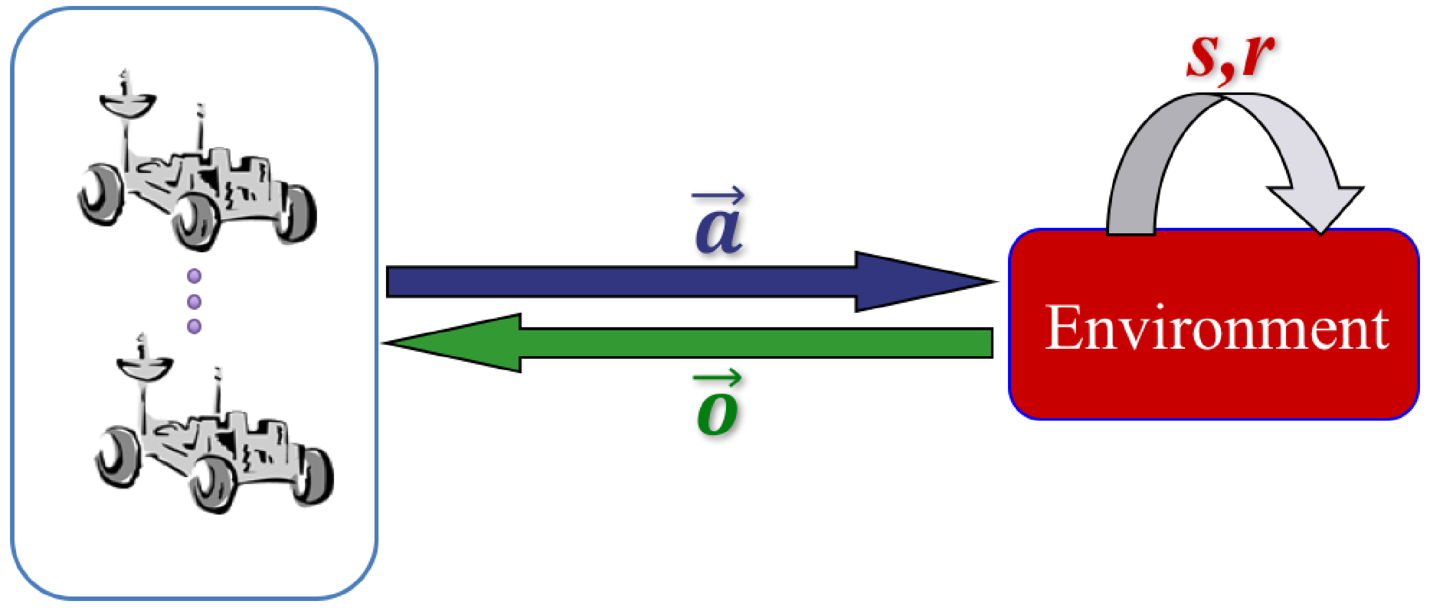


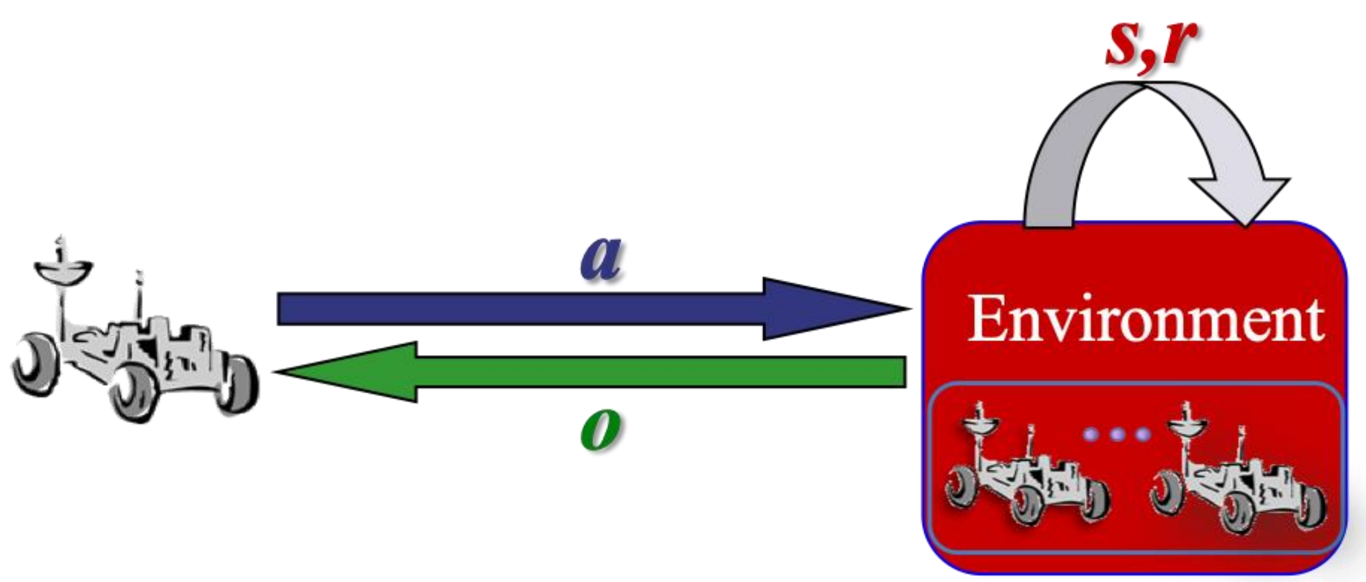
Emergent tool use (OpenAI)

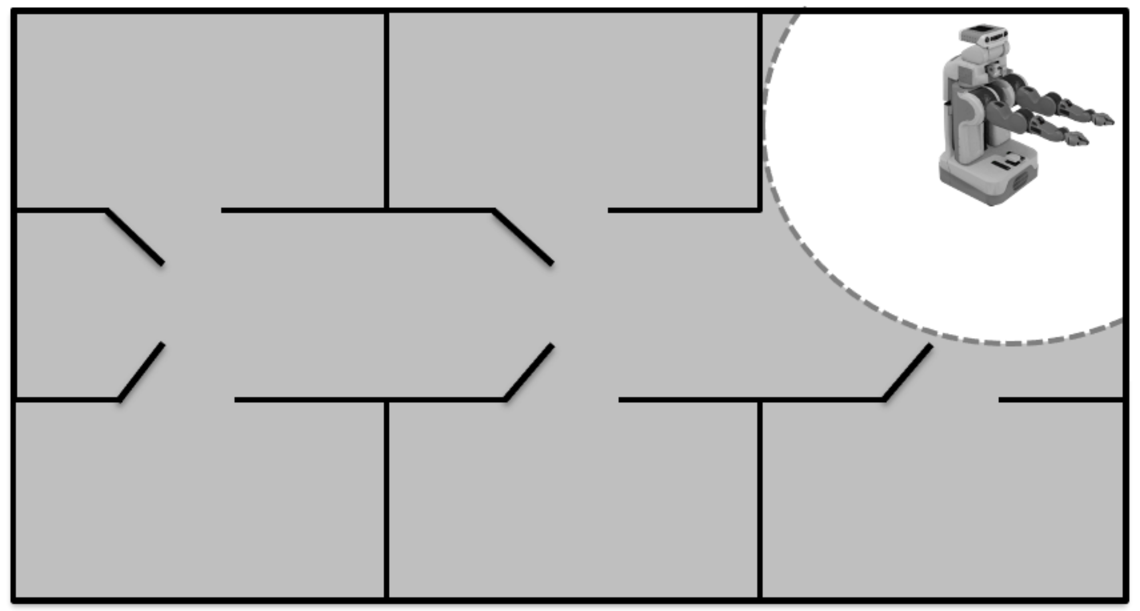
AlphaStar (Google DeepMind)

Multi-agent RL is hard

- How can we apply RL here?
  - *Centralized learning and control*? Need fast, perfect communication (cooperative case)
  - *Decentralized learning and control*? Limited knowledge of other agents and environmental nonstationarity
  - *Centralized training for decentralized execution?* Use centralized information offline but still execute in a decentralized way
- Almost always have partial observability





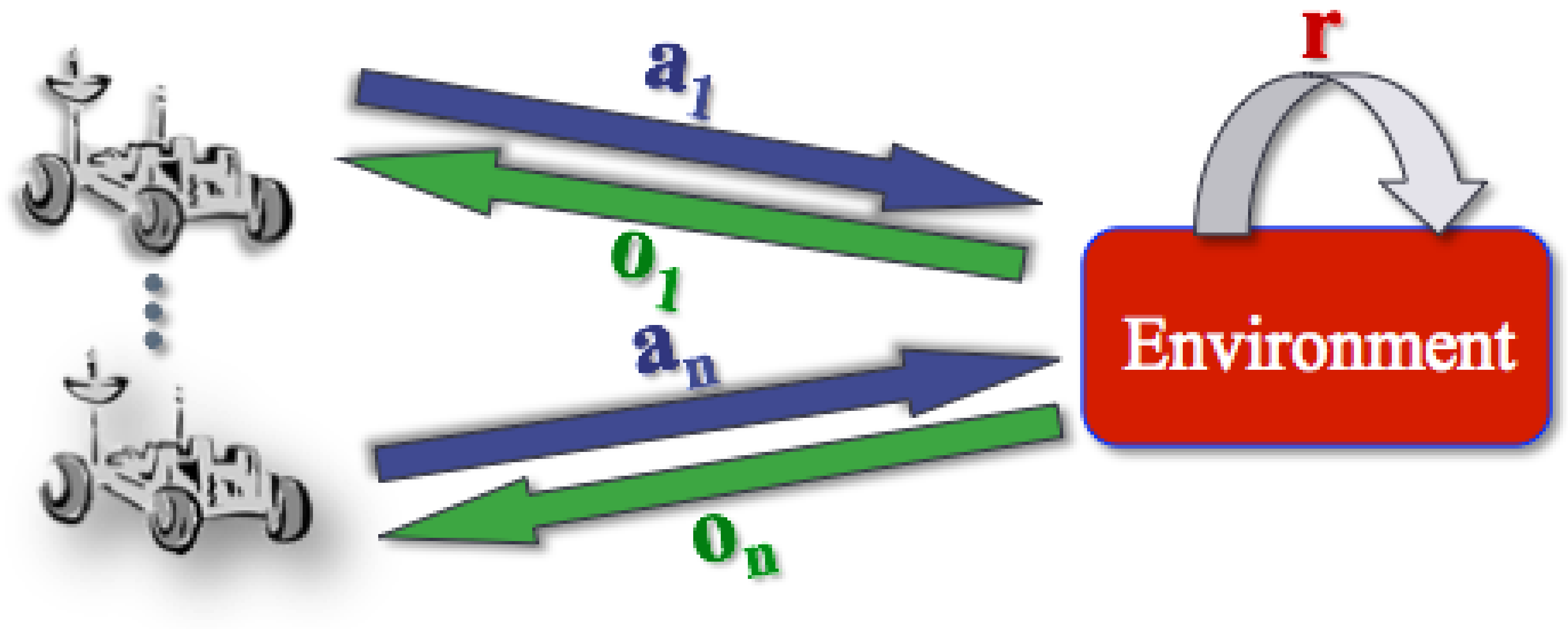


Overview

- Define the cooperative multi-agent RL (MARL) problem
- Discuss the current state-of-the art for the different classes of solutions
  - Centralized training and execution
  - Decentralized training and execution: IQL, decentralized REINFORCE, deep extensions
  - CTDE: VDN, QMIX, QPLEX, MADDPG, MAPPO
- Identify misconceptions/issues with current methods
- Applications, code, other topics, and the future (LLMs?)

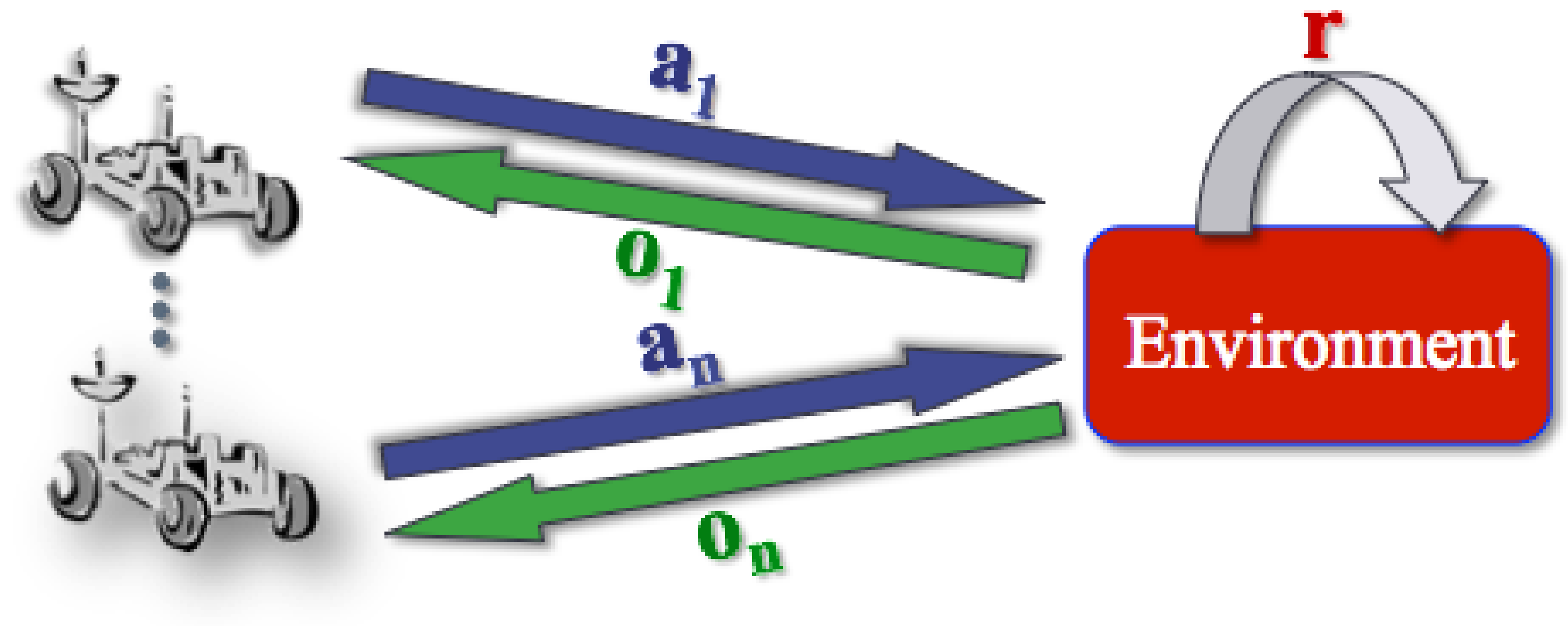
Cooperative MARL

- Cooperative case represented as Decentralized POMDP: <*I*, *S*, {A_i}, *T*, *R*, {Ω_i}, *O*, □>
  - *I*, a finite set of agents
  - *S*, a set of states
  - A_i, each agent's set of actions
  - *T*, the state transition model: $P(s'|s,\boldsymbol{a})$
  - *R*, the reward model: $R(s,\boldsymbol{a})$
  - Ω_i, each agent's finite set of observations
  - *O*, the observation model: $P(\boldsymbol{o}|s',\boldsymbol{a})$
  - *h*, horizon or discount □



Cooperative MARL

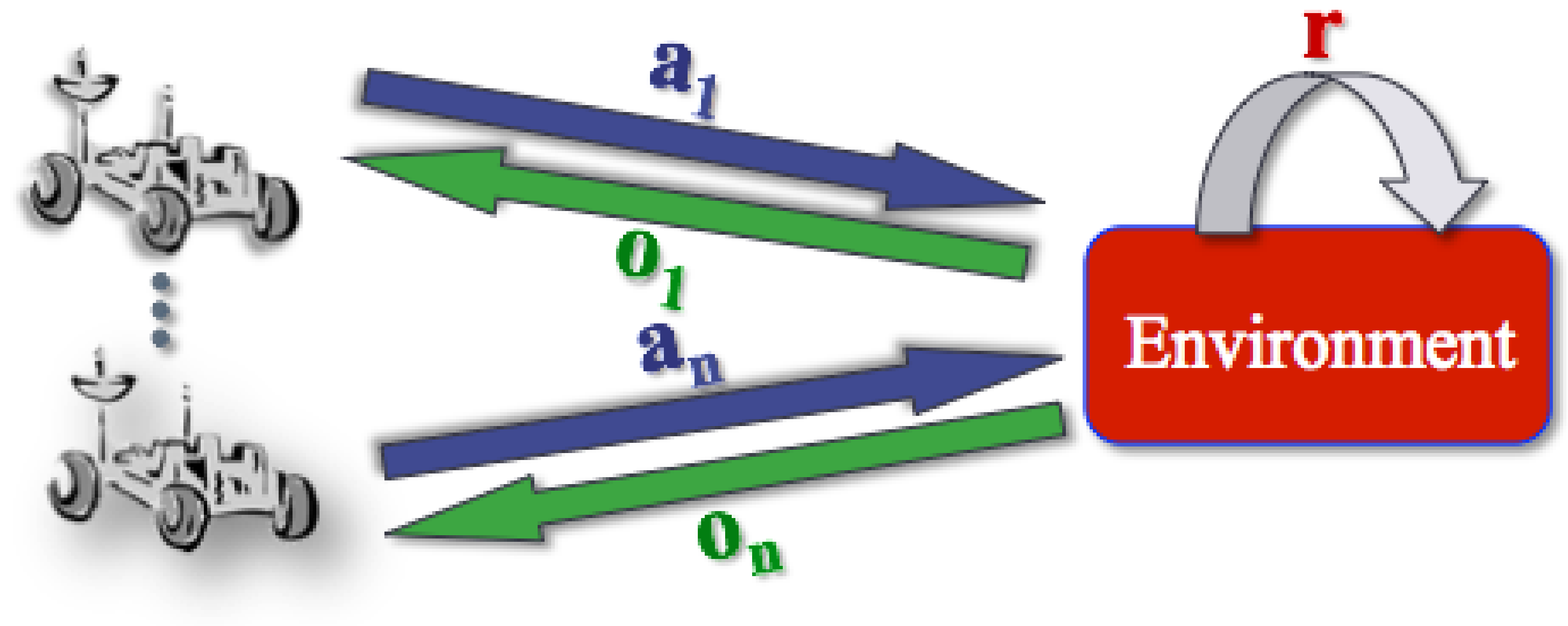
- Cooperative case represented as Decentralized POMDP: <*I*, *S*, {A_i}, *T*, *R*, {Ω_i}, *O*, ⍰>
  - *I*, a finite set of agents
  - *S*, a set of states
  - A_i, each agent's set of actions
  - *T*, the state transition model: $P(s'|s, \boldsymbol{a})$
  - *R*, the reward model: $R(s, \boldsymbol{a})$
  - Ω_i, each agent's finite set of observations
  - *O*, the observation model: $P(\boldsymbol{o}|s', \boldsymbol{a})$
  - *h*, horizon or discount ⍰



Objective: Maximize the (discounted) sum of future (joint) rewards   Cooperative

Cooperative MARL

- Cooperative case represented as Decentralized POMDP: <I, S, {A_i}, T, R, {Ω_i}, O, ⍰>
  - I, a finite set of agents
  - S, a set of states
  - A_i, each agent's set of actions
  - T, the state transition model: $P(s'|s, \boldsymbol{a})$
  - R, the reward model: $R(s, \boldsymbol{a})$
  - Ω_i, each agent's finite set of observations
  - O, the observation model: $P(\boldsymbol{o}|s', \boldsymbol{a})$
  - h, horizon or discount ⍰



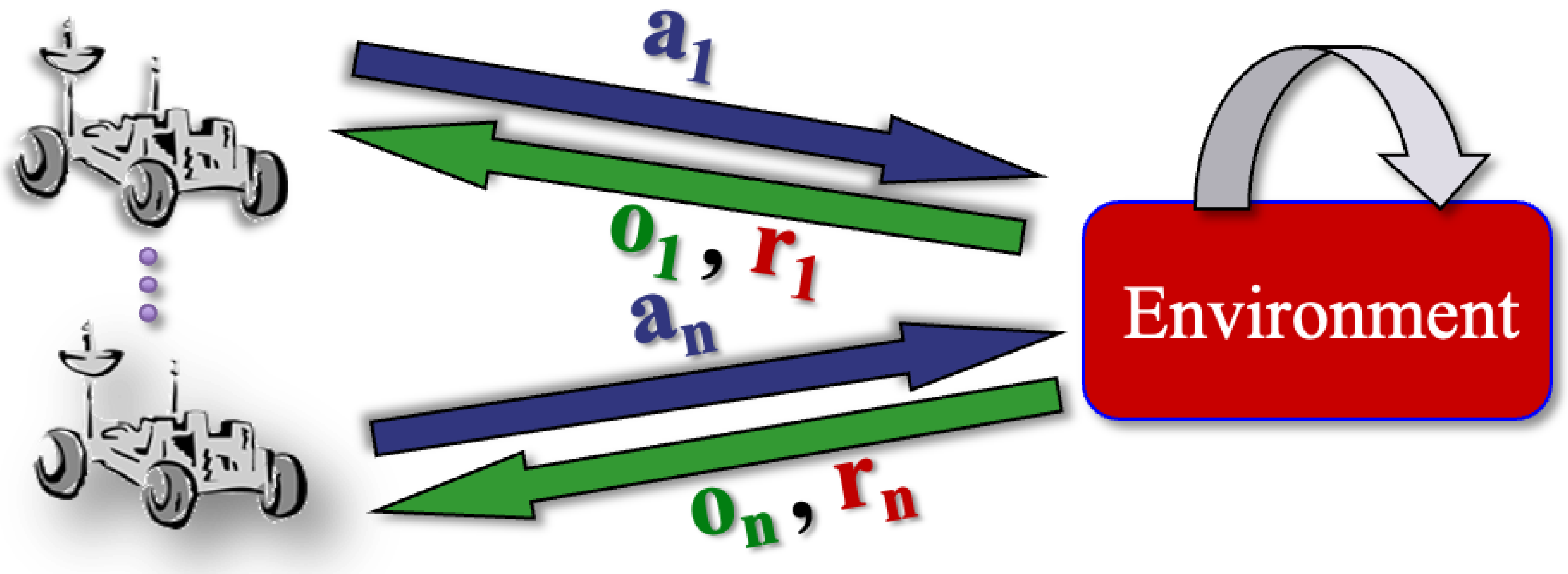
Objective: Maximize the (discounted) sum of future (joint) rewards

Calculate a set of optimal policies for each agent π_i^*: $H_i \rightarrow A_i$ that maximize joint objective

Decentralized partially observable execution

General MARL

- General case as Partially Observable Stochastic Game (POSG): <*I*, *S*, {A_i}, *T*, {R_i}, {Ω_i}, *O*, ⍰>
  - *I*, a finite set of agents
  - *S*, a set of states
  - A_i, each agent's set of actions
  - *T*, the state transition model: $P(s'|s,\boldsymbol{a})$
  - R_i, the reward model: $R_i(s,\boldsymbol{a})$
  - Ω_i, each agent's finite set of observations
  - *O*, the observation model: $P(\boldsymbol{o}|s',\boldsymbol{a})$
  - *h*, horizon or discount ⍰



Objective unclear: Some form of each agent maximizing the (discounted) sum of future individual rewards

Mixed/competitive

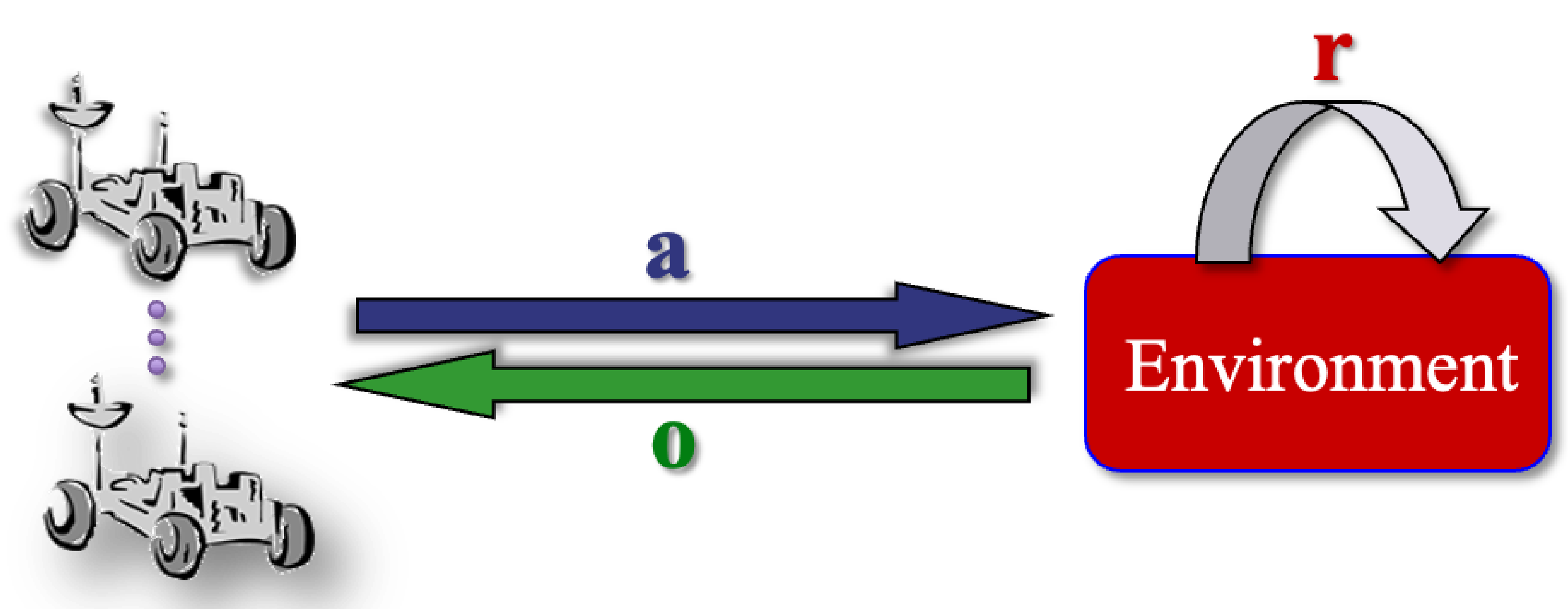
Centralized MARL

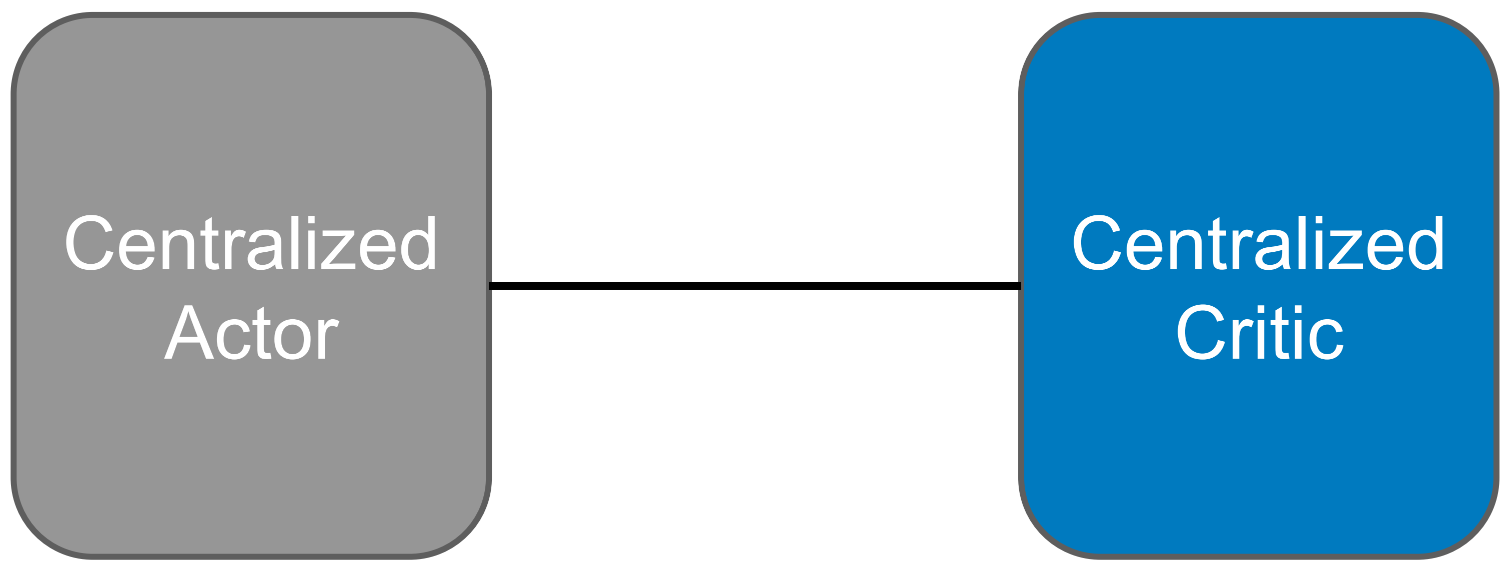
Models and methods

Centralized MARL

Assumptions:

- a centralized controller chooses actions for each agent, ***a***
- each agent takes the chosen actions ***a*** = $\langle a_1,\ldots,a_n \rangle$,
- the centralized controller observes the resulting observations ***o*** = $\langle o_1,\ldots,o_n \rangle$
- the (centralized) algorithm/controller observes ***o*** (and ***a***) and the joint reward r

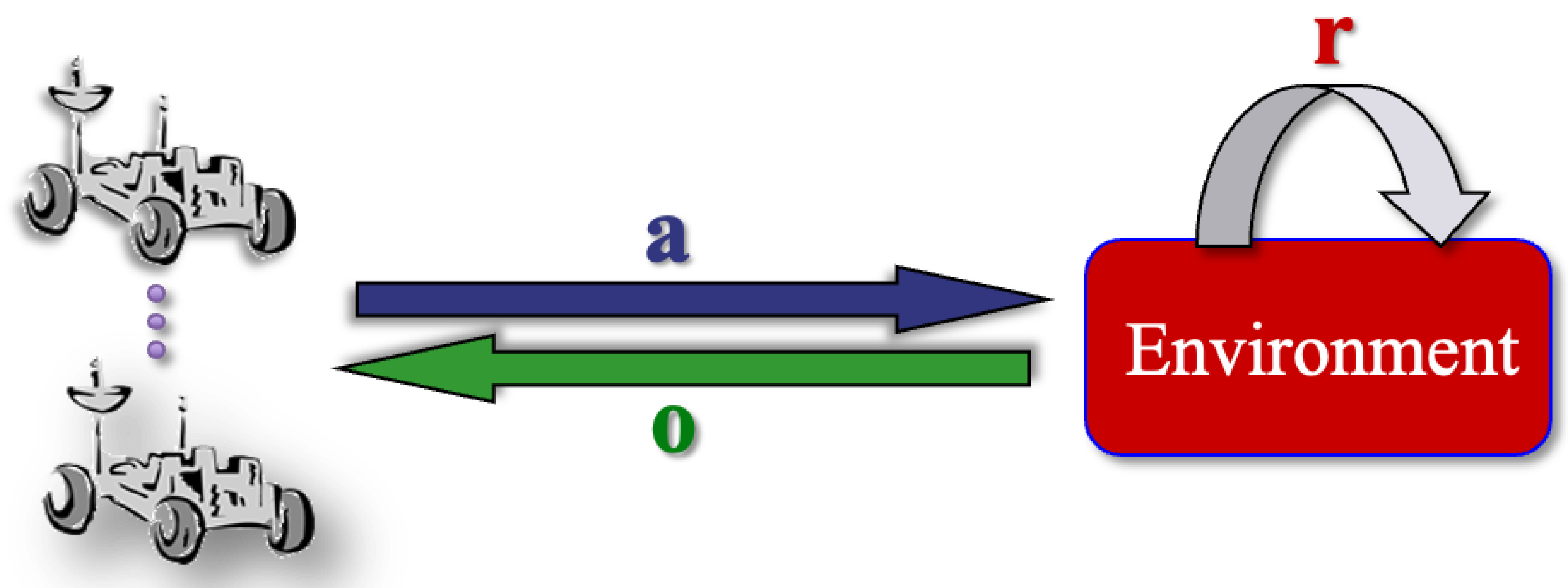




Note: Not a Dec-POMDP (or POSG) anymore since execution is centralized

Centralized MARL (partially observable)

- Cooperative case represented as MPOMDP: <*I*, *S*, {A_i}, *T*, *R*, {Ω_i}, *O*, □>
  - *I*, a finite set of agents
  - *S*, a set of states
  - A_i, each agent's set of actions
  - *T*, the state transition model: $P(s'|s, \boldsymbol{a})$
  - *R*, the reward model: $R(s, \boldsymbol{a})$
  - Ω_i, each agent's finite set of observations
  - *O*, the observation model: $P(\boldsymbol{o}|s', \boldsymbol{a})$
  - *h*, horizon or discount □

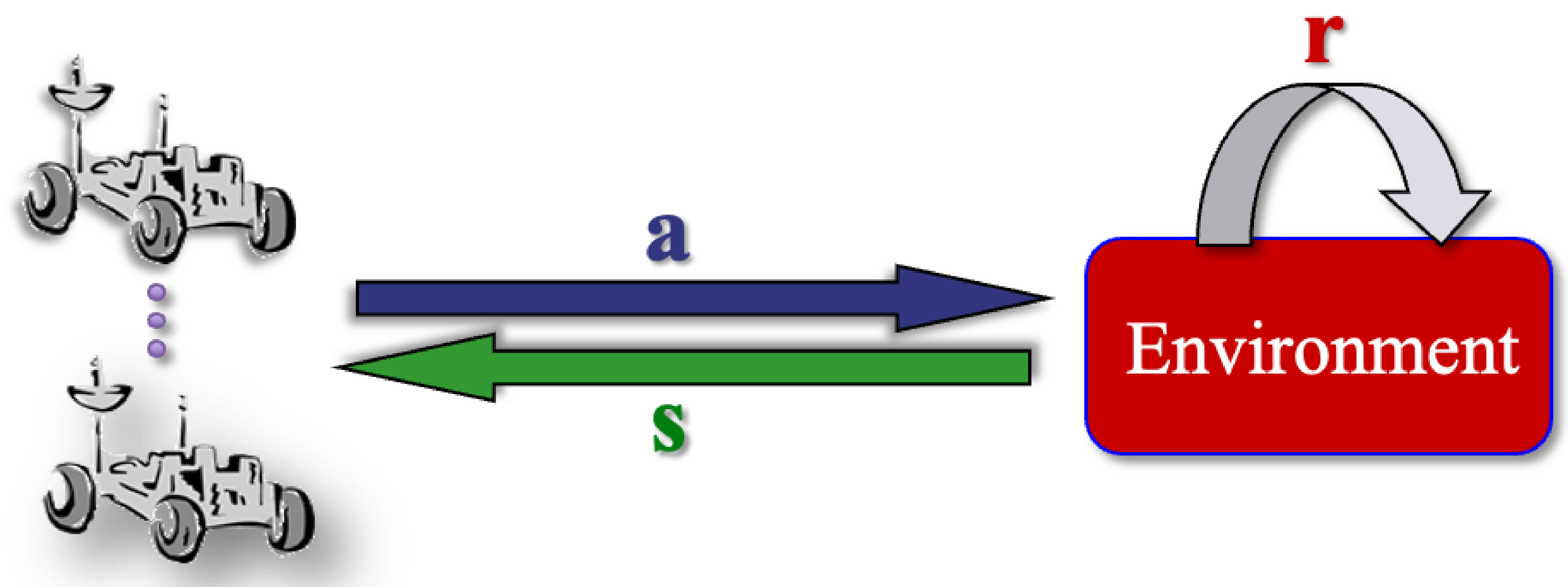


Objective: Maximize the (discounted) sum of future (joint) rewards

Calculate a single optimal policy for all agents π*: H → A that maximizes centralized objective

Centralized MARL (fully observable)

- Cooperative case represented as MMDP: <*I*, *S*, {A_i}, *T*, *R*, ⍰>
  - *I*, a finite set of agents
  - *S*, a set of states
  - A_i, each agent's set of actions
  - *T*, the state transition model: $P(s'|s, \boldsymbol{a})$
  - *R*, the reward model: $R(s, \boldsymbol{a})$
  - *h*, horizon or discount ⍰



Objective: Maximize the (discounted) sum of future (joint) rewards

Calculate a single optimal policy for all agents π^*: S → A that maximizes centralized objective

Centralized MARL (DRQN version)

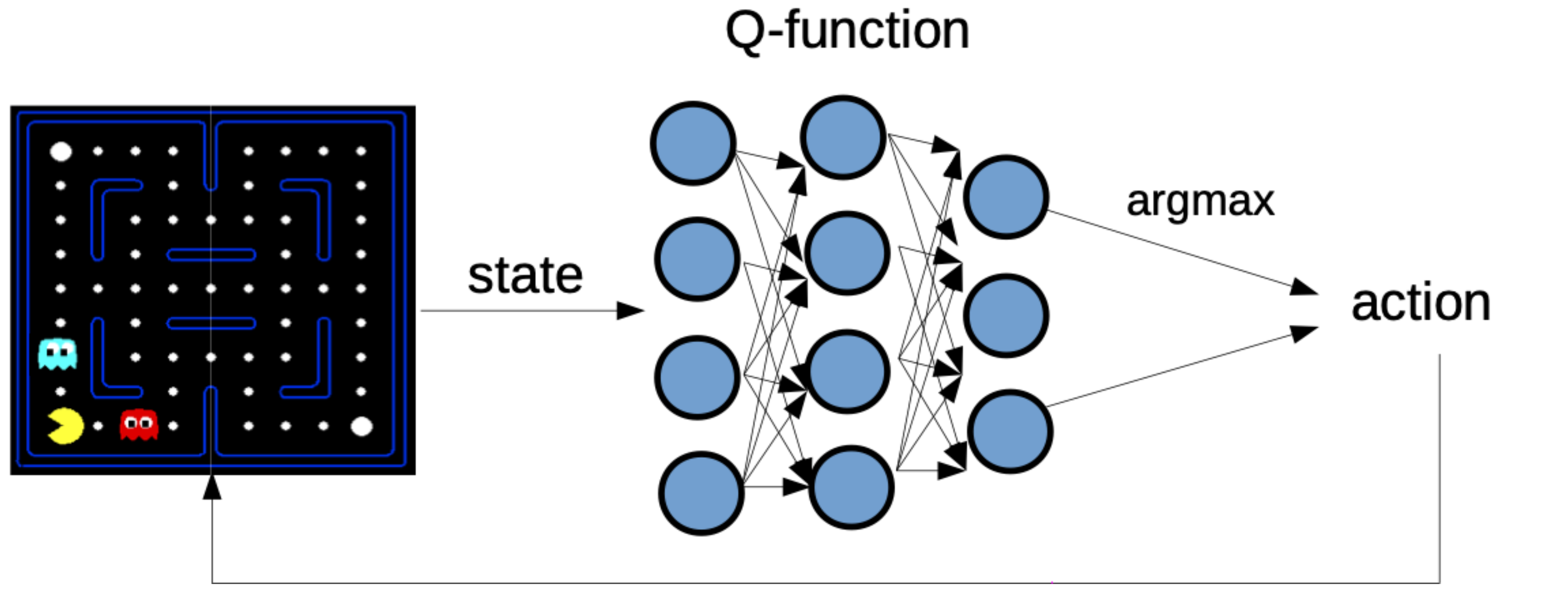
- Traditional Q-learning: estimate Q-value with (x can be state, observation or history)

$$Q(x,a) \leftarrow Q(x,a) + \alpha\delta$$

For learning rate α

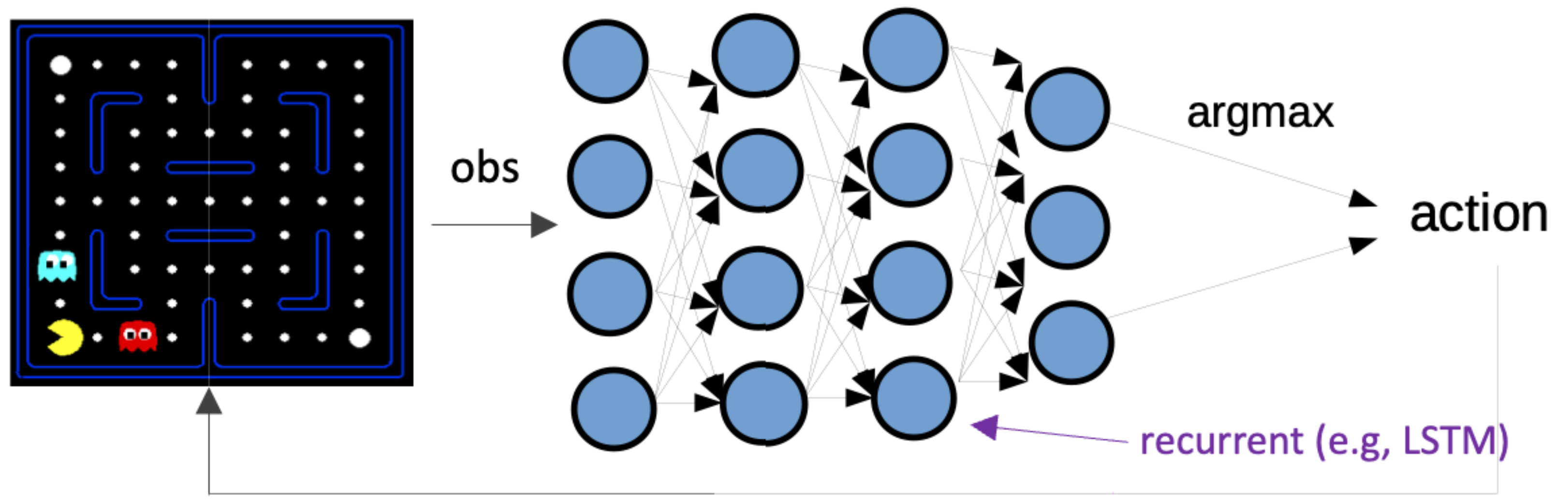
$$\delta = Q(x,a) - (r + \gamma \max_{a'} Q(x', a'))$$

- Deep Q-Networks (DQN) (Mnih et al., Nature 15) uses a neural net for function approximation



Helps with scalability

- DRQN (Hausknecht and Stone, arXiv 15) adds a recurrent layer for memory



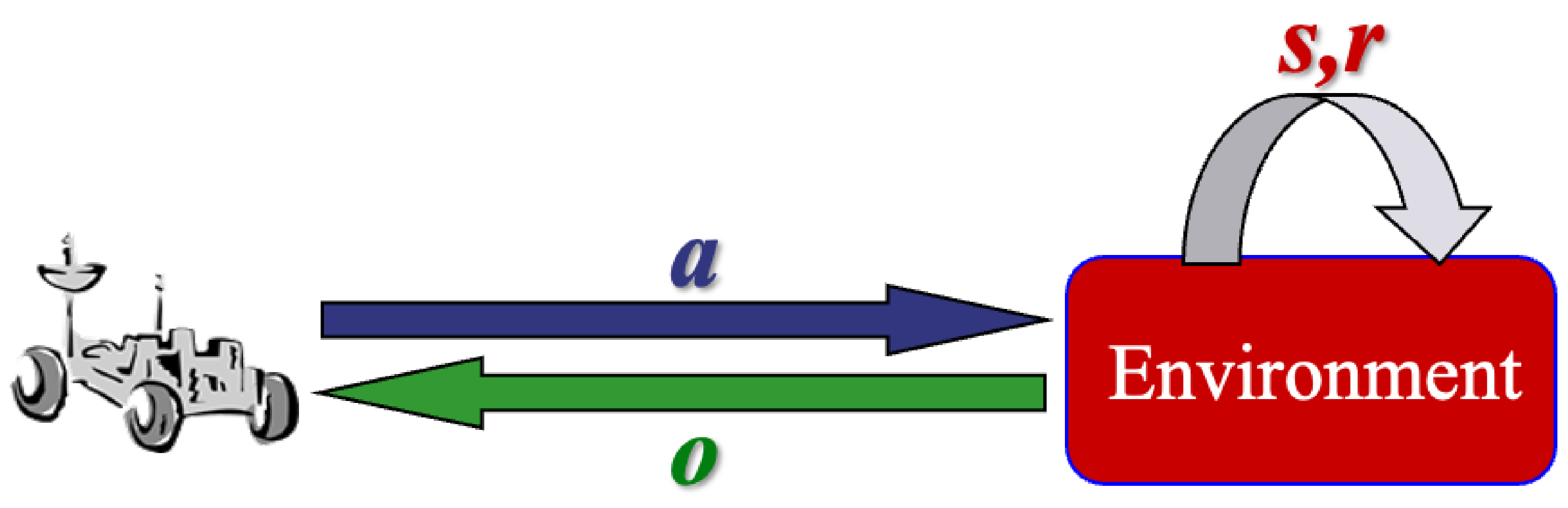
Helps with partial observability

Joint history

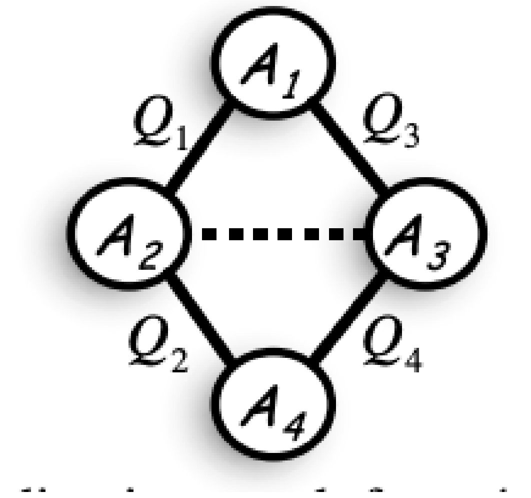
$$Q(\boldsymbol{h},\boldsymbol{a}) \leftarrow Q(\boldsymbol{h},\boldsymbol{a}) + \alpha\delta$$

$$\delta = Q(\boldsymbol{h},\boldsymbol{a}) - \left(r + \gamma \max_{\boldsymbol{a}'} Q(\boldsymbol{h},\boldsymbol{a}')\right)$$

Centralized MARL methods



- Now just a (factored) single-agent problem
  - Multi-agent MDP or POMDP (not Dec-POMDP/POSG)
  - Can use any single-agent RL method
  - But it doesn't scale well
  - And assumes centralized information and control
  - Some methods exploit multi-agent factorization but not very active
    - Coordination graphs [Guestrin et al., 2001]
    - AlphaStar [Vinyals et al., 2019]



Decentralizing centralized solutions

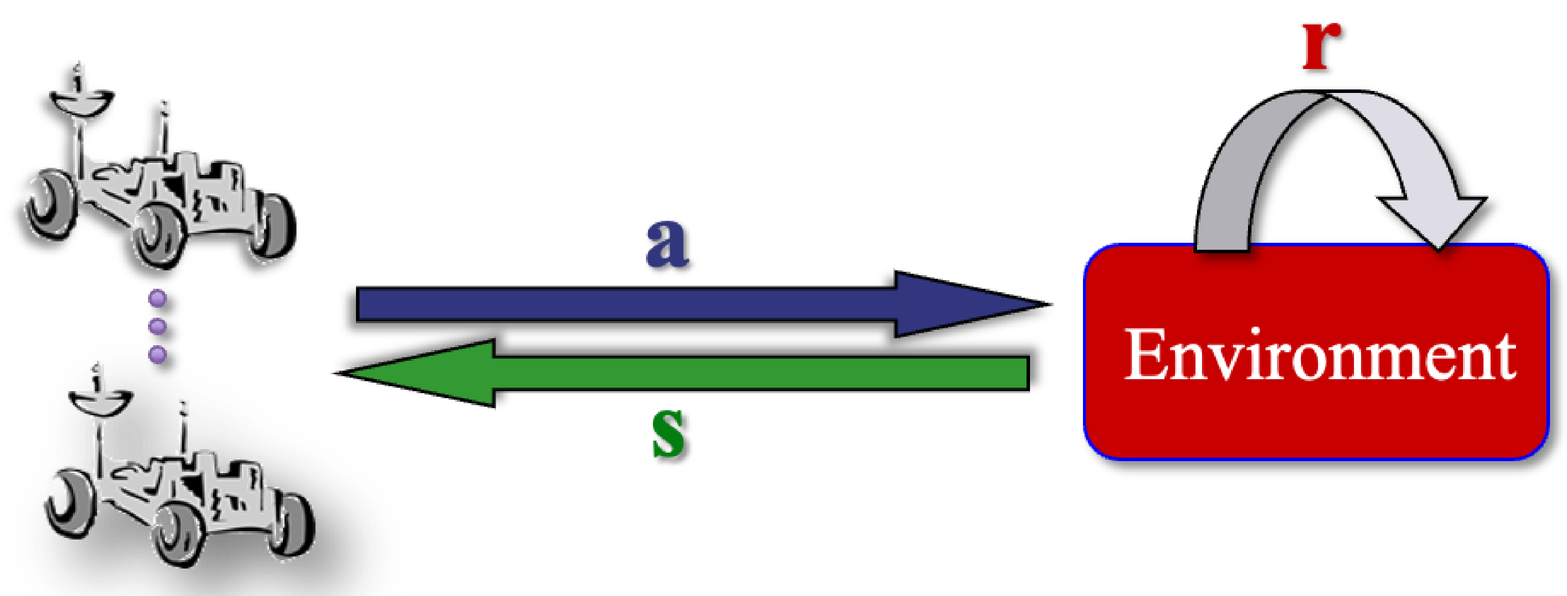
Easy to 'decentralize' in a MMDP or MPOMDP

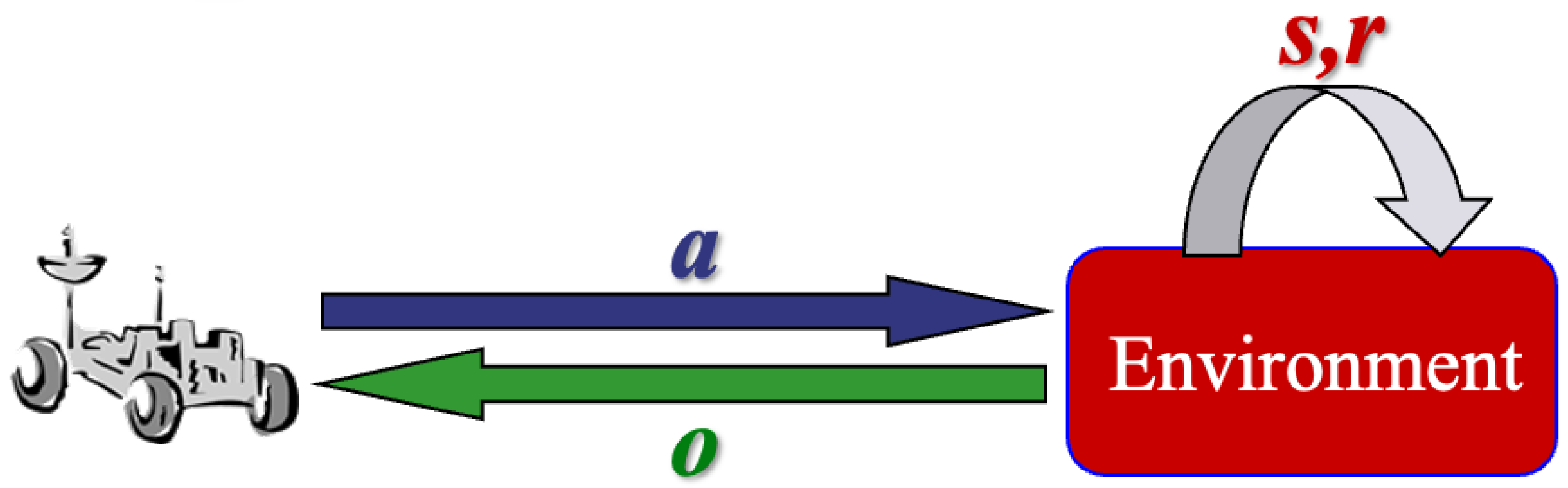
- MMDP
  - $S \rightarrow \boldsymbol{A}$ or $S \rightarrow A_i$
- MPOMDP
  - $\boldsymbol{H} \rightarrow \boldsymbol{A}$ or $\boldsymbol{H} \rightarrow A_i$

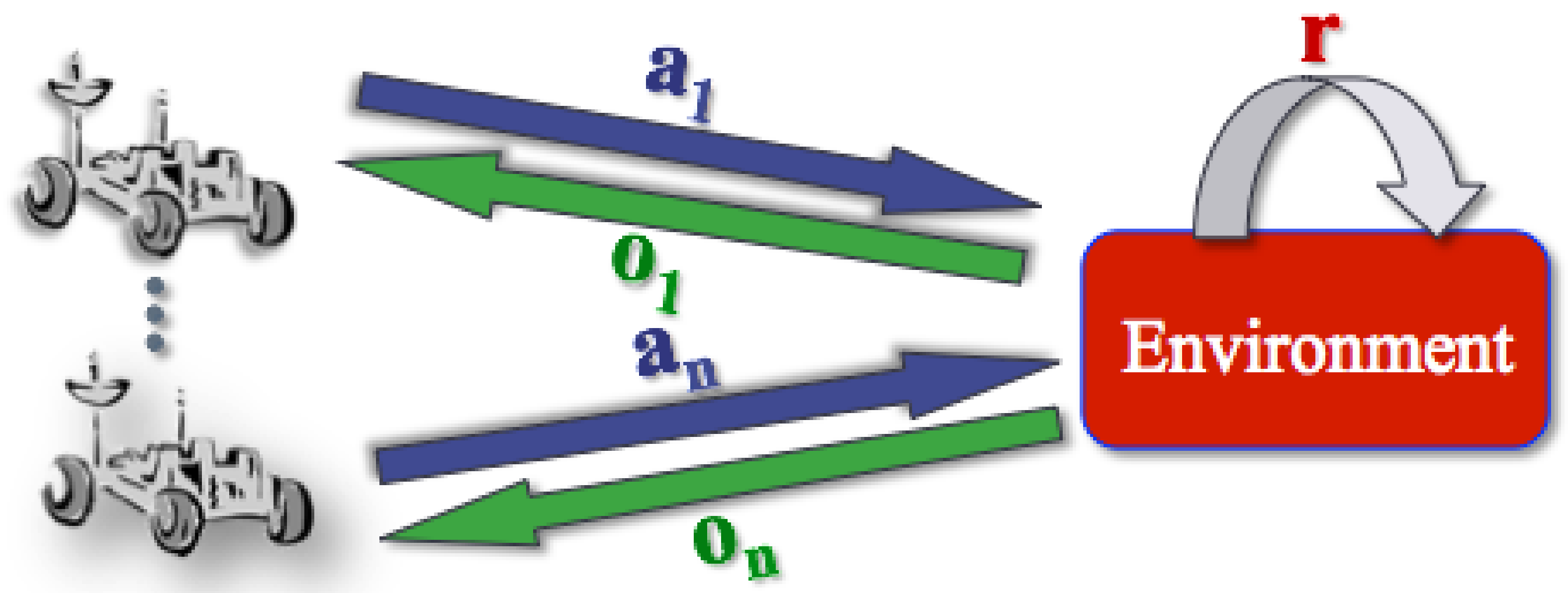
Hard in a Dec-POMDP

Once you have $\boldsymbol{H} \rightarrow \boldsymbol{A}$ how do you get $H_i \rightarrow A_i$?

Easy to decentralize control but not information







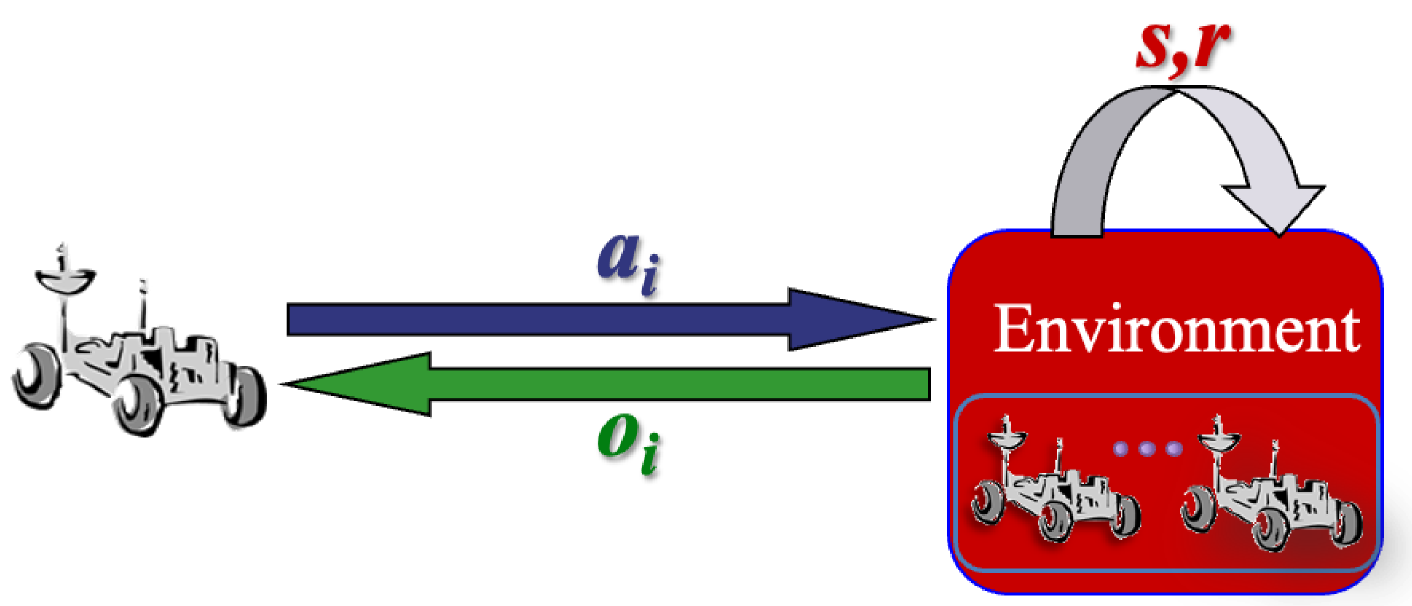
Decentralized MARL

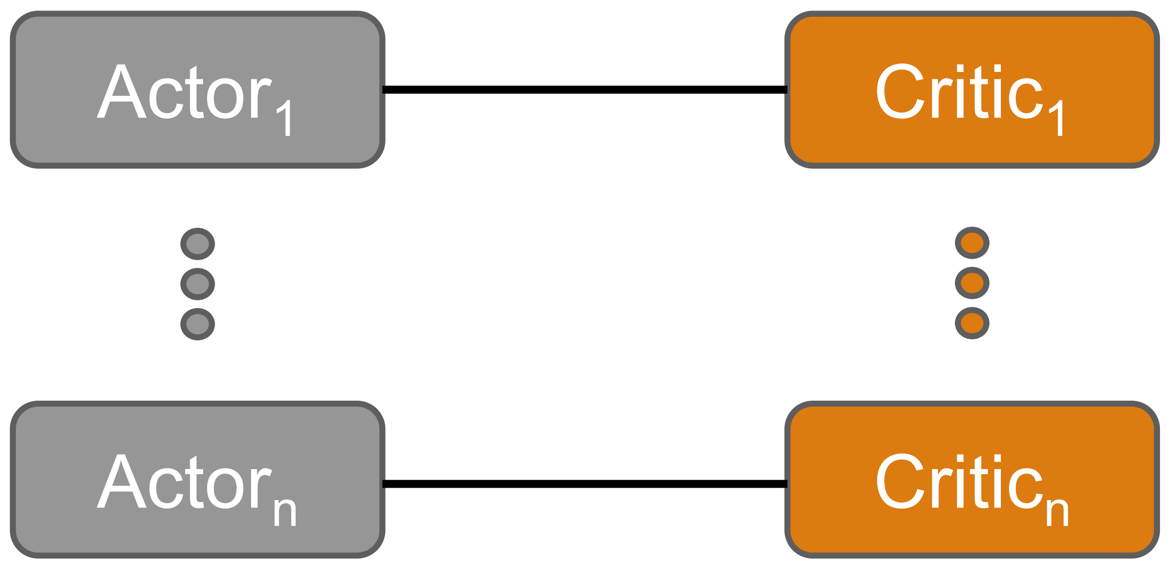
Models and methods

Decentralized MARL

Assumptions:

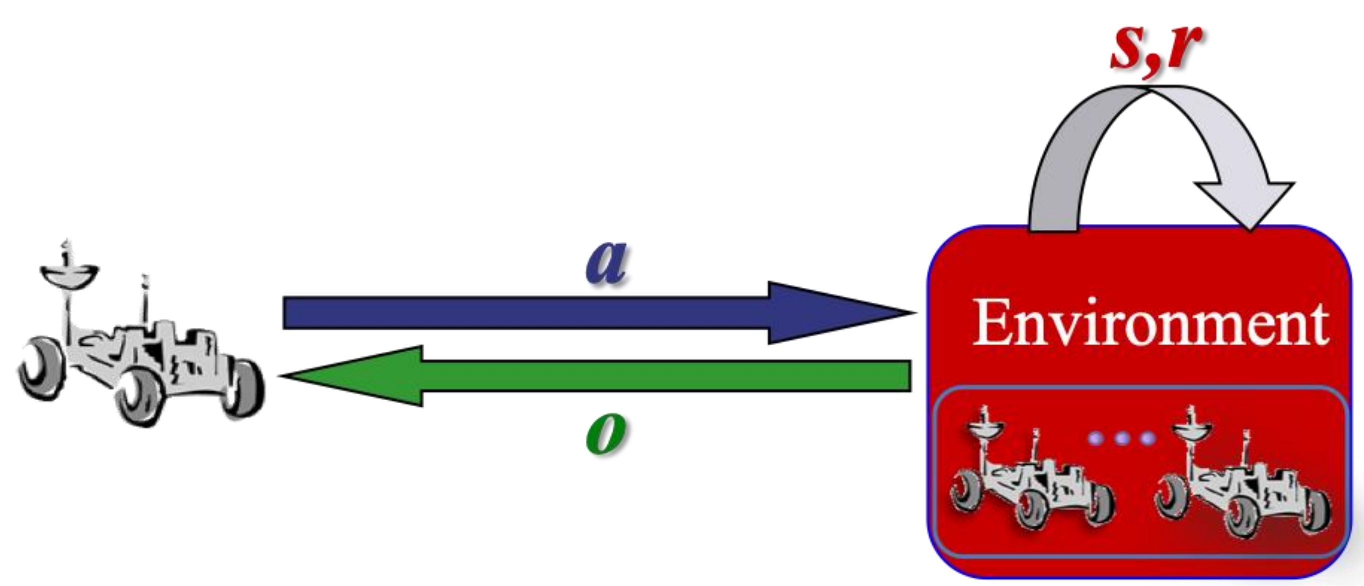
- each agent, i, observes its current observation, o_i, and takes action a_i at the resulting history, h_i,
- the (decentralized) algorithm/controller sees the same information (o_i and a_i) as well as the joint reward r.

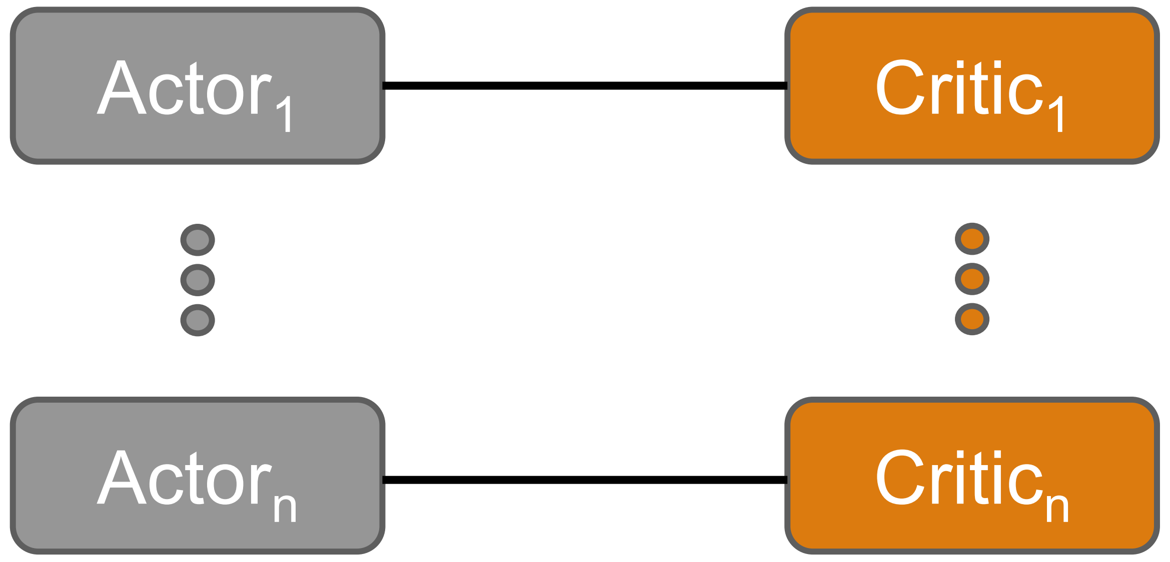




Decentralized MARL

- Agents each learn separately
  - Assumes training and execution are decentralized (e.g., lack of communication)
  - Is more scalable
  - The realistic case for POSGs and online learning in Dec-POMDPs
- Each agent *i* learns a policy that maps from *local histories* to *local actions* π_i: $H_i \rightarrow A_i$
- Can also use any single-agent method here
  - May be nonstationarity but there are many methods for dealing with that
  - Many improvements: Distributed Q, ICML-00; Hysteretic Q, IROS-07, ICML-17; Lenient Q JMLR-08, AAMAS-18; Likelihood Q, AAMAS-20; IPPO arxiv-20





Decentralized Action-Value Methods

IQL, Distributed Q, Hysteretic Q, Lenient Q
Deep extensions

Note: these methods were originally developed for the fully observable case

Independent Q-Learning (IQL) [Tan – ICML 93]

- Just apply Q-learning pretending the other agents don't exist

Algorithm 1 Independent Q-Learning for agent i (finite-horizon)

set α and ϵ (learning rate, exploration)
Initialize Q_i for all $h_i \in \mathbb{H}_i, a_i \in \mathbb{A}_i$
for all episodes **do**
  $h_i \leftarrow \emptyset$ {Empty initial history}
  for $t = 1$ to $\mathcal{H}$ **do**
    Choose a_i at h_i from $Q_i(h_i, \cdot)$ with exploration (e.g., ϵ-greedy)
    See joint reward r, local observation o_i {Depends on joint action $\mathbf{a}$}
    $h_i' \leftarrow h_i a_i o_i$
    $Q_i(h_i, a_i) \leftarrow Q_i(h_i, a_i) + \alpha \left[r + \gamma \max_{a_i'} Q_i(h_i', a_i') - Q_i(h_i, a_i)\right]$
    $h_i \leftarrow h_i'$
  end for
end for
return Q_i

Independent Q-Learning (IQL) [Tan – ICML 93]

- Just apply Q-learning pretending the other agents don't exist
- Where do the observations and joint rewards come from?

Algorithm 1 Independent Q-Learning for agent i (finite-horizon)

set α and ϵ (learning rate, exploration)
Initialize Q_i for all $h_i \in \mathbb{H}_i, a_i \in \mathbb{A}_i$ $\quad P(\boldsymbol{o}|s', \boldsymbol{a}) \quad P(s'|s, \boldsymbol{a})$
for all episodes **do**
    $h_i \leftarrow \emptyset$ $\quad$ {Empty initial history}
    for $t = 1$ to $\mathcal{H}$ **do**
$R(s, \boldsymbol{a})$ Choose a_i at h_i from $Q_i(h_i, \cdot)$ with exploration (e.g., ϵ-greedy)
        See joint reward r, local observation o_i $\quad$ {Depends on joint action $\mathbf{a}$}
        $h'_i \leftarrow h_i a_i o_i$
        $Q_i(h_i, a_i) \leftarrow Q_i(h_i, a_i) + \alpha \left[r + \gamma \max_{a'_i} Q_i(h'_i, a'_i) - Q_i(h_i, a_i) \right]$
        $h_i \leftarrow h'_i$
    end for
end for
return Q_i

Important hidden information

- Agents don't exist by themselves!
- Assumes other agents are acting according to some (fixed) policies
- Then learns as if in a POMDP where other agents are part of the environment:

$$Q_i(h_i, a_i) = \sum_{\mathbf{a} \in \mathbb{A}} \hat{P}(\mathbf{a}, \mathbf{h} | h_i, a_i) \left[r + \gamma \sum_{o_i} \hat{P}(o_i | \mathbf{h}, \mathbf{a}) \max_{a_i'} Q_i(h_i', a_i') \right]$$

- This is where non-stationarity comes from!
  - Other learning agents change their policies over time

$\hat{P}$s are empirical probabilities from data during training

IQL properties

- IQL may not converge (Tan ICML 93)
- Convergence properties of Q-learning in Dec-POMDPs is an open question!
- Usually performs poorly (often used as a baseline)
- Note even with optimal Q-values, agents may not select the optimal action without coordination when multiple actions are optimal (like equilibrium selection)

$$Q_1(h_1, a_1^1) = Q_1(h_1, a_1^2) \qquad Q_2(h_2, a_2^1) = Q_2(h_2, a_2^2)$$

$$Q(h_1, h_2, a_1^1, a_2^2) = Q(h_1, h_2, a_1^2, a_2^1) < Q(h_1, h_2, a_1^2, a_2^2) = Q(h_1, h_2, a_1^1, a_2^1)$$

Extension to the deep case - IDRQN

[Tampuu et al. – Plos one 17]

- Just DRQN applied to the multi-agent case
- Still needs other agents to act

Algorithm 2 Independent DRQN (IDRQN) for agent i (finite-horizon*)

set α, ϵ, and C (learning rate, exploration, and target update frequency)
Initialize network parameters θ and θ^- for Q_i
$\mathcal{D}_i \leftarrow \emptyset$
$e \leftarrow 1$ {episode index}
for all episodes **do**
    $h_i \leftarrow \emptyset$ {initial history is empty}
    for $t = 1$ to $\mathcal{H}$ **do**
        Choose a_i at h_i from $Q_i^\theta(h_i, \cdot)$ with exploration (e.g., ϵ-greedy)
        See joint reward r, local observation o_i {Depends on joint action $\mathbf{a}$}
        append a_i, o_i, r to $\mathcal{D}_i^e$
        $h_i \leftarrow h_i a_i o_i$ {update RNN state of the network}
    end for
    sample an episode from $\mathcal{D}$ Based on other agents
    for $t = 1$ to $\mathcal{H}$ **do**
        $h_i \leftarrow \emptyset$
        $a_i, o_i, r \leftarrow \mathcal{D}_i^e(t)$
        $h_i' \leftarrow h_i a_i o_i$
        $y = r + \gamma \max_{a_i'} Q_i^{\theta^-}(h_i', a_i')$
        Perform gradient descent on parameters θ with learning rate α and loss: $\left(y - Q_i^\theta(h_i, a_i)\right)^2$
        $h_i \leftarrow h_i'$
    end for
    if $e \mod C = 0$ **then**
        $\theta^- \leftarrow \theta$
    end if
    $e \leftarrow e + 1$
end for
return Q_i

Extension to the deep case - IDRQN [Tampuu et al. – Plos one 17]

- Just DRQN applied to the multi-agent case
- Still needs other agents to act
- Independent buffers cause poor performance (non-stationarity)

Algorithm 2 Independent DRQN (IDRQN) for agent i (finite-horizon*)

set α, ϵ, and C (learning rate, exploration, and target update frequency)
Initialize network parameters θ and θ^- for Q_i
$\mathcal{D}_i \leftarrow \emptyset$
$e \leftarrow 1$ {episode index}
for all episodes **do**
  $h_i \leftarrow \emptyset$ {initial history is empty}
  for $t = 1$ to $\mathcal{H}$ **do**
    Choose a_i at h_i from $Q_i^\theta(h_i, \cdot)$ with exploration (e.g., ϵ-greedy)
    See joint reward r, local observation o_i {Depends on joint action $\mathbf{a}$}
    append a_i, o_i, r to $\mathcal{D}_i^e$
    $h_i \leftarrow h_i a_i o_i$ {update RNN state of the network}
  end for
  sample an episode from $\mathcal{D}$
  for $t = 1$ to $\mathcal{H}$ **do**
    $h_i \leftarrow \emptyset$
    $a_i, o_i, r \leftarrow \mathcal{D}_i^e(t)$
    $h_i' \leftarrow h_i a_i o_i$
    $y = r + \gamma \max_{a_i'} Q_i^{\theta^-}(h_i', a_i')$
    Perform gradient descent on parameters θ with learning rate α and loss: $\left(y - Q_i^\theta(h_i, a_i)\right)^2$
    $h_i \leftarrow h_i'$
  end for
  if $e \mod C = 0$ **then**
    $\theta^- \leftarrow \theta$
  end if
  $e \leftarrow e + 1$
end for
return Q_i

Based on other agents

Decentralized MARL (Dec-HDRQN)

Omidshafiei, Pazis, Amato, How and Vian - ICML 17

- Traditional Q-learning: estimate Q-value with (x can be state, observation or history)

$$Q(x,a) \leftarrow Q(x,a) + \alpha\delta$$
$$\delta = Q(x,a) - (r + \gamma \max_{a'} Q(x',a'))$$

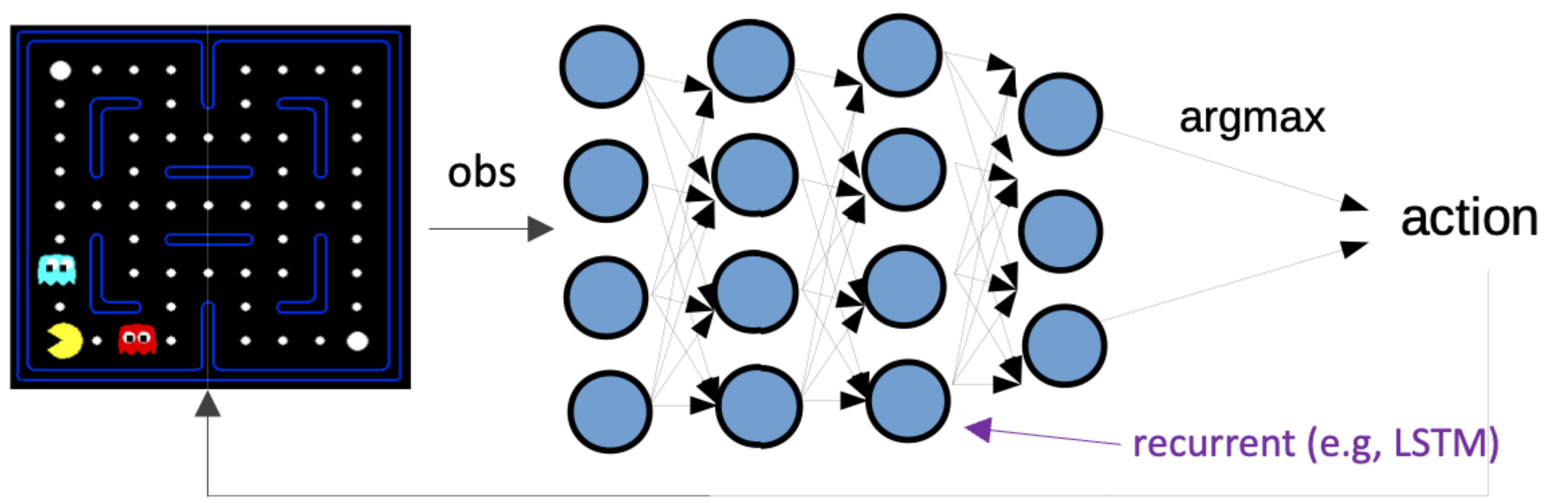
For learning rate α

- Hysteresis (Matignon et al., IROS 07): two learning rates $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (with $\boldsymbol{\beta} < \boldsymbol{\alpha}$)

$$Q(x,a) \leftarrow \begin{cases} Q(x,a) + \beta\delta & \text{if } \delta \le 0 \\ Q(x,a) + \alpha\delta & \text{otherwise} \end{cases}$$

Helps with coordination

- Still use DRQN (Hausknecht and Stone, arXiv 15) if partially observable



Helps with scalability

Helps with partial observability

Local history

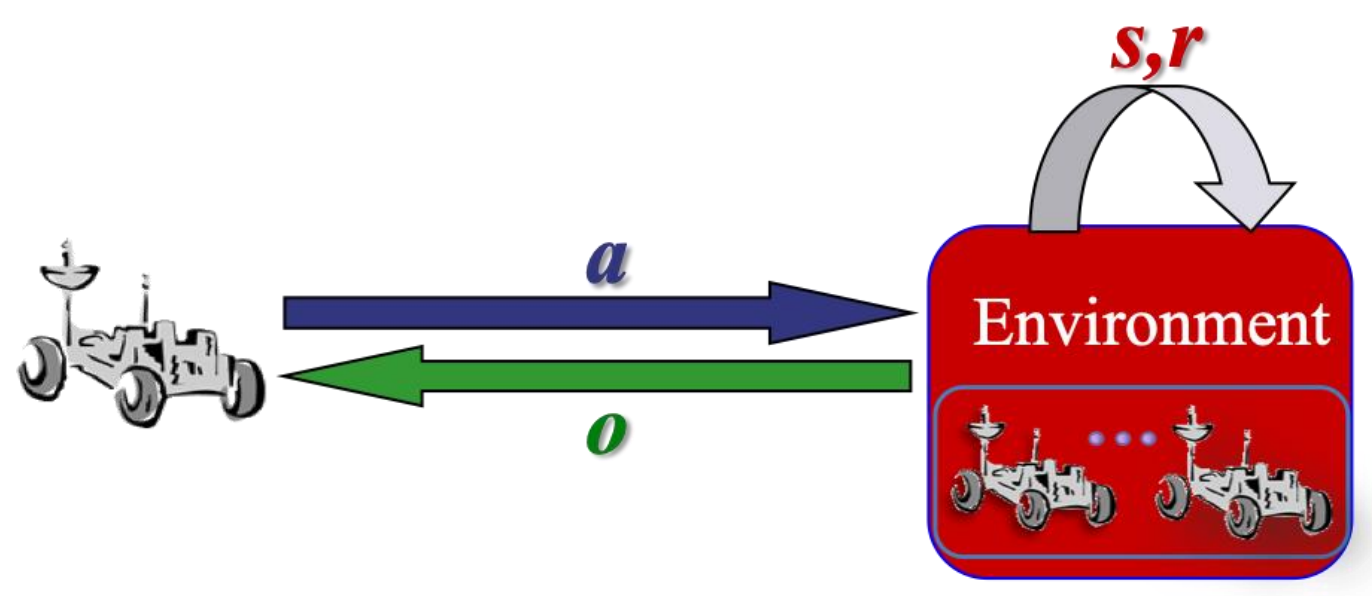
$$Q(h_i\, a_i) \leftarrow Q(h_i, a_i) + \alpha\delta$$

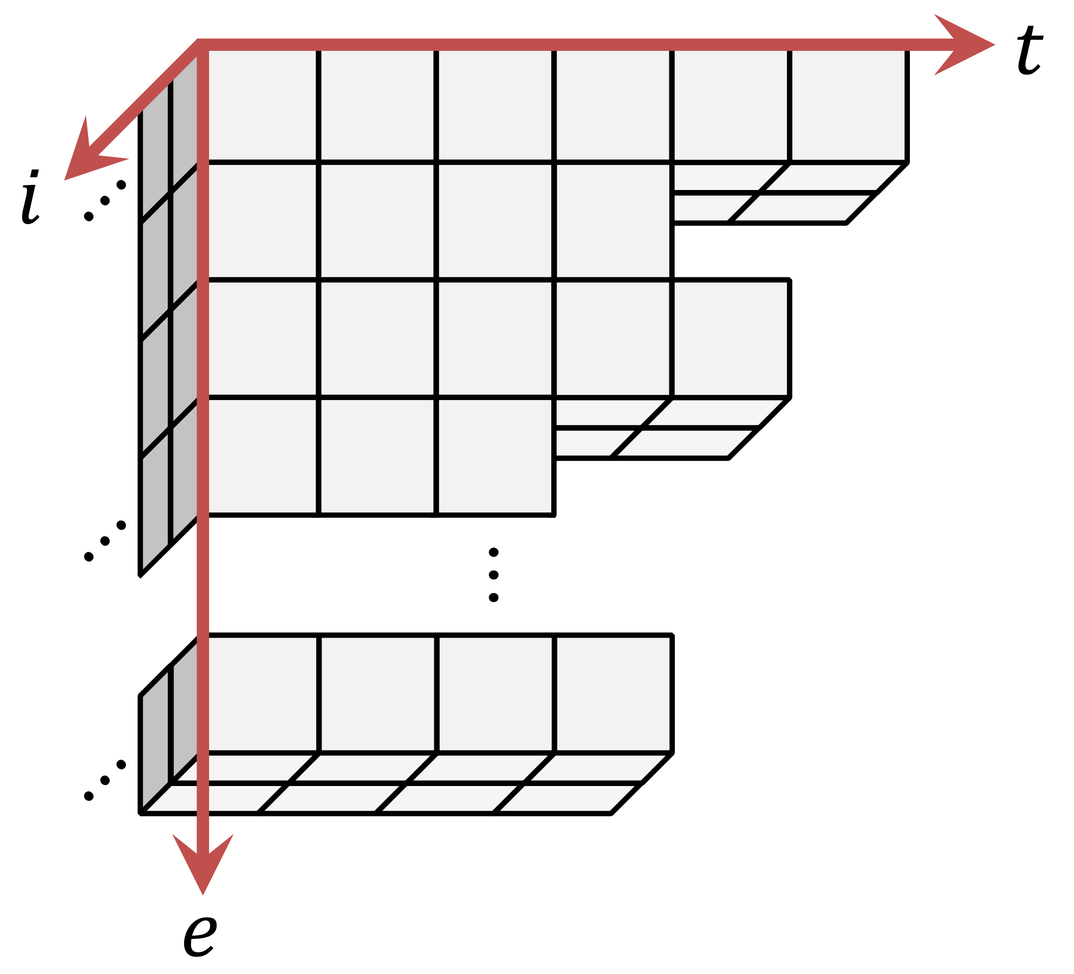
$$\delta = Q(h_i, a_i) - \left(r + \gamma \max_{a_i'} Q(h_i, a_i')\right)$$

Decentralized Hysteretic DQN (Dec-HDRQN)

[Omidshafiei, Pazis, Amato, How and Vian - ICML 17]

- Dec-HDRQN algorithm overview
  - Use idea from previous slide to help with cooperation, scalability and partial observability
  - Each agent learns ***concurrently*** (not independently)
  - Use decentralized Concurrent Experience Replay Trajectories (CERTs) (synchronized buffers) to stabilize learning
- Current decentralized methods (e.g., IPPO) also use some form of concurrent learning





Other deep decentralized methods

- Several other extensions of tabular and single agent methods
- Deep lenient Q-learning (Palmer et al. AAMAS 18)
  - Only for the fully observable case
  - Add leniency values to the replay buffer $(s_t, a_t, r_t, s_{t+1}, l(s_t, a_t))$ for $l(s_t, a_t) = 1 - e^{-K * T(\phi(s_t), a_t)}$
- Likelihood Q-learning (Lyu et al. AAMAS 20)
  - Uses distributional RL to estimate when other agents are exploring and use that info to adjust learning rate

Decentralized Policy Gradient Methods

Decentralized REINFORCE, IAC, IPPO

Decentralized REINFORCE [Peshkin et al. – UAI 00]

- Extends single agent REINFORCE (Williams 92)
- Simple but has convergence guarantees!
  - joint gradient can be decomposed into decentralized gradients
  - I.e., this algorithm converges to the same values as a centralized algorithm (over decentralized policies)
  - Assumes concurrent learning

Algorithm 3 Decentralized REINFORCE for agent i (finite-horizon)

Require: Individual actor models $\pi_i(a_i|h_i)$, parameterized by ψ_i

Policy but no value function

```
set α (learning rate)
for all episodes do
    h_{i,0} ← ∅                                          {Empty initial history}
    ep ← ∅                                               {Empty episode}
    for t = 0 to H − 1 do
        Choose a_{i,t} at h_{i,t} from π_i(a_i|h_{i,t})
        See joint reward r_t, local observation o_{i,t}  {Depends on joint action a}
        append a_{i,t}, o_{i,t}, r_t to ep
        h_{i,t+1} ← h_{i,t} a_{i,t} o_{i,t}              {Append new action and obs to previous history}
    end for
    for t = 0 to H − 1 do
        Compute return at t from ep: G_{i,t} ← Σ_{k=t}^{H−1} γ^{k−t} r_k
        Update parameters: ψ_i ← ψ_i + α γ^t G_{i,t} ∇ log π_i(a_i|h_{i,t})
    end for
end for
```

Based on other agents

Monte Carlo returns

Key steps: $h_{i,0} \leftarrow \emptyset$; $G_{i,t} \leftarrow \sum_{k=t}^{\mathcal{H}-1} \gamma^{k-t} r_k$; $\psi_i \leftarrow \psi_i + \alpha\gamma^t G_{i,t} \nabla \log \pi_i(a_i|h_{i,t})$

Note: this version generalizes the original algorithm which was defined for finite-state controllers

Independent actor critic (IAC) [Foerster et al. – AAAI 18]

- Extends Decentralized REINFORCE to the Actor Critic case

Policy and value model

Algorithm 4 Independent Actor-Critic (IAC) (finite-horizon)

Require: Individual actor models $\pi_i(a_i|h_i)$, parameterized by ψ_i
Require: Individual critic models $\hat{V}_i(h)$, parameterized by θ_i

```
for all episodes do
    h_{i,0} ← ∅                                                    {Empty initial history}
    for t = 0 to H − 1 do
        Choose a_{i,t} at h_{i,t} from π_i(a_i|h_{i,t})
        See joint reward r_t, local observation o_{i,t}            {Depends on joint action a}
        h_{i,t+1} ← h_{i,t} a_{i,t} o_{i,t}                          {Append new action and obs to previous history}
        Compute value TD error: δ_{i,t} ← r_t + γ V̂_i(h_{i,t+1}) − V̂_i(h_{i,t})     On-policy error
        Compute actor gradient estimate: γ^t δ_{i,t} ∇ log π_i(a_{i,t}|h_{i,t})
        Update actor parameters ψ_i using gradient estimate (e.g., ψ_i ← ψ_i + α γ^t δ_{i,t} ∇ log π_i(a_{i,t}|h_{i,t}))
                                                                     Update both models
        Compute critic gradient estimate: δ_{i,t} ∇ V̂_i(h_{i,t})
        Update critic parameters θ_i using gradient estimate (e.g., θ_i ← θ_i + β γ δ_{i,t} ∇ V̂_i(h_{i,t}))
    end for
end for
```

Where the listing prints them as formulas: $h_{i,0} \leftarrow \emptyset$; for $t = 0$ to $\mathcal{H} - 1$; $h_{i,t+1} \leftarrow h_{i,t}a_{i,t}o_{i,t}$; $\delta_{i,t} \leftarrow r_t + \gamma\hat{V}_i(h_{i,t+1}) - \hat{V}_i(h_{i,t})$; $\gamma^t\delta_{i,t}\nabla\log\pi_i(a_{i,t}|h_{i,t})$; $\psi_i \leftarrow \psi_i + \alpha\gamma^t\delta_{i,t}\nabla\log\pi_i(a_{i,t}|h_{i,t})$; $\delta_{i,t}\nabla\hat{V}_i(h_{i,t})$; $\theta_i \leftarrow \theta_i + \beta\gamma\delta_{i,t}\nabla\hat{V}_i(h_{i,t})$

Other decentralized PG methods

- Can extend any single-agent PG method to the multi-agent case
- Independent PPO (IPPO) (de Witt et al. 20)
  - A version of IAC with PPO as the base RL method
  - Yu et al. (22) version uses parameter sharing (not DTE)
  - More about IPPO and MAPPO in the CTDE discussion
- Not a very active area

Other topics

- **Parameter sharing**
  - Agents share the same copy of policy and/or value networks
  - I consider this a form of CTDE (since it assumes centralized info)
  - Decentralized methods can easily use parameter sharing to potentially improve performance
- **Relationship with CTDE**
  - Centralized PG equal to decentralized PG so maybe not that different?
- **Other forms of decentralization**
  - Communication during execution using ‘networked’ agents, e.g., (Zhang et al. 18)

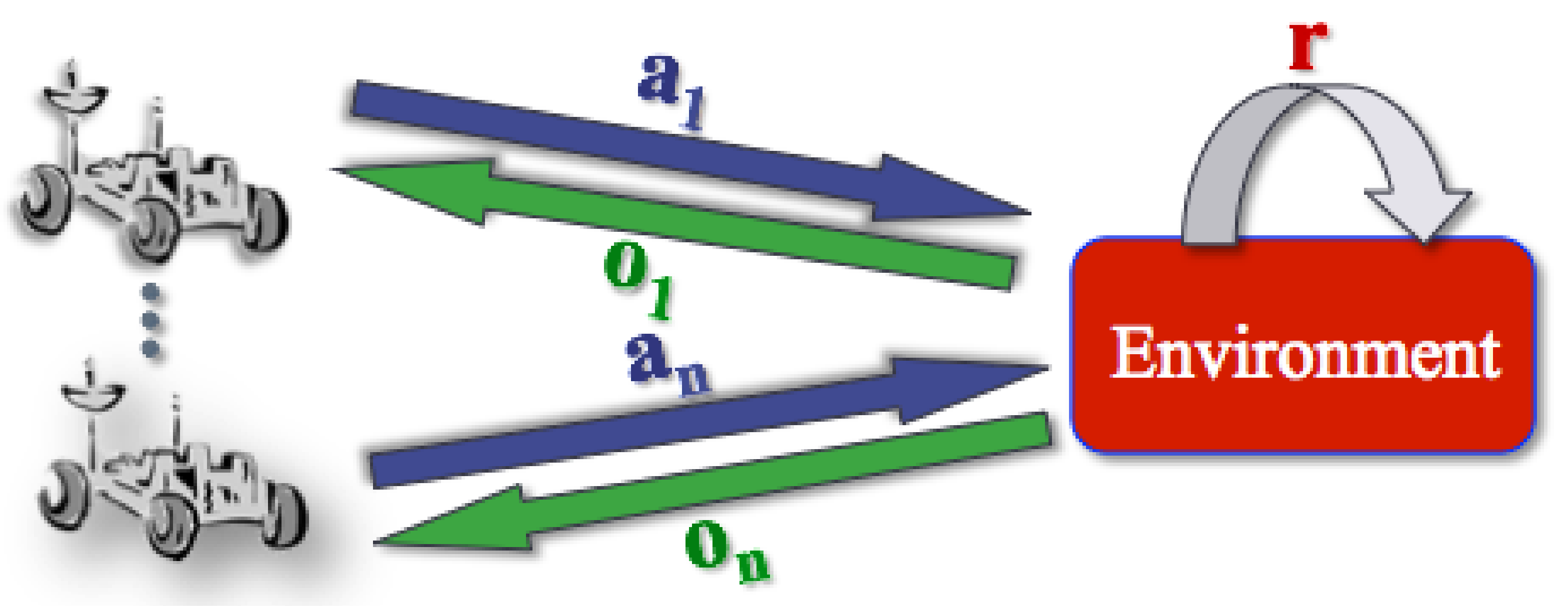
Centralized Training for Decentralized Execution (CTDE) MARL

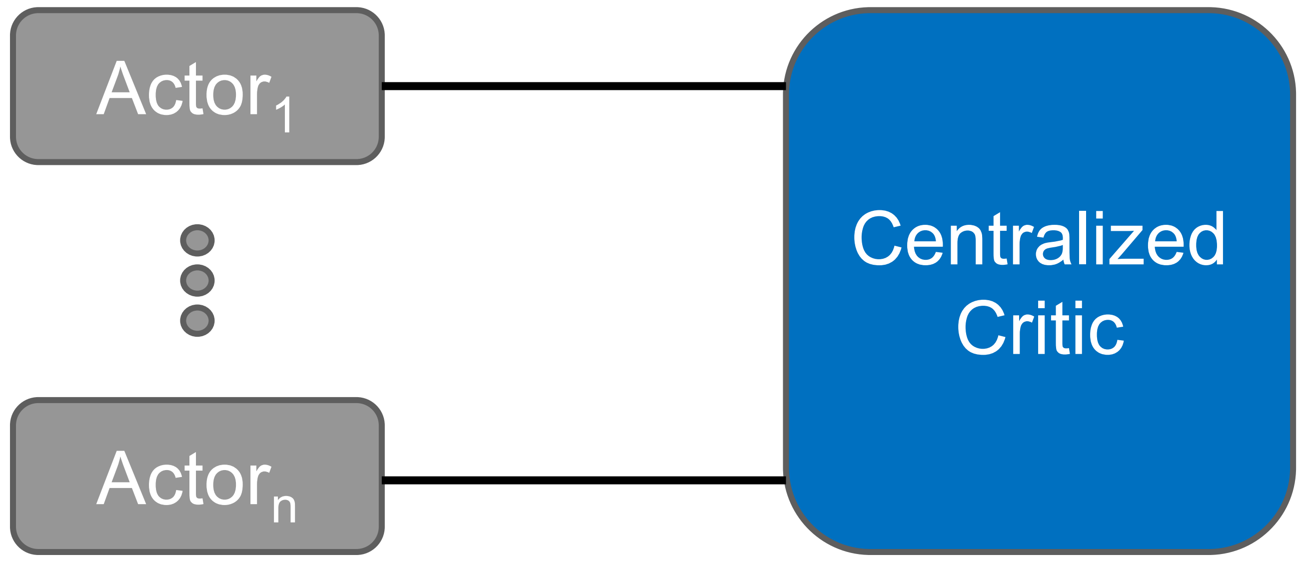
Models and methods

Centralized training for decentralized execution (CTDE)

Assumptions

- each agent, i, observes its current observation, o_i, and takes action a_i at the resulting history, h_i, like DTE
- the (centralized) algorithm/controller observes joint information $\boldsymbol{o}$ and $\boldsymbol{a}$ and the joint reward r (and possibly other information such as the underlying state s) like CTE

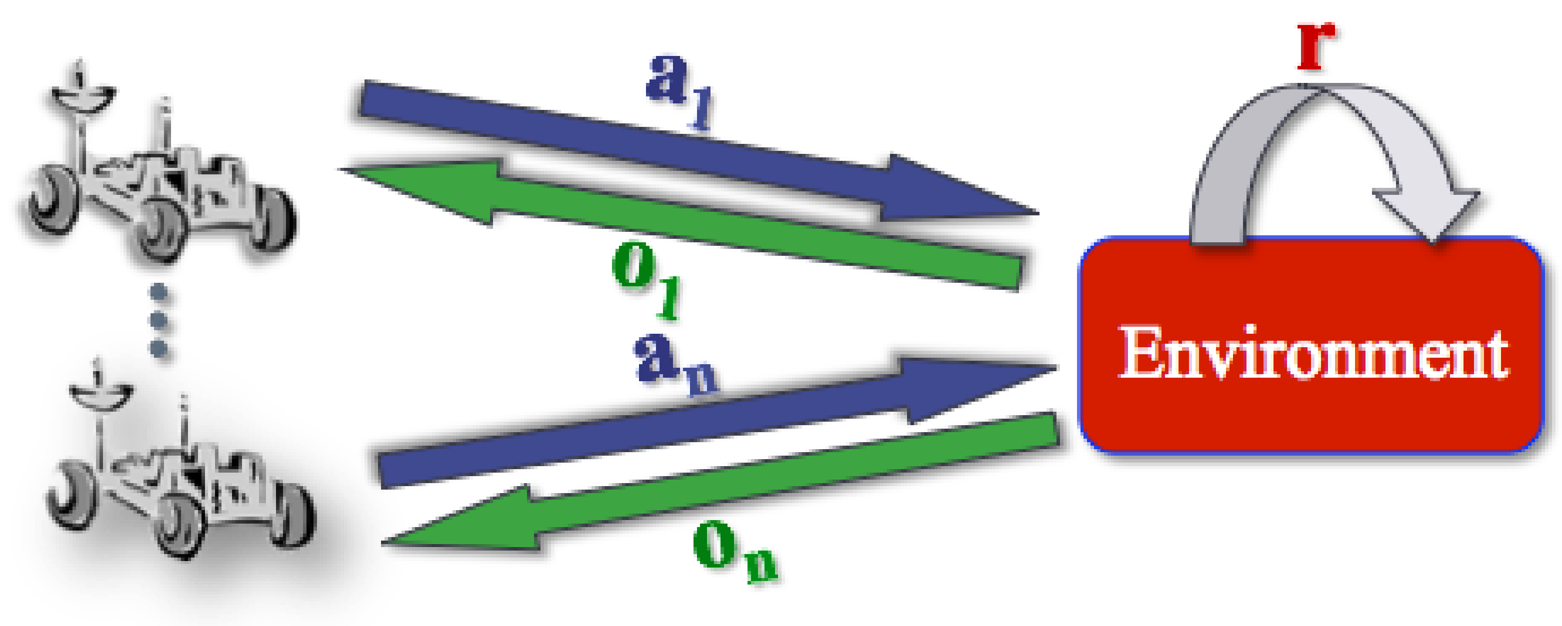


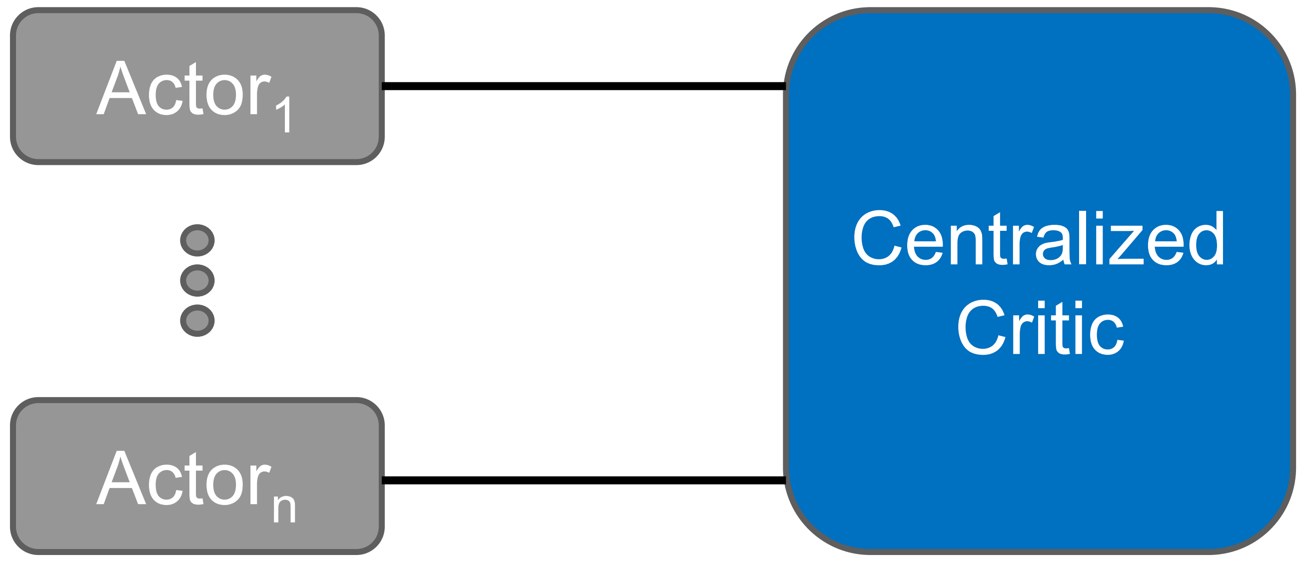


By far the most common type of (cooperative) MARL

Centralized training for decentralized execution (CTDE)

- Train offline for online execution
- Can use centralized info offline
- Still need to execute in a decentralized manner
- CTDE has become the dominant form of (cooperative) MARL
- Many methods: MADDPG, NeurIPS-17; COMA AAAI-18; QMIX, ICML-18; QPLEX, ICML-21; MAPPO, NeurIPS DB-22



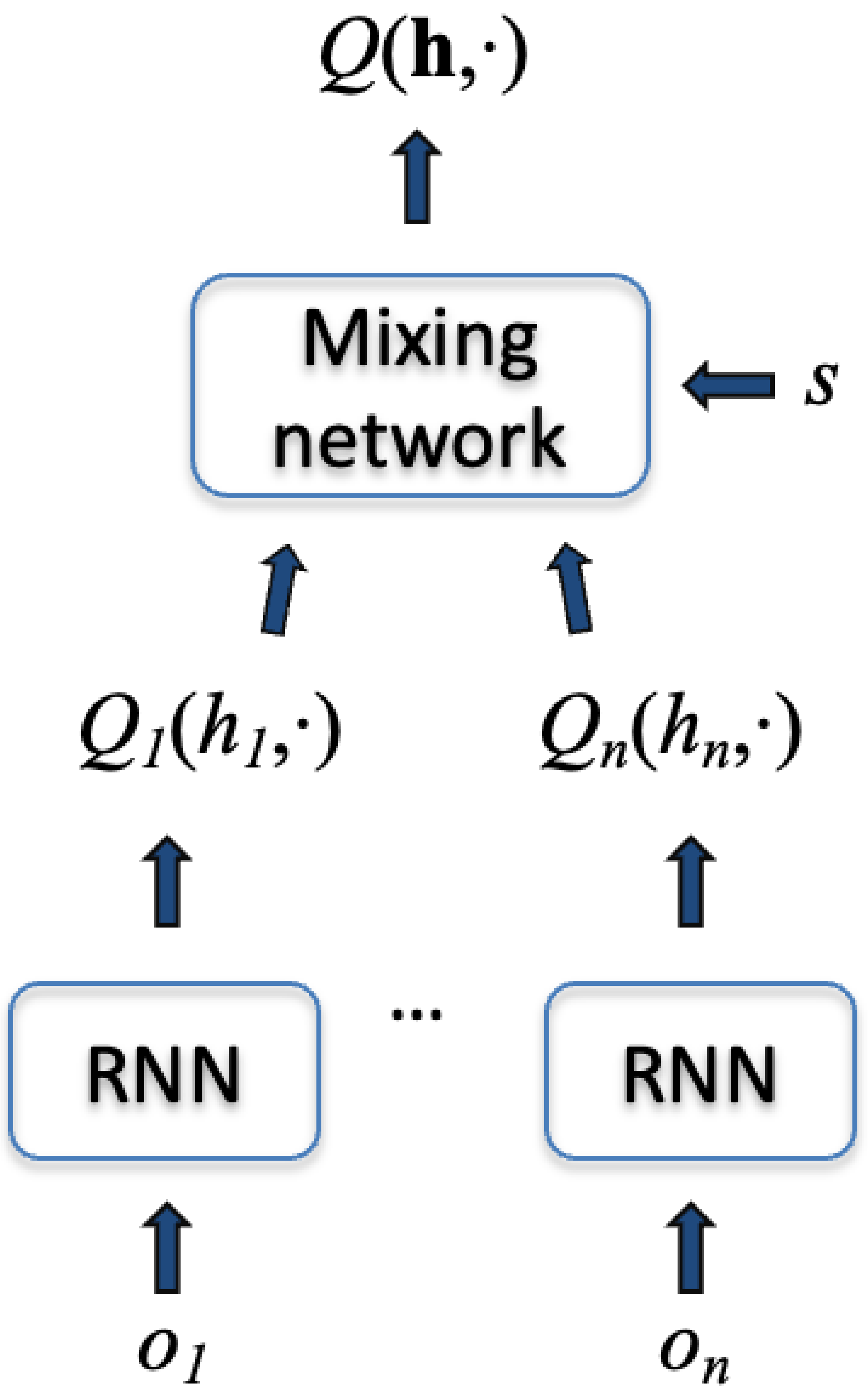


CTDE Action-Value Methods

Value function factorization: VDN, QMIX, and QPLEX

Value function factorization methods

- Basic idea:
  - Learn individual Q-values per agent as well as a form of joint Q-function
  - During training, learn individual Q-values from joint one
  - During execution, each agent uses individual Q-values to select actions



Value decomposition networks (VDN)

Sunehag et al. – arXiv 17

- The first deep value function factorization/decomposition method

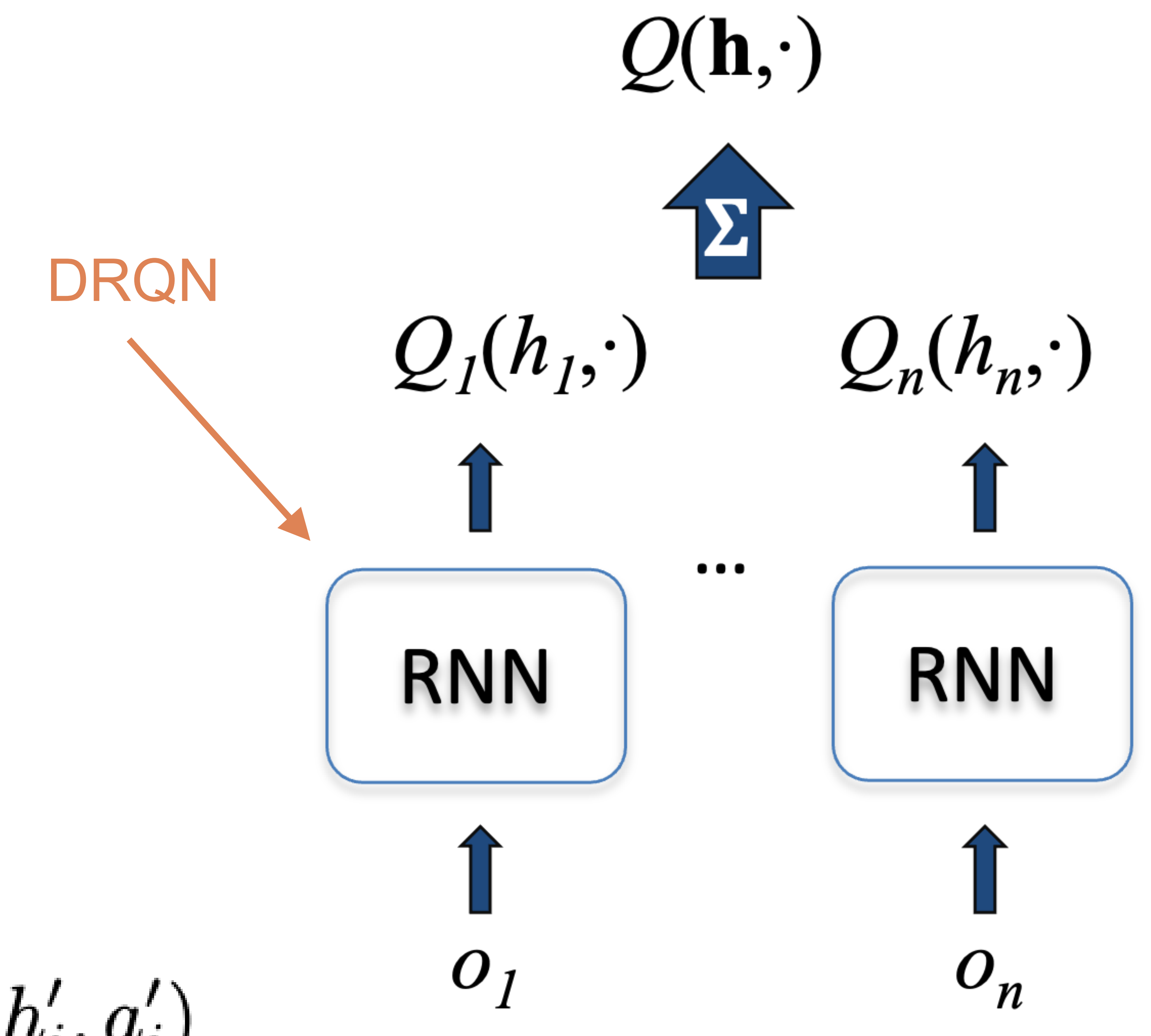
- Represents joint Q-value as a sum of individual Q-values:

$$\mathbf{Q}(\mathbf{h},\mathbf{a}) \approx \sum_{i \in \mathbb{I}}^{n} Q_i(h_i, a_i)$$

- Trains solely based on (joint) RL loss

$$\mathcal{L}(\theta) = \mathbb{E}_{<\mathbf{h},\mathbf{a},r,\mathbf{o}>\sim\mathcal{D}}\left[\left(y - \sum_{i}^{n} Q_i^{\theta}(h_i, a_i)\right)^2\right], \text{ where } \; y = r + \gamma \sum_{i}^{n} \max_{a_i'} Q_i^{\theta^-}(h_i', a_i')$$

- Simple, scalable, but limited joint Q-value representation



QMIX [Rashid et al. – ICML 18](#)

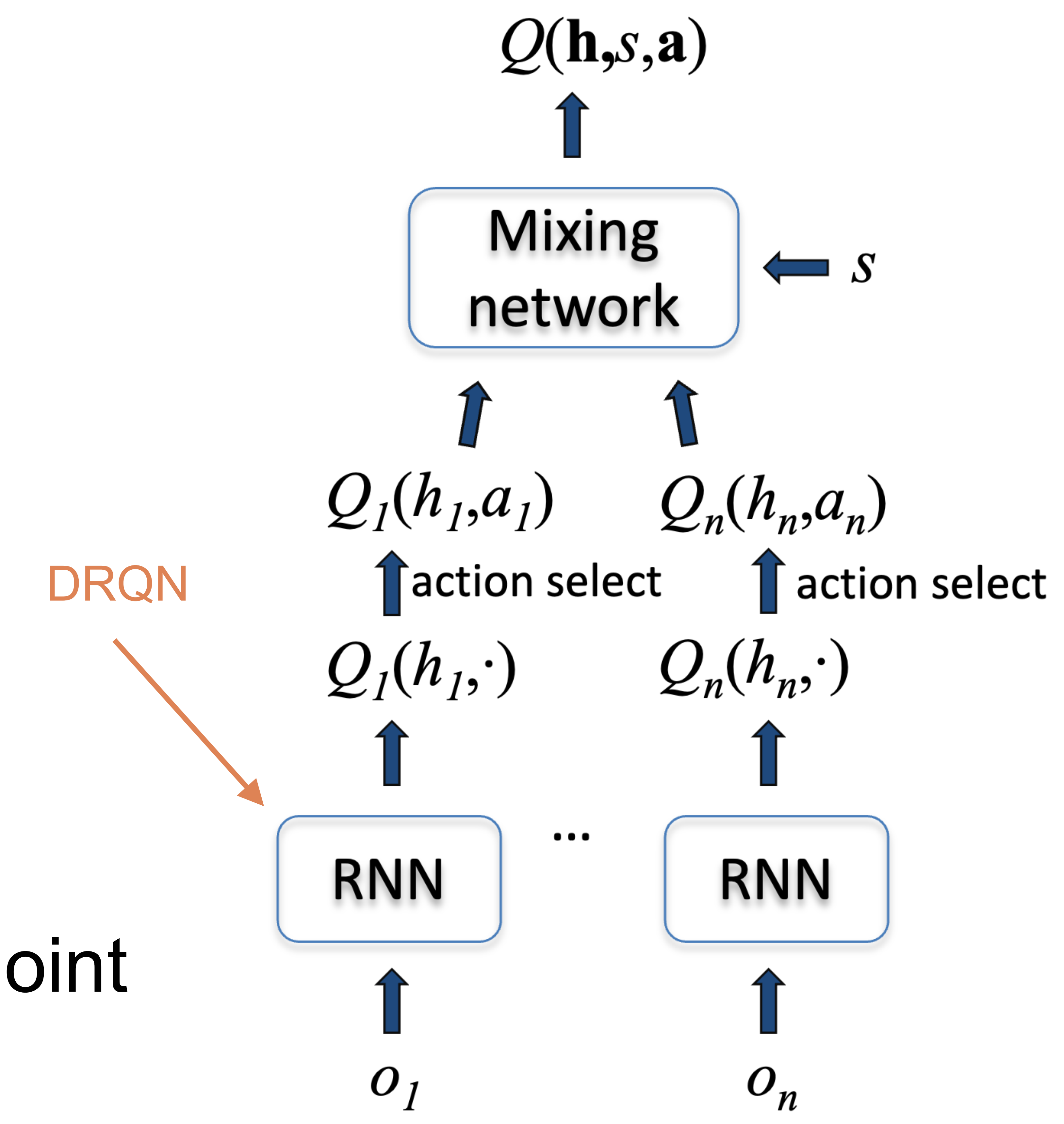
- Extends VDN to represent monotonic functions

$$\mathbf{Q}(\mathbf{h},\mathbf{a}) \approx f_{mono}(Q_i(h_1,a_1),\ldots,Q_n(h_n,a_n))$$

- (implemented with positive weights in mixer)
- Also, use state as input to mixer (with hypernetwork)
- Still argmax over indiv. Q-functions and train based on the joint loss

$$\mathcal{L}(\theta) = \mathbb{E}_{<\mathbf{h},s,\mathbf{a},r,\mathbf{o},s'>\sim\mathcal{D}}\left[\left(y - \mathbf{Q}^{\theta}(\mathbf{h},s,\mathbf{a})\right)^2\right], \text{where } y = r + \gamma \mathbf{Q}^{\theta^-}(\mathbf{h}',s',\tilde{\mathbf{a}}'),$$
$$\text{and } \tilde{\mathbf{a}}' = \langle \operatorname*{argmax}_{a'_1} Q_1(h'_1,a'_1),\ldots,\operatorname*{argmax}_{a'_n} Q_n(h'_n,a'_n)\rangle$$

- Can't represent all Q-functions but still a state-of-the-art method



Individual Global-Max (IGM) Son et al.– ICML 19 (QTRAN)

Definition: Individual-Global-Max

For a joint action-value function **Q**(**h**,**a**) where **h**$= \langle h_1, \ldots, h_n \rangle$ is a joint action-observation history, if there exist individual functions $[Q_i]$ such that:

$$\underset{\mathbf{a}}{\operatorname{argmax}}\, \mathbf{Q}(\mathbf{h}, \mathbf{a}) = \begin{pmatrix} \operatorname{argmax}_{a_1} Q_1(h_1, a_1) \\ \vdots \\ \operatorname{argmax}_{a_n} Q_n(h_n, a_n) \end{pmatrix}$$

Then $[Q_i]$ satisfy IGM for **Q** at **h**

- This is the main principle of value factorization/decomposition methods: the argmax of the joint value function is the same as the argmax of the individual Q-functions
- VDN and QPLEX satisfy this (as do QTRAN, QPLEX, etc.)

QPLEX Wang et al.– ICLR 21

Extends IGM to the advantage case

Definition: Advantage-based IGM

For joint and individual advantages:

$\mathbf{A}(\mathbf{h},\mathbf{a}) = \mathbf{Q}(\mathbf{h},\mathbf{a})-\mathbf{V}(\mathbf{h})$ where $\mathbf{V}(\mathbf{h})= \max_{\mathbf{a}}\mathbf{Q}(\mathbf{h},\mathbf{a})$ and $A_i(h_i,a_i)= Q_i(h_i,a_i)-V_i(h_i)$ where $V_i(h_i)=\max_{a_i}Q_i(h_i,a_i)$

For a joint action-value function $\mathbf{Q}(\mathbf{h},\mathbf{a})$ where $\mathbf{h}= \langle h_1, \dots, h_n \rangle$ is a joint action-observation history, if there exist individual functions $[Q_i]$ such that:

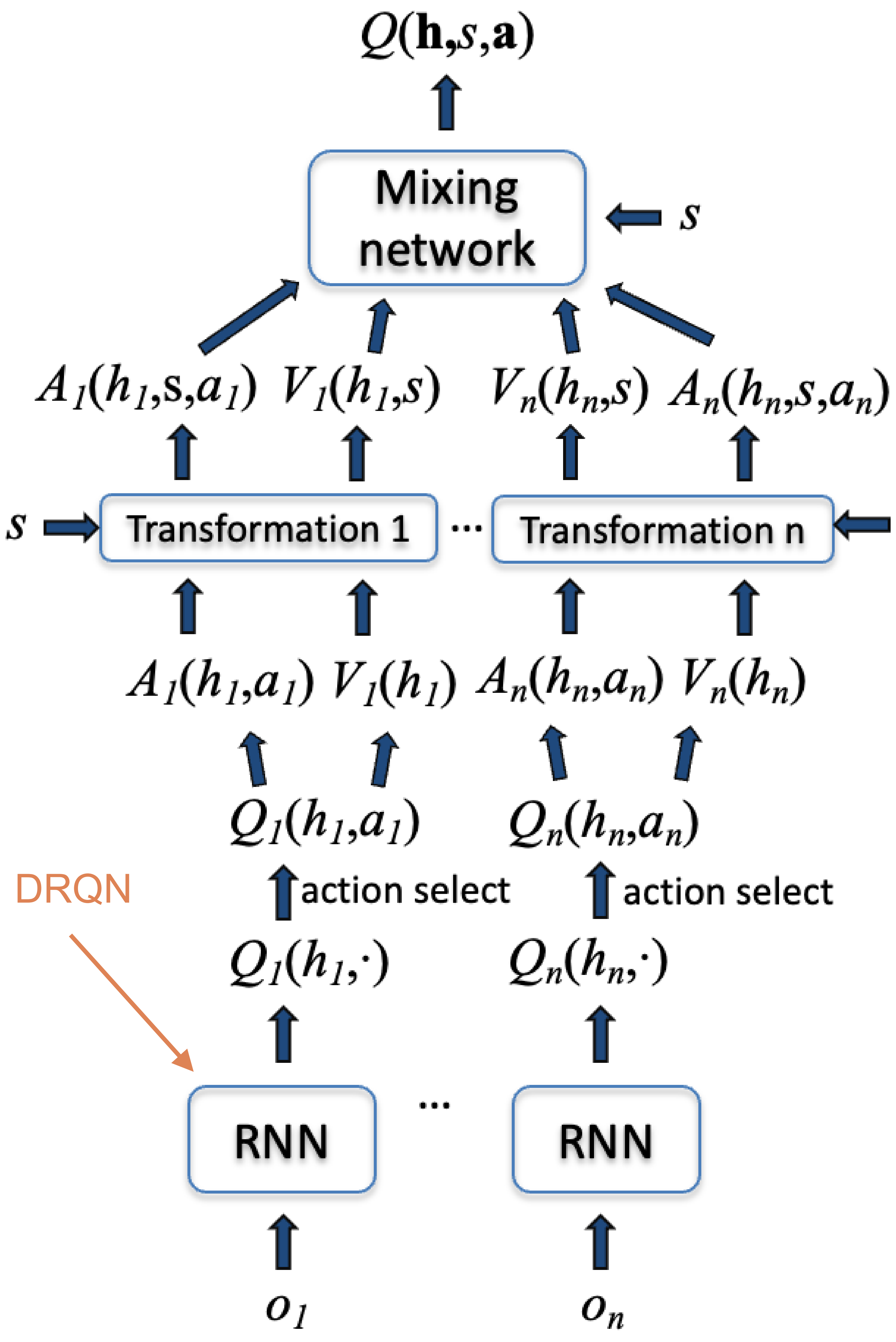
$$\arg\max_{\mathbf{a}} \mathbf{A}(\mathbf{h},\mathbf{a}) = \begin{pmatrix} \arg\max_{a_1} A_1(h_1, a_1) \\ \vdots \\ \arg\max_{a_n} A_n(h_n, a_n) \end{pmatrix}$$

Then $[Q_i]$ satisfy IGM for $\mathbf{Q}$ at $\mathbf{h}$

- This is subtle but important! Non-standard advantage makes them 0 for optimal action and negative otherwise! Used as a constraint to represent the full IGM function class

QPLEX architecture [Wang et al.– ICLR 21]

- Architecture is a bit complicated but it performs well
- Can sometimes outperform QMIX and is a state-of-the-art method
- Other recent value factorization/decomposition methods but not clear they outperform QMIX and QPLEX

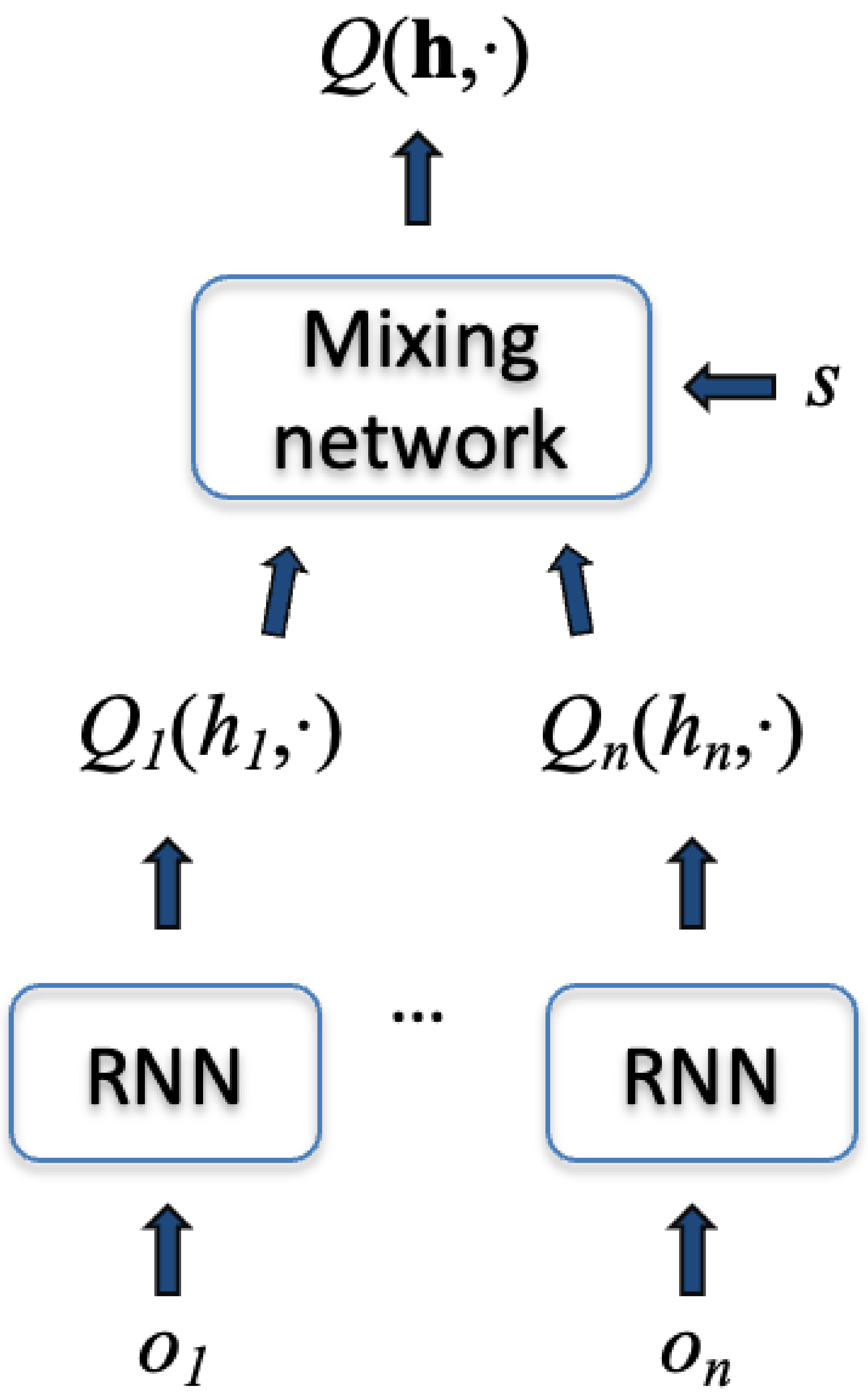


State in value function factorization

Marchesini et al.,--AAMAS 25

- Is it cheating/wrong to use state during training?

- QMIX: Sound since state information gets marginalized out
- QPLEX:
  - Sound since similar to QMIX
  - Less general with state (can't represent all IGM functions)
- Weighted QMIX: Probably not sound as uses separate state-conditioned weights



Note: The paper also introduces a new algorithm DualMIX which I don't discuss here

State in value function factorization

Marchesini et al.,--AAMAS 25

Why is the state helpful?

Benefit of state unclear in theory but may be helpful in practice

Tried the methods with state (s), a random (r) value, or a 0 value

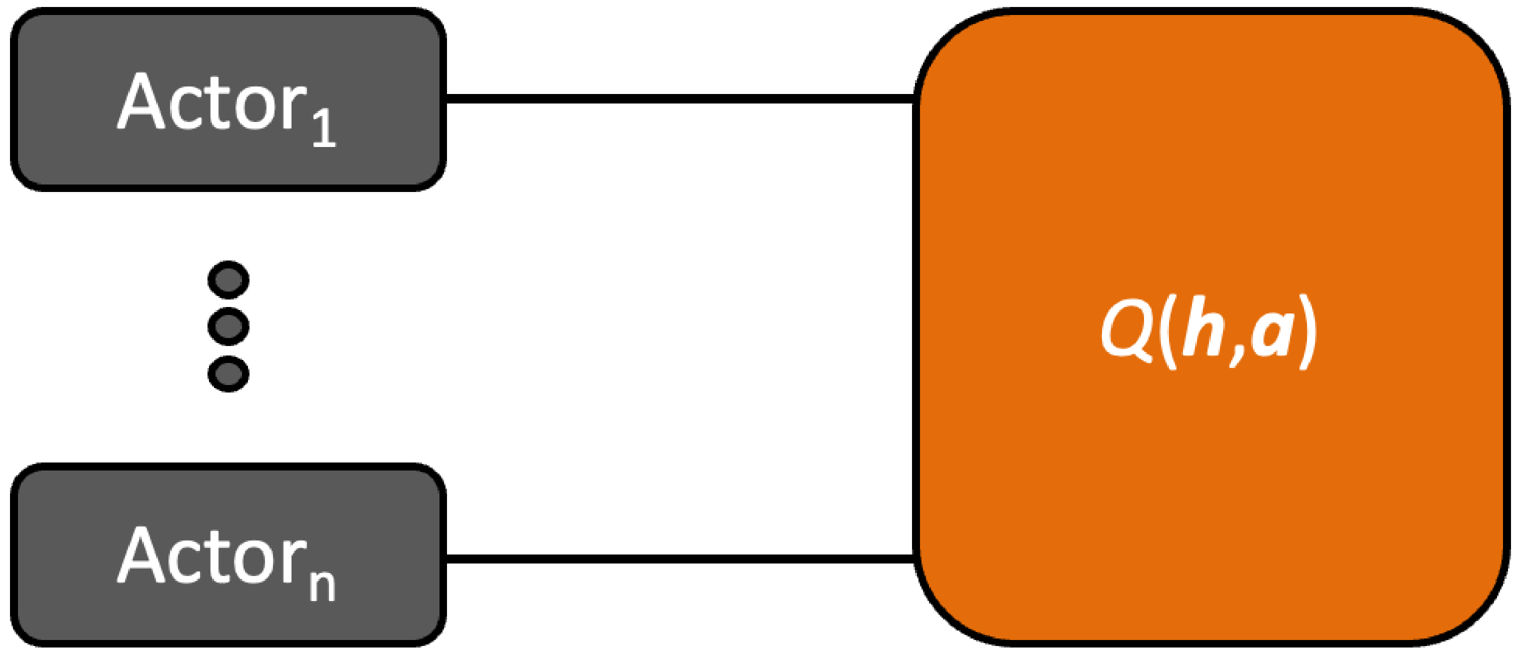
Other information can outperform state info!

(fine-tuned $\downarrow$)		**5s10z**
QMIX	s	$\mathbf{15.8 \pm 0.4}$
	r	14.5 ± 1.4
	c	14.7 ± 0.1
QPLEX	s	16.2 ± 2.1
	r	18.0 ± 0.6
	c	$\mathbf{18.3 \pm 0.8}$

CTDE Policy Gradient Methods

Centralized critics: MADDPG, COMA, and MAPPO

Actor critic with a centralized critic



- Have an actor for each agent
- Learn a 'centralized' Q-function
- Update each actor using this joint Q-value:

$$\nabla_{\psi_i} J = \mathbb{E}_{<\mathbf{h},\mathbf{a}>\sim\mathcal{D}}\left[\mathbf{Q}^{\pi}(\mathbf{h},\mathbf{a})\nabla_{\psi_i}\log\pi_i(a_i|h_i)\right]$$

- Update the joint Q-value using the joint info:

$$\mathcal{L}(\theta) = \mathbb{E}_{<\mathbf{h},\mathbf{a},r,\mathbf{h}'>\sim\mathcal{D}}\left[\left(y - \hat{\mathbf{Q}}(\mathbf{h},\mathbf{a})\right)^2\right], \text{where } \; y = r + \gamma\hat{\mathbf{Q}}(\mathbf{h}',\mathbf{a}')$$

A basic centralized critic approach

A policy network for each agent

A joint value network

Joint error calculation

The gradient using

$\mathcal{L}(\theta) = \mathbb{E}_{<\mathbf{h},\mathbf{a},r,\mathbf{h}'>\sim\mathcal{D}}\left[\left(y - \hat{\mathbf{Q}}(\mathbf{h},\mathbf{a})\right)^2\right]$, where $y = r + \gamma\hat{\mathbf{Q}}(\mathbf{h}',\mathbf{a}')$

Loop over agents

Use joint Q to update agent policies

Algorithm 6 Independent Actor Centralized Critic (IACC) (finite-horizon)

Initialize individual actor models $\pi_i(a_i|h_i)$, parameterized by ψ_i
Initialize centralized critic model $\hat{\mathbf{Q}}(\mathbf{h},\mathbf{a})$, parameterized by θ
for all episodes **do**
    $h_{i,0} \leftarrow \emptyset$ {Empty initial history}
    Denote $\mathbf{h}_t$ as $\langle h_{1,0},\ldots,h_{n,0}\rangle$ {Notation for joint variables}
    for all i, choose $a_{i,0}$ at $h_{i,0}$ from $\pi_i(a_i|h_{i,0})$
    Store $\mathbf{a}_t$ as $\langle a_{1,0},\ldots,a_{n,0}\rangle$
    for $t = 0$ to $\mathcal{H}-1$ **do**
        Take joint action $\mathbf{a}_t$, see joint reward r_t, and observations $\mathbf{o}_t$
        for all i, $h_{i,t+1} \leftarrow h_{i,t}a_{i,t}o_{i,t}$ {Append new action and obs to previous history}
        for all i, choose $a_{i,t+1}$ at $h_{i,t+1}$ from $\pi_i(a_i|h_{i,t+1})$
        Store $\mathbf{a}_{t+1}$ as $\langle a_{1,t+1},\ldots,a_{n,t+1}\rangle$
        $\delta_t \leftarrow r_t + \gamma\hat{\mathbf{Q}}(\mathbf{h}_{t+1},\mathbf{a}_{t+1}) - \hat{\mathbf{Q}}(\mathbf{h}_t,\mathbf{a}_t)$ {Compute centralized TD error}
        Compute critic gradient estimate: $\delta_t\nabla_\theta\hat{\mathbf{Q}}(\mathbf{h}_t,\mathbf{a}_t)$
        Update critic parameters θ using gradient estimate (e.g., $\theta \leftarrow \theta + \beta\delta_t\nabla_\theta\hat{\mathbf{Q}}(\mathbf{h}_t,\mathbf{a}_t)$ for learning rate β)
        for each agent i **do**
            Compute actor gradient estimate: $\gamma^t\hat{\mathbf{Q}}(\mathbf{h}_t,\mathbf{a}_t)\nabla_{\psi_i}\log\pi_i(a_{i,t}|h_{i,t})$
            Update actor parameters ψ_i using gradient estimate (e.g., $\psi_i \leftarrow \psi_i + \alpha\gamma^t\hat{\mathbf{Q}}(\mathbf{h},\mathbf{a})\nabla_{\psi_i}\log\pi_i(a_{i,t}|h_{i,t})$ for learning rate α)
        end for
    end for
end for

MADDPG [Lowe et al.—NeurIPS 17]

- Designed for competitive or cooperative problems
- Off-policy (so uses reply buffer like DQN)
- Continuous action, so uses a Deterministic PG (Silver et al., ICML-14)

$$\nabla_{\psi_i} J = \mathbb{E}_{x,\mathbf{a}\sim\mathcal{D}}\left[\nabla_{\psi_i}\mu_i(o_i)\nabla_{\mathbf{a}}\mathbf{Q}^{\pi}(x,\mathbf{a})\;|_{a_i=\mu_i(o_i)}\right]$$

- Defined policies based on a single observation but should be:

$$\nabla_{\psi_i} J = \mathbb{E}_{x,\mathbf{a}\sim\mathcal{D}}\left[\nabla_{\psi_i}\mu_i(h_i)\nabla_{\mathbf{a}}\mathbf{Q}^{\pi}(\mathbf{h},\mathbf{a})\;|_{a_i=\mu_i(h_i)}\right]$$

- Learn centralized critic from the reply buffer and using target network θ^-

$$\mathcal{L}(\theta) = \mathbb{E}_{<\mathbf{h},\mathbf{a},r,\mathbf{h}'>\sim\mathcal{D}}\left[\left(y - Q_\theta(\mathbf{h},\mathbf{a})\right)^2\right], \text{ where } \; y = r + \gamma Q_{\theta^-}(\mathbf{h}',\mathbf{a}')\;|_{a_i=\mu^-(h_i)\;\forall i\in\mathbb{I}}$$

- MADDPG is no longer widely used but the centralized critic have been adopted

Note: For the cooperative CTDE case we assume a single shared critic among agents, do not consider learning policy models of the other agents, and do not consider ensembles of other agent policies to improve robustness.

Counterfactual Multi-Agent Policy Gradients (COMA) [Foerster et al.–AAAI 18]

- Centralized critic along with a counterfactual baseline to potentially help with variance and credit assignment

- Calculate a per-agent advantage considering that difference between with the agent did and the expected Q-value from policy and fixing other agents:

$$A_i(\mathbf{h}, \mathbf{a}) = \mathbf{Q}(\mathbf{h}, \mathbf{a}) - \sum_{a'_i} \pi_i(a'_i | h_i) \mathbf{Q}(\mathbf{h}, a'_i, \mathbf{a}_{-i})$$

- Is implemented with agent ids to only require a single centralized critic network (rather than one per agent)

- On-policy so the critic is updated as usual: $\mathcal{L}(\theta) = \mathbb{E}_{<\mathbf{h},\mathbf{a},r,\mathbf{h}'>\sim\mathcal{D}}\left[\left(y - \hat{\mathbf{Q}}(\mathbf{h}, \mathbf{a})\right)^2\right]$, where $y = r + \gamma \hat{\mathbf{Q}}(\mathbf{h}', \mathbf{a}')$

- Policy network update uses A_i instead of Q: $\gamma^t A_i(\mathbf{h}_t, \mathbf{a}_t) \nabla_{\psi_i} \log \pi_i(a_{i,t} | h_{i,t})$

- COMA is also not widely used but very influential

Note: COMA originally used state instead of history in the advantage and Q-values but this is incorrect as I'll discuss later.

MAPPO [Yu et al. -- NeurIPS DB&B 22]

- MAPPO is a form of a centralized critic method
- Just use PPO as the base RL method
- Actor loss: $\mathcal{L}_{clip}^{MAPPO}(\psi_i) = \min\left(r_{\psi_i,i}\mathbf{A}, \text{clip}(r_{\psi_i,i}, 1-\epsilon, 1+\epsilon)\mathbf{A}\right)$
  - Uses joint advantage: $\mathbf{A}(\mathbf{h},\mathbf{a}) = \mathbf{Q}(\mathbf{h},\mathbf{a}) - \mathbf{V}(\mathbf{h})$
    - Use GAE but can be computed from **V** as **δ=** $r_t + \gamma\hat{\mathbf{V}}(\mathbf{h}_{t+1}) - \hat{\mathbf{V}}(\mathbf{h}_t)$
  - Uses joint value function and local policy ratio: $r_{\psi_i,i} = \frac{\pi_{\psi_i}(a_i|h_i)}{\pi_{\psi_{i,old}}(a_i|h_i)}$
- Critic loss: $\mathcal{L}^{MAPPO}(\theta) = \max\left[(\mathbf{V}(\mathbf{h}_t) - \hat{R}_t)^2, \left(\text{clip}(\mathbf{V}(\mathbf{h}), \mathbf{V}_{old}(\mathbf{h}) - \epsilon, \mathbf{V}_{old}(\mathbf{h}) + \epsilon) - \hat{R}_t\right)^2\right]$
- Can use other centralized info in the critic (more later)
- Simple, but works well and some form of this often works best

Note: actual details in the paper are unclear so this is a more general version

IPPO [de Witt et al. –arXiv 20]

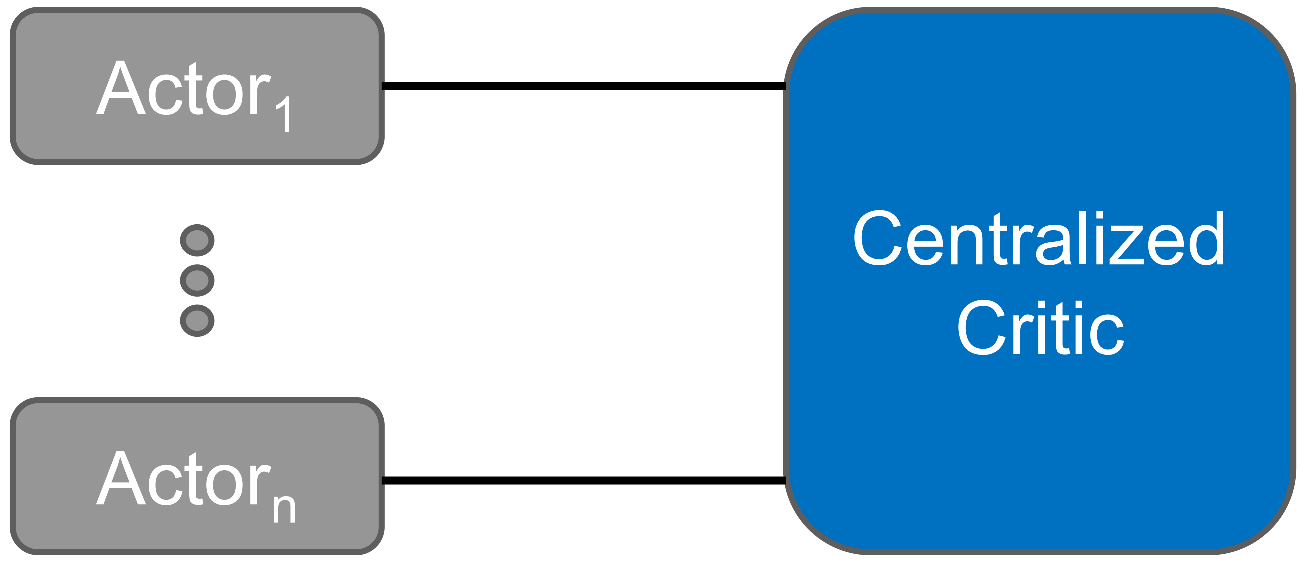
- Actor loss: $\mathcal{L}^{IPPO}_{clip}(\psi_i) = \min\left(r_{\psi_i,i}A_i, \text{clip}(r_{\psi_i,i}, 1-\epsilon, 1+\epsilon)A_i\right)$
  - Uses local advantage: $\hat{A}_i = r_t + \gamma \hat{V}_i(h_{i,t+1}) - \hat{V}_i(h_{i,t})$
    - Can also use GAE or other methods (e.g., n-step)
  - Ratio same as before: $r_{\psi_i,i} = \frac{\pi_{\psi_i}(a_i|h_i)}{\pi_{\psi_{i,old}}(a_i|h_i)}$
  - The only difference is the use of A_i instead of **A**
- Critic loss (with clipping):

$$\mathcal{L}^{IPPO}(\theta) = \max\left[(V_i(h_{i,t})) - \hat{R}_t)^2, \left(\text{clip}(V_i(h_{i,t})), V_{i,old}(h_{i,t})) - \epsilon, V_{i,old}(h_{i,t})) + \epsilon) - \hat{R}_t\right)^2\right]$$

- Often performs similarly to MAPPO but sometimes lower

Contrasting Centralized and Decentralized Critics
in Multi-Agent Policy Gradient [Lyu, Xiao, Daley and Amato – AAMAS21 Best Paper Nomination]

- Centralized critic widely use but misunderstood
- We show in theory:
  - Centralized Critic does not foster cooperation any better than Decentralized Critics
    - Both unbiased estimates of the decentralized policy
  - Centralized Critic exhibits more variance in policy gradient
- In practice:
  - Centralized Critic – less bias, more variance
  - Decentralized Critics – more bias, less variance

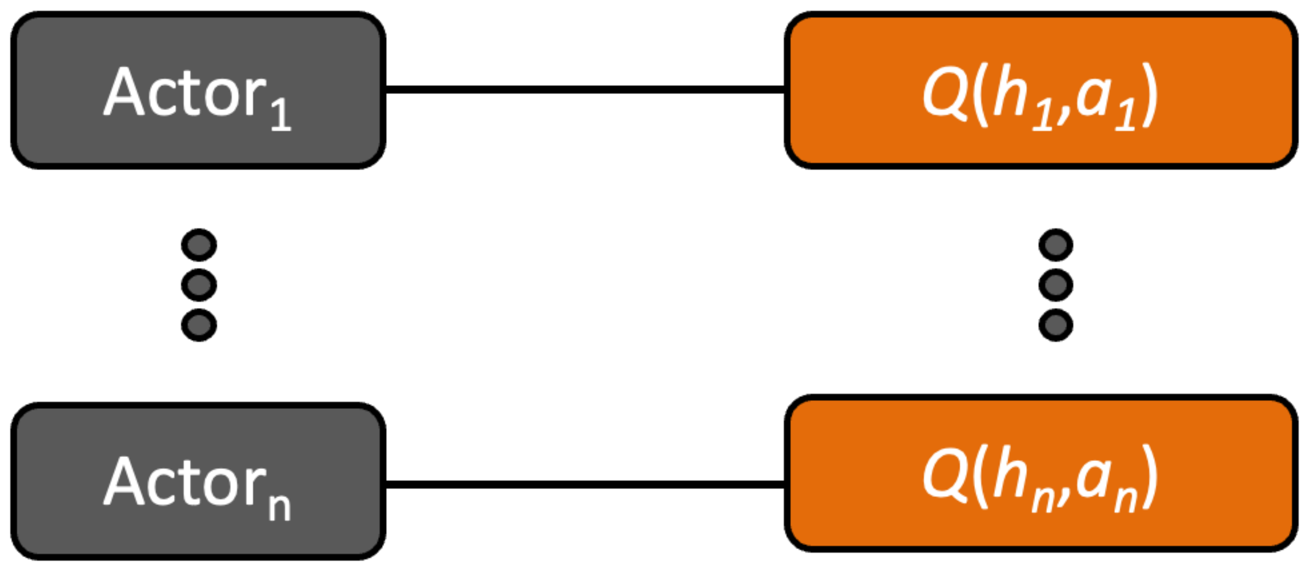


Multi-Agent Actor Critic

Decentralized and Centralized Critic

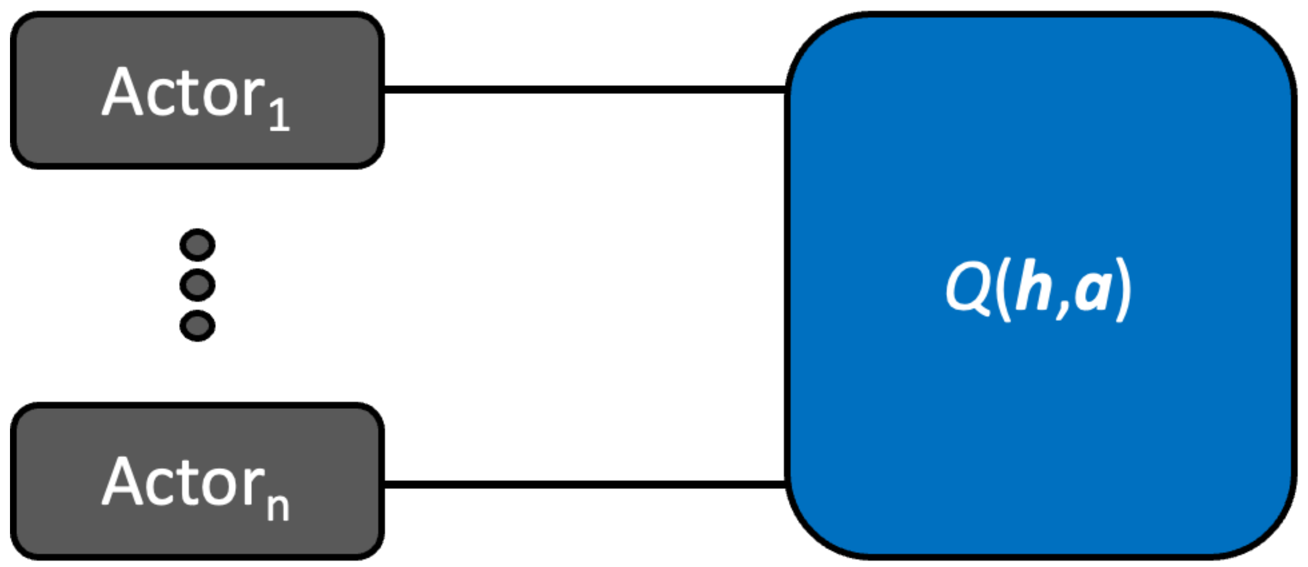
Initialize θ, ϕ
for each training rollout e **do**
  Empty and fill buffer with experience data using actors π
  for Each batch t **do**
    Unroll RNN using observations, actions and rewards
    for each agent i **do**
      Calculate TD targets y_t^i
      $\phi_i = \phi_i - \alpha \nabla_{\phi^i} (y_t^i - Q^i(h_t^i, \mathbf{a}))^2$ // update critic weights
      $\theta^i = \theta^i + \alpha \nabla_{\theta^i} \log \pi^i(a \mid h_t^i) Q^i(h_t^i, a_t^i)$ // update actor weights
    end for
  end for
end for

Decentralized actor and critic: pretend the other agents are part of the environment (independent per agent)

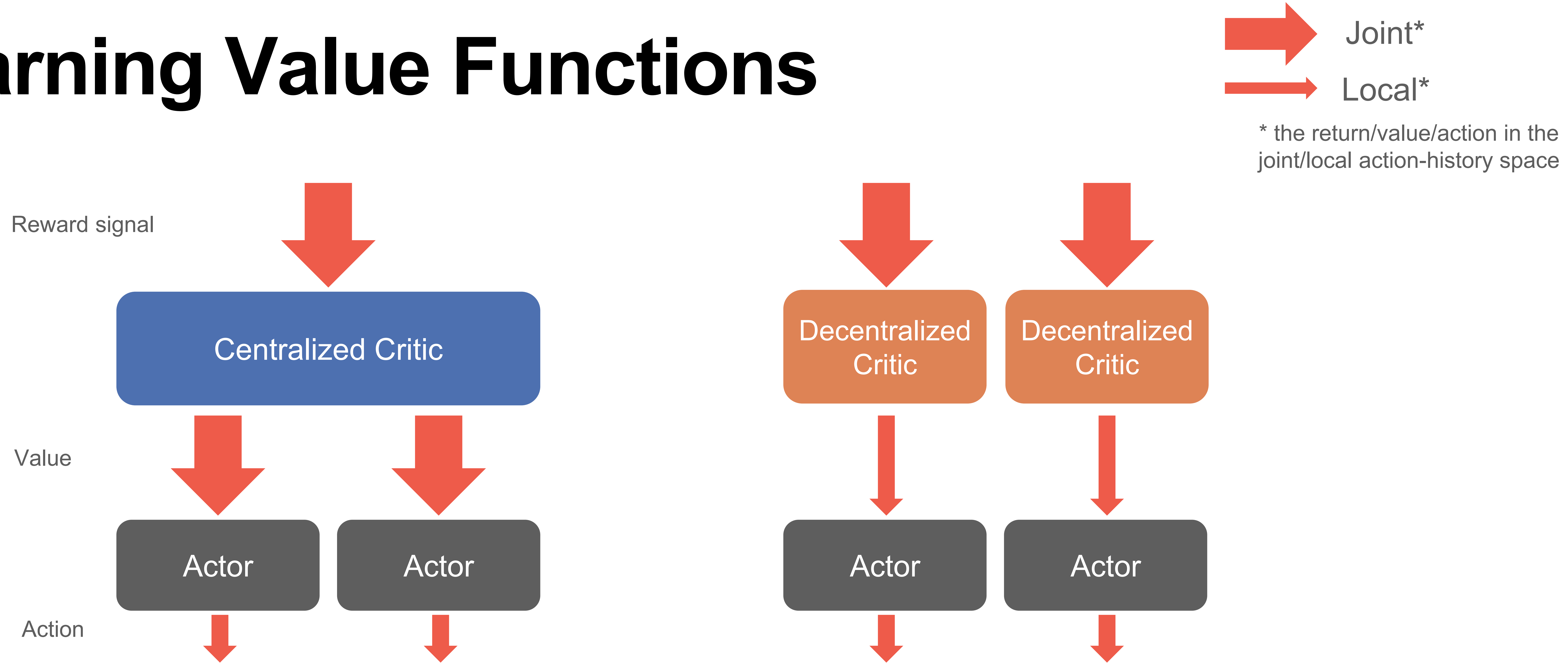


Initialize θ, ϕ
for each training rollout e **do**
  Empty and fill buffer with experience data using actors π
  for Each batch t **do**
    Unroll RNN using observations, actions and rewards
    Calculate TD targets y_t
    $\phi = \phi - \alpha \nabla_{\phi} (y_t - \mathbf{Q}(\mathbf{h}_t, \mathbf{a}_t))^2$ // update critic weights
    for each agent i **do**
      $\theta^i = \theta^i + \alpha \nabla_{\theta^i} \log \pi^i(a \mid h_t^i) \mathbf{Q}(\mathbf{h}_t, \mathbf{a}_t)$ // update actor weights
    end for
  end for
end for

Decentralized actor and centralized critic: update critic based on centralized Q-value and then update each agent's actor



Learning Value Functions

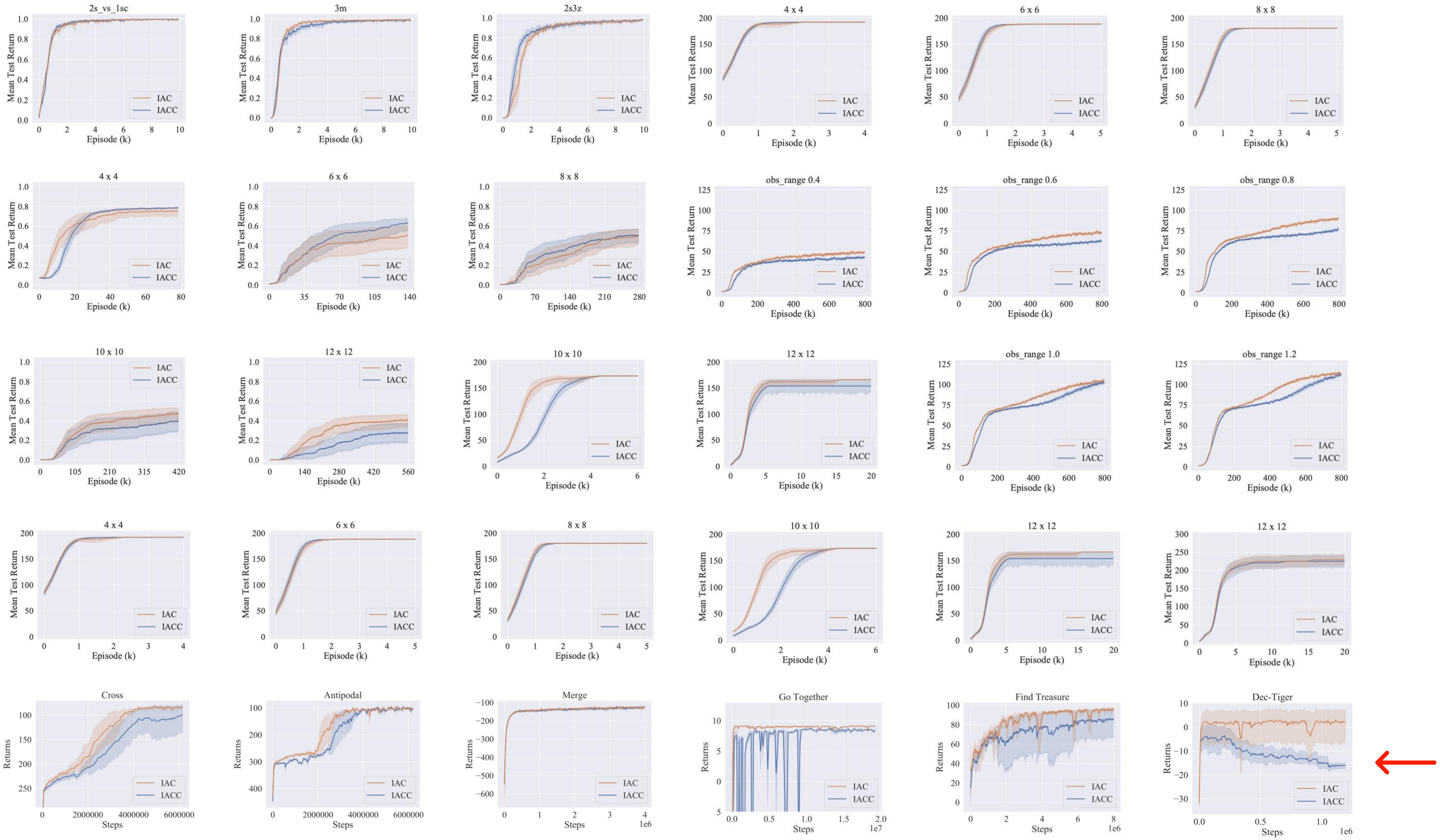


$$\nabla_{\theta_i} J_c(\theta_i) = \mathbb{E}_{\boldsymbol{a},\boldsymbol{h}}[\nabla \log \pi_i(a_i \mid h_i; \theta_i) Q^{\boldsymbol{\pi}}(\boldsymbol{h}, \boldsymbol{a}; \phi)]$$
$$= \mathbb{E}_{a_i,h_i}\Big[\nabla \log \pi_i(a_i \mid h_i; \theta_i) \underline{\mathbb{E}_{a_{-i},h_{-i}}}[Q^{\boldsymbol{\pi}}(h_i, h_{-i}, a_i, a_{-i})]\Big]$$

Both estimating and updating decentralized policies

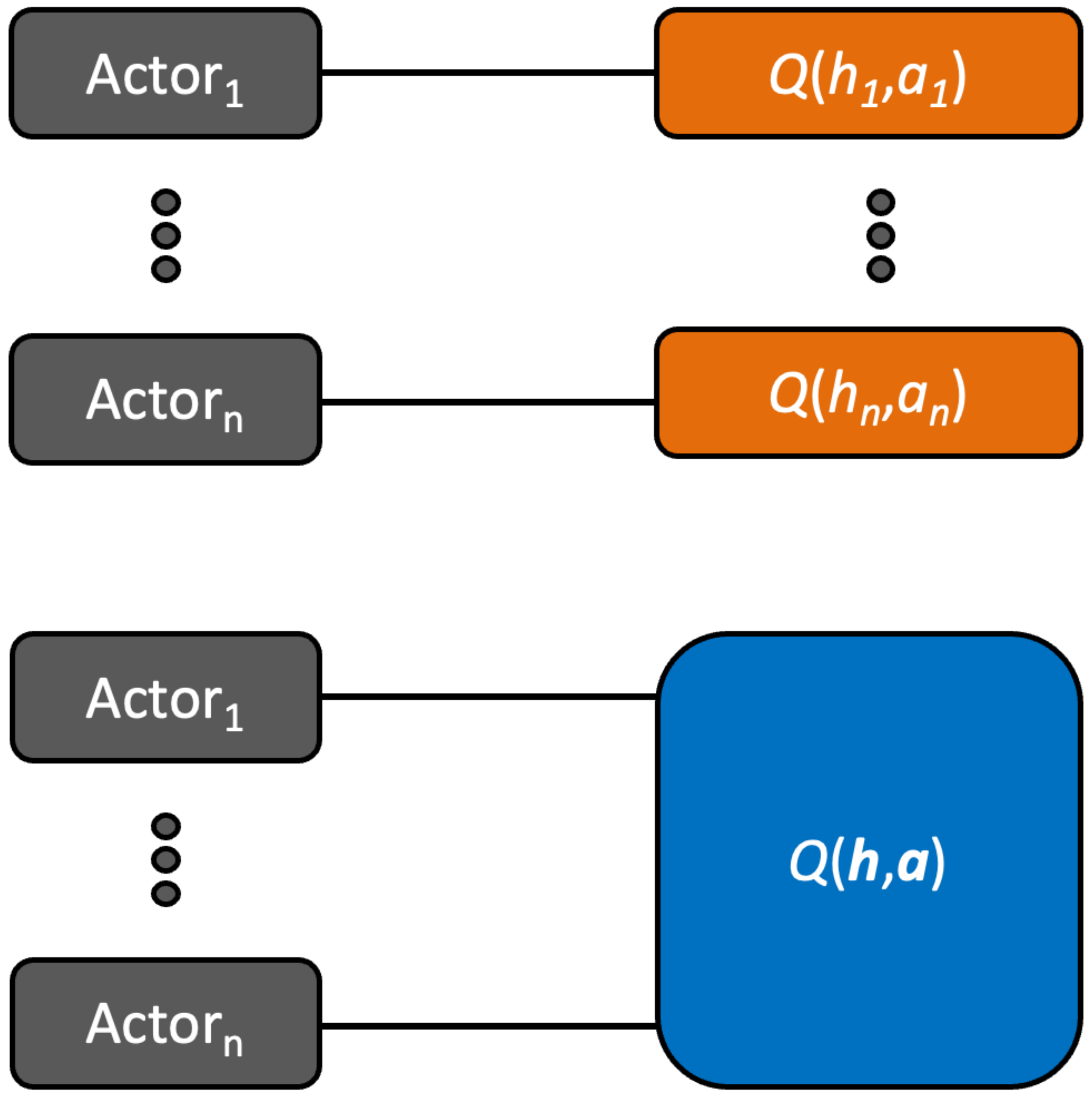
Centralized and Decentralized Critic Performance

on StarCraft Multi-Agent Challenge (SMAC), Box Pushing, Particle environments, Target Capture, etc.



Decentralized vs centralized critics

- Theoretically equivalent
  - But that assumes learned critics
- Decentralized critics can be harder to learn
  - When other agents change policies
  - Higher bias
- Centralized critics can be harder to learn
  - Large domains (action, obs, agents)
  - Higher variance to marginalize out other agents



State-based Centralized Critics

State information is often available offline in a simulator

Implemented by pioneering Centralized Critic methods

- COMA (Foerster et al. 2018), MADDPG (Lowe et al. 2017)

Followed by later methods

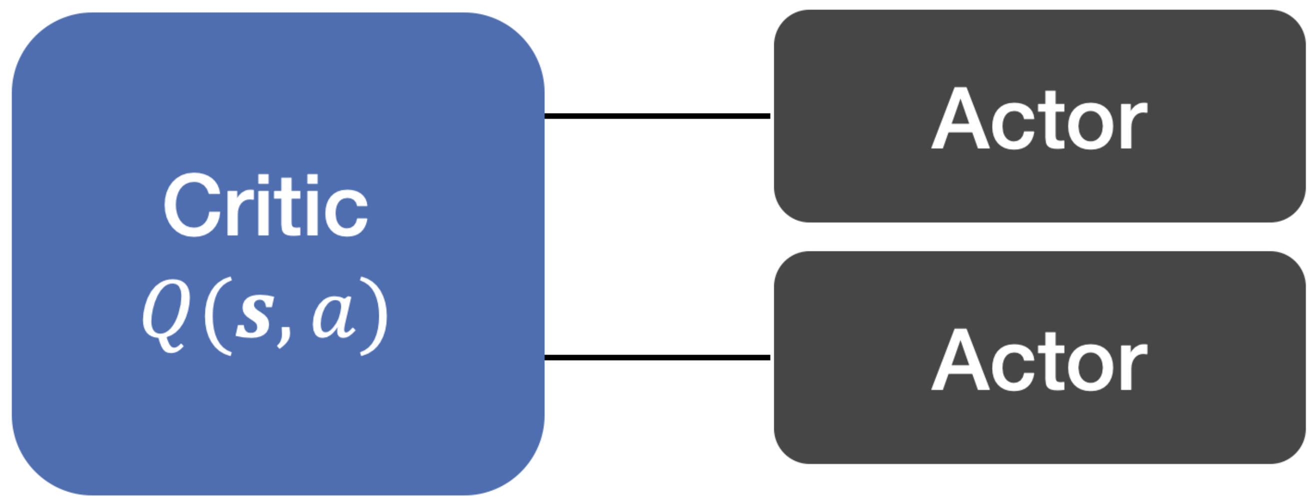
- SQDDPG (Wang et al. 2020), LIIR (Du et al. 2019), LICA (Zhou et al. 2020),VDAC-mix (Su, Adams, and Beling 2021), DOP (Wang et al. 2021) and MACKRL (Schroeder de Witt et al. 2019)

Obvious Advantages of State-based Centralized Critic

- Compact, Fully Observable

Obvious Disadvantages of History-based Centralized Critic

- Complexity from (potentially long) time horizon
- Complexity from combining observations (and actions) from multiple agents
- Partially Observable



A Deeper Understanding of State-Based Critics
in Multi-Agent Reinforcement Learning

Lyu, Baisero, Xiao and Amato – AAAI22

State-based critics in MARL are popular but misunderstood

We show in theory:

- State-based critics may be biased compared to History-based Critics
- State-based critics may produce higher variance

We show empirically:

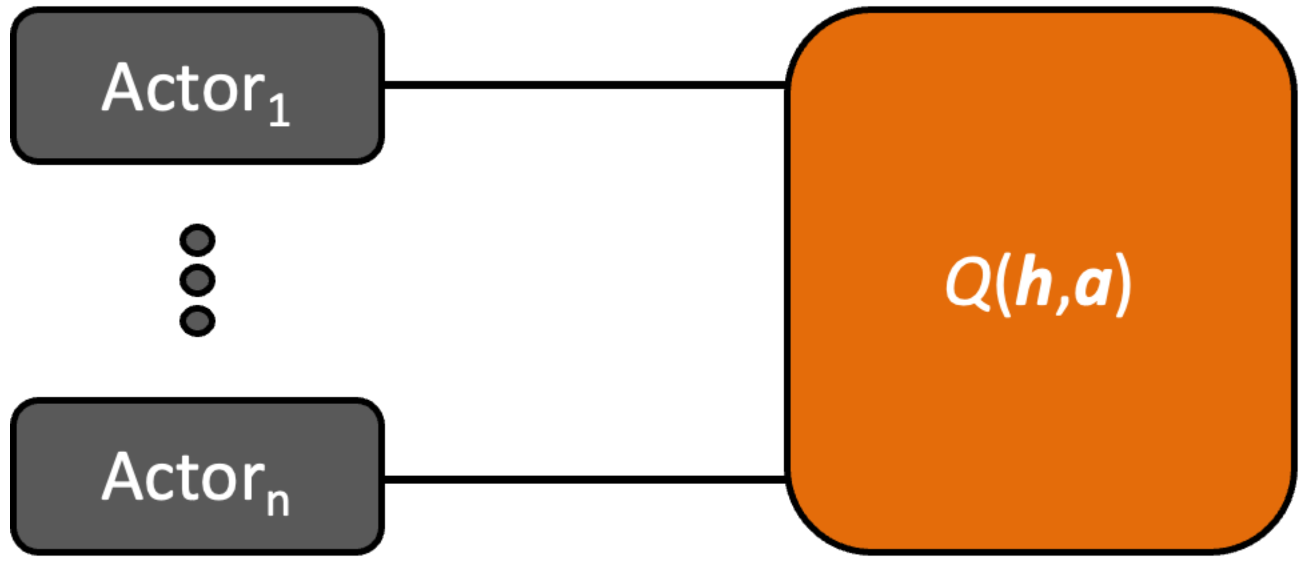
- Both critics work well in different domains
- Common benchmarks lack partial observability
- The state-history-based critic is robust to various domains

Centralized critics

Centralized critic

Conditions on history of all agents (joint history $\boldsymbol{h}$)

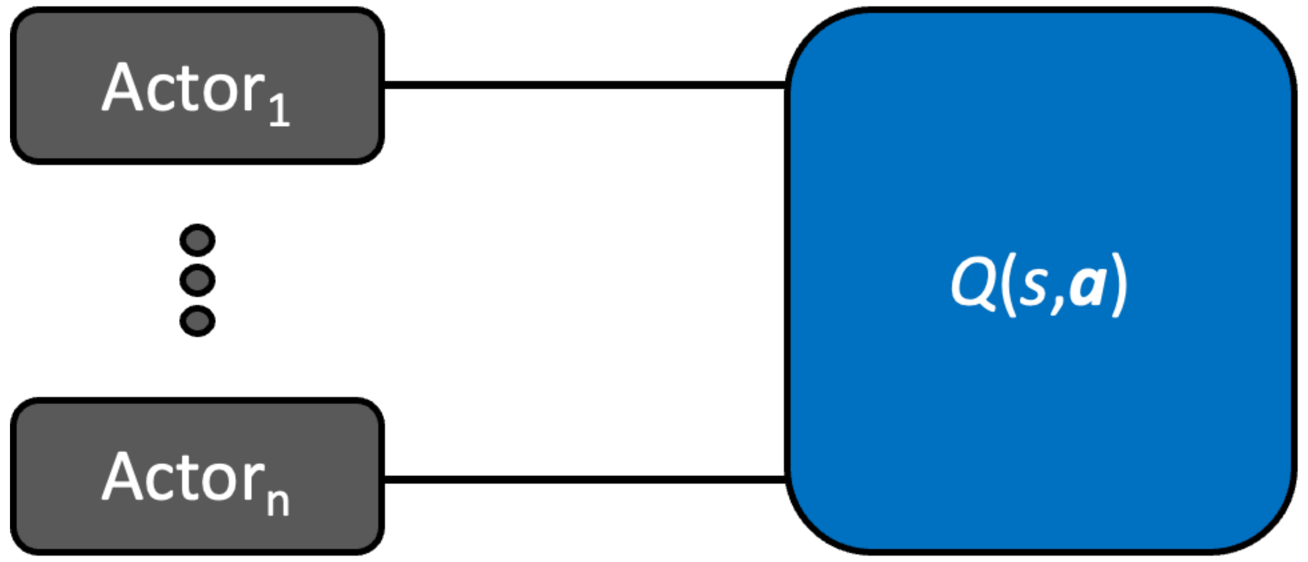
$$\nabla_i J_{\boldsymbol{h}} = \mathbb{E}_{\boldsymbol{h}\sim\rho(\boldsymbol{h}),\boldsymbol{a}\sim\boldsymbol{\pi}(\boldsymbol{h})}\left[Q^{\boldsymbol{\pi}}(\boldsymbol{h},\boldsymbol{a})\nabla_{\theta_i}\log\pi_i(a_i;h_i)\right]$$



State-based centralized critic

Conditions on the world state s

$$\nabla_i J_{s} = \mathbb{E}_{\boldsymbol{h},s\sim\rho(\boldsymbol{h},s),\boldsymbol{a}\sim\boldsymbol{\pi}(\boldsymbol{h})}\left[Q^{\boldsymbol{\pi}}(s,\boldsymbol{a})\nabla_{\theta_i}\log\pi_i(a_i;h_i)\right]$$

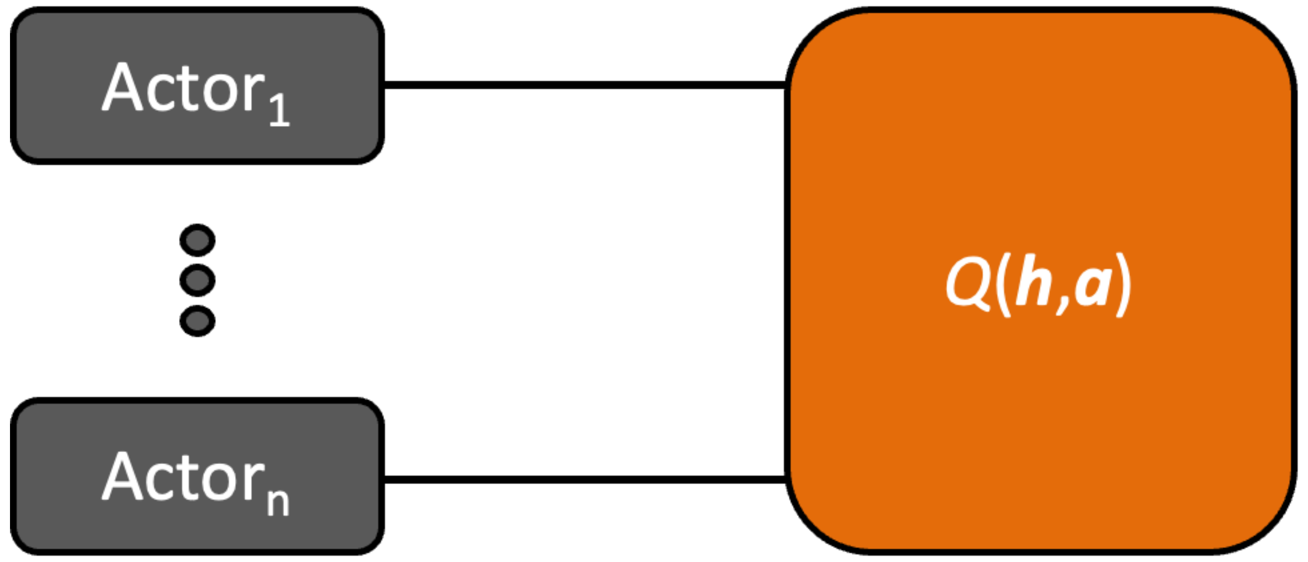


Centralized critics

Centralized critic

Conditions on history of all agents (joint history $\boldsymbol{h}$)

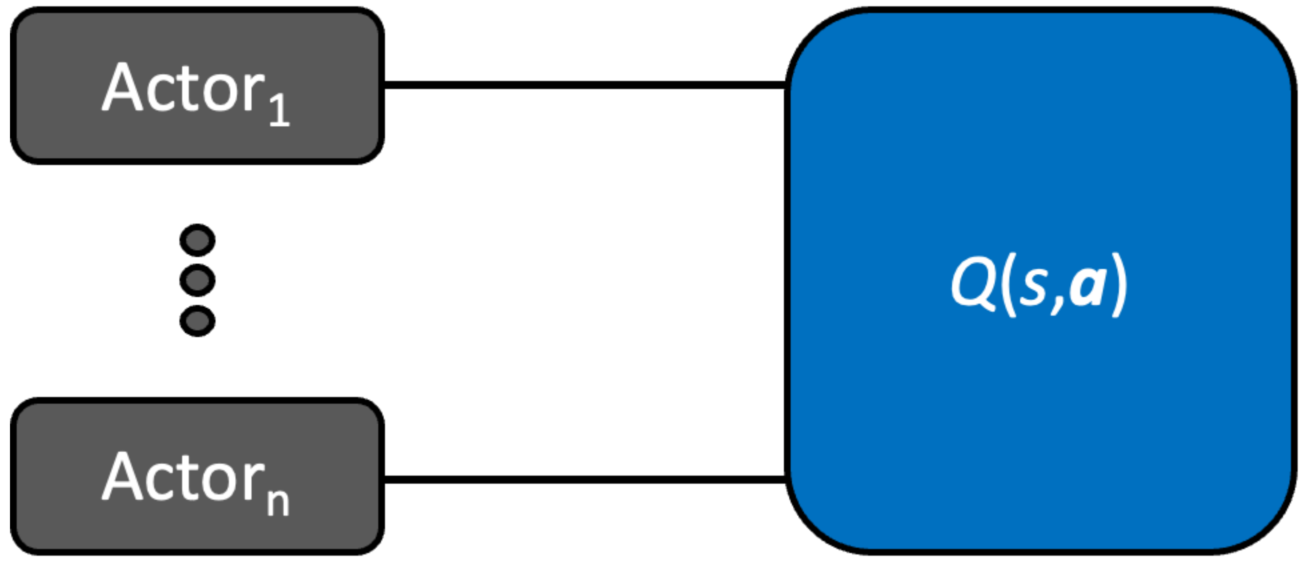
$$\nabla_i J_{\boldsymbol{h}} = \mathbb{E}_{\boldsymbol{h}\sim\rho(\boldsymbol{h}),\boldsymbol{a}\sim\boldsymbol{\pi}(\boldsymbol{h})}\left[Q^{\boldsymbol{\pi}}(\boldsymbol{h},\boldsymbol{a})\nabla_{\theta_i}\log\pi_i(a_i;h_i)\right]$$



State-based centralized critic

Conditions on the world state s

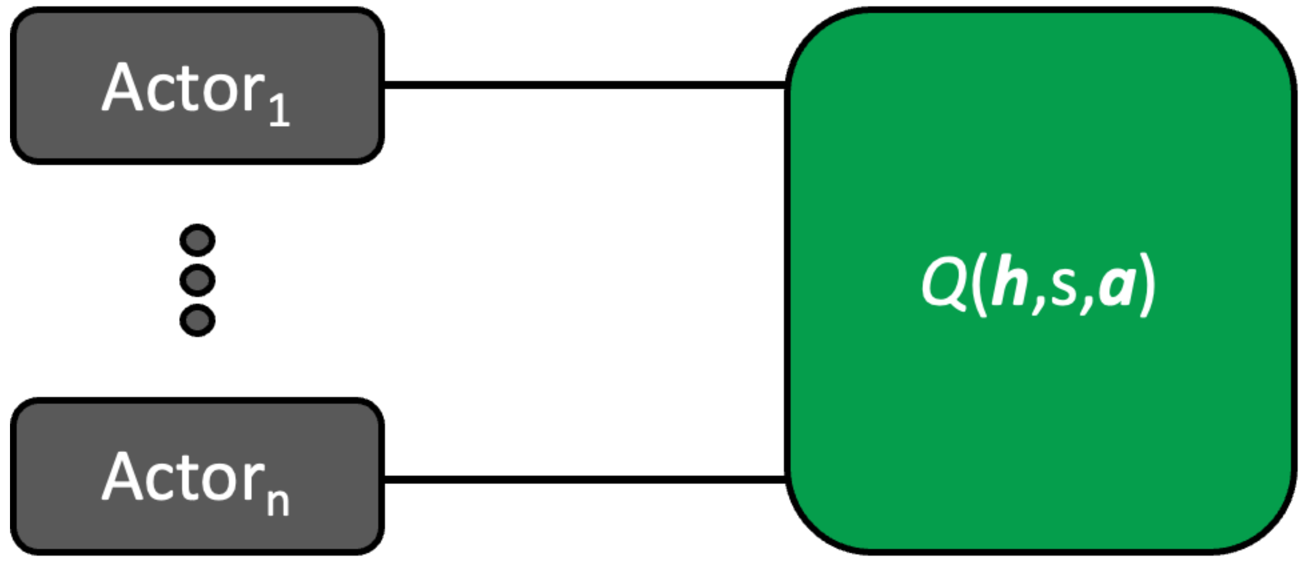
$$\nabla_i J_{s} = \mathbb{E}_{\boldsymbol{h},s\sim\rho(\boldsymbol{h},s),\boldsymbol{a}\sim\boldsymbol{\pi}(\boldsymbol{h})}\left[Q^{\boldsymbol{\pi}}(s,\boldsymbol{a})\nabla_{\theta_i}\log\pi_i(a_i;h_i)\right]$$



History-state-based centralized critic

Conditions on the joint history $\boldsymbol{h}$ and world state s

$$\nabla_i J_{s} = \mathbb{E}_{\boldsymbol{h},s\sim\rho(\boldsymbol{h},s),\boldsymbol{a}\sim\boldsymbol{\pi}(\boldsymbol{h})}\left[Q^{\boldsymbol{\pi}}(s,\boldsymbol{h},\boldsymbol{a})\nabla_{\theta_i}\log\pi_i\left(a_i;h_i\right)\right]$$



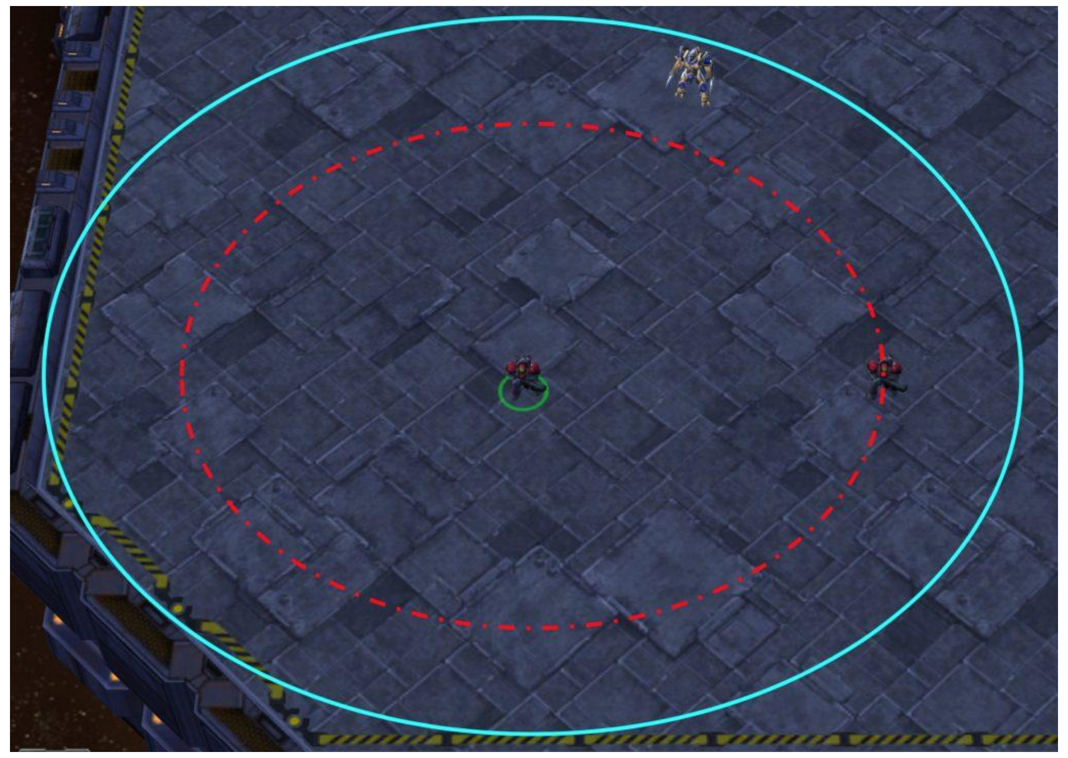
Experiments

Tested with advantage actor critic (A2C)

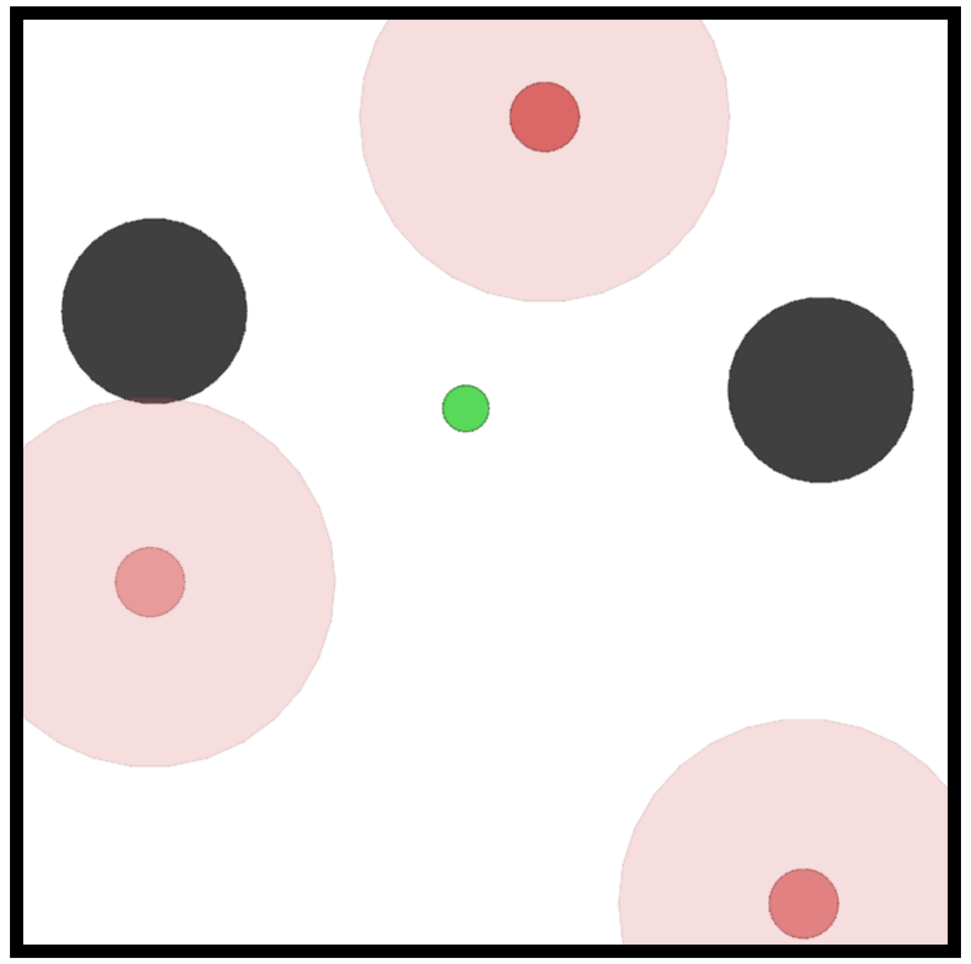
- History critic
- State critic
- State-history critic

Used standard domains: small common domains, SMACv1 (Starcraft) and partially observable particle environments

Have additional experiments and base actor-critic methods in the paper

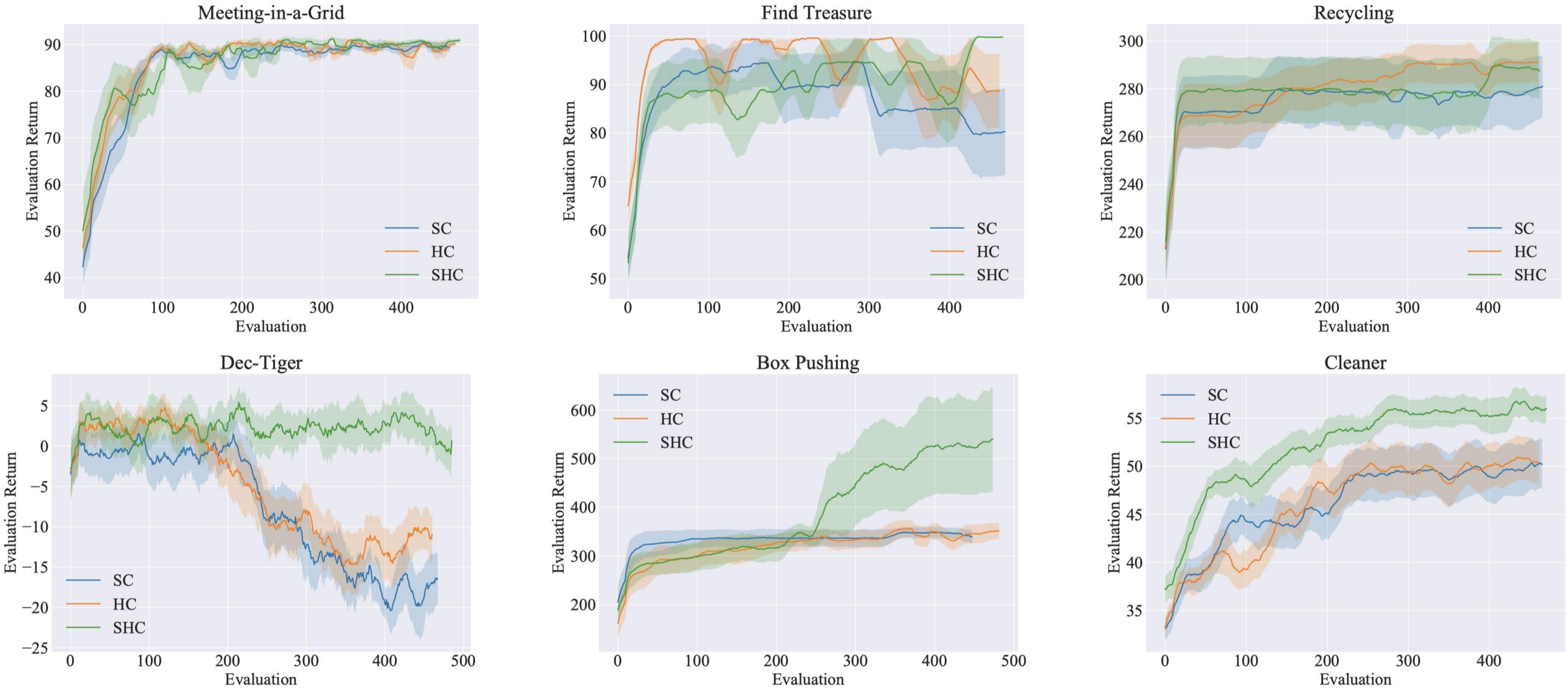


SMAC v1

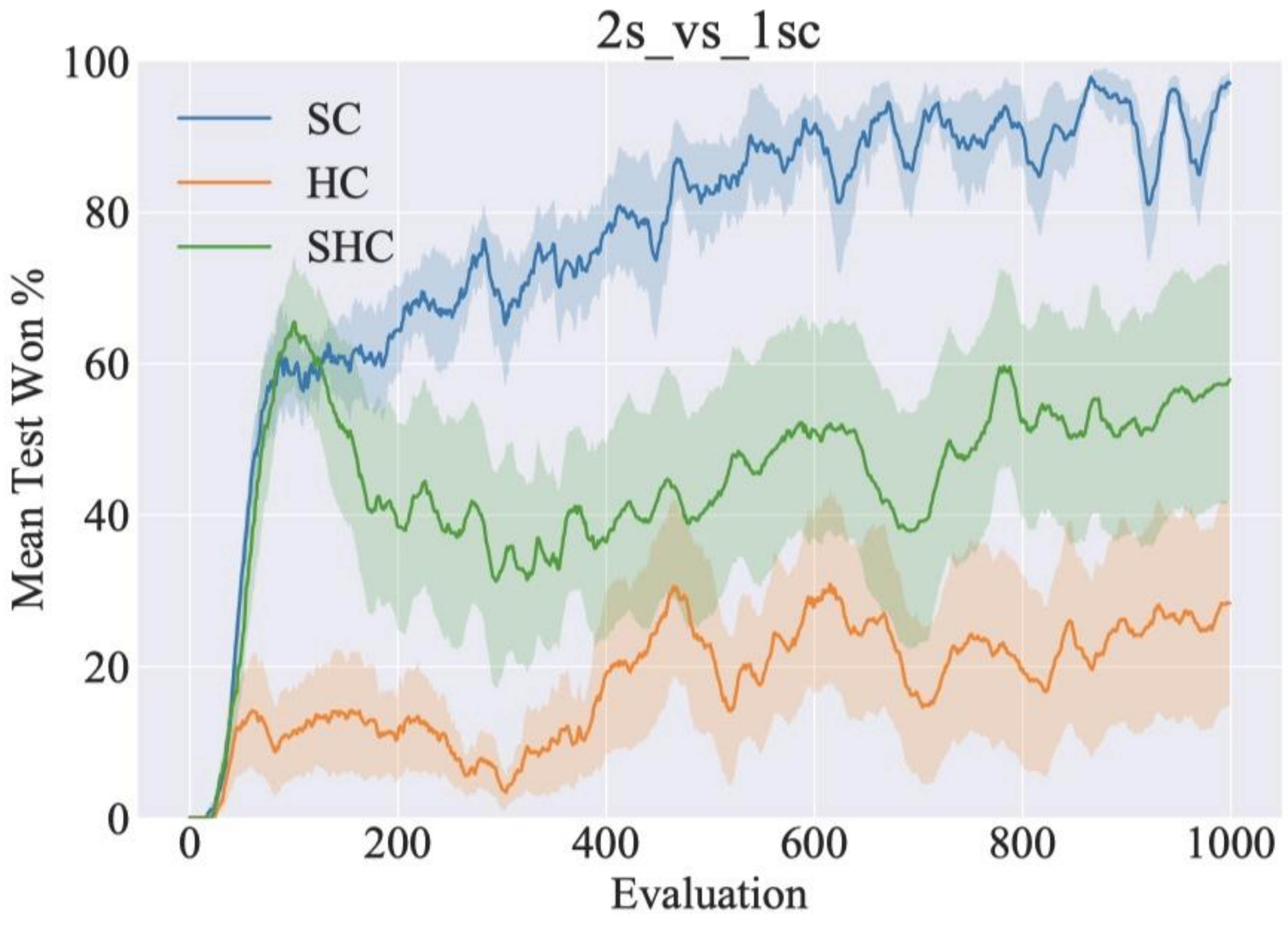


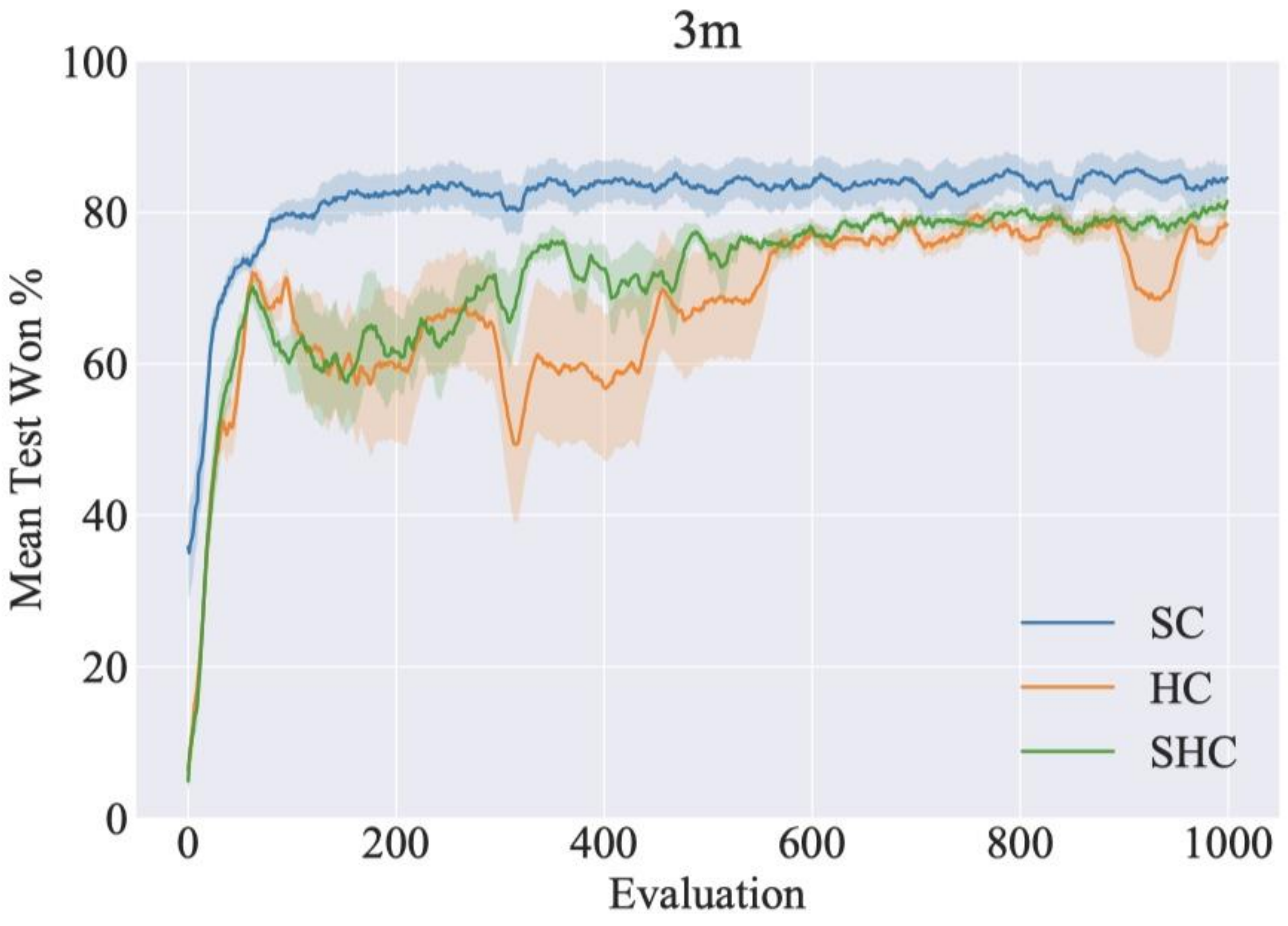
Partially observable particle envs

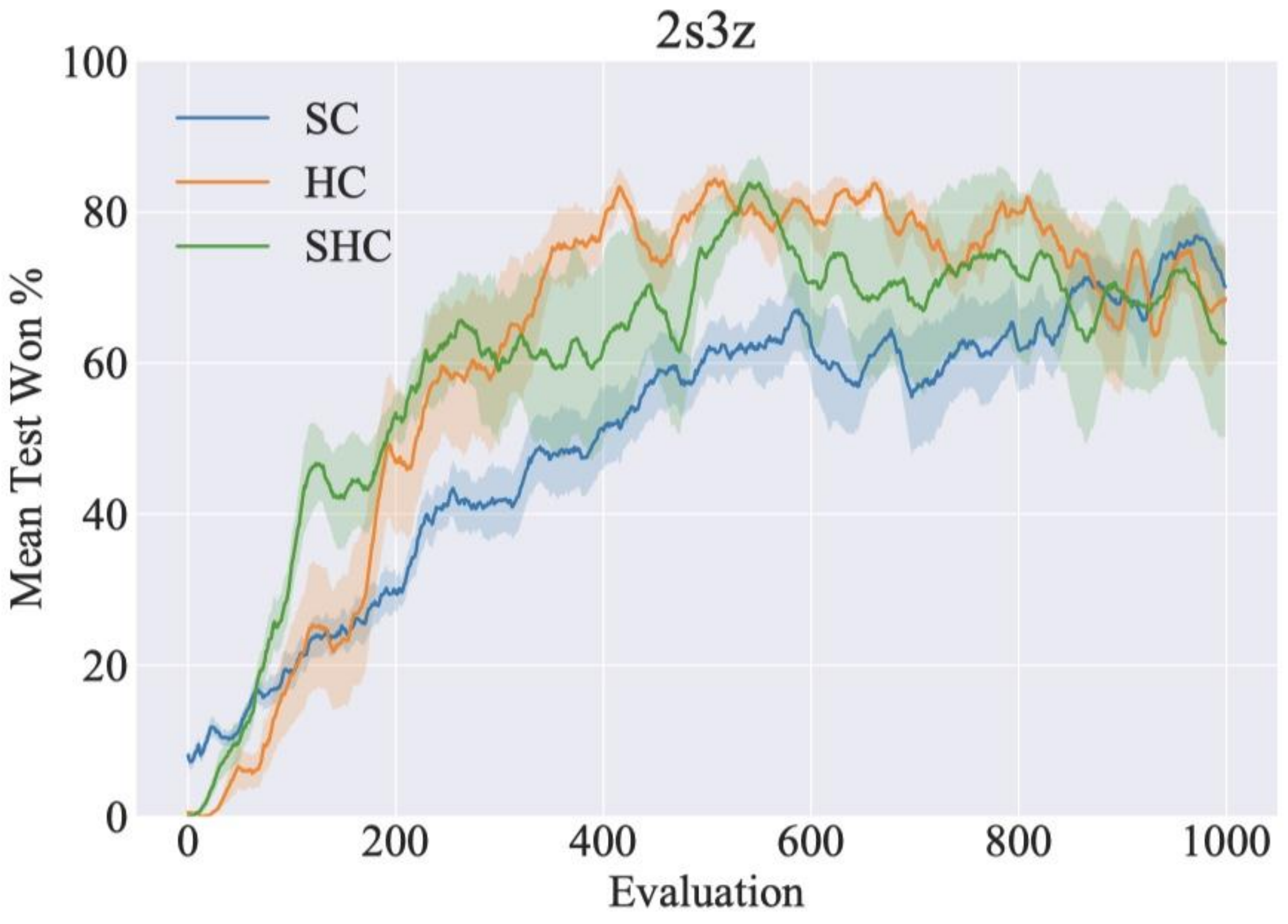
Common small environments

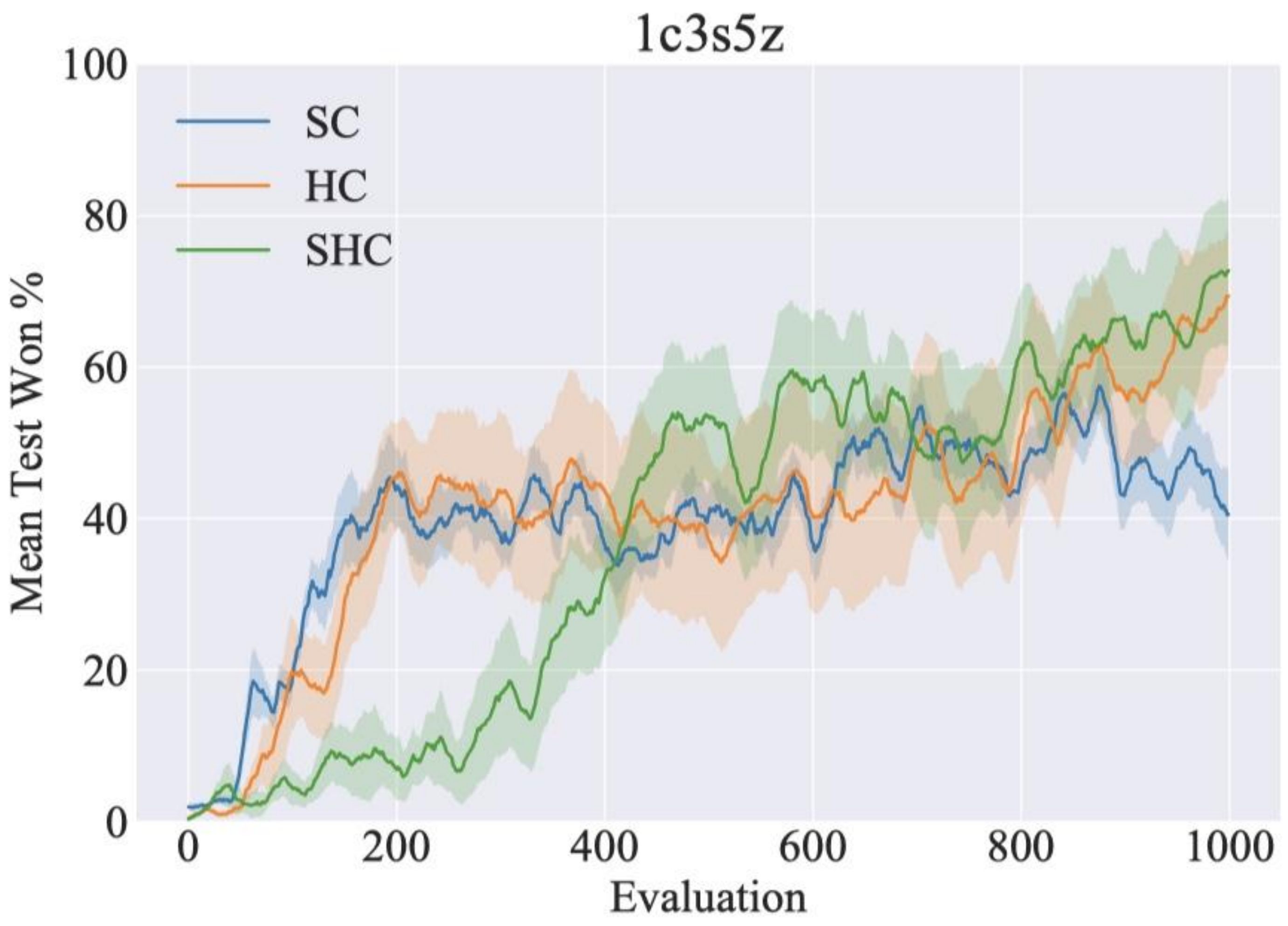


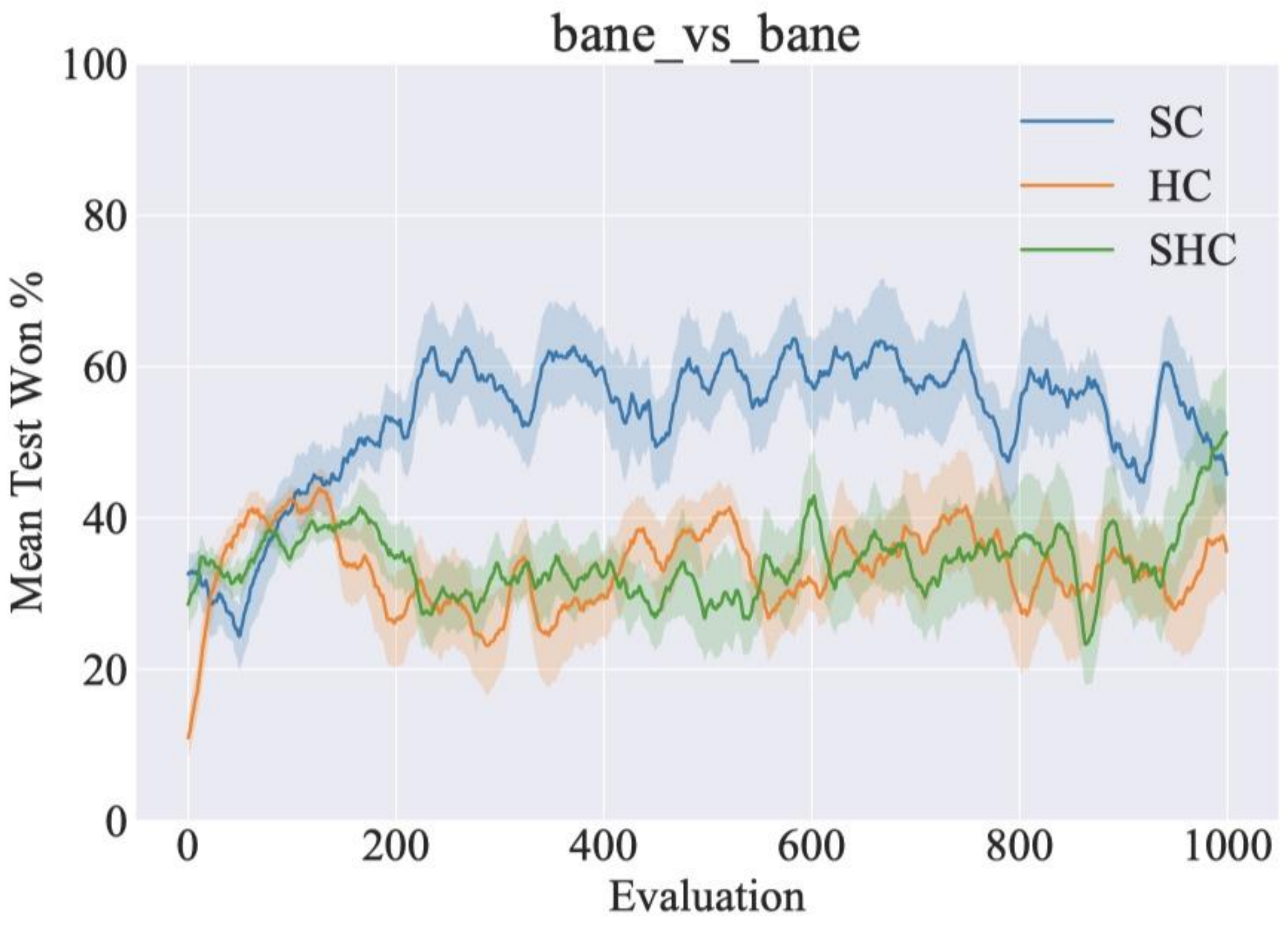
SMAC - StarCraft Multi-Agent Challenge





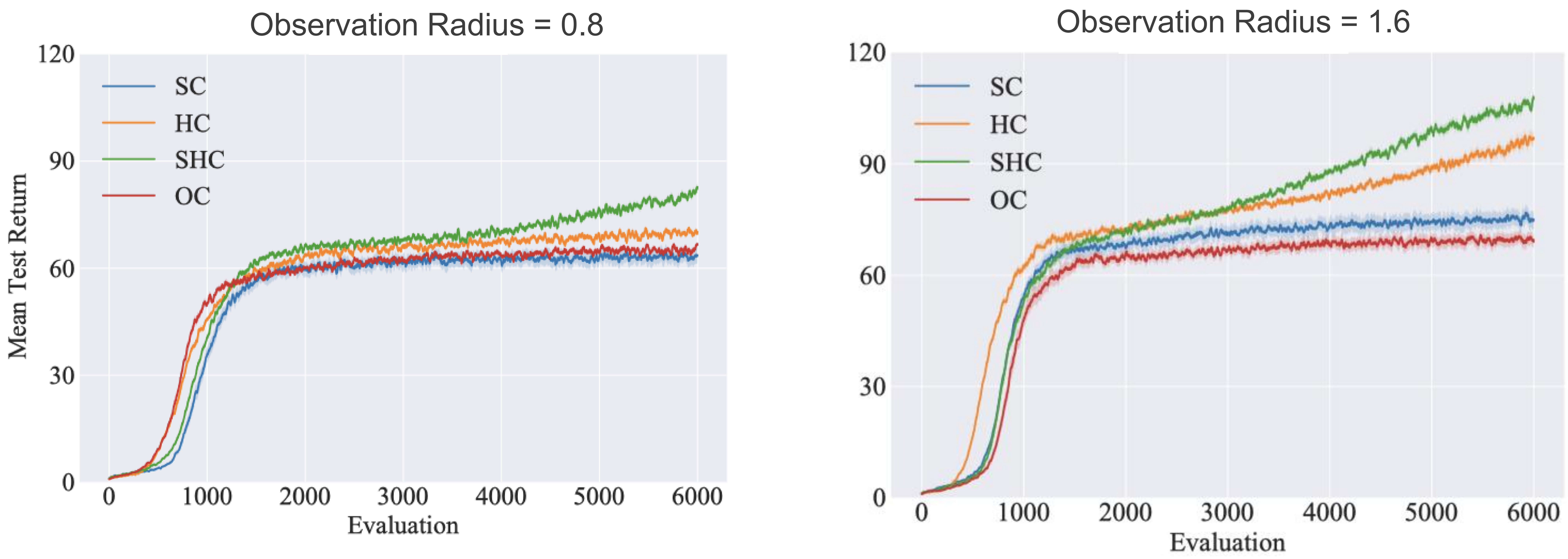






Partially Observable Particle Environments

Predator and Prey



Takeaways

Benchmark problems

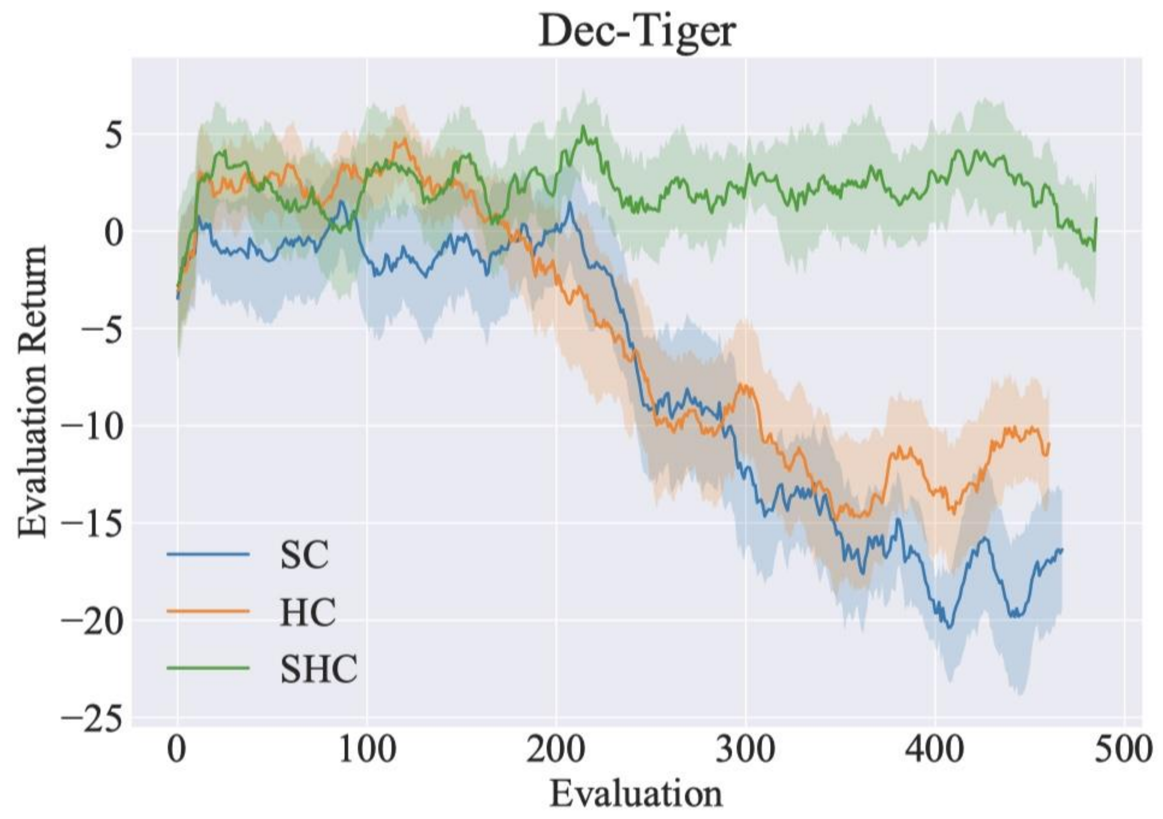
- We need harder, more partially observable problems

Methods to use

- Decentralized critics and (centralized) state-history-based often work the best
- MAPPO paper had a similar result
- Not really clear why

CTDE

- What is the best way to perform centralized training for decentralized execution (that's both principled and performs well)?



(a) Dec-Tiger (Nair et al. 2003)

Other CTDE methods

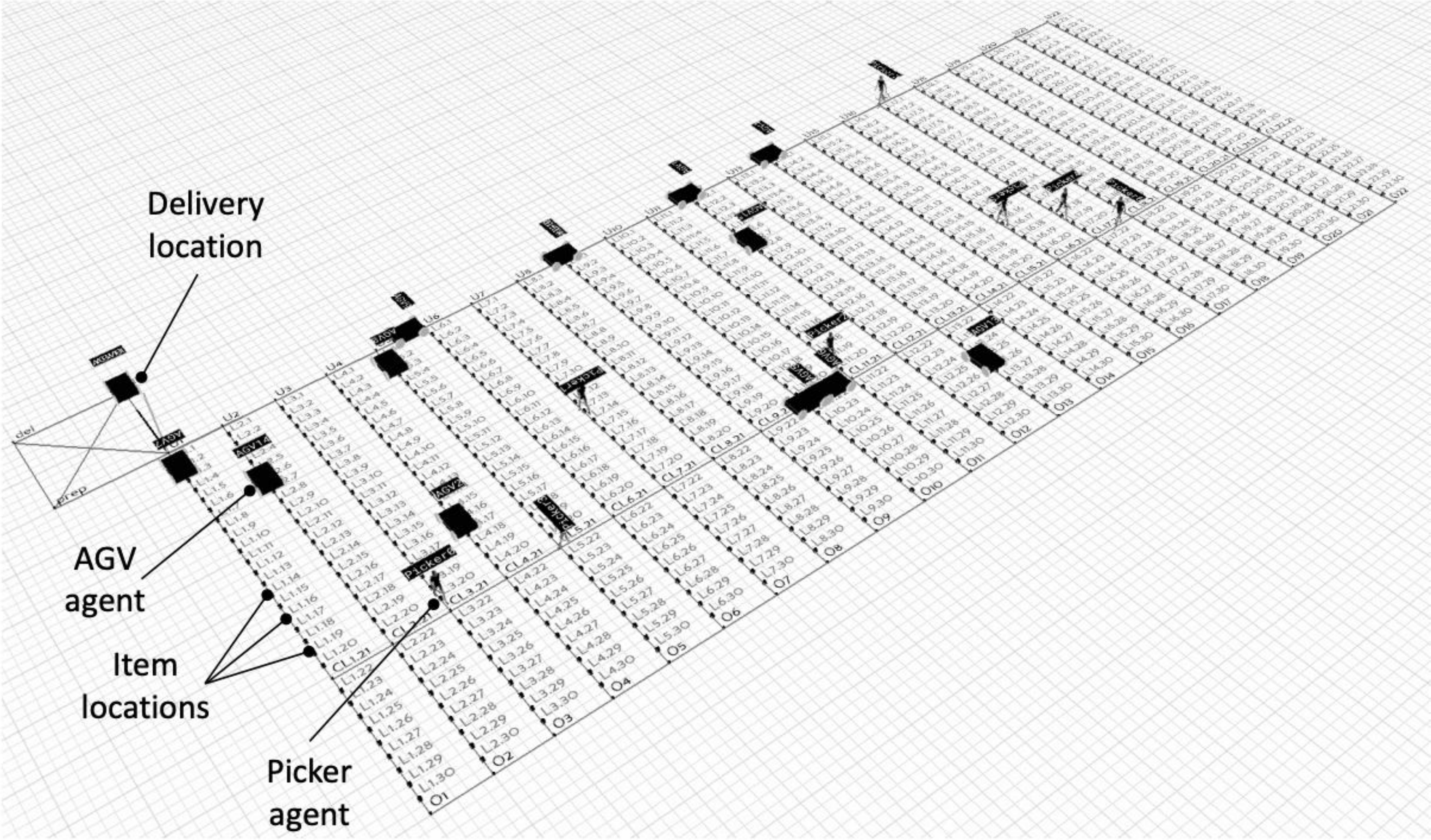
- Many other extensions and approaches:
  - E.g., FACMAC: Use a factored critic (doesn’t need IGM) (Peng et al., 2021)
- **Parameter Sharing**
- **Alternating learning**
  - (Banerjee et al., 2012, Su et al., 2024)
  - Sequential agent updates as in HATRPO and HAPPO (Kuba et al. 2022)
- **Other agent modeling**, e.g., LOLA (Foerster et al. 2018a)

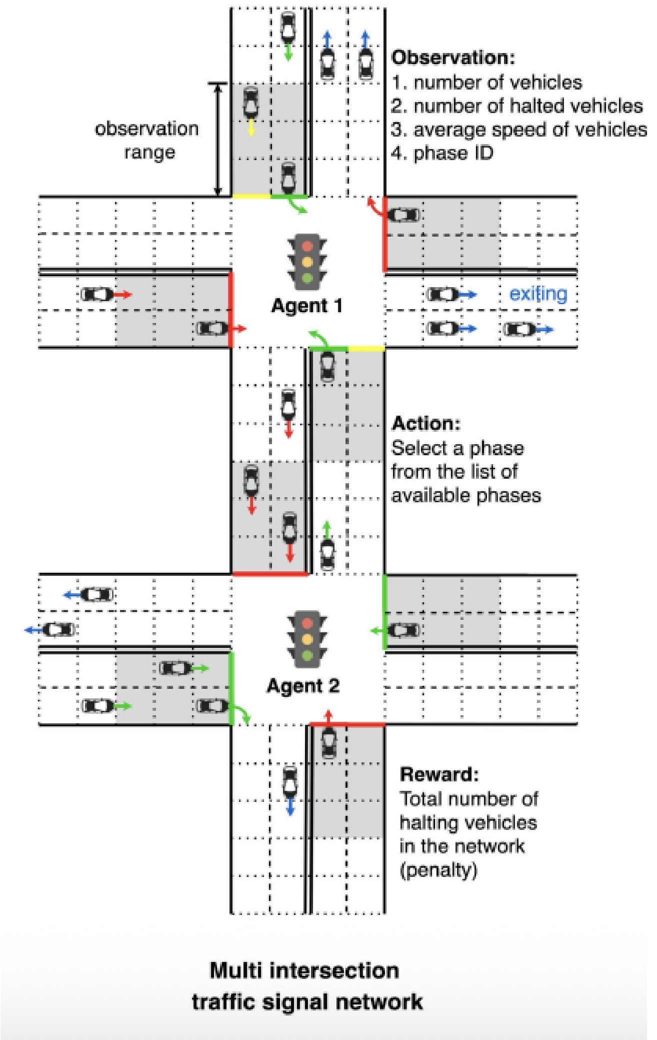
Other topics

Many other topics in (cooperative) MARL that we don't have time to cover

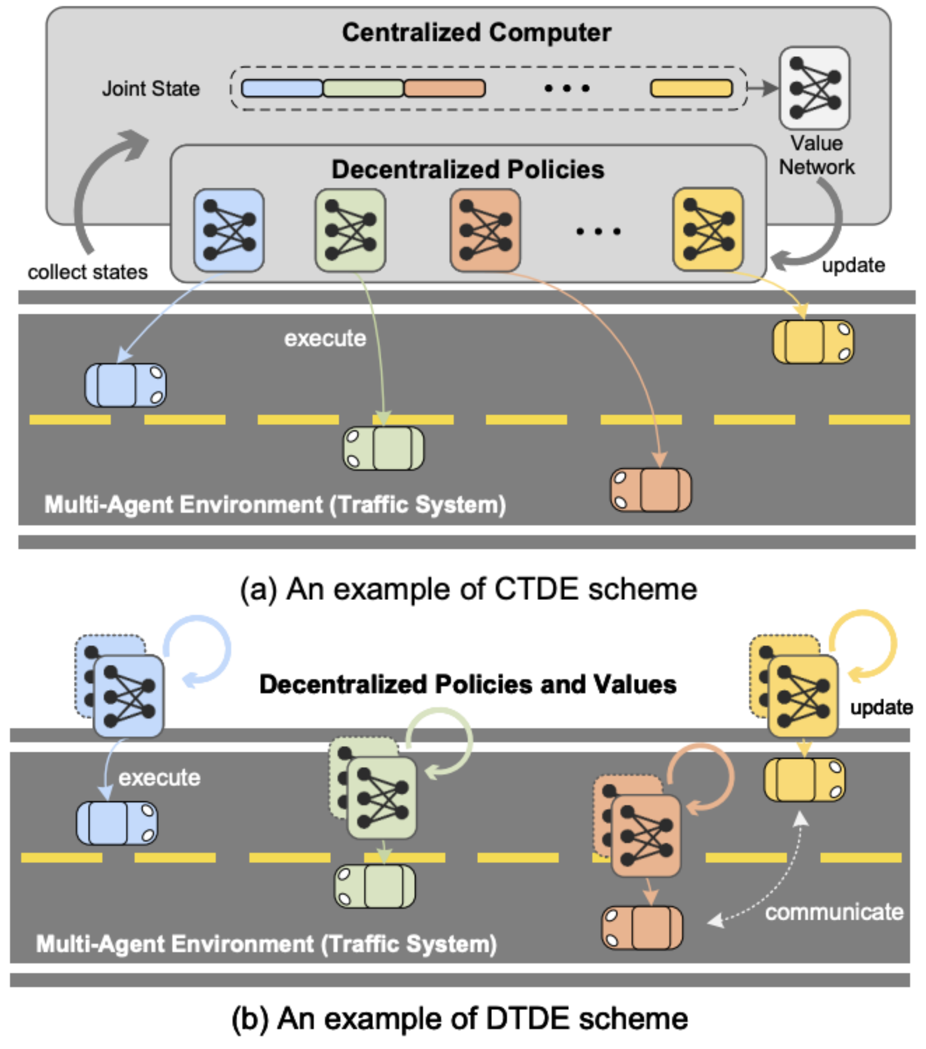
- Communication (Zhu et al., 2024)
- Ad hoc teamwork (Mirsky et al., 2022),
- Model-based methods (Wang et al., 2022)
- Exploration, offline methods, model-based methods, hierarchical methods, role decomposition, multi-task approaches, etc.

Applications





(Bokade et al., 2023)



(a) An example of CTDE scheme

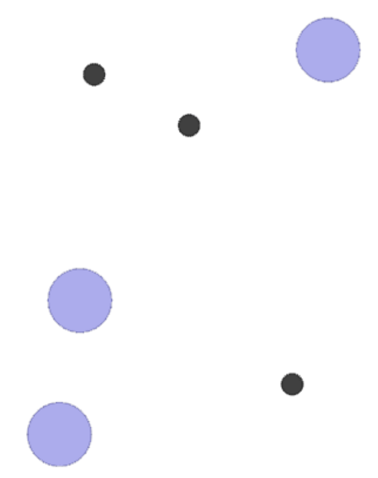
(b) An example of DTDE scheme

- Video games (e.g.,AlphaStar (Vinyals et al., 2019)
  - Centralized MARL for a team
- Warehouse robots (Krnjaic et al. 2024)
  - Hierarchical CTDE approach
- Traffic signal control (e.g., survey by Wei et al. 2021)
- Autonomous vehicle control (e.g., survey by Zhang et al. 2024)
- Power systems, etc!

Multi-agent RL with macro-actions

[Xiao, Hoffman, Xia and Amato – ICRA20]

Benchmarks

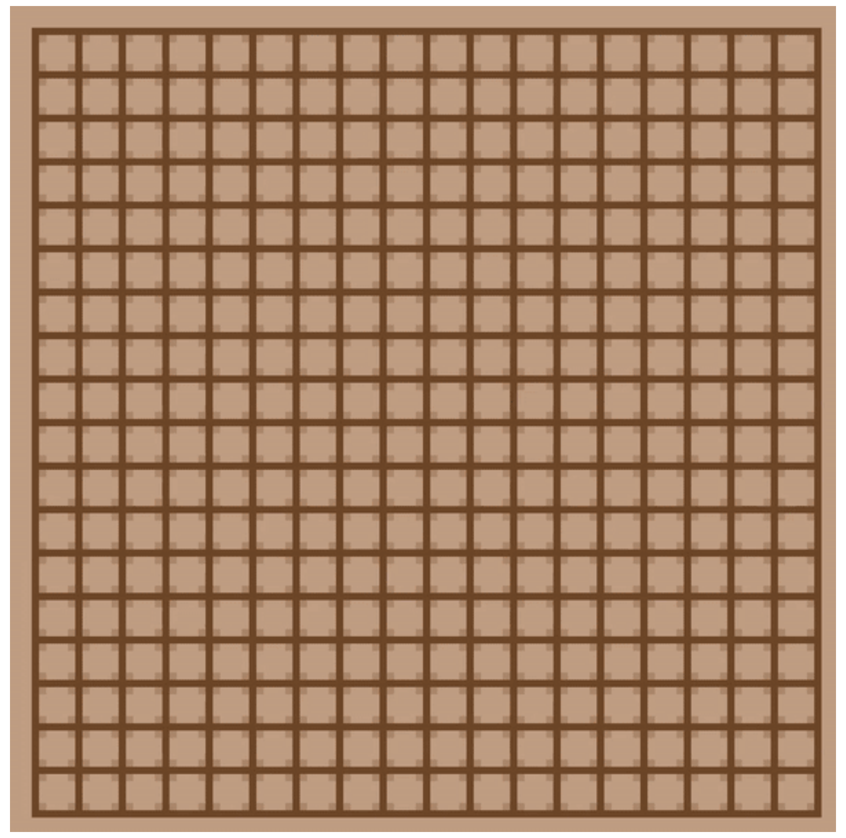


- Standard domains:
  - Multi-agent Particle Envs (MPE) (PyTorch and JAX)
  - Overcooked (PyTorch and JAX)

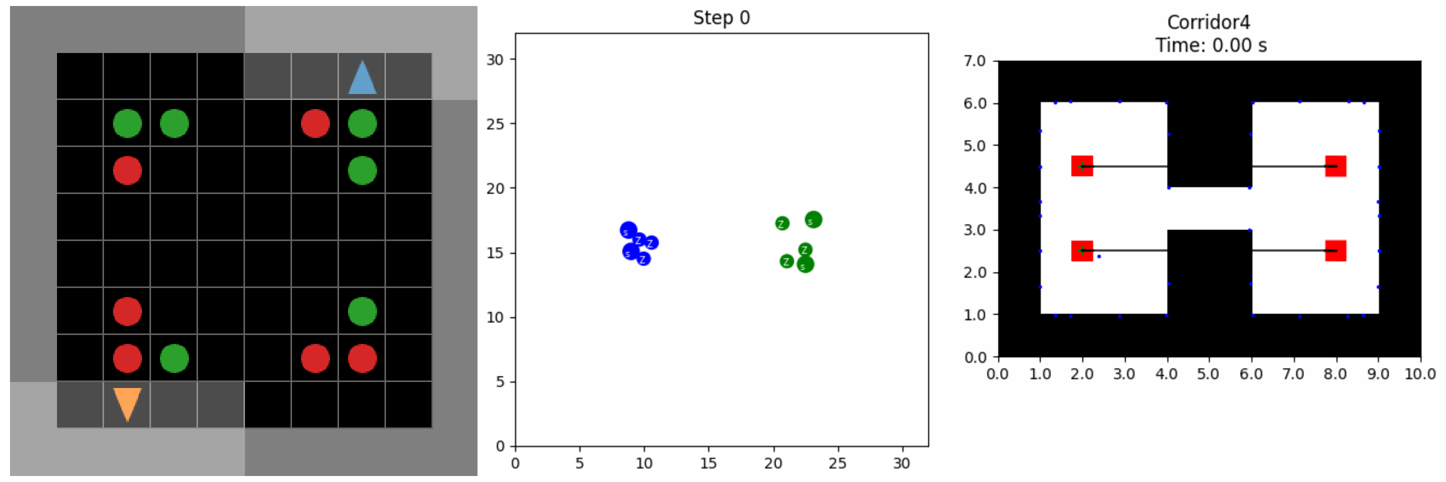
  - SMAC v1 and v2 (PyTorch and JAX)

- Many many more inspired by applications

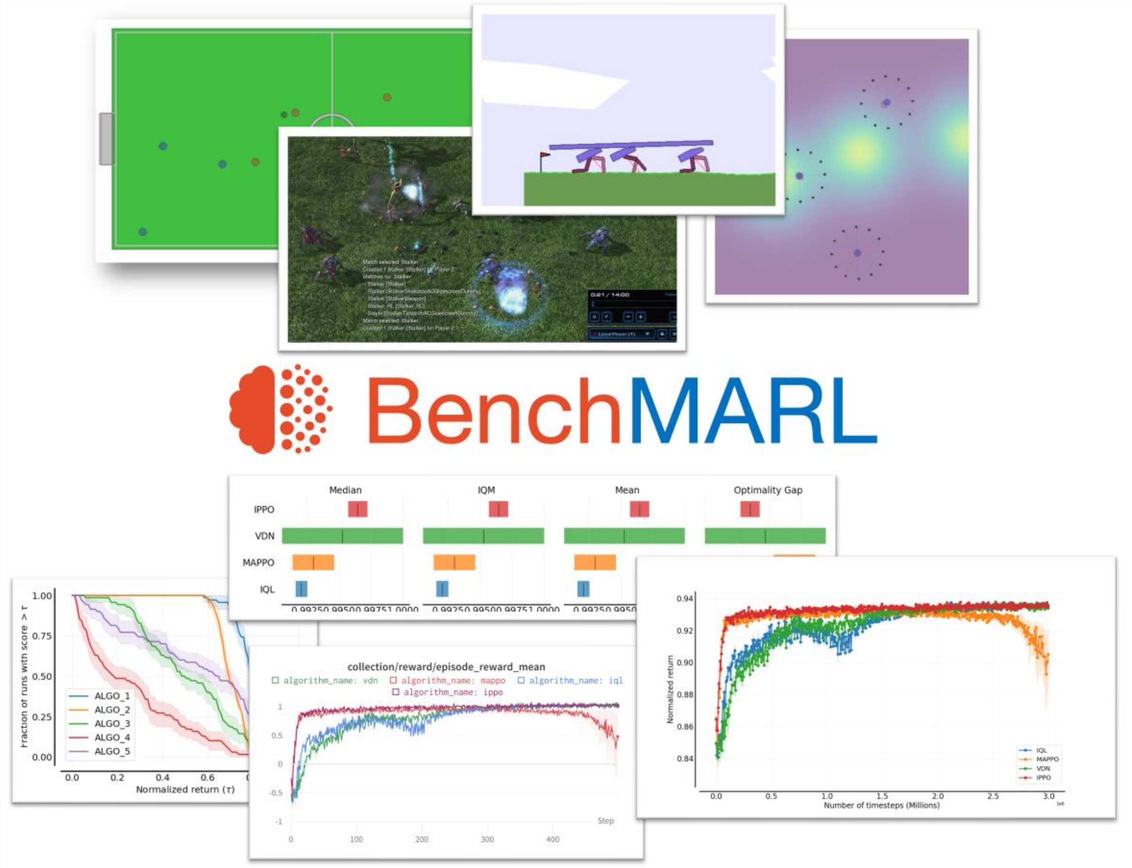
Environments and code



- PettingZoo
  - Multi-agent version of gym
  - Interface and some environments
  - https://pettingzoo.farama.org/
- JAXMARL
  - Efficient (JAX-based) baseline methods and environments
  - https://github.com/FLAIROx/JaxMARL/tree/main/jaxmarl/environments/smax



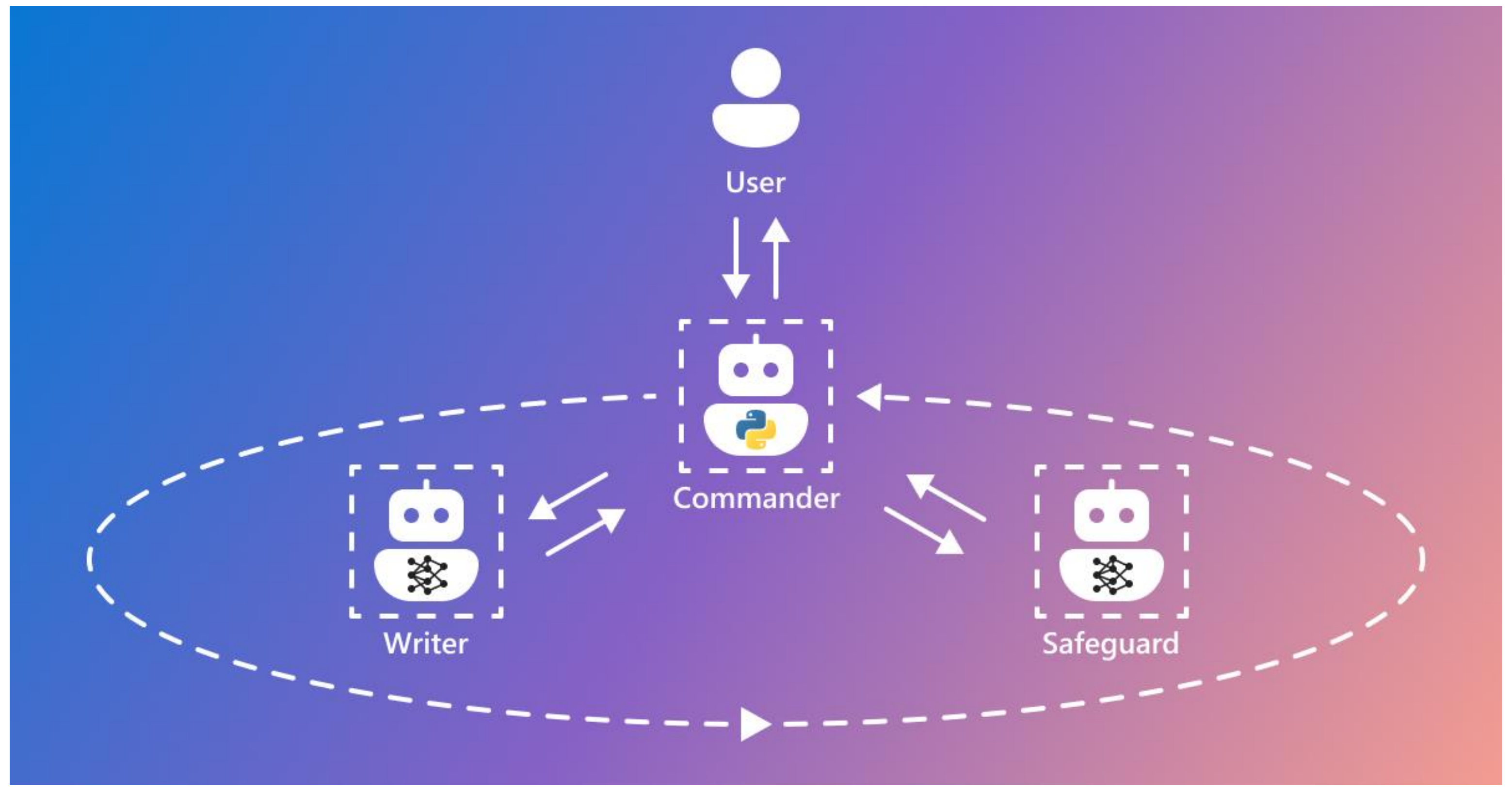
- BenchMARL
  - PyTorch baseline methods and environments
  - https://github.com/facebookresearch/BenchMARL
- Several more…



MARL and LLMs

- RL is widely used for LLMs
- MARL is ***not*** currently used for multi-agent LLMs (to best of my knowledge)
- There is no reason it couldn't be
- Open questions
    - Use cases
    - Control scheme
    - MARLHF
    - Training
- Benefits: specialization, robustness, scalability/performance
- Disconnect between academia and industry

https://developer.nvidia.com/blog/introduction-to-llm-agents/



https://www.microsoft.com/en-us/research/project/autogen/

Conclusion

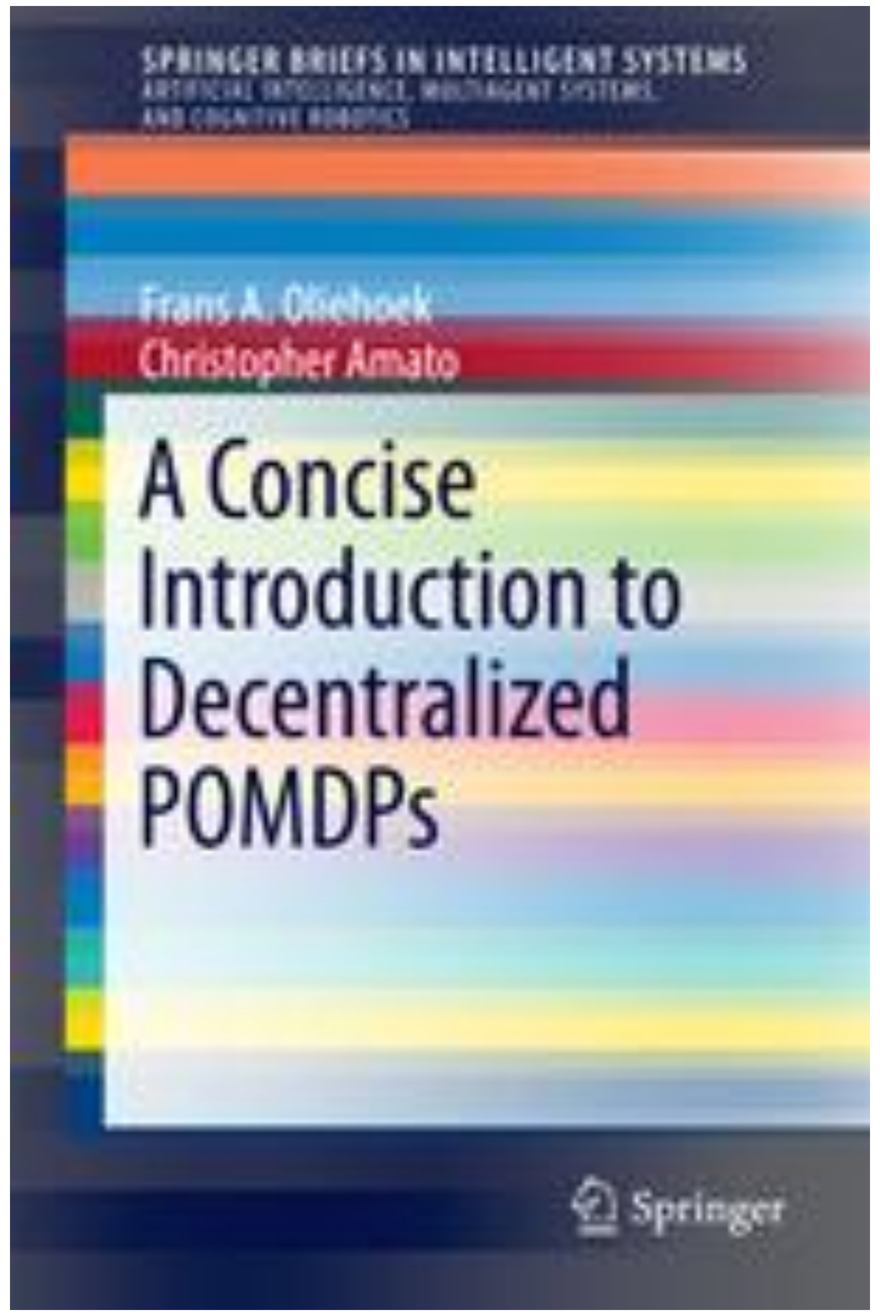
- Cooperative multi-agent reinforcement learning is a very general setting that fits with lots of applications
- A lot of work cooperative MARL
  - Centralized training and execution
  - Decentralized training and execution
  - Centralized training for decentralized execution (CTDE)
- Academia and industry are working on improved methods to improve scalability and performance

Conclusion

- Many open questions
  - MARL for LLM agents
  - Very scalable MARL
  - Optimal MARL
  - How to best do CTDE
  - Multiagent approaches to ML (e.g., GANs, decentralized methods)

Our resources

- Dec-POMDP book
  - Background on models and planning methods
- Book draft (An Initial Introduction to Cooperative Multi-Agent Reinforcement Learning): https://arxiv.org/abs/2405.06161
  - Let us know what you think and what should be changed/added for the final version!
- Slides will be available
  - https://www.khoury.northeastern.edu/home/camato/tutorials.html



Contents