

# Multi-Agent Reinforcement Learning

## Multi-Agent Reinforcement Learning in Games: First Steps and Challenges

---

Stefano V. Albrecht, Filippos Christianos, Lukas Schäfer

Slides by: Leonard Hinckeldey

This lecture is based on

## **Multi-Agent Reinforcement Learning: Foundations and Modern Approaches**

by Stefano V. Albrecht, Filippos Christianos and  
Lukas Schäfer

MIT Press, 2024

Download book, slides, and code at:

[www.marl-book.com](http://www.marl-book.com)

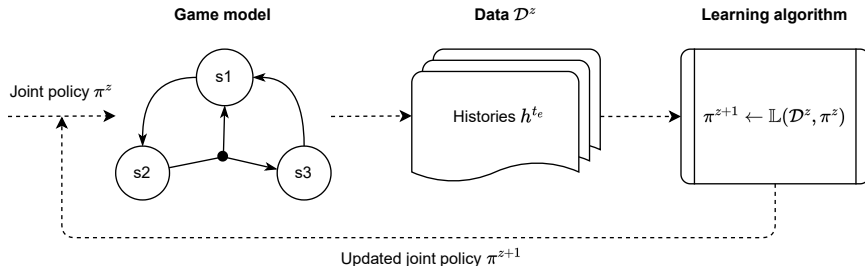


- Learning framework for MARL
- Independent learning
- Central learning
- Modes of learning
- Challenges of MARL

# MARL Learning Framework

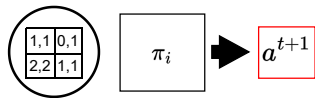
---

# MARL Learning Process



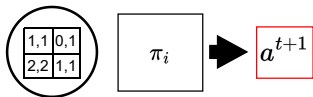
- The game model defines the learning environment
- Interaction data from joint policy  $\pi^z$  are collected as  $\mathcal{D}^z = \{h^{te} \mid e = 1, \dots, z\}, z \geq 0$
- A learning algorithm updates the joint policy as  $\pi^{z+1} = \mathbb{L}(\mathcal{D}^z, \pi^z)$
- The learning goal is a chosen solution concept, e.g. Nash equilibrium

# Inputs of Policies Depend on Game Model

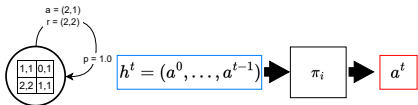


Normal-form games

# Inputs of Policies Depend on Game Model

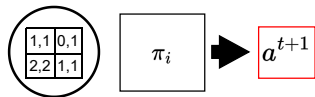


Normal-form games

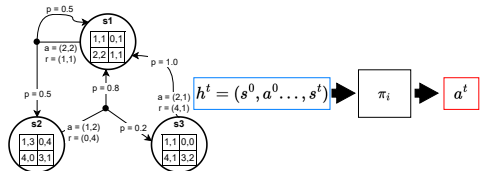


Repeated normal-form  
games

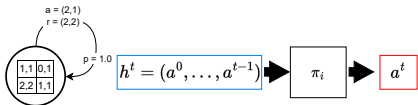
# Inputs of Policies Depend on Game Model



Normal-form games



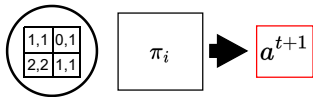
Stochastic games



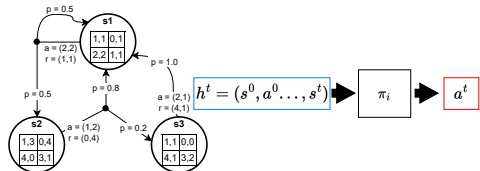
Repeated normal-form  
games



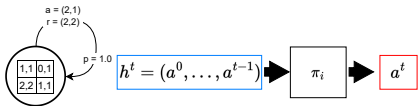
# Inputs of Policies Depend on Game Model



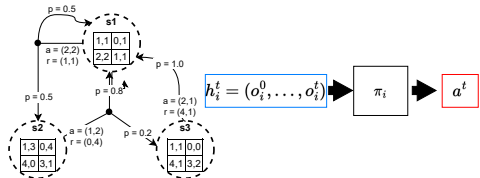
Normal-form games



Stochastic games



Repeated normal-form  
games



Partially observable stochastic games

# Convergence

To evaluate the learning algorithm, we typically assess whether the learnt joint policy has **converged** to an optimal joint policy:

$$\lim_{z \rightarrow \infty} \pi^z = \pi^*$$

- Optimal joint policies may differ depending on the solution concept
- There may be many valid solutions  $\Rightarrow$  e.g. multiple Nash equilibria
- In practice, we cannot collect infinite data  $z \rightarrow \infty$   
 $\Rightarrow$  Learning typically stops after a predefined 'budget' (e.g. training steps)

# Single-Agent RL Reductions

The simplest way to apply RL algorithms in multi-agent settings is to reduce them to single-agent problems.

# Single-Agent RL Reductions

The simplest way to apply RL algorithms in multi-agent settings is to reduce them to single-agent problems.

## Central learning:

- Apply one single-agent RL algorithm to control all agents centrally  
⇒ A central policy is learned over the joint action space

# Single-Agent RL Reductions

The simplest way to apply RL algorithms in multi-agent settings is to reduce them to single-agent problems.

## Central learning:

- Apply one single-agent RL algorithm to control all agents centrally  
⇒ A central policy is learned over the joint action space

## Independent learning:

- Apply single-agent RL algorithms to each agent independently  
⇒ Agents do not explicitly consider or represent each other

## Central Learning

---

# Central Learning

**Central learning:** learn a single central policy  $\pi_c$  which receives observations of all agents and selects an action for each agent (i.e. joint action  $(a_1, \dots, a_n)$ ).

- Requires transforming the joint reward  $(r_1, \dots, r_n)$  into a single scalar reward  $r$

**Central learning:** learn a single central policy  $\pi_c$  which receives observations of all agents and selects an action for each agent (i.e. joint action  $(a_1, \dots, a_n)$ ).

- Requires transforming the joint reward  $(r_1, \dots, r_n)$  into a single scalar reward  $r$
- This can be easy in some settings, e.g. in games with common rewards  $r = r_i$ , but difficult in zero-sum or general-sum games



**Central learning:** learn a single central policy  $\pi_c$  which receives observations of all agents and selects an action for each agent (i.e. joint action  $(a_1, \dots, a_n)$ ).

- Requires transforming the joint reward  $(r_1, \dots, r_n)$  into a single scalar reward  $r$
- This can be easy in some settings, e.g. in games with common rewards  $r = r_i$ , but difficult in zero-sum or general-sum games
- May not scale well with the number of agents, as the joint action space may grow exponentially with the number of agents

**Central learning:** learn a single central policy  $\pi_c$  which receives observations of all agents and selects an action for each agent (i.e. joint action  $(a_1, \dots, a_n)$ ).

- Requires transforming the joint reward  $(r_1, \dots, r_n)$  into a single scalar reward  $r$
- This can be easy in some settings, e.g. in games with common rewards  $r = r_i$ , but difficult in zero-sum or general-sum games
- May not scale well with the number of agents, as the joint action space may grow exponentially with the number of agents
- May also not be suitable in environments that require agents to act independently based on local observations

---

## Algorithm Central Q-learning

---

- 1: Initialize:  $Q(s, a) = 0$  for all  $s \in S$  and  $a \in A = A_1 \times \dots \times A_n$
  - 2: Repeat for every episode:
  - 3: **for**  $t = 0, 1, 2, \dots$  **do**
  - 4:   Observe current state  $s^t$
  - 5:   With probability  $\epsilon$ : choose random joint action  $a^t \in A$
  - 6:   Otherwise: choose joint action  $a^t \in \arg \max_a Q(s^t, a)$
  - 7:   Apply joint action  $a^t$ , observe rewards  $r_1^t, \dots, r_n^t$  and next state  $s^{t+1}$
  - 8:   Transform  $r_1^t, \dots, r_n^t$  into scalar reward  $r^t$
  - 9:    $Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \alpha[r^t + \gamma \max_{a'} Q(s^{t+1}, a') - Q(s^t, a^t)]$
-

# Independent Learning

---

# Independent Learning

**Independent learning:** each agent  $i$  learns its policy  $\pi_i$  using only its local history of observations.

- From the perspective of each agent  $i$ , the environment transition function looks like this:

$$\mathcal{T}_i(s^{t+1}|s^t, a_i) \propto \sum_{a_{-i} \in A_{-i}} \mathcal{T}(s^{t+1}|s^t, \langle a_i, a_{-i} \rangle) \prod_{j \neq i} \pi_j(a_j|s^t)$$

- As each agent  $j$ 's policies are updated, the action probabilities  $\pi_j$  change  
 $\Rightarrow$  From agent  $i$ 's perspective, the transition function  $\mathcal{T}_i$  appears non-stationary!

# Independent Q-learning

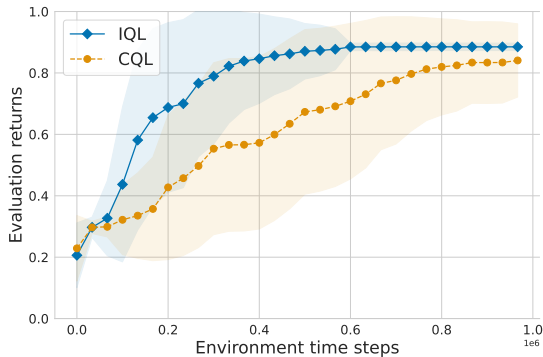
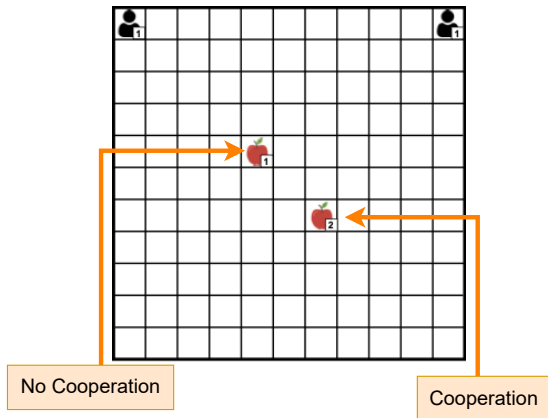
---

## Algorithm Independent Q-learning (IQL) for stochastic games

---

- 1: *// Algorithm controls agent  $i$*
  - 2: Initialize:  $Q_i(s, a_i) = 0$  for all  $s \in S, a_i \in A_i$
  - 3: Repeat for every episode:
  - 4: **for**  $t = 0, 1, 2, \dots$  **do**
  - 5:   Observe current state  $s^t$
  - 6:   With probability  $\epsilon$ : choose random action  $a_i^t \in A_i$
  - 7:   Otherwise: choose action  $a_i^t \in \arg \max_{a_i} Q_i(s^t, a_i)$
  - 8:   (meanwhile, other agents  $j \neq i$  choose their actions  $a_j^t$ )
  - 9:   Observe own reward  $r_i^t$  and next state  $s^{t+1}$
  - 10:    $Q_i(s^t, a_i^t) \leftarrow Q_i(s^t, a_i^t) + \alpha[r_i^t + \gamma \max_{a'_i} Q_i(s^{t+1}, a'_i) - Q_i(s^t, a_i^t)]$
-

# IQL and CQL in Level-Based Foraging



- IQL can learn more quickly, as CQL needs to explore  $6^2 = 36$  actions in each state

# Modes of Operation in MARL

Modes of operation in MARL:

## Self-play:

- *Algorithm self-play*: all agents use the same learning algorithm (and parameters)
- *Policy self-play*: agent's policy is trained directly against itself

## Mixed-play:

- Agents use different learning algorithms



## MARL Challenges

---

## Singe-Agent RL Challenges

- Unknown environment dynamics
- Exploration-exploitation dilemma
- Non-stationarity from bootstrapping
- Temporal credit assignment

# MARL Challenges

## Singe-Agent RL Challenges

- Unknown environment dynamics
- Exploration-exploitation dilemma
- Non-stationarity from bootstrapping
- Temporal credit assignment

## Multi-Agent RL Challenges

- Non-stationarity from multiple learning agents
- Equilibrium selection
- Multi-agent credit assignment
- Scaling to many agents

# Non-Stationarity

A stochastic process  $X_{t \in \mathbb{N}^0}^t$  is stationary if:

- The probability distribution of  $X^{t+\tau}$  does not depend on  $\tau \in \mathbb{N}^0$ , where  $t$  and  $t + \tau$  are time indices
- This means that the dynamics of the process do not change over time

# Non-Stationarity

A stochastic process  $X^t_{t \in \mathbb{N}^0}$  is stationary if:

- The probability distribution of  $X^{t+\tau}$  does not depend on  $\tau \in \mathbb{N}^0$ , where  $t$  and  $t + \tau$  are time indices
- This means that the dynamics of the process do not change over time

Consider:  $X^t$  samples the state  $s^t$  at each time step  $t$ :

- In a MDP,  $X^t$  is completely defined by the state transition function  $\mathcal{T}(s^t | s^{t-1}, a^{t-1})$  and the agent's policy  $\pi$  which selects an action  $a \sim \pi(.|s)$

# Non-Stationarity

A stochastic process  $X_{t \in \mathbb{N}^0}^t$  is stationary if:

- The probability distribution of  $X^{t+\tau}$  does not depend on  $\tau \in \mathbb{N}^0$ , where  $t$  and  $t + \tau$  are time indices
- This means that the dynamics of the process do not change over time

Consider:  $X^t$  samples the state  $s^t$  at each time step  $t$ :

- In a MDP,  $X^t$  is completely defined by the state transition function  $\mathcal{T}(s^t | s^{t-1}, a^{t-1})$  and the agent's policy  $\pi$  which selects an action  $a \sim \pi(.|s)$
- If  $\pi$  does not change, then this process is stationary (i.e. independent of  $t$ ) because  $s^t$  depends only on  $s^{t-1}$ ,  $a^{t-1}$  and  $a^{t-1}$  depends only on  $s^{t-1}$  via  $\pi(.|s^{t-1})$

# Non-Stationarity

A stochastic process  $X^t_{t \in \mathbb{N}^0}$  is stationary if:

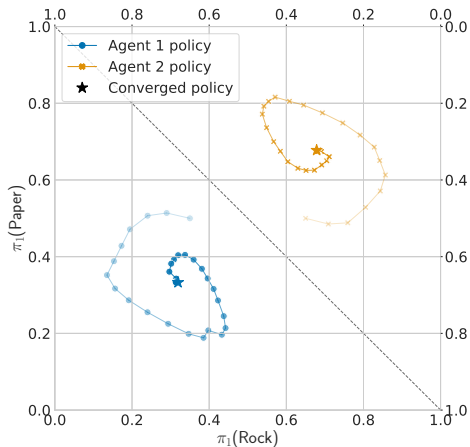
- The probability distribution of  $X^{t+\tau}$  does not depend on  $\tau \in \mathbb{N}^0$ , where  $t$  and  $t + \tau$  are time indices
- This means that the dynamics of the process do not change over time

Consider:  $X^t$  samples the state  $s^t$  at each time step  $t$ :

- In a MDP,  $X^t$  is completely defined by the state transition function  $\mathcal{T}(s^t | s^{t-1}, a^{t-1})$  and the agent's policy  $\pi$  which selects an action  $a \sim \pi(\cdot | s)$
- If  $\pi$  does not change, then this process is stationary (i.e. independent of  $t$ ) because  $s^t$  depends only on  $s^{t-1}$ ,  $a^{t-1}$  and  $a^{t-1}$  depends only on  $s^{t-1}$  via  $\pi(\cdot | s^{t-1})$
- However, in RL the policy does change over time through the learning  $\pi^{z+1} = \mathbb{L}(\mathcal{D}^z, \pi^z)$ , which leads to **non-stationarity** in  $X^t$

# Non-Stationarity in Multi-Agent Settings

In MARL, non-stationarity is exacerbated by multiple agents changing their policies!



- $\pi^{z+1} = \mathbb{L}(\mathcal{D}^z, \pi^z)$  updates an *entire* joint policy  $\pi^z = (\pi_1^z, \dots, \pi_n^z)$
- Environment is non-stationary from each agent's perspective
- Can cause cyclic learning dynamics where agents co-adapt to each other's changing policies



# Equilibrium Selection

**Equilibrium selection:** a game may have multiple equilibria, which can yield different expected returns to the agents.

- **Example:** Stag Hunt matrix game
- Two hunters choose: cooperate to hunt stag (S) or go solo for hare (H)

	S	H
S	4,4	0,3
H	3,0	2,2

# Equilibrium Selection

**Equilibrium selection:** a game may have multiple equilibria, which can yield different expected returns to the agents.

- **Example:** Stag Hunt matrix game
- Two hunters choose: cooperate to hunt stag (S) or go solo for hare (H)
- Nash equilibria: reward-dominant (S,S)  
maximizes reward, risk-dominant (H,H)  
minimizes risk

	S	H
S	4,4	0,3
H	3,0	2,2

# Equilibrium Selection

**Equilibrium selection:** a game may have multiple equilibria, which can yield different expected returns to the agents.

- **Example:** Stag Hunt matrix game
- Two hunters choose: cooperate to hunt stag (S) or go solo for hare (H)
- Nash equilibria: reward-dominant (S,S) maximizes reward, risk-dominant (H,H) minimizes risk
- (S,S) requires trust; (H,H) offers a safe, lower reward

	S	H
S	4,4	0,3
H	3,0	2,2

# Equilibrium Selection

**Equilibrium selection:** a game may have multiple equilibria, which can yield different expected returns to the agents.

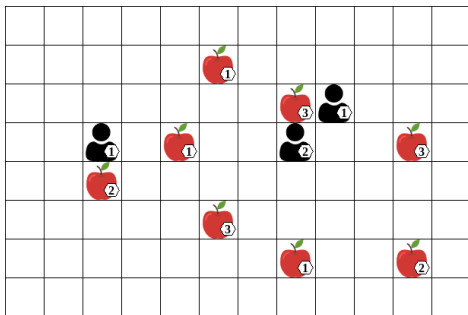
- **Example:** Stag Hunt matrix game
- Two hunters choose: cooperate to hunt stag (S) or go solo for hare (H)
- Nash equilibria: reward-dominant (S,S) maximizes reward, risk-dominant (H,H) minimizes risk
- (S,S) requires trust; (H,H) offers a safe, lower reward

	S	H
S	4,4	0,3
H	3,0	2,2

- Indep. Q-learning may bias towards (H,H) due to initial action uncertainty
- Early random actions can reinforce (H,H) since deviating from H is penalized if the other agent chooses H

# Multi-Agent Credit Assignment

**Multi-agent credit assignment:** which agent's actions contributed to received rewards?



- At time step  $t$  all agents perform *collect*, each receiving reward  $+1$
- Whose actions led to the reward?
- The agent on the left did not contribute
- For a learning agent that only observes  $s^t, a^t, r^t, s^{t+1}$ , disentangling the action contributions is difficult

# Joint Actions for Addressing Multi-Agent Credit Assignment

**Joint actions** can help disentangle agent contributions. Consider the RPS game:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

# Joint Actions for Addressing Multi-Agent Credit Assignment

**Joint actions** can help disentangle agent contributions. Consider the RPS game:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

1. Agents choose actions  $(a_1, a_2) = (R, S) \Rightarrow$  agent 1 receives reward +1

# Joint Actions for Addressing Multi-Agent Credit Assignment

**Joint actions** can help disentangle agent contributions. Consider the RPS game:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

1. Agents choose actions  $(a_1, a_2) = (R, S) \Rightarrow$  agent 1 receives reward +1
2. Then agents choose  $(a_1, a_2) = (R, P) \Rightarrow$  agent 1 receives reward -1



# Joint Actions for Addressing Multi-Agent Credit Assignment

**Joint actions** can help disentangle agent contributions. Consider the RPS game:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

1. Agents choose actions  $(a_1, a_2) = (R, S) \Rightarrow$  agent 1 receives reward +1
2. Then agents choose  $(a_1, a_2) = (R, P) \Rightarrow$  agent 1 receives reward -1
3. Action value  $Q(a_1)$  does not model agent 2, value for action  $R$  appears to be 0

# Joint Actions for Addressing Multi-Agent Credit Assignment

**Joint actions** can help disentangle agent contributions. Consider the RPS game:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

1. Agents choose actions  $(a_1, a_2) = (R, S) \Rightarrow$  agent 1 receives reward +1
2. Then agents choose  $(a_1, a_2) = (R, P) \Rightarrow$  agent 1 receives reward -1
3. Action value  $Q(a_1)$  does not model agent 2, value for action  $R$  appears to be 0
4. **Joint action value model**  $Q_1(a_1, a_2)$  can represent the effect of agent 2:  
 $Q_1(R, S) = +1$  and  $Q_1(R, P) = -1$

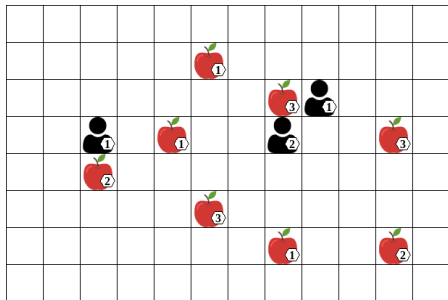
## Problem

- **Joint action space** can grow **exponentially** with number of agents:

$$|A| = |A_1| \cdot \dots \cdot |A_n|$$

- If agents have associated features in  $s$  (e.g. agent position) then  $|S|$  also increases exponentially  
⇒ How to scale efficiently to many agents?

# Scaling to Many Agents



Changing number of agents from 3 to 5 increases the number of joint actions from 216 to 7776!

## We covered:

- MARL learning process
- Independent and central learning
- Modes of operation in MARL
- Challenges of MARL

## Next we'll cover:

- Foundational algorithms in MARL