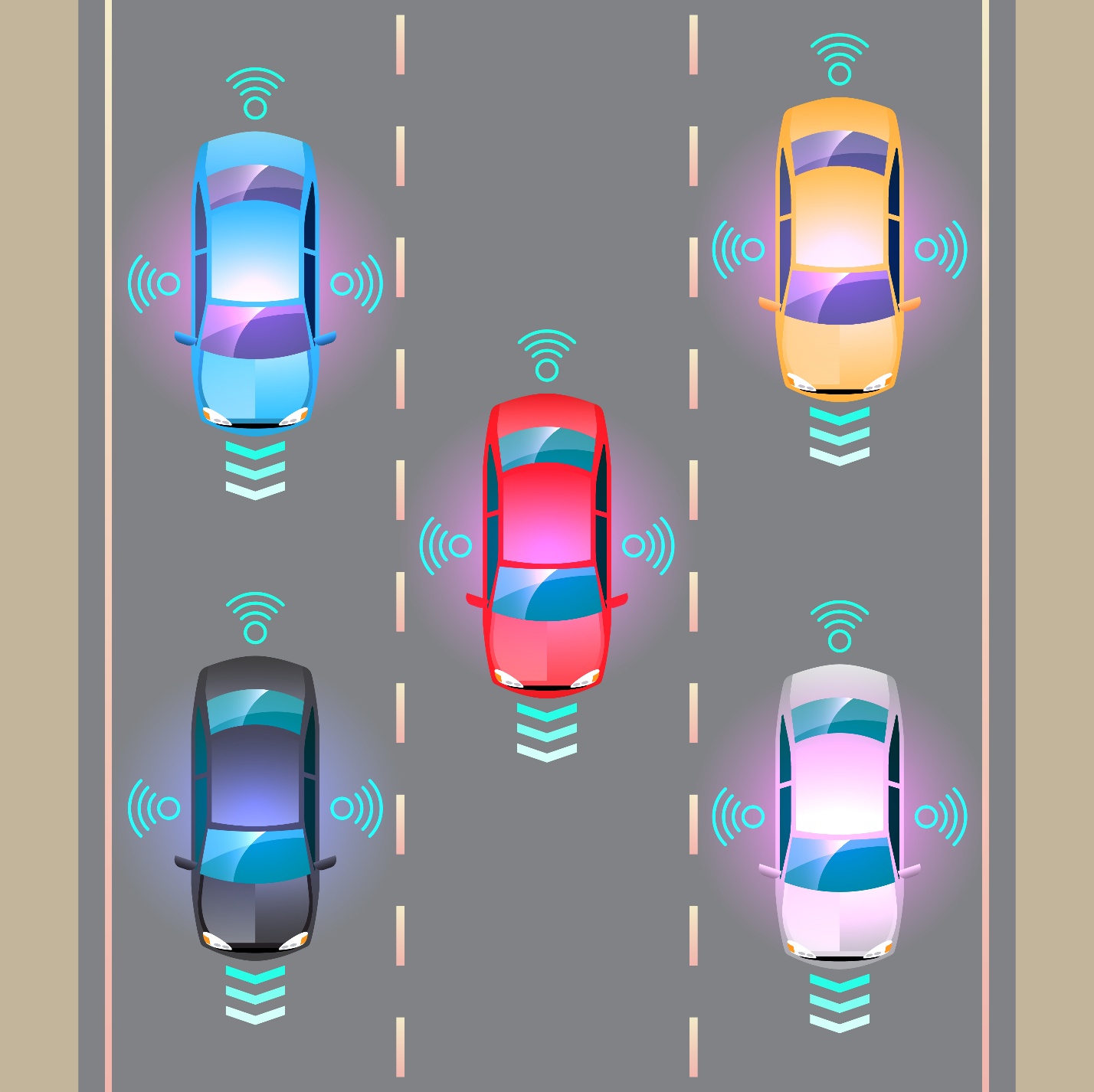
**An introduction to Multi-Agents Reinforcement Learning (MARL)**

In the first unit, we learned to train agents in a single-agent system. When our agent was alone in its environment: **it was not cooperating or collaborating with other agents**.

When we do multi-agents reinforcement learning (MARL), we are in a situation where we have multiple agents **that share and interact in a common environment**.

first, let’s understand the different types of multi-agent environments.

### 1. **Cooperative Environments**

In **cooperative environments**, all agents work together towards a **shared goal**. The reward is often **distributed equally** or based on joint performance. These systems emphasize **collaboration**, with agents needing to **coordinate** and optimize their actions to benefit all agents involved.

#### Example:

* **Warehouse Automation**:
  + **Scenario**: Multiple robots work together to load and unload packages efficiently.
  + **Goal**: All robots cooperate to maximize throughput and reduce delays.
  + **Reward**: The total benefit (speed and efficiency) is shared among robots, encouraging teamwork.

#### Key Characteristics:

* **Shared rewards**: Agents collaborate to reach a common objective.
* **Coordination needed**: Agents must synchronize actions (e.g., avoid collisions, allocate tasks).
* **Joint reward maximization**: Success is measured by collective performance.

****

### 2. **Competitive/Adversarial Environments**

In **competitive environments**, agents have **conflicting goals** and are in direct competition with each other. Each agent seeks to maximize its own **reward** by minimizing the opponent's reward. These systems often involve **adversarial** interactions where agents "compete" to outperform others.

#### Example:

* **Tennis Game**:
  + **Scenario**: Two agents play a game of tennis where each agent wants to win.
  + **Goal**: Each agent aims to maximize their own score and **minimize the opponent's score**.
  + **Reward**: The reward is typically positive when winning and negative when losing, emphasizing individual success.

#### Key Characteristics:

* **Conflicting rewards**: Agents benefit when others do poorly (zero-sum game).
* **Strategic decision-making**: Agents often need to predict and counteract opponents’ actions.
* **Competition-driven**: Success depends on outsmarting or out-performing the other agents.



### 3. **Mixed Environments**

A **mixed environment** is where agents experience both cooperative and competitive dynamics. Some agents might collaborate, while others are in competition. This type of environment requires agents to **adjust their strategies** depending on the situation (sometimes cooperating, sometimes competing).

#### Example:

* **Team Sports** (like Soccer or Basketball):
  + **Scenario**: Teams of agents compete against each other, but teammates must cooperate within their team while competing with the opposing team.
  + **Goal**: Team members cooperate to score points, but still compete against the other team to win the game.
  + **Reward**: Team members share rewards for scoring, but individual performance may also matter.

#### Key Characteristics:

* **Mixed rewards**: Cooperation within teams, but competition between teams.
* **Dual objectives**: Agents must balance cooperation with competition.
* **Complex strategy**: Agents must determine when to cooperate or act competitively.

Let’s look at some examples of MARL in different domains:

1. Autonomous driving: MARL is used to simulate and train

autonomous vehicles to navigate complex traffic scenarios, where

each vehicle (agent) must consider the actions of others to avoid

collisions and optimize traffic flow.

2. Robotics: In collaborative robotics, MARL enables multiple robots

to work together on tasks such as assembly lines or search-and-

rescue missions, coordinating their actions for efficiency and

effectiveness.

3. E-commerce: Online platforms use MARL to model interactions

between buyers, sellers, and recommendation systems,

optimizing for user satisfaction and revenue.

4. Multiplayer online games: Game developers use MARL to create

more intelligent and adaptable non-player characters (NPCs) that

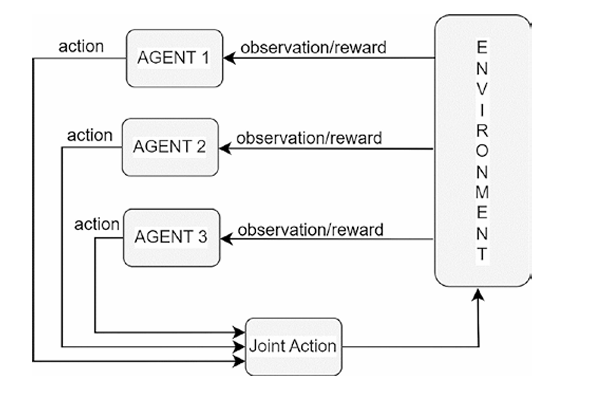
can interact with human players or other NPCs in complex game

environments.

5. Financial markets: MARL models the behavior of traders in

financial markets, helping to understand market dynamics and

develop strategies for trading and investment

The next question that comes to mind is how do these agents learn in a setup like this? It is actually very similar to the RL you have been seeing so far.

**This picture illustrates the basic MARL training loop.**

Each of the agents picks an individual action, and the combination of these actions is called joint action.

The joint action affects the state of the environment according to the environment dynamics

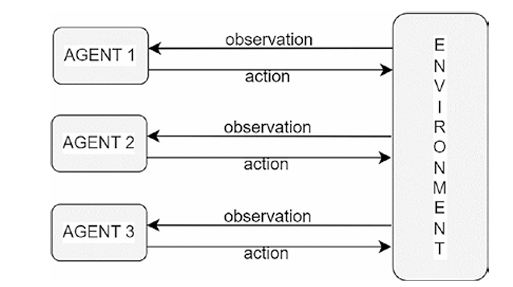
and the agents get individual rewards from this change, as well as individual observations about the new environment state.

This loop repeats until a stopping criterion is met such as one agent losing a game or all agents collaboratively reaching an end goal.

A full run of this loop from the initial state to the final state is called an episode.

**The data generated from multiple separate episodes—that is, the observed observations, actions, and rewards in each episode—are used to constantly improve the agents’ policy/reward.**

After the agent has been trained, in actual runs, the agents may still act independently , or jointly.



Could you still solve it using what you have learned so far for single agent setup?

Theoretically, there is a way. Think of the three agents as a combined single agent. And think of combining individual rewards into a collective single reward. The same view can be extended to observations and actions as well.

With this approach you can recast a multi-agent setup to a single agent setup. Though theoretically possible, it has a few challenges when viewed as a single combined-agent RL. I briefly talk about these challenges.

Suppose each agent has a single dimension action with five discrete values and further suppose that you are combining three such agents together. What is the action space for the combined agent? It will be 5x5x5=125 actions. The same applies to state/ observation as well. Therefore, using a single agent RL approach leads to exponential growth of the possible action space and state/observation space.

Another challenge is when agents act sequentially, like in a game of Chess or Go. Each agent takes a turn to act and the action depends on the action of other player. In a setup like this, the agents’ actions are not independent. There is no easy way to look at it as a single agent RL setup.

The third challenge is with the combination of state/observations. Consider the case of a bunch of self-driving cars. If they are treated as a single agent RL, you need to combine the local observation of each car together into a big observation vector. For two cars which are, say, a few kilometers apart, the decision to steer each car will mostly depend on the local state of that car. The state/observation of the car far away will have no bearing (unless it involves a police chase!). However, by combining the states of both the car together, you are providing a whole lot of irrelevant information, thus making learning tougher or almost impossible

Therefore, there is a need to have a separate approach and a separate set of algorithms that recognize the presence of multiple agents and their unique challenges and then carry out planning effectively.

**However, depending on the abstraction used, treating each agent as an individual entity under MARL has a different set of challenges.**

***Key Challenges in MARL:***

**Moving and changing targets**

in RL, unlike supervised learning where the training data is given beforehand, new data/ observations appear as agents learn a behavior and explore the environment. The training data is not fixed, so learning a target is like aiming at a moving target

This problem is even bigger in MARL.

 Since there are multiple agents, agent tries to adjust to the policies of the other agents and vice versa, in a cycle. This creates an unstable cyclic learning dynamic that requires MARL algorithms to deal with this non-stationary aspect in a more reliable way than single agent RL algorithms.

**The best policy and the balance**

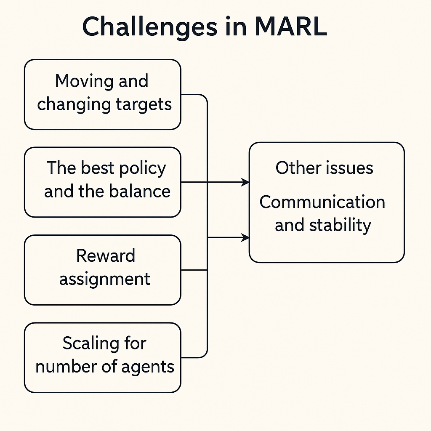
In a single agent RL, the best policy means getting the highest expected returns in each state. But when there are multiple agents together, the return for each agent depends on the return for the others. So, what is the best result? Is it for one agent, all agents together or relative to what? How do you combine the returns of different agents? There may be more than one policy with the same total return but different individual returns. If the environment is cooperative or competitive, how does that affect the idea of the best? The whole idea of the best depends on the context and that needs to be abstracted in MARL.

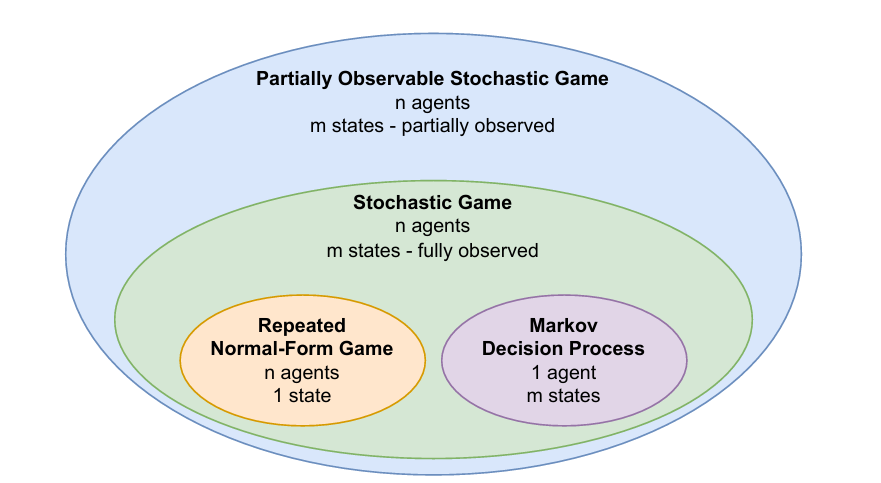
**Reward assignment**

In RL, reward assignment, also known as temporal credit assignment in RL, is the challenge of determining which of the previous actions contributed to the reward agent receives at a given instance. In MARL, this challenge is even more complicated because it also requires identifying who performed the action that resulted in the reward. With only this state/action/reward information, it can be very hard to distinguish the influence of each agent on the received reward, especially if an agent did not contribute to the reward since its action was irrelevant. Although theoretical ideas based on counterfactual reasoning can address this problem, it is still an open problem how to do multi-agent reward assignment in an effective and scalable way.

**Scaling for number of agents**

in a MARL environment, as the number of agents grows, the number of combinations of agent actions also increase and could do so in an exponential way. The current state-of-the-art MARL approaches does not handle a large number of agents. It is an active area of research.



**Game Models**

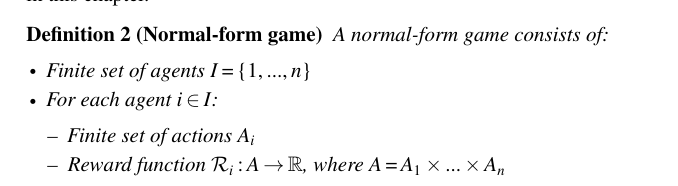
**You have been mostly learning about Markov Decision Processes (MDP) introduced which involves single agent in the environment.**

It is based on the action and response of the environment to the action; the agent can move from one state to another.

and we introduced multi arm bandit, which is a special case of MDP with one agent and one state. After the agent takes an action, the agent gets a reward based on the action and then the agent state is reset to the beginning.

Just like multi arm bandit was the foundational building block of MDP, repeated normal form game is the foundational block of MARL.  
in this chapter, I use “environment” and “game” interchangeably. The word “game” comes from the discipline of “game theory” which is the foundational concept on which MARL models are built.

Normal-Form Games



In a normal-form game, there are no time steps or states. Agents choose an action and observe a reward.

The game proceeds as follows:

1. Each agent samples an action ai ∈ Ai with probability πi(ai)

2. The resulting actions from all agents form a joint action, a = (a1,...,an)

3. Each agent receives a reward based on its individual reward function and the

joint action, ri = Ri(a)

Normal form games can be further sub-classified based on the type impacting the reward structure.

• Zero-sum game: The sum of agents’ rewards is zero.

In the case of a two-agent setup, the gain of one agent is a loss of the same amount

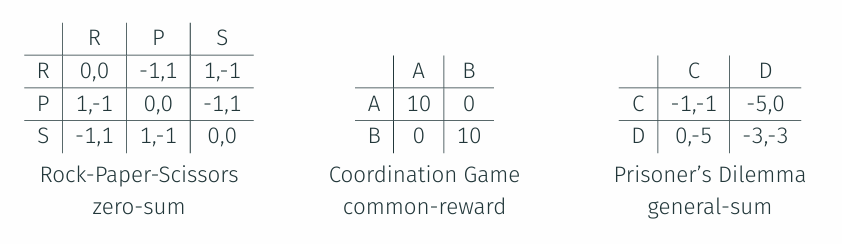
for the other one.

 • Common-reward game: All agents receive the same reward. Most

cooperative behaviors would fall under this category.

• General-sum game: All other types that do not fit into the previous

two categories

Two-player normal form games are also known as Matrix games because the actions of one player can be represented as rows of a matrix and the actions of the second agent can be represented as columns of the matrix. The reward in this case is shown in the cells of the matrix as a tuple, with the first element of the tuple showing the reward of the first agent and the second element of the tuple showing the reward of the second agent.

### The Prisoner’s Dilemma is a classic scenario in game theory that shows how two individuals acting in their own self-interest can end up worse off—even when cooperation would have given them a better outcome. **Key Idea:**

* For each individual, **defecting always seems better**, since it either frees them or reduces prison time.
* But if **both defect**, **both lose more** than if they had cooperated.

This means **"Defect" is a dominant strategy** — it gives a better outcome **no matter what** the other player does.

The dilemma shows how **lack of trust** leads to both players choosing the worse outcome, even though **trust and cooperation would benefit both**.

Repeated normal-form game

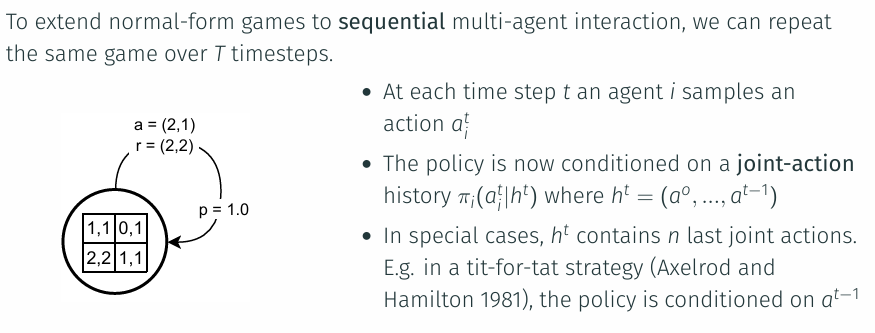
**"Repeated normal-form game is nothing but repeating this in a loop similar to taking multiple repeated actions, such as with the multi-arm bandit."**

When I talk about repeated actions, the time horizon also becomes important. Games with finite time may demonstrate different behavior as the game progresses and gets close to the end. **The agents may choose different actions closer to the end of the game versus the actions toward the start.**

With infinite time internal actions, after some stabilization over certain time periods, the agent actions will converge to the same distribution as kind of a steady state. As discussed earlier, with infinite interval games, you use the concept of γ, the discounting factor, to add the rewards in order to keep the sum of the rewards bounded.

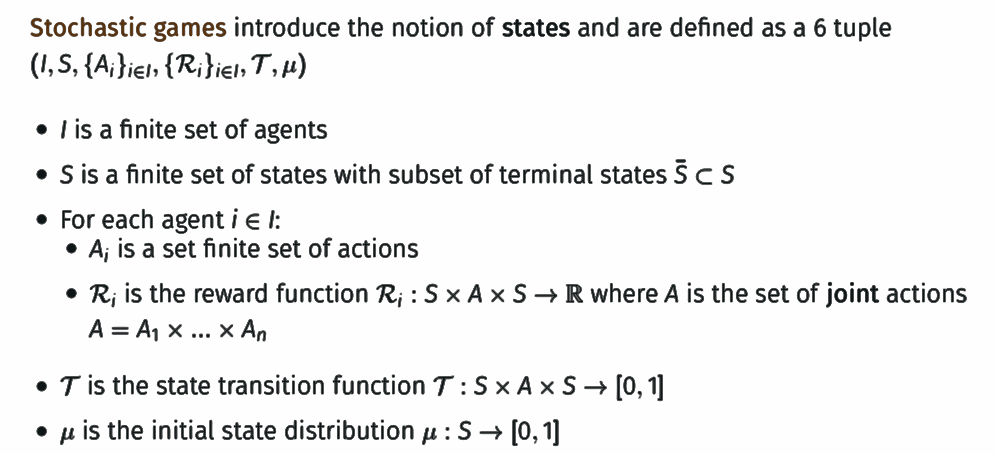
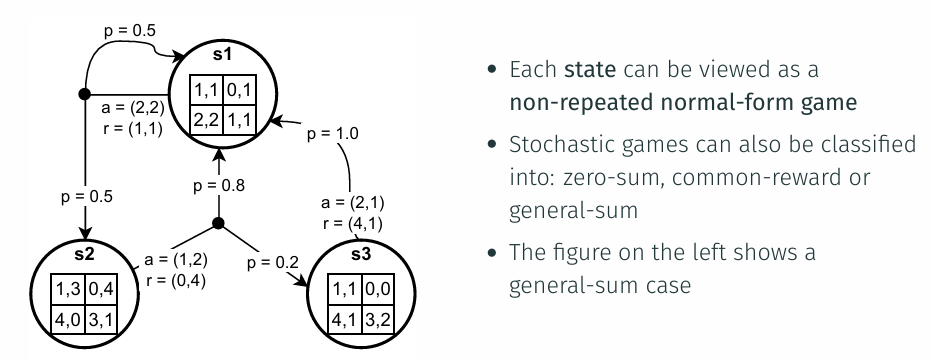
یعنی اگر بازی برای همیشه ادامه داشته باشه، در نهایت رفتار عامل‌ها به یک الگوی پایدار و قابل پیش‌بینی تبدیل می‌شه.

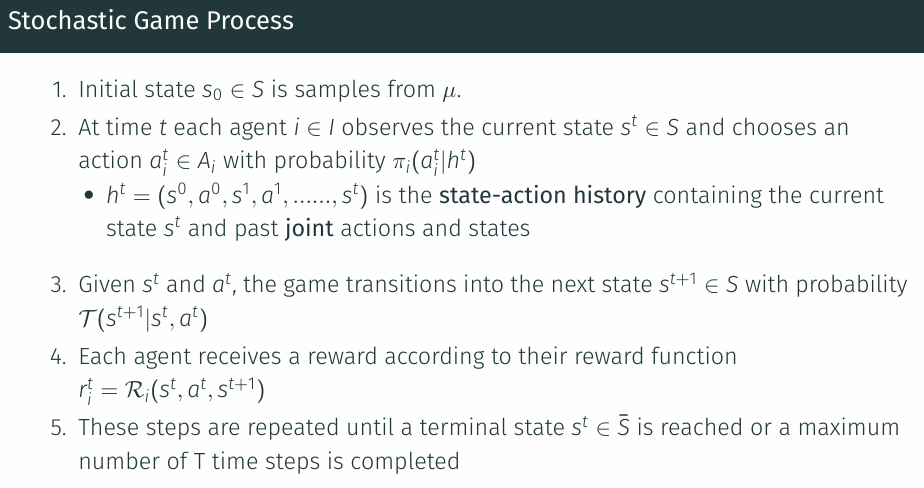
For example, the repeated Prisoner’s Dilemma has been extensively studied in the game theory literature (e.g., Axelrod 1984)...

.

توضیح: در این استراتژی ساده، هر عامل در هر مرحله دقیقاً همان کاری را انجام می‌دهد که عامل دیگر در مرحله‌ی قبل انجام داده. مثلاً اگر قبلاً همکاری کردی، من هم همکاری می‌کنم. اگر خیانت کردی، من هم خیانت می‌کنم. این یک مثال معروف در نظریه بازی‌هاست برای تشویق به همکاری بلندمدت

Stochastic Games

Next in the hierarchy is the concept of Stochastic games. This setup extends the repeated-normal-form with a single state to multiple states, very similar to the extension of multi-arm-bandit to MDP. This is a full multi-agent system, defined as MARL. Just like a single agent MDP, you now have each agent acting based on current state. These actions are combined to get joint action. The joint action is passed over to the environment, which responds with reward and the next state for each individual agent. The agents then use the new state to reinitiate the action-reward-next state loop. Similar to MDP, stochastic games have a Markov property—that is, the probability of the next state and action for an agent being conditionally independent of the past states and joint action given the current states and joint action. It is the same Markov Property you saw for RL and a single agent. Now, it is extended from one agent to a multi-agent setup. repeated normal-form games are a special case of stochastic games with only one state.



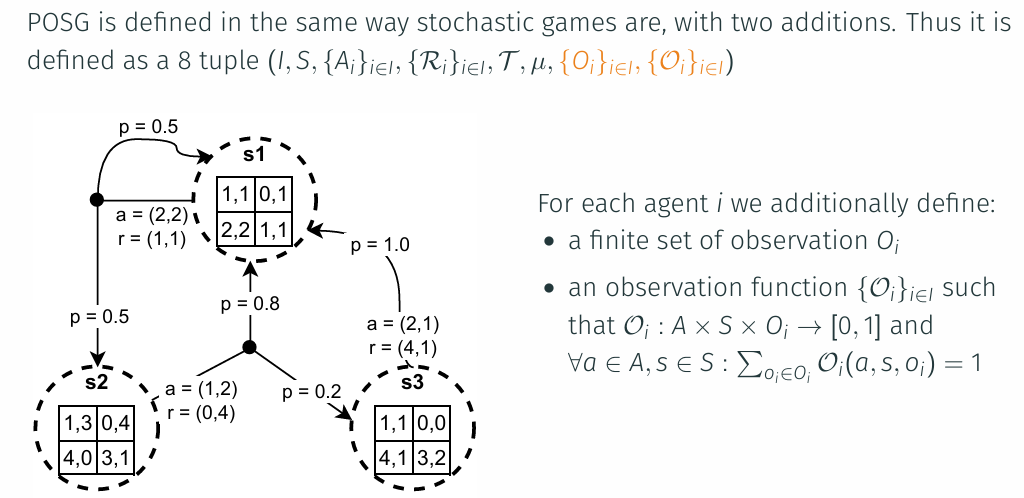
partially observable stochastic games

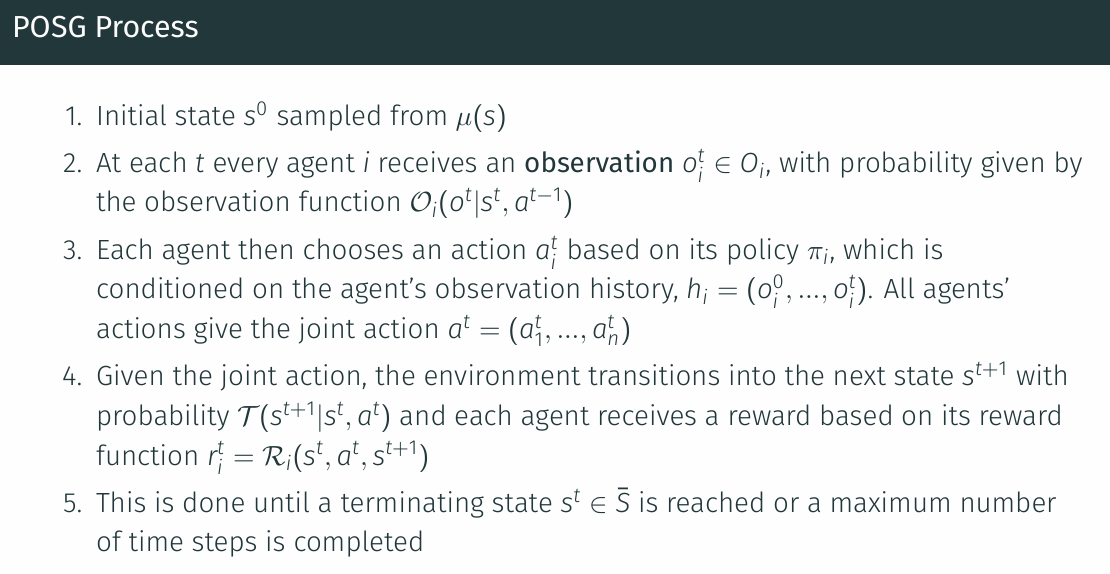
Extending the generalization, you can move from fully observable stochastic games to partially observable stochastic games.

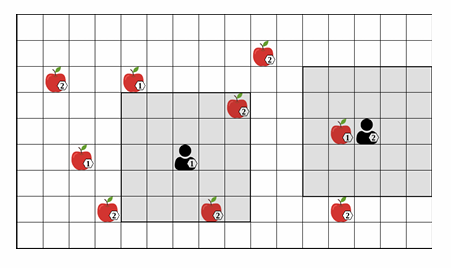
**while in stochastic games, the agents are able to directly observe the state of the environment, in Partially Observable Stochastic Games (POSG), the agent gets an observation, which is an incomplete information of the actual state.**

Usually for each agent i, an individual observation function Oi is defined which specifies the probabilities over the agent’s possible observations oi given the actual environment state st and joint action in previous time step at − 1. This is represented as . Accordingly, each agent’s action/policy depends on the history of the observations now. This distinction of zero-sum games, common-reward games and general-sum reward games also extends to POSG. The common reward games are also known as decentralized POMDP. The observation function Oi can be used to present diverse situations.

In some other cases the agents may only observe a subset of the state and joint actions. An example of this is self-driving cars.



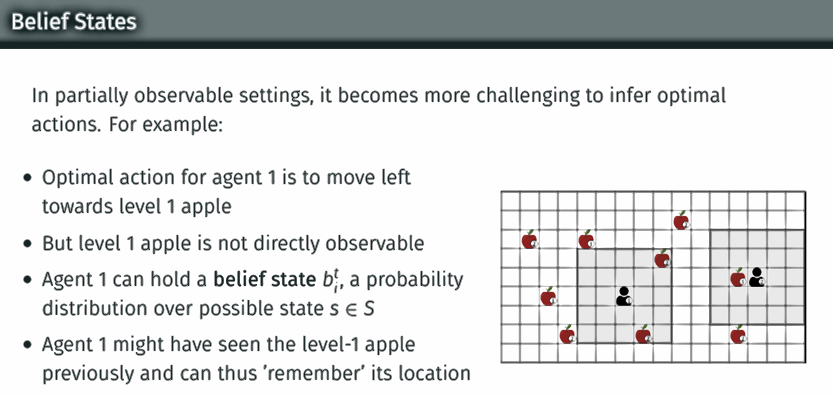
**For POSG, the states are dashed to represent that agents do not directly observe the current state of the game; instead, agents receive partial/noisy observations about the state."**

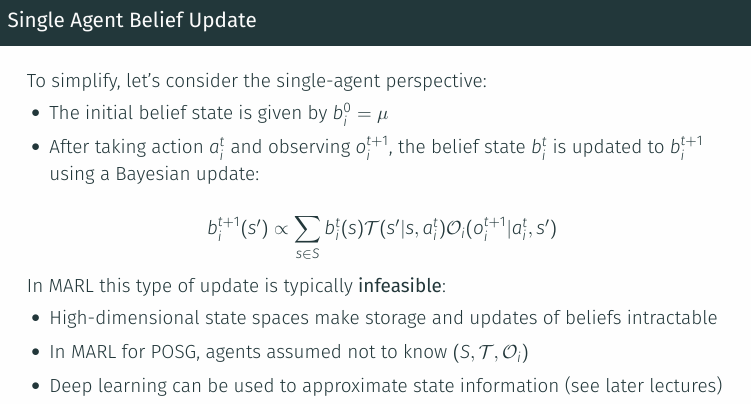


Belief States

in a POSG, since an agent’s observation only provides partial information about the current state of the environment, it is typically not possible to choose optimal actions based only on the current observation.

In general, if the environment is only partially observed, the agents must maintain estimates of the possible current states and their relative likelihoods based on the history of observations.





The process of updating belief states is also known as (belief state) filtering.

in a POSG with more than one agent, the definition of belief states becomes significantly more complex.

agents now also have to infer probabilities over the possible observations and actions of other agents.

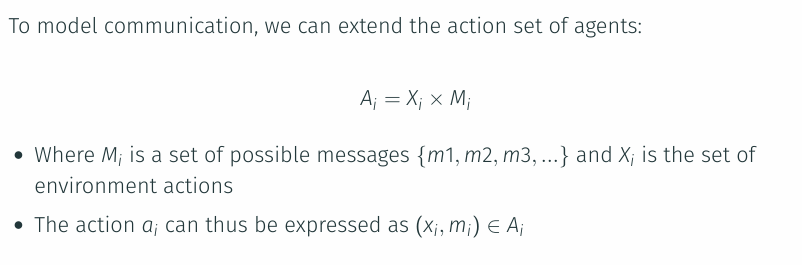
**To achieve some form of filtering without knowledge of these elements, deep RL algorithms often use recurrent neural networks.** **The output vector of the recurrent network learns to encode information about the current state...**

In **partially observable** settings (e.g., StarCraft II, fog-of-war, local vision), agents need some form of **belief estimation** or memory. This is typically done using **RNNs** (like LSTM/GRU) to approximate belief states.

Modeling Communication

Stochastic games and POSGs can also model communication between agents by including communication actions that agents can use to send messages to other agents.

Intuitively, we can view communication as a type of action that can be observed by other agents but does not affect the state of the environment.

In contrast to environment actions, communication actions do not affect the next state of the environment. Communications are ephemeral—they last only for a single time step—but agents can remember them using the history .

The agents may use messages to communicate observations, goals, or plans.

**However, in reinforcement learning, agents do not know the meaning of communication actions and must learn them.**

| **Algorithm** | **MARL Application** | **LLM Application** |
| --- | --- | --- |
| **MADDPG (Multi-Agent Deep Deterministic Policy Gradient)** | Centralized training with decentralized execution, enabling communication and coordination in multi-agent environments. | Principles of decentralized execution and coordination can be used in **multi-agent interaction** in LLM tasks. |
| **COMA (Counterfactual Multi-Agent Policy Gradient)** | Addresses **credit assignment** in cooperative multi-agent environments, where agents learn how to share rewards. | Can be adapted to determine responsibility in **collaborative dialogue systems** or shared tasks in LLMs. |
| **QMIX (Value Decomposition Networks)** | Decomposes global Q-values into individual Q-values for cooperative tasks in multi-agent settings. | Can be used for **task decomposition** in LLMs, e.g., multi-step reasoning or collaboration in dialogue systems. |
| **CommNet (Communication Network)** | Models agent communication as a learned protocol to improve performance in multi-agent environments. | Learning communication protocols for **agents** to exchange messages, useful in **multi-agent dialogue**. |
| **DIAL (Differentiable Inter-Agent Learning)** | Enables differentiable communication learning between agents, improving coordination. | Can be applied to **dialogue systems** where communication between agents is learned and optimized. |
| **Hierarchical Reinforcement Learning (HRL)** | Breaks down tasks into smaller subtasks for cooperative agents, improving task management in complex environments. | Can be applied in LLMs to **break down tasks** into subgoals (e.g., goal setting and execution in dialogues). |
| **Self-Play Algorithms (e.g., AlphaZero, OpenAI Five)** | Agents learn from interaction with each other, improving performance in **competitive and cooperative tasks**. | LLMs can use self-play for **zero-shot learning** or **multi-turn conversation systems** where models interact. |
| **Recurrent Neural Networks (RNNs) / LSTMs** | Used for **memory-based decision-making** in partially observable environments, retaining historical information. | Core in **sequence processing** for LLMs in tasks like **dialogue** and **language modeling** (multi-turn interaction). |
| **Transformers (e.g., GPT, BERT)** | Can be adapted to **multi-agent environments** where agents process sequences of actions and communications. | Central to **modern NLP architectures**, such as **dialogue systems**, **language modeling**, and **question answering**. |
| **Proximal Policy Optimization (PPO)** | Ensures **stable policy updates** in multi-agent settings, improving training performance in MARL. | Can optimize **response generation** in LLMs for better dialogue performance. |

بخش دوم

Solution Concepts

What is a solution in MARL? A previous section introduced the basic environment setup for multi-agent scenarios—how they act, what ways they could pass messages, and so on.

**However, this on its own does not help in developing a learning process. You need one more thing, which is the objective that you are trying to optimize.**

**In single RL, you used total discounted reward, also known as return, as a metric to be maximized.**

Similarly, you need to define an objective that you want to maximize in a MARL setup.

As there are multiple agents involved in MARL, the objective that you need to optimize has its own peculiarities.

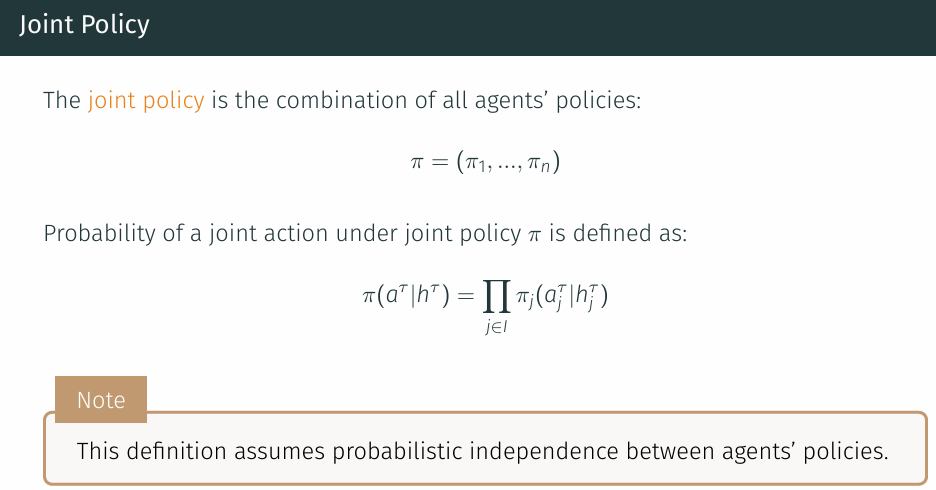
For common-reward games, in which all agents receive the same reward, a straightforward definition for a solution is to maximize the expected return received by all agents.

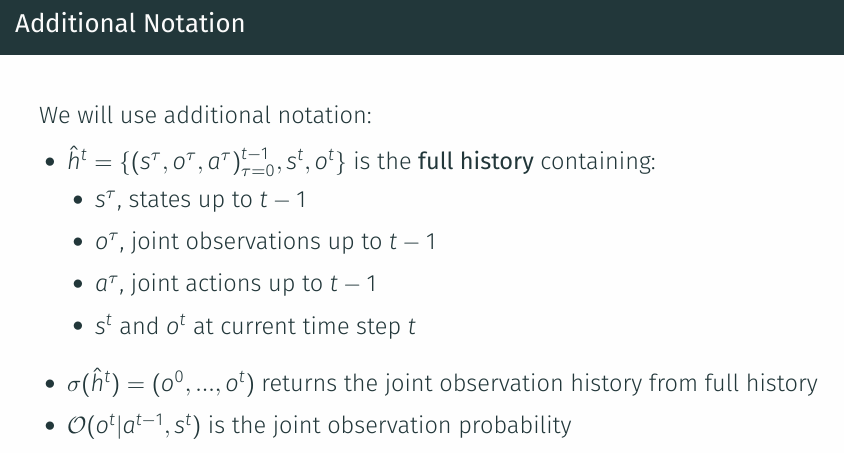
However, if the agents have differing rewards, the definition of a solution concept becomes more complicated.

In general, a solution to a game is a joint policy that consists of one policy for each agent and satisfies certain properties.

### Note that our definitions of solution concepts and their stated existence properties assume finite game model.

### In particular, we assume finite state, action, and observation spaces, and a finite number of agents.

Joint Policy

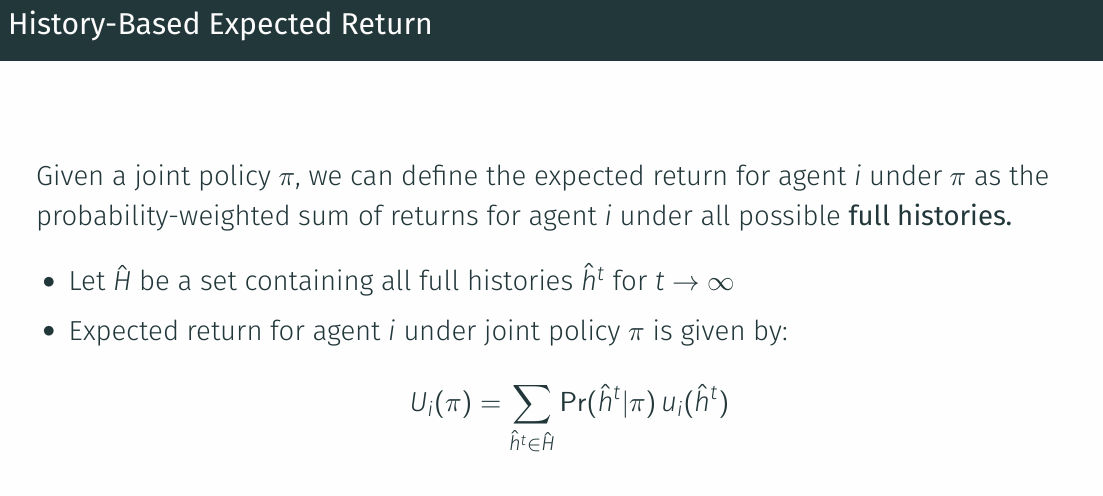
**we will define the expected return in the context of the POSG model , which is the most general game model used in this book and includes both stochastic games and normal-form games.**

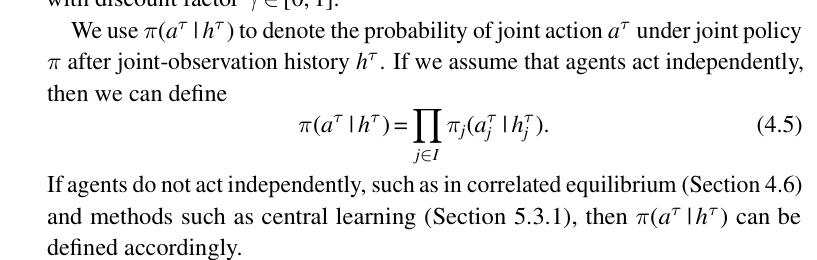
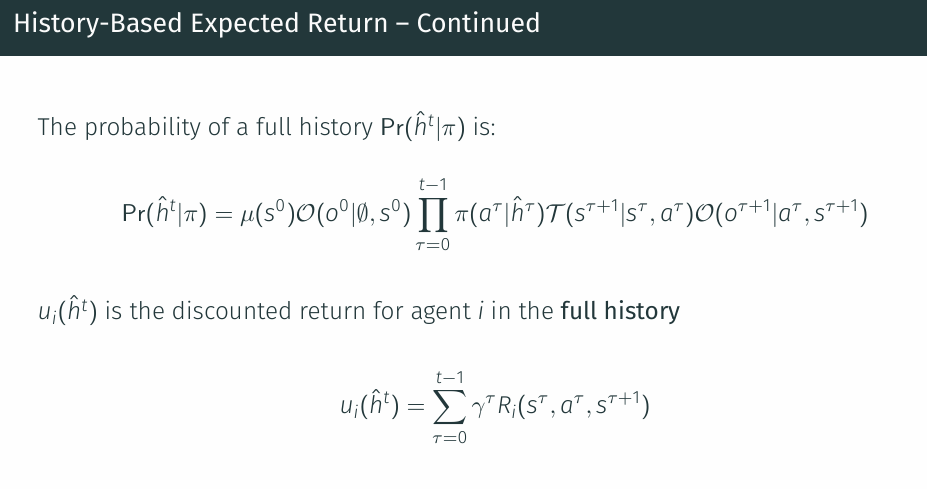
Expected Return

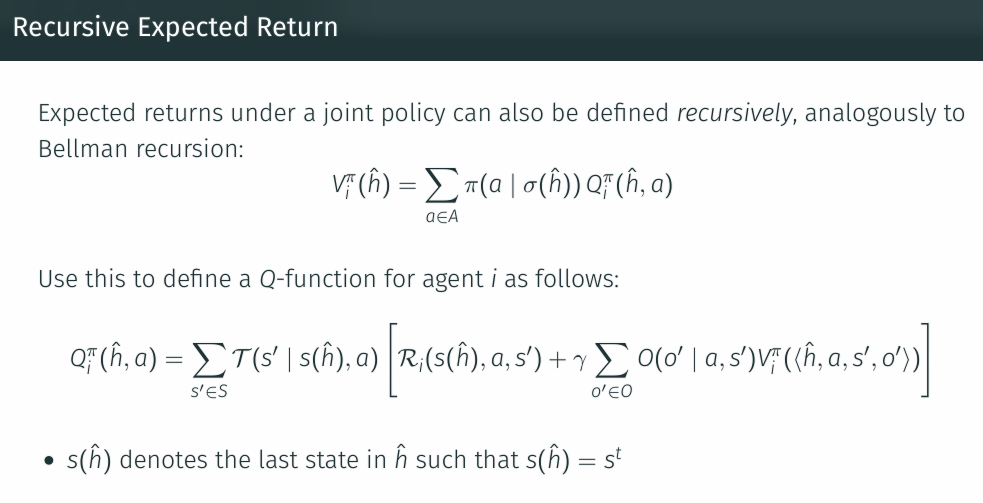
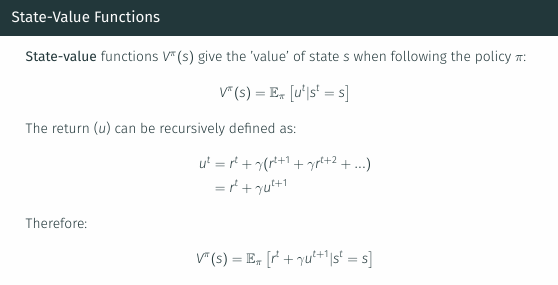
**"To make our definitions valid for both finite and infinite time steps, we use discounted returns and assume the standard convention of absorbing states "That is, once an absorbing state has been reached, the game will subsequently always transition into that same state with probability 1 and give a reward of 0 to all agents."**

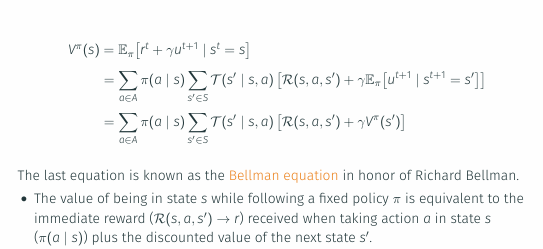
**"To simplify our definitions below, we also assume that the observation functions and policies of all agents become deterministic (i.e., assign probability 1 to a certain observation and action, respectively) once an absorbing state has been reached."**

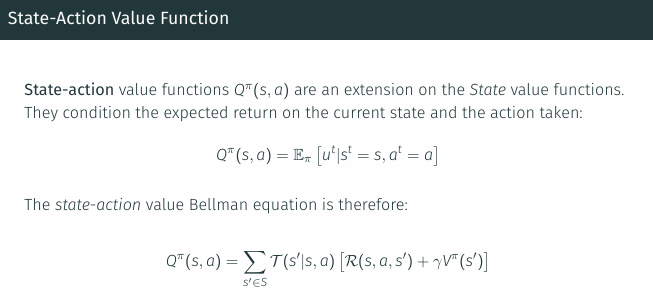
In the following, we provide two equivalent definitions of expected returns. The first definition is based on enumerating all full histories in the game, while the second definition is based on a Bellman-style recursion of value computations.

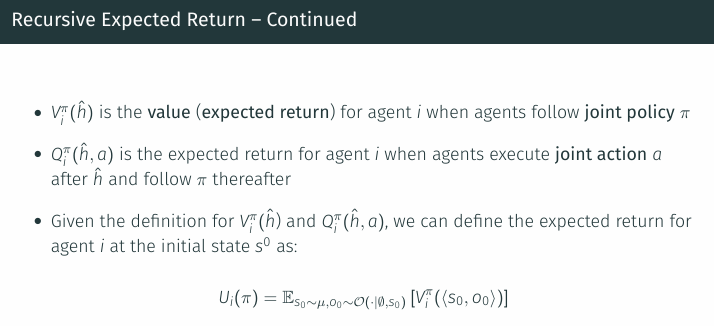
These definitions are equivalent, but can provide different perspectives and have been used in different ways. **"In particular, the first definition resembles a linear sum and may be easier to interpret, while the second definition uses a recursion that can be operationalized such as in value iteration for games.**



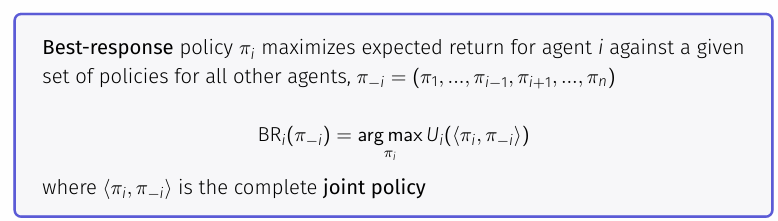








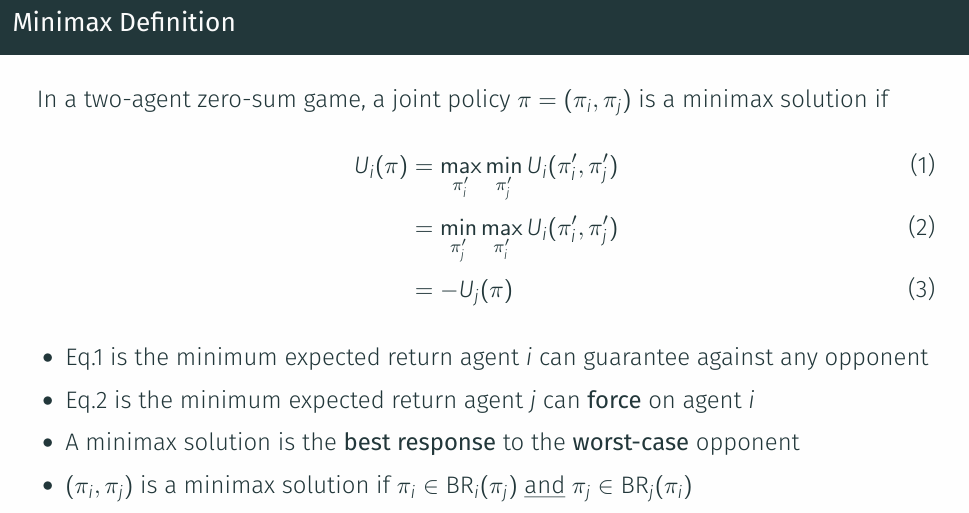
Best response

Given a set of policies for all agents other than agent i, denoted by π–i = (π₁, ..., πᵢ₋₁, πᵢ₊₁, ..., πₙ), a best response for agent i to π–i is a policy πᵢ that maximizes the expected return for i when played against π–i.

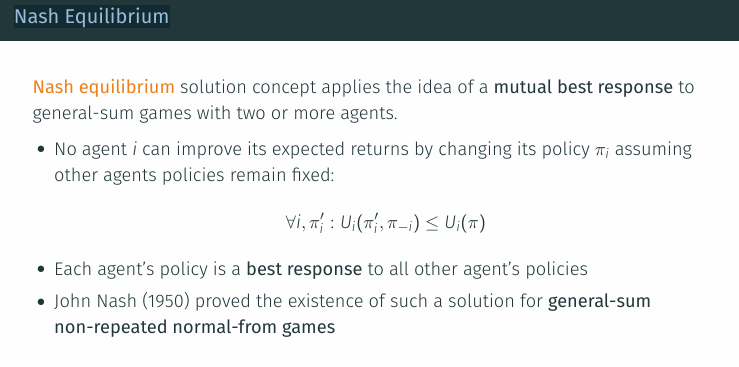
Note that the best response to a given π–i may not be unique, meaning that BRᵢ(π–i) may contain more than one best-response policy.

MINMAX

Minimax is a solution concept defined for two-agent zero-sum games, in which one agent’s reward is the negative of the other agent’s reward.

The existence of minimax solutions for normal-form games was first proven in the foundational game theory work of von Neumann (1928)

Moreover, while more than one minimax solution may exist in a game, all minimax solutions yield the same unique value Ui(π) for agent i (and, thus, agent j).

Nash Equilibrium

In the game theory literature, deterministic and probabilistic equilibria are also called “pure equilibria” and “mixed equilibria,”

Second, a game may have multiple Nash equilibria, and each equilibrium may entail different expected returns for the agents.

This leads to the important question of which equilibrium the agents should converge to during learning and how this may be achieved.

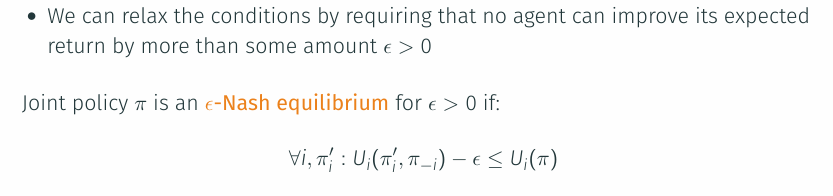
**ϵ-Nash Equilibrium**

The strict requirement of Nash equilibrium — that no agent can gain anything by unilaterally deviating from the equilibrium — can lead to practical issues when used in a computational system.

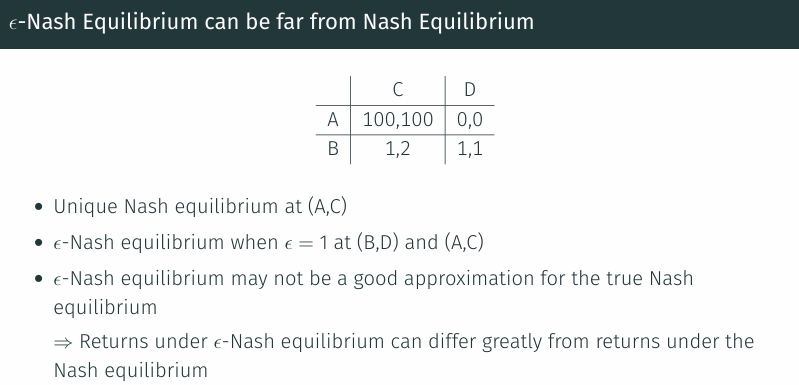
It is known that for games with more than two agents, the action probabilities specified by the policies in the equilibrium may be irrational numbers (i.e., cannot be represented as a fraction of two integers).

However, computer systems cannot fully represent irrational numbers using finite-precision floating-point approximations.

Exact Nash equilibria are often computationally too expensive.

The ϵ-Nash equilibrium relaxes the strict Nash equilibrium by requiring that no agent can improve its expected returns by more than some amount ϵ > 0 when deviating from its policy in the equilibrium.

However, although it may be tempting to view ϵ-Nash equilibrium as an approximation of Nash equilibrium, it is important to note that an ϵ-Nash equilibrium may not be close to any real Nash equilibrium, in terms of the expected returns produced by the equilibrium.



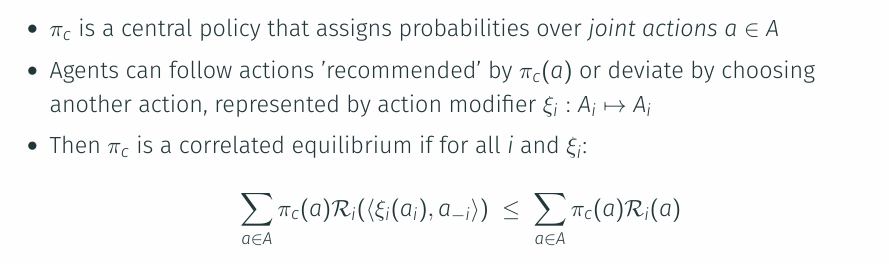
**Correlated Equilibrium**

"Nash equilibrium assumes policies are independent → this can limit expected returns"

Correlated equilibrium (Aumann 1974) generalizes Nash equilibrium by allowing for correlation between policies.

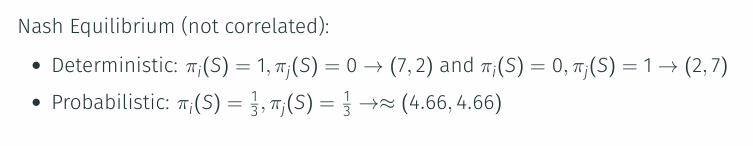
In the general definition of correlated equilibrium, each agent i’s policy is additionally conditioned on the outcomes of a private random variable di for the agent, which are governed by a joint probability distribution over (d1,...,dn) that is commonly known by all agents.

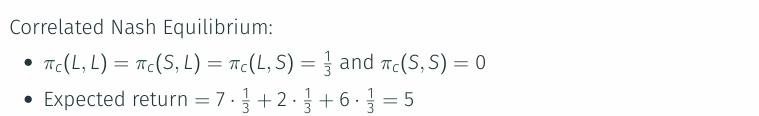
(Correlated equilibrium): In a general-sum normal-form game with n agents, let πc(a) be a joint policy that assigns probabilities to joint actions a ∈ A.



This Equation states that in a correlated equilibrium, in which every agent knows the probability distribution πc(a) and its own recommended action ai (but not the recommended actions for other agents), no agent can unilaterally deviate from its recommended actions in order to increase its expected return.



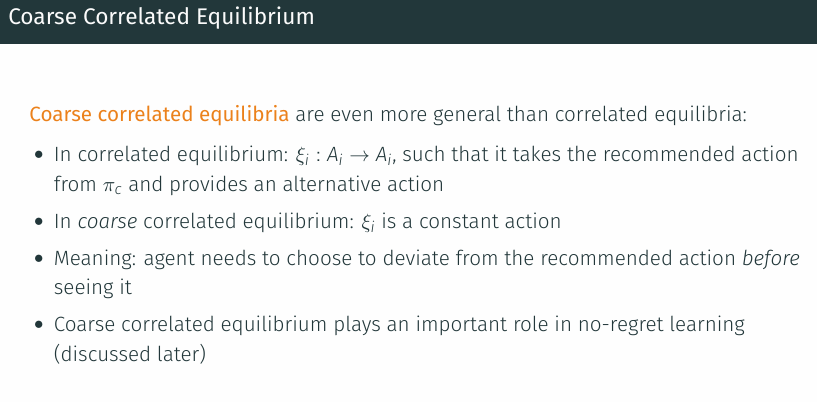




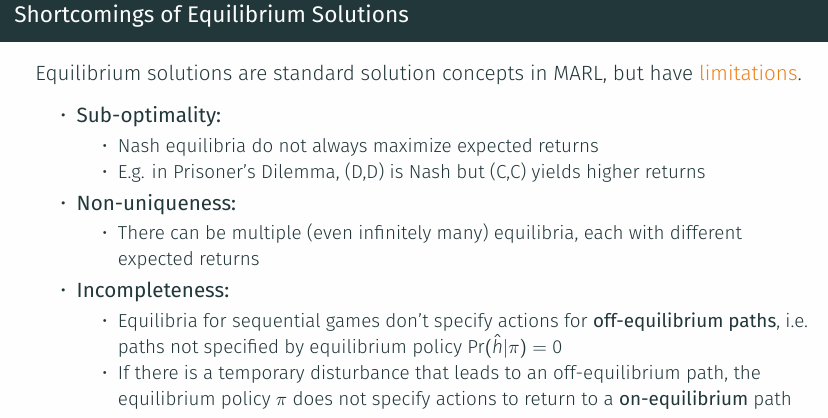
Coarse Correlated Equilibrium

 توی **Correlated Equilibrium**، بازیکن بعد از دیدن پیشنهاد (توصیه‌ی عملش) تصمیم می‌گیره که گوش بده یا نه.

 ولی توی **Coarse Correlated Equilibrium**، بازیکن **قبل از دیدن** پیشنهاد باید تصمیم بگیره که **کل سیاست پیشنهادی رو بپذیره یا نه.**

This solution concept means that each agent has to decide upfront, before seeing its recommended action, whether to follow the joint policy πc assuming that the other agents follow it.

| **ویژگی** | **Nash Equilibrium** | **Correlated Equilibrium** | **Coarse Correlated Equilibrium** |
| --- | --- | --- | --- |
| **هماهنگی بین بازیکنان** | ❌ هیچ هماهنگی‌ای نیست | ✅ هماهنگی از طریق یک داور مرکزی | ✅ هماهنگی از طریق داور مرکزی، ولی با محدودیت بیشتر |
| **اطلاع از توصیه‌ها قبل از تصمیم‌گیری** | ❌ اصلاً توصیه‌ای وجود نداره | ✅ بازیکن توصیه را می‌بینه و بعد می‌تونه تصمیم بگیره | ❌ بازیکن باید قبل از دیدن توصیه تصمیم بگیره که منحرف بشه یا نه |
| **فرم تصمیم‌گیری بازیکن (ξi)** | ندارد | تابع از پیشنهاد به انتخاب دیگر (ξi: Ai → Ai) | فقط یک انتخاب ثابت (مثل "من همیشه L می‌زنم") |
| **قدرت هماهنگی/بازده ممکن** | کم | متوسط | زیاد (از لحاظ تعمیم و کاربرد در یادگیری) |
| **کاربرد در یادگیری** | کمتر استفاده میشه | در مدل‌های تعاملی کاربرد داره | بسیار مهم در no-regret learning |
| **مثال شهودی** | هر کسی مسیر خودش رو رانندگی می‌کنه | چراغ راهنمایی بهمون می‌گه کی حرکت کنیم | باید از قبل بگی "اگه چراغ قرمز شد، من همیشه وایمیستم" بدون اینکه بدونی چراغ چی نشون می‌ده |



Non-uniqueness :

One approach is to use criteria like Pareto optimality or fairness..

Incompleteness:

To address such incompleteness, game-theorists have developed refinement concepts

**Subgame Perfect Equilibrium** و **Trembling-hand Perfect Equilibrium**.

### **Pareto Optimality**

### 

"پارتو بهینه" یعنی به سقف کارایی رسیدیم که ارتقا برای یک عامل، حتماً به هزینه عامل دیگر تموم میشه

هیچ "پیشرفت"ی ممکن نیست مگر اینکه کسی "پسرفت" کنه.

## مشکل تعادل نش

**زیر بهینه بودن (Sub-optimality)   
تعادل نش تضمین نمی‌کنه که بهترین نتیجه ممکن برای همه عوامل به دست بیاد.  
مثال کلاسیک: بازی معمای زندانی (Prisoner’s Dilemma)  
تعادل نش = (خیانت، خیانت) با سود پایین  
در حالی که (سکوت، سکوت) سود بالاتری برای هر دو نفر داره، ولی تعادل نش نیست چون افراد وسوسه می‌شن منحرف شن.**

**چندگانگی تعادل (Non-uniqueness)   
بازی ممکنه چندین تعادل نش داشته باشه با نتایج متفاوت، و هیچ راهی نداریم بفهمیم کدومش «بهتر»ه.**

**رفتار تعریف‌نشده در مسیرهای خارج از تعادل (Incompleteness)   
تعادل نش فقط مسیرهای احتمالی خودش رو پوشش می‌ده؛ اگر بازی از این مسیر منحرف بشه (مثلاً به‌خاطر خطا یا نویز)، نمی‌دونیم باید چی کار کنیم.**

## **نقش پرتو در حل مشکلات**

### **مقابله با زیر بهینه بودن**

**تضمین می‌کنه که هیچ سیاستی نیست که بتونه سود یکی از بازیکن‌ها رو بیشتر کنه بدون اینکه به بقیه ضرر بزنه**

**بنابراین اگر چند تعادل نش داریم، می‌تونیم بین اون‌ها پارتو بهینه‌ها رو انتخاب کنیم تا مطمئن شیم در «بالاترین سطح کارایی جمعی» هستیم**

**یعنی: تعادل نش فقط ثبات رو تضمین می‌کنه، ولی پارتو بهینگی کمک می‌کنه تعادل‌هایی انتخاب کنیم که واقعاً ارزشمند و سودآور هم باشن**

### **مقابله با چندگانگی**

**می‌تونه به ما کمک کنه تصمیم بگیریم کدومشون برای جامعه یا سیستم بهتره  
"از بین تعادل‌های نش، اون‌هایی که پارتو بهینه هستن رو انتخاب کن**

### **مکمل در مسیرهای خارج از تعادل**

**در بازی‌های ترتیبی، وقتی بازی از مسیر تعادل منحرف می‌شه، پارتو منطق تصمیم‌گیری بهتر و هماهنگ‌تری ارائه می‌ده چون مبتنی بر بهبود جمعی است، نه صرفاً منافع فردی**

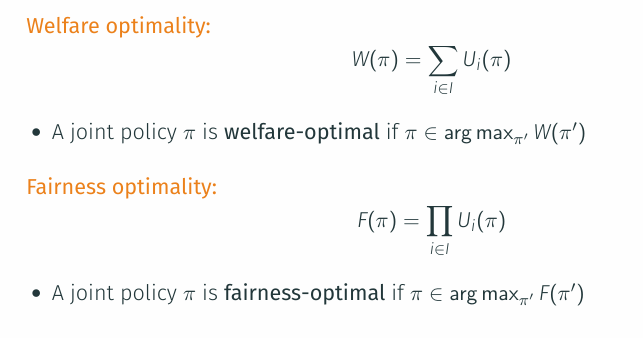
Social Welfare and Fairness

Pareto optimality states that there is no other solution in which at least one agent is better off without making other agents worse off.

However, it does not make any statements about the total amount of rewards and their distribution among the agents.

Thus, we may consider concepts of social welfare and fairness to further constrain the space of desirable solutions.

**The term welfare usually refers to some notion of totality of the agents’ returns, while the term fairness relates to the distribution of returns among agents.**



سه سیاست مشترک متفاوت در یک بازی دو عامله که بازده‌های مورد انتظار (1,5)، (2,4) و (3,3) را به‌دست می‌دهند، رفاه برابر ۶ دارند ولی انصاف آن‌ها به‌ترتیب ۵، ۸ و ۹ است.

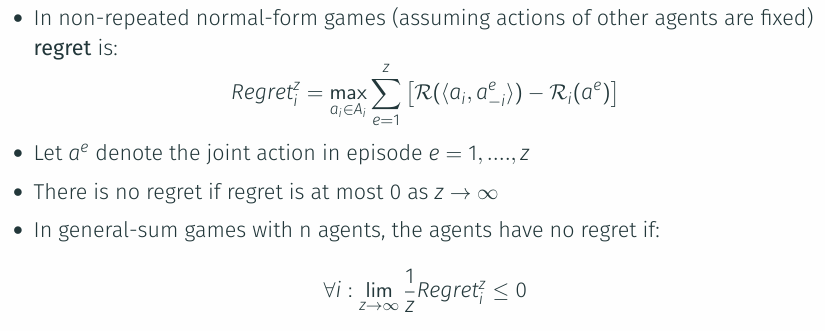
No-Regret

**Regret measures the difference between the rewards an agent received versus the rewards it would have received by choosing a different action in past episodes.**

An agent is said to have no-regret if, in the limit of infinitely many episodes, the agent’s average regret across the episodes is at most zero.

There are multiple ways in which regret can be defined.

We will first give a standard definition of regret for non-repeated normal-form games, which is based on comparing the rewards of different actions in the episodes.



As a solution concept, no-regret requires that all agents in the game have no-regret.

## **Complexity of Computing Equilibria**

عدم وجود الگوریتم ساده و سریع

وجود راه‌حل تضمین شده ولی سخت‌یافت

چندگانگی تعادل‌ها

Complexity Considerations for MARL

افزایش ابعاد مسأله با تعداد عامل‌ها

غیرایستا بودن محیط

هماهنگی بین عامل‌ها

وجود چندین هدف

بخش سوم

The preceding chapters introduced game models as a formalism of multi-agent interaction, and solution concepts to define what it means for the agents to act optimally in a game.

In this chapter, we will begin to explore methods to compute solutions for games.

The principal method by which we seek to compute solutions is via reinforcement learning (RL), in which the agents repeatedly try actions, make observations, and receive rewards.

Analogous to the standard RL terminology , we use the term episode to refer to each independent run of a game starting in some initial state.

The agents learn their policies based on data (i.e., observations, actions, and rewards) obtained from multiple episodes in a game.

We will then introduce two basic approaches of applying RL in games, called central learning and independent learning, both of which reduce the multi-agent problem to a single-agent problem.

Central learning applies single-agent RL directly to the space of joint actions to learn a central policy that chooses actions for each agent, while independent learning applies single-agent RL to each agent independently to learn agent policies, essentially ignoring the presence of other agents.

General Learning Process

In machine learning, learning is a process that optimizes a model or function based on data.

In our setting, the model is a joint policy usually consisting of policies for each agent, and the data (or “experiences”) consist of one or more histories in the game.

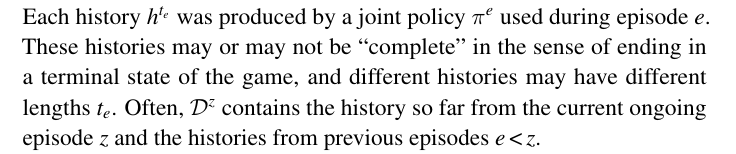
The learning goal is a solution of the game, defined by a chosen solution concept.

Thus, this learning process involves several elements:

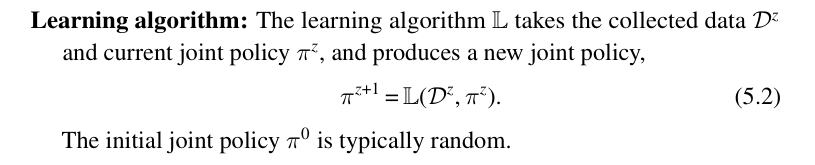
1.Game model: The game model defines the multi-agent environment and how agents may interact. Game models include non-repeated normal-form games, repeated normal-form games, stochastic games, and partially observable stochastic games (POSG).

2. The data used for learning consist of a set of z histories.





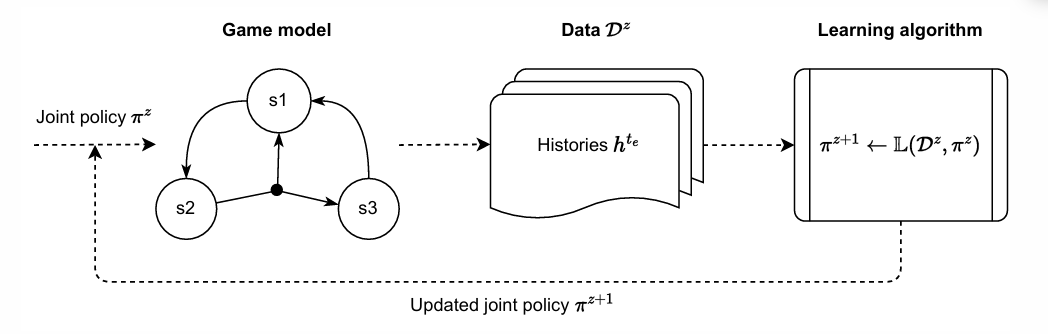
3.Learning algorithm:

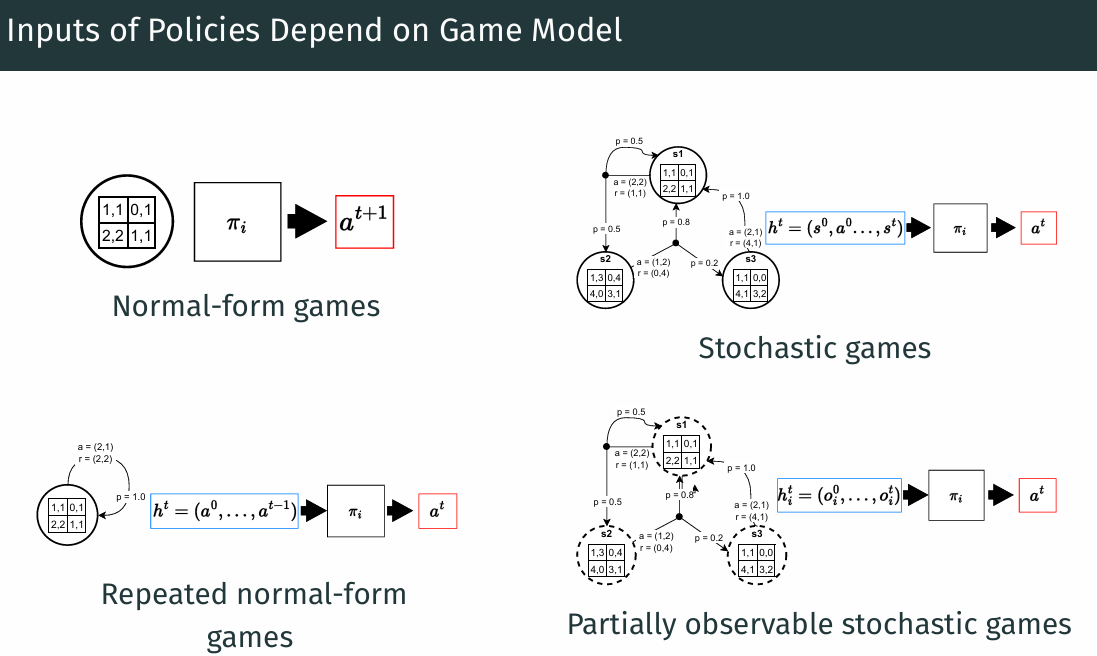


4.Learning goal: The goal of learning is a joint policy π∗ which satisfies the

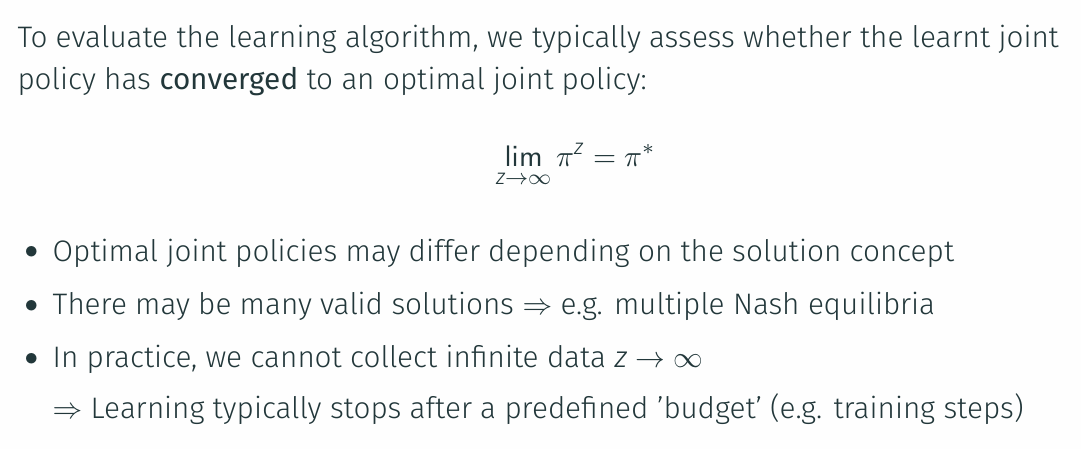
properties of a chosen solution concept. We introduced a range of

possible solution concepts, such as Nash equilibrium..



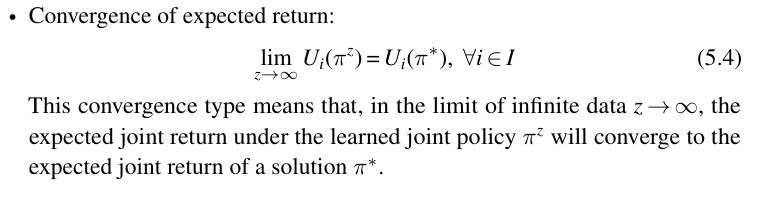
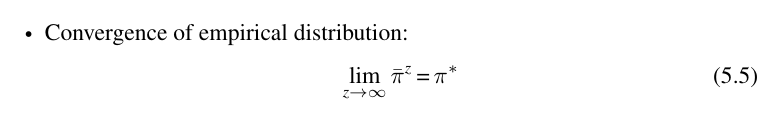
The chosen game model determines the conditioning of the learned joint policy.

Convergence



Weaker types of convergence

Central learning

The most basic approach to using RL to learn agent policies in multi-agent systems is to essentially reduce the multi-agent learning problem to a single-agent learning problem.

Central learning trains a single central policy πc, which receives the local observations of all agents and selects an action for each agent, by selecting joint actions from A = A1 × ... × An.

This essentially reduces the multi-agent problem to a single-agent problem, and we can apply existing single-agent RL algorithms to train πc.

An example of central learning based on Q-learning, called central Q-learning (CQL). This algorithm maintains joint action values Q(s,a) for joint actions a ∈ A.

central learning can be useful because it circumvents the multi-agent aspects of the non-stationarity and credit assignment problems.

However, in practice, this approach has a number of limitations.

1.The first limitation to note is that, in order to apply single-agent RL, central learning requires transforming the joint reward (r1, ..., rn) into a single scalar reward r.

For the case of common-reward games, in which all agents receive identical rewards, we can use r = ri for any i.

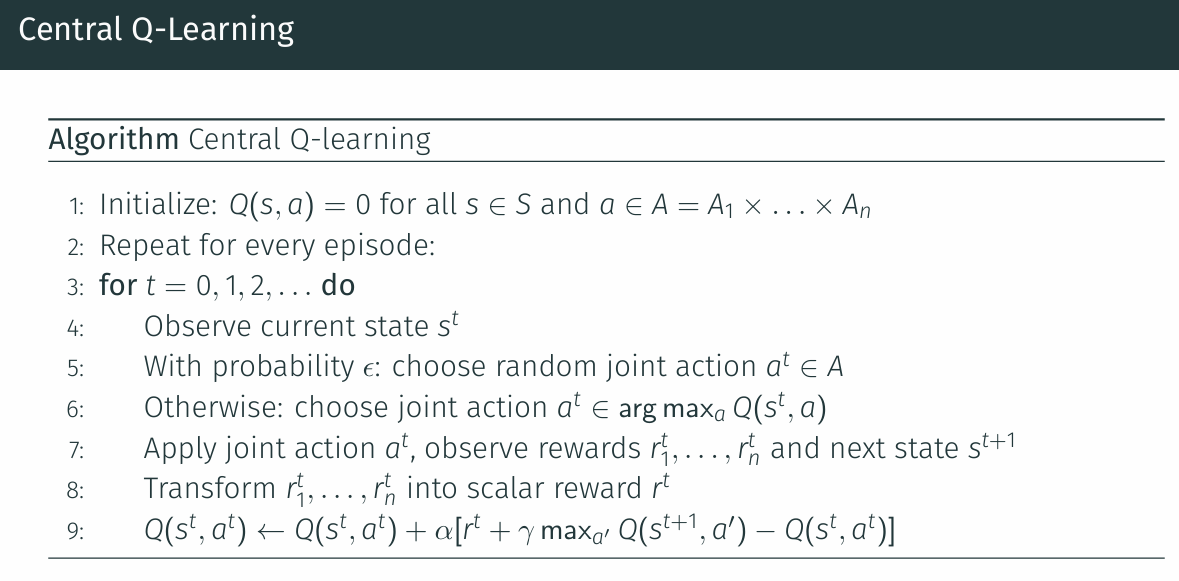
The optimality of the single-agent RL algorithm means that πc achieves maximum expected returns in each state s ∈ S.

Therefore, since the reward is defined as r = ri for all i, we know that πc is Pareto-optimal because there can be no other policy that achieves a higher expected return for any agent.

This can be difficult in zero-sum or general-sum games.

2.The second limitation is that by training a policy over the joint-action space, we now have to solve a decision problem with an action space that grows exponentially in the number of agents.

3. Finally, a fundamental limitation of central learning is due to the inherent structure of multi-agent systems. Agents are often localized entities that are physically or virtually distributed. In such settings, communication from a central policy πc to the agents and vice versa may not be possible or desirable, for various reasons.





Independent Learning

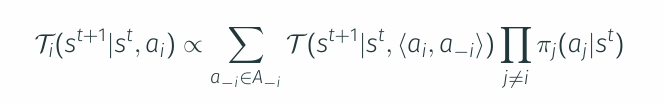
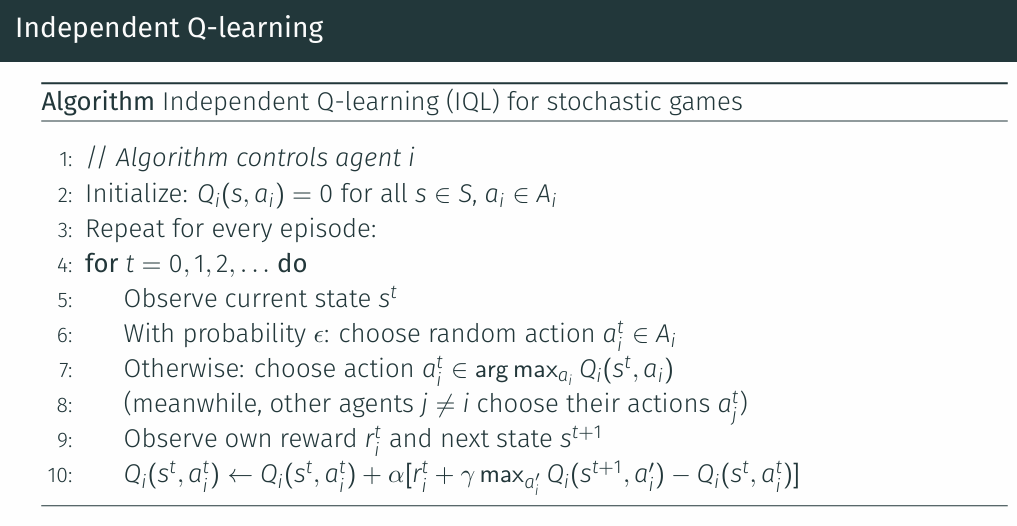
In independent learning (often abbreviated as IL), each agent i learns its own policy πᵢ using only its local history of own observations, actions, and rewards, while ignoring the existence of other agents .

Agents do not observe or use information about other agents, and the effects of other agents’ actions are simply part of the environment dynamics from the perspective of each learning agent. An example of independent learning based on Q-learning, called independent Q-learning (IQL).

**Independent learning naturally avoids the exponential growth in action spaces that plagues central learning.**

**It also does not require a scalar transformation of the joint reward, as is the case in central learning.**

**The downside of independent learning is that it can be significantly affected by non-stationarity caused by the concurrent learning of all agents.**

In an independent learning algorithm such as IQL, from the perspective of each agent i the policies πⱼ of other agents j ≠ i become part of the environment’s state transition function via.

