Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning: Foundational Algorithms

Stefano V. Albrecht, Filippos Christianos, Lukas Schäfer

MARL Book

Multi-Agent Reinforcement Learning: Foundations and Modern Approaches

Stefano V. Albrecht, Filippos Christianos, Lukas Schäfer

Published by MIT Press
(print version will be released on 17 December 2024)

This lecture is based on *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches* by Stefano V. Albrecht, Filippos Christianos and Lukas Schäfer

The book can be downloaded for free at www.marl-book.com.

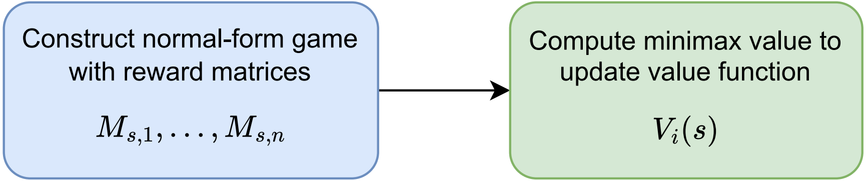
Lecture Outline

- Dynamic Programming for Games: Value Iteration
- Temporal-Difference Learning for Games: Joint-Action Learning
- Agent Modeling
- Policy-Based Learning
- No-Regret Learning

Dynamic Programming for Games: Value Iteration

Shapley (1953) proposed value iteration to compute minimax joint policy in zero-sum stochastic games with two agents

- Algorithm makes two sweeps over states $s \in S$ and agents $i \in I$:



- Converges to minimax values $V_i^*(s)$ of the stochastic game

Value Iteration Pseudocode

Algorithm Value iteration for stochastic games

1: Initialize: $V_i(s) = 0$ for all $s \in S$ and $i \in I$
2: Repeat until all V_i have converged:

Value Iteration Pseudocode

Algorithm Value iteration for stochastic games

1: Initialize: $V_i(s) = 0$ for all $s \in S$ and $i \in I$
2: Repeat until all V_i have converged:
3: **for all** states $s \in S$, agents $i \in I$, joint actions $a \in A$ **do**

$$M_{s,i}(a) \leftarrow \sum_{s' \in S} \mathcal{T}(s' \mid s, a) \left[\mathcal{R}_i(s, a, s') + \gamma V_i(s')\right]$$

Value Iteration Pseudocode

Algorithm Value iteration for stochastic games

Initialize: $V_i(s) = 0$ for all $s \in S$ and $i \in I$
Repeat until all V_i have converged:
for all states $s \in S$, agents $i \in I$, joint actions $a \in A$ **do**

$$M_{s,i}(a) \leftarrow \sum_{s' \in S} \mathcal{T}(s' \mid s, a) \left[\mathcal{R}_i(s, a, s') + \gamma V_i(s')\right]$$

for all states $s \in S$, agents $i \in I$ **do**

$$V_i(s) \leftarrow Value_i(M_{s,1}, ..., M_{s,n}) \quad \textit{// Minimax value for agent i}$$

Obtaining Minimax Policies for the Stochastic Game

To obtain minimax policies $\pi_1, ..., \pi_n$ for the stochastic game:

- Given converged state minimax values V_i^* and a state s
- Construct the normal-form game $M_{s,1}^*, ..., M_{s,n}^*$
- Compute the minimax policies $\pi_1^*, ..., \pi_n^*$ of this normal-form game
- Set action probabilities $\pi_i(a_i|s) \leftarrow \pi_i^*(a_i)$ for all $a_i \in A_i$

From Dynamic Programming to Temporal-Difference Learning

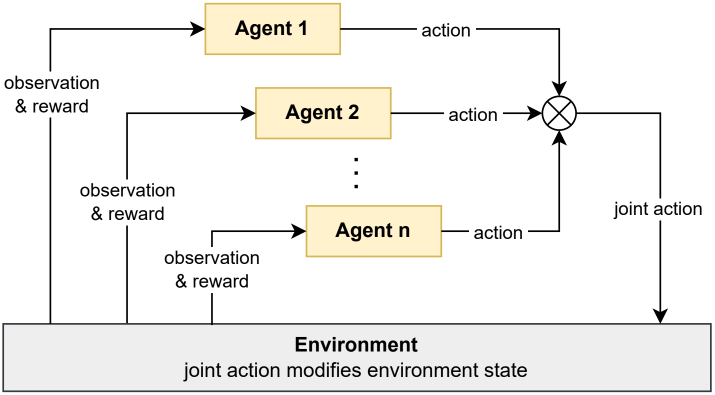
Problem

Dynamic programming requires full knowledge of game

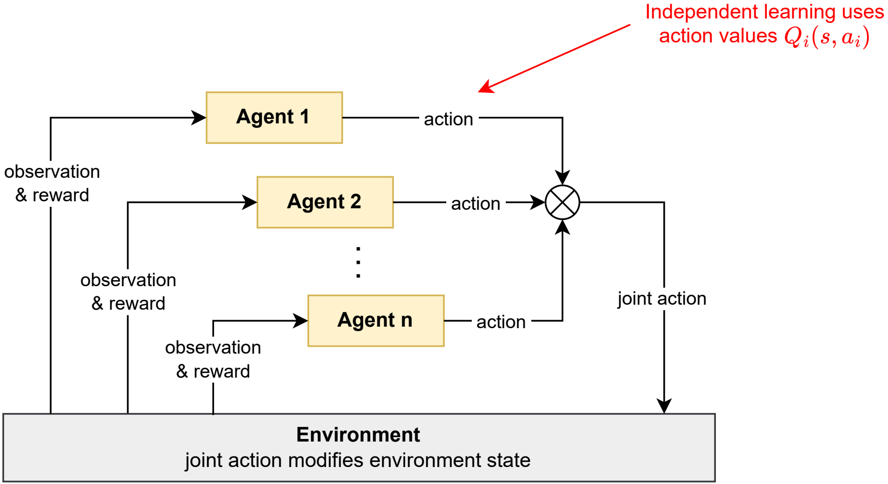
- Including reward functions $\mathcal{R}_i$ and state transition function $\mathcal{T}$
- May not be available!

Can we *learn* equilibrium joint policy via temporal-difference learning?

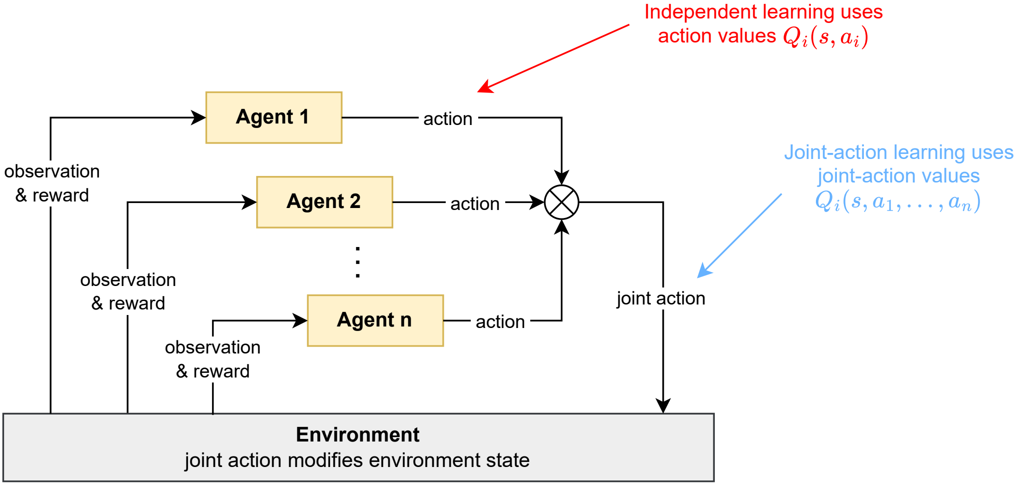
Temporal-Difference Learning for Games: Joint-Action Learning



Temporal-Difference Learning for Games: Joint-Action Learning



Temporal-Difference Learning for Games: Joint-Action Learning



Temporal-Difference Learning for Games: Joint-Action Learning

Problem

$Q_i(s, a_1, \ldots, a_n)$ is not enough to find optimal action for agent i

- Cannot evaluate $\max_{a_i} Q_i(s, a_1, \ldots, a_n)$

  $\rightarrow$ Optimal action depends on actions of other agents!

Temporal-Difference Learning for Games: Joint-Action Learning

Problem

$Q_i(s, a_1, \ldots, a_n)$ is not enough to find optimal action for agent i

- Cannot evaluate $\max_{a_i} Q_i(s, a_1, \ldots, a_n)$

  $\rightarrow$ Optimal action depends on actions of other agents!

We have to define:

1. How to select action from Q_i?
2. How to update Q_i?

Joint-Action Learning with Game Theory

Idea: joint-action value functions define a normal-form game:

- Each agent stores Q-function Q_j for every agent $j \in I$ (assumes agents can observe all agents' actions and rewards)

Joint-Action Learning with Game Theory

Idea: joint-action value functions define a normal-form game:

- Each agent stores Q-function Q_j for every agent $j \in I$
(assumes agents can observe all agents' actions and rewards)

- Define reward functions for normal-form game in state s as

$$\mathcal{R}_j(a_1, ..., a_n) = Q_j(s, a_1, ..., a_n)$$

Joint-Action Learning with Game Theory

Idea: joint-action value functions define a normal-form game:

- Each agent stores Q-function Q_j for every agent $j \in I$ (assumes agents can observe all agents' actions and rewards)
- Define reward functions for normal-form game in state s as

$$\mathcal{R}_j(a_1, ..., a_n) = Q_j(s, a_1, ..., a_n)$$

- We can *solve* the normal-form game defined by

$$\Gamma_s = \Big(\mathcal{R}_1 = Q_1(s, \cdot), \cdots, \mathcal{R}_n = Q_n(s, \cdot)\Big)$$

Joint-Action Learning with Game Theory

Solution of Γ_s is a joint policy $\pi_s^* = (\pi_{s,1}^*, ..., \pi_{s,n}^*)$ with certain properties

- e.g. compute minimax solution or Nash equilibrium of Γ_s

  $\rightarrow$ Use $\pi_{s,i}^*$ to select action for agent i

Solution of Γ_s is a joint policy $\pi_s^* = (\pi_{s,1}^*, \ldots, \pi_{s,n}^*)$ with certain properties

- e.g. compute minimax solution or Nash equilibrium of Γ_s

  $\rightarrow$ Use $\pi_{s,i}^*$ to select action for agent i

Value of Γ_s for agent j is its expected reward under joint policy π_s^*

$$Value_j(\Gamma_s) = \sum_{a \in A} Q_j(s, a)\, \pi_s^*(a)$$

$\rightarrow$ Update Q_j towards update target: $r_j + \gamma\, Value_j(\Gamma_{s'})$

JAL-GT Pseudocode

Algorithm Joint-action learning with game theory (JAL-GT)

// Algorithm controls agent i

Initialize: $Q_j(s, a) = 0$ for all $j \in I$ and $s \in S, a \in A$

Repeat for every episode:

for $t = 0, 1, 2, \ldots$ **do**

    Observe current state s^t

    With probability ϵ: choose random action a_i^t

    Otherwise: solve Γ_{s^t} to get policies $(\pi_1, \ldots, \pi_n)$, then sample action $a_i^t \sim \pi_i$

    Observe joint action $a^t = (a_1^t, \ldots, a_n^t)$, rewards $r_1^t, \ldots, r_n^t$, next state s^{t+1}

    for all $j \in I$ **do**

        $Q_j(s^t, a^t) \leftarrow Q_j(s^t, a^t) + \alpha \left[r_j^t + \gamma Value_j(\Gamma_{s^{t+1}}) - Q_j(s^t, a^t) \right]$

Minimax-Q, Nash-Q, CE-Q

Minimax-Q solves Γ_s via minimax solution (Littman, 1994)

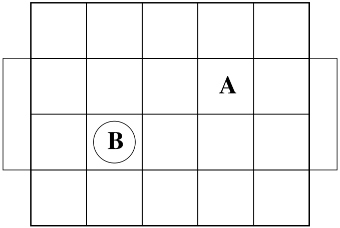
- Converges to unique minimax values in two-agent zero-sum stochastic games
- Minimax profile can be computed with linear programming (LP)

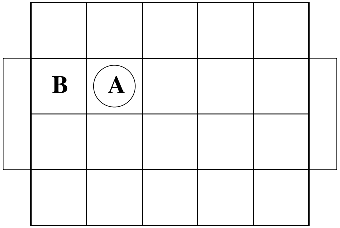
Nash-Q solves Γ_s via Nash equilibrium (Hu and Wellman, 2003)
CE-Q solves Γ_s via correlated equilibrium (Greenwald and Hall, 2003)

- Converges to equilibrium under highly restrictive conditions

  $\rightarrow$ Problem: often no unique equilibrium value in general-sum games
- Compute CE with LP, compute NE with quadratic programming

Example: Minimax-Q in Grid Soccer (Littman, 1994)





- Episodes start in left state with random ball assignment
- Agent wins episode if it moves the ball into opponent goal
- Agent loses ball to opponent if it moves into opponent's location

Against *unknown* opponent, optimal policy must randomise (right state; why?)

Example: Minimax-Q in Grid Soccer (Littman, 1994)

	minimax Q		independent Q	
	% won	ep. len.	% won	ep. len.
vs. random				
vs. hand-built				
vs. optimal				

- random: uniform-random opponent policy
- hand-built: manual opponent policy
- optimal: Q-learning opponent policy trained against final policy of minimax Q / independent Q

Example: Minimax-Q in Grid Soccer (Littman, 1994)

	minimax Q		independent Q	
	% won	ep. len.	% won	ep. len.
vs. random	99.3	13.89		
vs. hand-built	53.7	18.87		
vs. optimal	37.5	22.73		

- minimax Q learns "safe" policy that works against any opponent
  $\rightarrow$ minimax policy guarantees minimum average 50% win
- lower % win against optimal because minimax Q did not fully converge during training, so could be exploited by optimal opponent

Example: Minimax-Q in Grid Soccer (Littman, 1994)

	minimax Q		independent Q	
	% won	ep. len.	% won	ep. len.
vs. random	99.3	13.89	99.5	11.63
vs. hand-built	53.7	18.87	76.3	30.30
vs. optimal	37.5	22.73	0	83.33

Problem

- Independent Q-learning can learn strong performance
- **But:** overfits to opponent, does not generalise well to other opponents
  $\rightarrow$ "optimal" opponent exploits deterministic policy learned by independent Q-learning, resulting in 0% wins

Agent Modeling & Best Response

Game theory solutions are normative: they *prescribe* how agents *should* behave

- e.g. minimax assumes worst-case opponent

Agent Modeling & Best Response

Game theory solutions are normative: they *prescribe* how agents *should* behave

- e.g. minimax assumes worst-case opponent

What if agents don't behave as prescribed by solution?

- e.g. minimax-Q was unable to exploit hand-built opponent in soccer example

Agent Modeling & Best Response

Game theory solutions are normative: they *prescribe* how agents *should* behave

- e.g. minimax assumes worst-case opponent

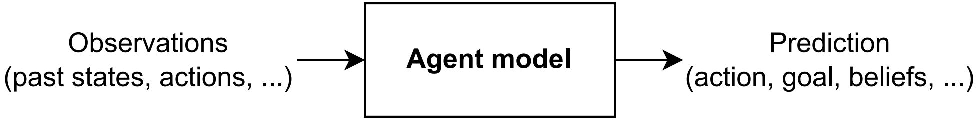
What if agents don't behave as prescribed by solution?

- e.g. minimax-Q was unable to exploit hand-built opponent in soccer example

Other approach: **agent modeling with best response**

- Learn models of other agents to predict their actions
- Compute optimal action (best response) against agent models

Agent Modeling



Many kinds of agent modeling:

- Policy reconstruction
- Type-based reasoning
- Classification
- Plan recognition
- Recursive reasoning
- Graphical methods
- Group modeling
- Implicit modeling

S. Albrecht, P. Stone. Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems. *Artificial Intelligence*, 2018

Policy Reconstruction & Best Response

Policy reconstruction: learn model $\hat{\pi}_j \approx \pi_j$ from past observed actions

In general, can train model with supervised learning on data $\{(s^\tau, a_j^\tau)\}_{\tau=1}^{t-1}$

- E.g. look-up table, neural network, finite state machine, ...
- Model should support incremental updating

Policy Reconstruction & Best Response

Policy reconstruction: learn model $\hat{\pi}_j \approx \pi_j$ from past observed actions

In general, can train model with supervised learning on data $\{(s^\tau, a_j^\tau)\}_{\tau=1}^{t-1}$

- E.g. look-up table, neural network, finite state machine, ...
- Model should support incremental updating

Given models for other agents $\hat{\pi}_{-i} = \{\hat{\pi}_j\}_{j \neq i}$, compute best response

$$\pi_i \in \mathrm{BR}_i(\hat{\pi}_{-i})$$

Fictitious Play

Fictitious play (Brown 1951) algorithm for non-repeated normal-form games

Each agent i models other agents j as stationary distribution:

$$\hat{\pi}_j(a_j) = \frac{C(a_j)}{\sum_{a'_j \in A_j} C(a'_j)}$$

$C(a_j)$ is number of times agent j chose action a_j in prior episodes

Fictitious Play

Fictitious play (Brown 1951) algorithm for non-repeated normal-form games

Each agent i models other agents j as stationary distribution:

$$\hat{\pi}_j(a_j) = \frac{C(a_j)}{\sum_{a'_j \in A_j} C(a'_j)}$$

$C(a_j)$ is number of times agent j chose action a_j in prior episodes

In each episode, agents choose best-response action:

$$\mathrm{BR}_i(\hat{\pi}_{-i}) = \arg\max_{a_i \in A_i} \sum_{a_{-i} \in A_{-i}} \mathcal{R}_i(\langle a_i, a_{-i} \rangle) \prod_{j \neq i} \hat{\pi}_j(a_j)$$

Fictitious Play Convergence

- If agents' actions converge, then the converged actions form a NE

Fictitious Play Convergence

- If agents' actions converge, then the converged actions form a NE
- If in any episode the agents' actions form a NE, then they will always remain in the equilibrium

Fictitious Play Convergence

Fictitious Play Convergence

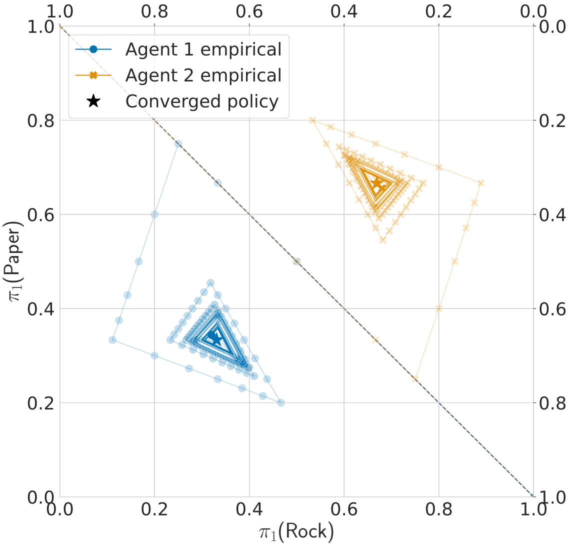
- If agents' actions converge, then the converged actions form a NE
- If in any episode the agents' actions form a NE, then they will always remain in the equilibrium
- If empirical distribution of agents' actions converges, then the distributions converge to a NE

Fictitious Play Convergence

Fictitious Play Convergence

- If agents' actions converge, then the converged actions form a NE
- If in any episode the agents' actions form a NE, then they will always remain in the equilibrium
- If empirical distribution of agents' actions converges, then the distributions converge to a NE
- The empirical distributions converge in several game classes, e.g. in two-agent zero-sum finite games

Fictitious Play in Rock-Paper-Scissors



Fictitious Play in Rock-Paper-Scissors

Episode e	Joint action (a_1^e, a_2^e)	Agent model $\hat{\pi}_2$	Agent 1 action values
1	R,R	(0.33, 0.33, 0.33)	(0.00, 0.00, 0.00)
2	P,P	(1.00, 0.00, 0.00)	(0.00, 1.00, –1.00)
3	P,P	(0.50, 0.50, 0.00)	(–0.50, 0.50, 0.00)
4	P,P	(0.33, 0.67, 0.00)	(–0.67, 0.33, 0.33)
5	S,S	(0.25, 0.75, 0.00)	(–0.75, 0.25, 0.50)
6	S,S	(0.20, 0.60, 0.20)	(–0.40, 0.00, 0.40)
7	S,S	(0.17, 0.50, 0.33)	(–0.17, –0.17, 0.33)
8	S,S	(0.14, 0.43, 0.43)	(0.00, –0.29, 0.29)
9	S,S	(0.13, 0.38, 0.50)	(0.12, –0.38, 0.25)
10	R,R	(0.11, 0.33, 0.56)	(0.22, –0.44, 0.22)

Joint-Action Learning with Agent Modeling

Extend fictitious play approach to stochastic games by using joint-action learning with agent models

Joint-Action Learning with Agent Modeling

Extend fictitious play approach to stochastic games by using joint-action learning with agent models

Agents models other agents j, this time conditioned on states s:

$$\hat{\pi}_j(a_j \mid s) = \frac{C(s, a_j)}{\sum_{a'_j \in A_j} C(s, a'_j)}$$

Joint-Action Learning with Agent Modeling

Extend fictitious play approach to stochastic games by using joint-action learning with agent models

Agents models other agents j, this time conditioned on states s:

$$\hat{\pi}_j(a_j \mid s) = \frac{C(s, a_j)}{\sum_{a'_j \in A_j} C(s, a'_j)}$$

Given models $\{\hat{\pi}_j\}_{j \neq i}$, action values are defined as:

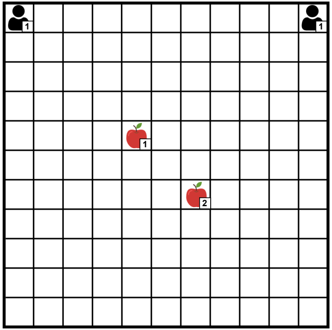
$$AV_i(s, a_i) = \sum_{a_{-i} \in A_{-i}} Q_i(s, \langle a_i, a_{-i} \rangle) \prod_{j \neq i} \hat{\pi}_j(a_j \mid s)$$

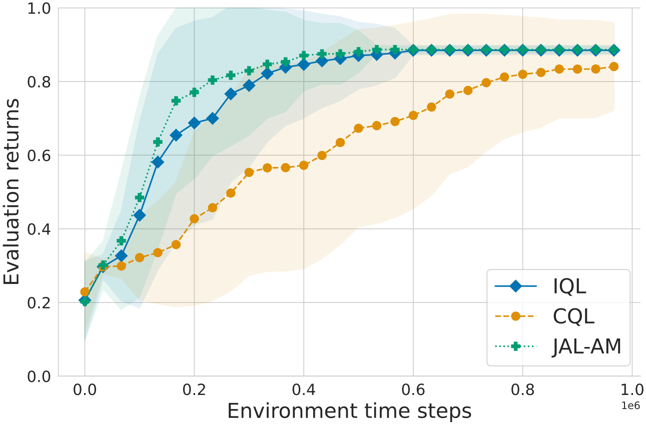
$\rightarrow$ Use AV_i to select optimal actions and as learning update targets

JAL-AM Pseudocode

Algorithm Joint-action learning with agent modeling (JAL-AM)

Initialize:
    $Q_i(s, a) = 0$ for all $s \in S, a \in A$
    Agent models $\hat{\pi}_j(a_j \mid s) = \frac{1}{|A_j|}$ for all $j \neq i, a_j \in A_j, s \in S$
Repeat for every episode:
for $t = 0, 1, 2, \ldots$ **do**
    Observe current state s^t
    With probability ϵ: choose random action a_i^t
    Otherwise: choose best-response action $a_i^t \in \arg\max_{a_i} AV_i(s^t, a_i)$
    Observe joint action $a^t = (a_1^t, \ldots, a_n^t)$, reward r_i^t, next state s^{t+1}
    Update agent models $\hat{\pi}_j$ with new observations (e.g., (s^t, a_j^t))
    $Q_i(s^t, a^t) \leftarrow Q_i(s^t, a^t) + \alpha \left[r_i^t + \gamma \max_{a_i'} AV_i(s^{t+1}, a_i') - Q_i(s^t, a^t) \right]$


1
1
1
2


1.0
0.8
0.6
0.4
0.2
0.0
Evaluation returns
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Environment time steps
IQL
CQL
JAL-AM

Policy-Based Learning

All algorithms so far derive policies from learned action-value functions

- Has important limitations, e.g. algorithms using best-response actions from values (e.g. fictitious play, JAL-AM) cannot represent probabilistic equilibria
- Fictitious play unable to represent uniform-random NE in Rock-Paper-Scissors

Policy-Based Learning

All algorithms so far derive policies from learned action-value functions

- Has important limitations, e.g. algorithms using best-response actions from values (e.g. fictitious play, JAL-AM) cannot represent probabilistic equilibria
- Fictitious play unable to represent uniform-random NE in Rock-Paper-Scissors

Policy-based learning instead uses learning data to directly optimise policies

- Use *parameterised policies* that are differentiable
- Use *gradient-ascent techniques* to optimise parameters

  $\rightarrow$ Can directly learn action probabilities in policies!

Gradient-ascent learning in non-repeated normal-form games with two agents i, j and two actions

Reward matrices:

$$\mathcal{R}_i = \begin{bmatrix} r_{1,1} & r_{1,2} \\ r_{2,1} & r_{2,2} \end{bmatrix} \qquad \mathcal{R}_j = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}$$

Policies with parameters $\alpha, \beta \in [0, 1]$:

$$\pi_i = (\alpha, 1 - \alpha) \qquad \pi_j = (\beta, 1 - \beta)$$

Gradient Ascent in Expected Reward

Update policy in direction of gradient in expected reward using step size $\kappa > 0$:

$$\alpha^{k+1} = \alpha^k + \kappa \frac{\partial U_i(\alpha^k, \beta^k)}{\partial \alpha^k}$$
$$\beta^{k+1} = \beta^k + \kappa \frac{\partial U_j(\alpha^k, \beta^k)}{\partial \beta^k}$$

Partial derivative of an agent's expected reward with respect to its policy:

$$\frac{\partial U_i(\alpha, \beta)}{\partial \alpha} = \beta u + (r_{1,2} - r_{2,2})$$
$$\frac{\partial U_j(\alpha, \beta)}{\partial \beta} = \alpha u' + (c_{2,1} - c_{2,2}).$$

where $u = r_{1,1} - r_{1,2} - r_{2,1} + r_{2,2}$ and $u' = c_{1,1} - c_{1,2} - c_{2,1} + c_{2,2}$

Learning Dynamics of Infinitesimal Gradient Ascent

What joint policy will agents converge to?

$\rightarrow$ Can analyse learning dynamics via dynamical systems theory!

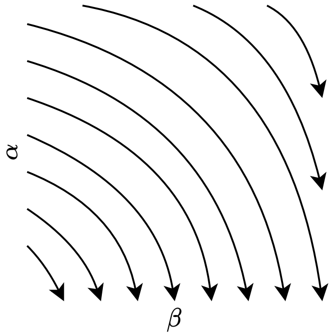
Infinitesimal Gradient ascent (IGA): use infinitesimal step size $\kappa \rightarrow \infty$

Joint policy given by $(\alpha(t), \beta(t))$ will follow continuous trajectory according to differential equation:

$$\begin{bmatrix} \frac{\partial \alpha}{\partial t} \\ \frac{\partial \beta}{\partial t} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & u \\ u' & 0 \end{bmatrix}}_{F} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} + \begin{bmatrix} (r_{1,2} - r_{2,2}) \\ (c_{2,1} - c_{2,2}) \end{bmatrix}$$

Learning Dynamics of Infinitesimal Gradient Ascent

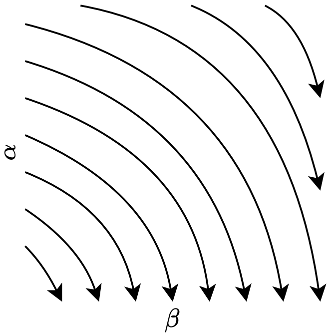
Learning dynamics of (α, β) will follow one of three types of trajectories:



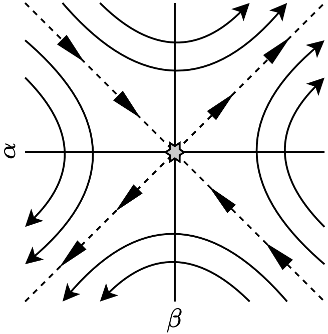
(a) F not invertible

Learning Dynamics of Infinitesimal Gradient Ascent

Learning dynamics of (α, β) will follow one of three types of trajectories:



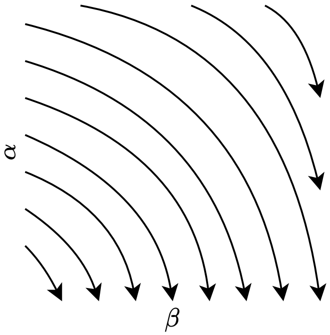
(a) *F* not invertible



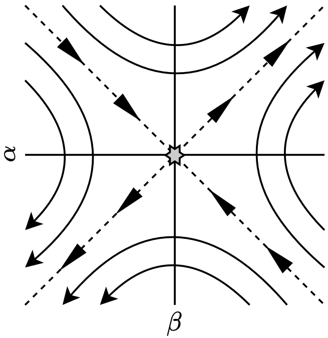
(b) *F* has purely real eigenvalues

Learning Dynamics of Infinitesimal Gradient Ascent

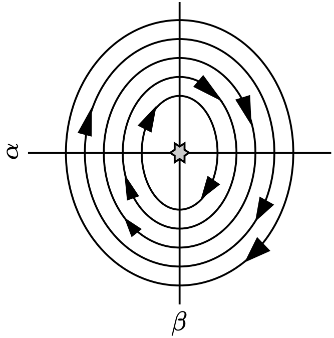
Learning dynamics of (α, β) will follow one of three types of trajectories:



(a) *F* not invertible



(b) *F* has purely real eigenvalues



(c) *F* has purely imaginary eigenvalues

IGA Convergence

IGA Convergence (Singh, Kearns, Mansour 2000)

- (α, β) does not converge in all cases

IGA Convergence

IGA Convergence (Singh, Kearns, Mansour 2000)

- (α, β) does not converge in all cases
- If (α, β) does not converge, then average rewards during learning converge to expected rewards of some NE

IGA Convergence (Singh, Kearns, Mansour 2000)

- (α, β) does not converge in all cases
- If (α, β) does not converge, then average rewards during learning converge to expected rewards of some NE
- If (α, β) converges, then converged joint policy is a NE

Win or Learn Fast – Variable Learning Rate

By using a variable step size κ, we can ensure that IGA policies *always* converge to NE

Win or Learn Fast (WoLF): (Bowling and Veloso 2002)

- learn fast (use larger κ) when "losing"
- learn slow (use smaller κ) when "winning"

Winning/losing depends on current expected reward compared to NE rewards

Win or Learn Fast – Variable Learning Rate

Modify learning rule (analogous for agent j):

$$\alpha^{k+1} = \alpha^k + l_i^k \kappa \frac{\partial U_i(\alpha^k, \beta^k)}{\partial \alpha^k}$$

with variable step size $l_i^k \in [l_{\mathsf{min}}, l_{\mathsf{max}}] > 0$

$$l_i^k = \begin{cases} l_{\mathsf{min}} & \text{if } U_i(\alpha^k, \beta^k) > U_i(\alpha^e, \beta^k) \quad \text{(winning)} \\ l_{\mathsf{max}} & \text{otherwise} \quad \text{(losing)} \end{cases}$$

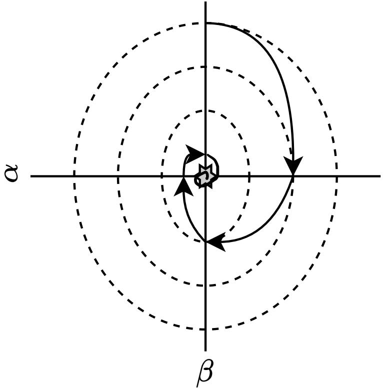
where α^e is a policy from some NE

WoLF-IGA Convergence

IGA does not converge if F is invertible and has imaginary eigenvalues

- In WoLF-IGA, trajectories of (α, β) are piecewise elliptical, each quadrant tightens ellipse by factor $\sqrt{\frac{l_{\text{min}}}{l_{\text{max}}}} < 1$
- Using variable learning rate, WoLF-IGA converges!

WoLF-IGA guaranteed to learn NE in two-agent two-action normal-form game



Win or Learn Fast with Policy Hill Climbing (WoLF-PHC)

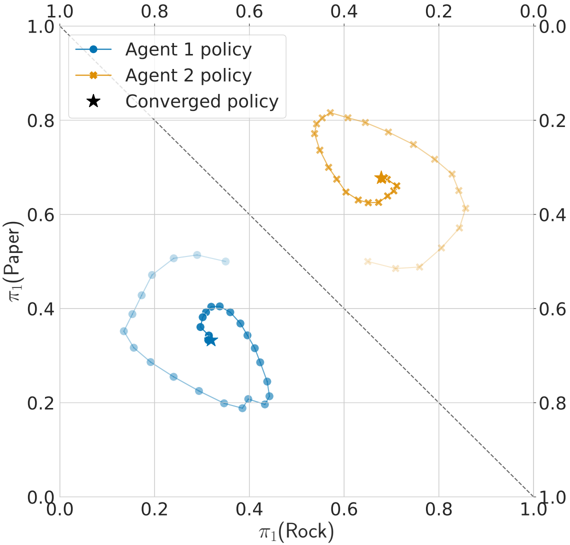
WoLF-PHC algorithm (Bowling and Veloso 2002) applies WoLF principle in stochastic games:

- Can learn in general-sum games with any number of agents and actions
- Does not require knowledge of reward functions and policies

To determine winning/losing, it compares average reward of current policy π_i to "average" policy $\bar{\pi}_i$ over past policies:

$$\delta = \begin{cases} l_w & \text{if } \sum_{a'_i} \pi_i(a'_i \mid s) Q(s, a'_i) > \sum_{a'_i} \bar{\pi}_i(a'_i \mid s) Q(s, a'_i) \\ l_l & \text{otherwise} \end{cases}$$

WoLF-PHC in Rock-Paper-Scissors



No-Regret Learning

JAL-GT and JAL-AM algorithms use solution concepts and agent modeling to learn joint policies

Now: no-regret learning algorithms that use regret definitions to learn policies

- We consider two simple regret matching algorithms (Hart and Mas-Colell, 2000)
- In normal-form games, their empirical action distributions converge to set of (coarse) correlated equilibrium

Unconditional Regret Matching

Unconditional regret matching: compute action probabilities proportional to (positive) average *unconditional* regrets of actions $a_i \in A_i$

$$Regret_i^z(a_i) = \sum_{e=1}^{z} \left[\mathcal{R}_i(\langle a_i, a_{-i}^e \rangle) - \mathcal{R}_i(a^e)\right]$$

a^e is joint action from past episodes $e = 1, ..., z$.

Each agent i starts with a random initial policy π_i^1, then update π_i^z to

$$\pi_i^{z+1}(a_i) = \frac{[\bar{R}_i^z(a_i)]_+}{\sum_{a_i' \in A_i} [\bar{R}_i^z(a_i')]_+} \quad \text{with } \bar{R}_i^z(a_i) = \frac{1}{z} Regret_i^z(a_i)$$

where $[x]_+ = \max[x, 0]$.

Conditional Regret Matching

Conditional regret matching: compute action probabilities proportional to (positive) average *conditional* regrets with respect to most recent selected action a_i'

$$Regret_i^z(a_i', a_i) = \sum_{e: a_i^e = a_i'} \left[\mathcal{R}_i(\langle a_i, a_{-i}^e\rangle) - \mathcal{R}_i(a^e)\right]$$

Each agent i starts with a random initial policy π_i^1, then update π_i^z to

$$\pi_i^{z+1}(a_i) = \begin{cases} \frac{1}{\eta}[\bar{R}_i^z(a_i^z, a_i)]_+ & \text{if } a_i \neq a_i^z \\ 1 - \sum_{a_i' \neq a_i^z} \pi_i^{z+1}(a_i') & \text{otherwise} \end{cases} \quad \text{with } \bar{R}_i^z(a_i', a_i) = \frac{1}{z} Regret_i^z(a_i', a_i)$$

where $\eta > 2 \cdot \max_{a \in A} |\mathcal{R}_i(a)| \cdot (|A_i| - 1)$ is a parameter.

Regret Matching Convergence

In both types of regret matching, the average regrets are bounded by $\kappa \frac{1}{\sqrt{z}}$ for some constant $\kappa > 0$

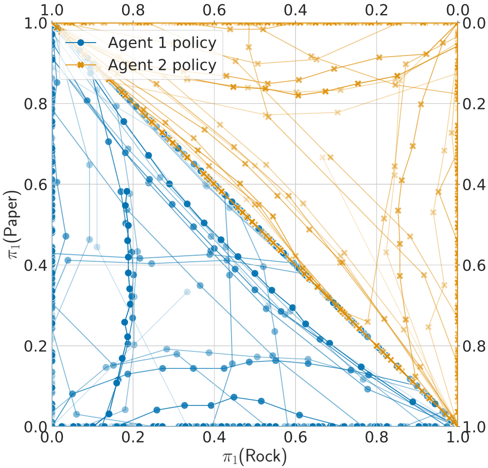
- For infinite episodes $z \rightarrow \infty$, the average regrets $\bar{R}_i^z$ will be at most 0

  $\rightarrow$ Thus, agents learn no-regret joint policy!
- Does not require any assumptions about the actions of other agents

Empirical action distributions converge to:

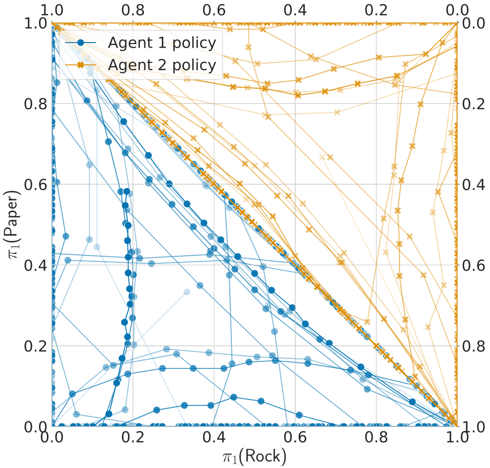
- Unconditional regret matching $\rightarrow$ coarse correlated equilibrium
- Conditional regret matching $\rightarrow$ correlated equilibrium

Unconditional Regret Matching in Rock-Paper-Scissors

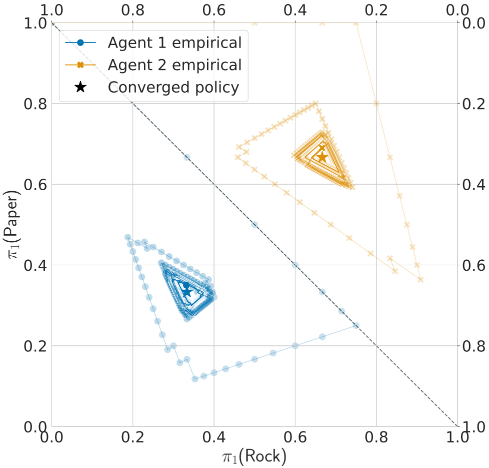


(a) Policies π_i^z

Unconditional Regret Matching in Rock-Paper-Scissors

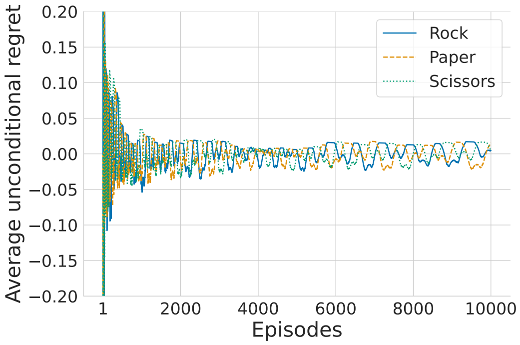


(a) Policies π_i^z

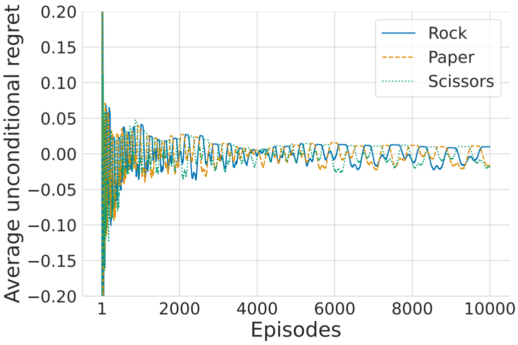


(b) Empirical distributions $\bar{\pi}_i^z$

Unconditional Regret Matching in Rock-Paper-Scissors



(a) Agent 1



(b) Agent 2

Summary

We covered:

- Value iteration for stochastic games
- Joint-action value learning algorithms
    - JAL-GT: temporal-difference learning with game theory solution concepts
    - JAL-AM: temporal-difference learning with agent models and best responses
- Learning policies by optimising policy parameters with gradient ascent
- Learning policies by minimising notions of regret

Next we'll cover:

- Deep learning