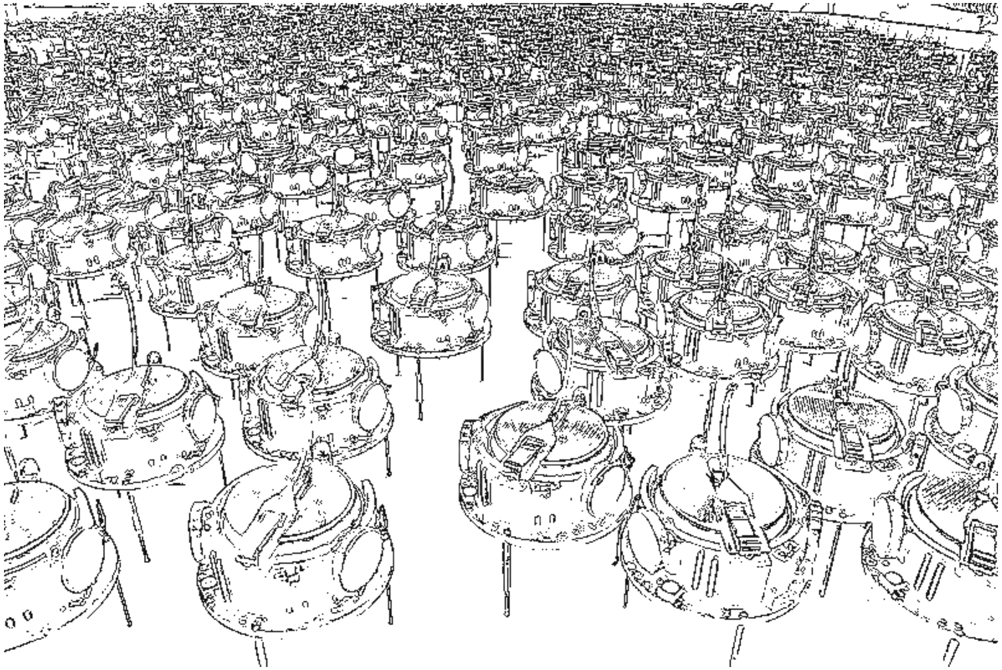
Intelligent Multi Agent Systems



Multi-Agent Reinforcement Learning

Gerhard Neumann

From Single-Agent RL to MARL

So far, we looked only at a single agent

However, learning with multiple agents can be inherently more challenging

- Large State Space
- Stability
- Non-Stationarity
- Exploration-Exploitation

Outline

1. **Challenges of Multi-Agent Reinforcement Learning**
2. **Algorithms:**
   - Single-Agent Approaches
   - Opponent Modelling Approaches
   - Equilibrium Approaches
3. **Analysis**
   - Dynamics of Learning Algorithms

Markov Games

n-player game: $\langle n, S, A_1, \ldots, A_n, \mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{P} \rangle$

- S : set of states
- A_i : action set for player i
- $\mathcal{R}_i$: reward for player i
- $\mathcal{P}$: transition probability

The reward function $\mathcal{R}_i : S \times A_1 \times \cdots \times A_n \to \mathbb{R}$ maps the joint action to an immediate reward value

The transition probability $\mathcal{P} : S \times A_1 \times \cdots \times A_n \times S \to [0, 1]$ depends on the joint action vector

Challenges: State Space

- Convergence criteria in *Q*-Learning include infinitely many visits of each state-action pair
- State-action space grows exponentially in number of states and actions
- In MARL also exponentially in the number of agents
- Curse of dimensionality

Questions:

- How to represent state-action space?
- How to ensure convergence?

Challenges: Stability

- Correlation of the returns for agents
- Independent maximization often not possible
- Highly dynamic and stochastic environments

Learning goals:

- Stability of the learning process (convergence to stationary strategies, e.g. Nash-Equilibrium)
- Adaption to other (learning) agents
- Tradeoff between of stability and adaption

Challenges: Non-Stationarity

- All learning is simultaneous
- Changes in strategy of one agent might affect strategy of other agents („nonstationarity”)
- Moving-Target Problem

Challenges: Exploration-Exploitation

- Exploitation of learned strategy
- Exploration of new strategies
- Balance of exploration and exploitation

MARL:

- Explore environment and other agents
- Too much exploration may lead to unstable learning processes

Outline

1. Challenges of Multi-Agent Reinforcement Learning
2. Algorithms:
   - Single-Agent Approaches
   - Opponent Modelling Approaches
   - Equilibrium Approaches
3. Analysis
   - Dynamics of Learning Algorithms

MDP-Approaches

Use reinforcement learning methods for Markov Decision Processes to learn in Stochastic Games: *Q-learning, Sarsa, LSPI…*

- Some success with this approach (Tan, 93; Sen *et al*, 94).

Pros:

- Simple implementation.

Cons:

- Cannot learn stochastic policies (MDP optimal is deterministic).
- Environment is not stationary from the agent's point of view (MDP methods assume stationarity).

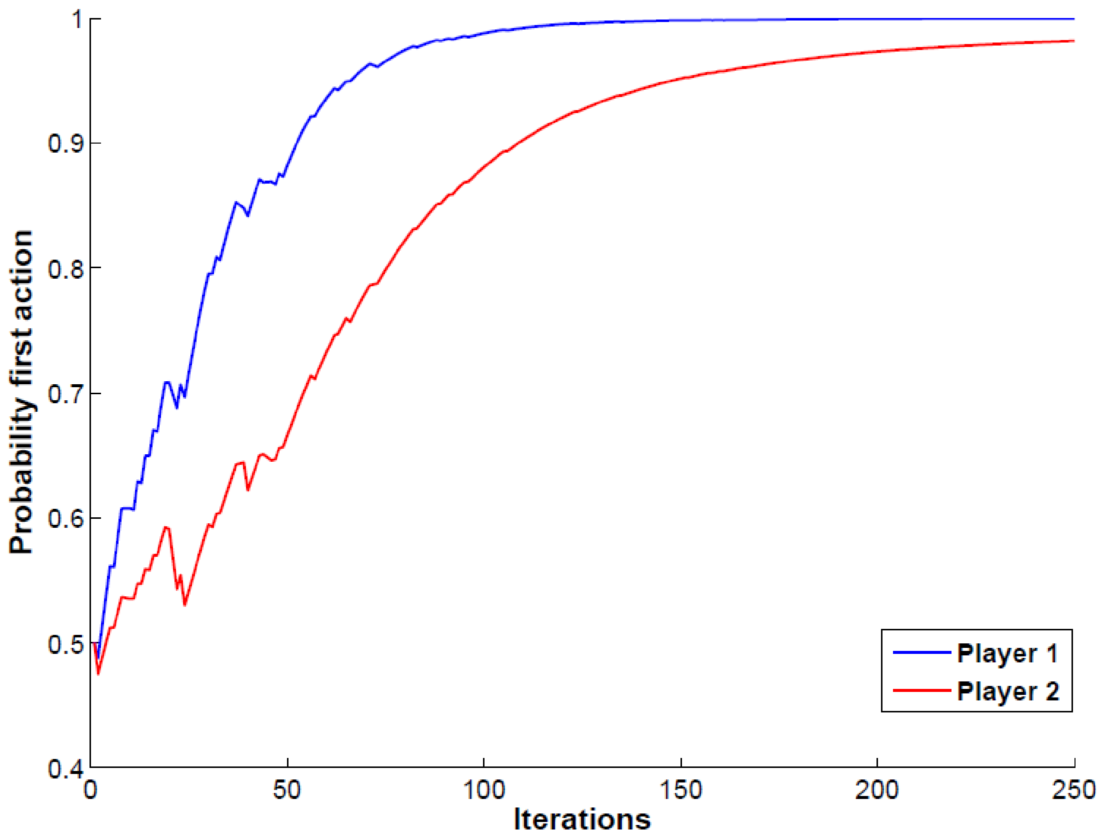
MARL vs RL

Case Study: Battle of the Sexes

- 2 Independent Reinforcement Learners (Q-Learners)
- Naïve extension to multi-agent setting
- Independent learners mutually ignore each other
- Perceive interaction with other agents as noise

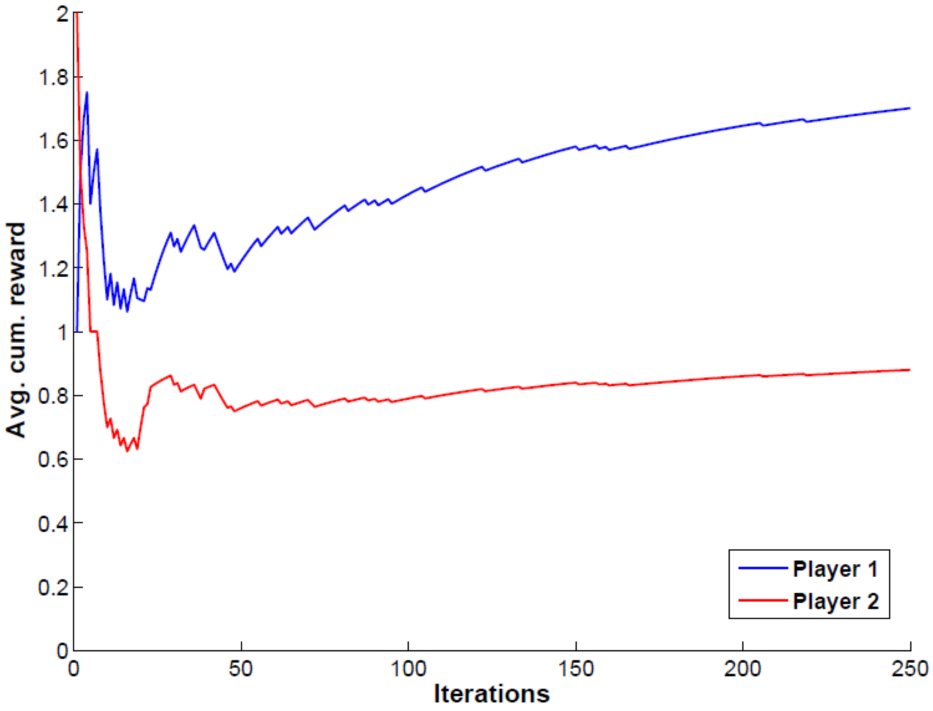
	B	S
B	2, 1	0, 0
S	0, 0	1, 2

Learning in Matrix Games



	B	S
B	2, 1	0, 0
S	0, 0	1, 2

Learning in Matrix Games



	B	S
B	2, 1	0, 0
S	0, 0	1, 2

Very slow convergence (or no convergence at all)!

Outline

1. **Challenges of Multi-Agent Reinforcement Learning**
2. **Algorithms:**
   - Single-Agent Approaches
   - Opponent Modelling Approaches
   - Equilibrium Approaches
3. **Analysis**
   - Dynamics of Learning Algorithms

Opponent-Modelling Methods

Similarly as in single-agent learning, we can compute the Q-function of the (optimal policy)

$$Q_i(s, \langle a_1 \ldots a_n \rangle) = r_i(s, \langle a_1 \ldots a_n \rangle) + \gamma \mathbb{E}[V_i(s') | s, \langle a_1 \ldots a_2 \rangle]$$

➡ The Q-Function depends on the joint action vector

How to evaluate *$V_i(s')$* in the multi-agent setup?

Joint-Action Learner

Learn Q-values based on joint actions:

$$Q_i(s, \langle a_1 \dots a_n \rangle) = r_i(s, \langle a_1 \dots a_n \rangle) + \gamma \mathbb{E}[V_i(s')|s, \langle a_1 \dots a_n \rangle]$$

- Maintain statistics of the opponents actions
  - Agent i's estimate of agent j's policy $\hat{\pi}_j^i(a_j|s) = \frac{n_{sa_j}^j}{n_s}$
  - $n_{sa_j}^j$ … number of times agent j has taken action a_j in state *s*
  - n_s … number of times we visited state *s*
- For evaluating the actions of agent *i* we average over the actions of the other agents

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

$$V_i(s') = \max_{a_i'} \hat{Q}_i(s', a_i')$$

Joint-Action Learner

Pros:

- Use information of the other players.

Cons:

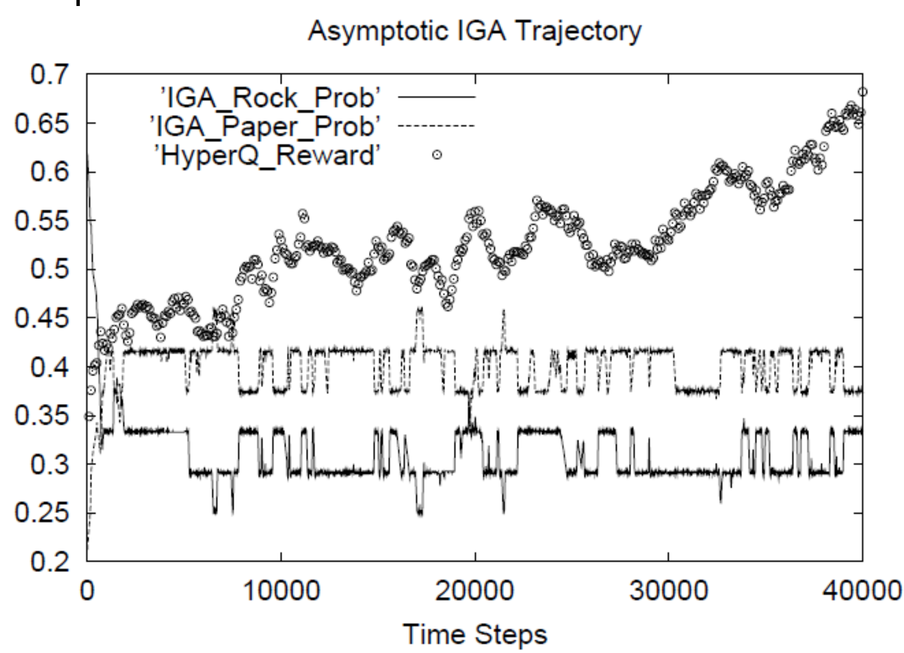
- Also only learn deterministic policies (max operator).

Hyper-Q Learner

Put the estimated strategies as argument in the Q-function

$$Q_i(s, \hat{\pi}, \langle a_1 \ldots a_n \rangle) = r_i(s, \langle a_1 \ldots a_n \rangle) + \gamma \mathbb{E}[V_i(s', \hat{\pi}') | s, \langle a_1 \ldots a_n \rangle]$$

- Learn to react to any (useful) strategy of the opponent
- The space of policies is a continuous space
- Use linear function approximation
  - Still, a very hard problem!
  - Only feasible for toy tasks



Outline

1. **Challenges of Multi-Agent Reinforcement Learning**
2. **Algorithms:**
   - Single-Agent Approaches
   - Opponent Modelling Approaches
   - Equilibrium Approaches
3. **Analysis**
   - Dynamics of Learning Algorithms

Equilibrium based Methods

Compute the Q-function of the (optimal policy)

$$Q(s, \langle a_1, a_2 \rangle) = r(s, \langle a_1, a_2 \rangle) + \gamma \mathbb{E}[V(s') | s, \langle a_1, a_2 \rangle]$$

How to evaluate V(s') in the multi-agent setup?

At each state, the Q-Values define an own matrix game

- called stage game

We can use different solution concepts to compute V(s')

- Minimax Q-Learning
- Nash Q-Learning

Can be used with any (off-policy) Q-Function based learning method (Q-Learning, LSPI)

MiniMax Q-Learning

Algorithm for zero-sum markov games:

➡ **Simple**: Q-Function of opponent is the negative of “mine”

➡ Learn the Q-function of the optimal min-max policy:

$$Q(s_t, \langle a_{t,1}, a_{t,2}\rangle) = (1-\alpha)Q(s_t, \langle a_{t,1}, a_{t,2}\rangle) + \alpha\Big(r(s_t, \langle a_{t,1}, a_{t,2}\rangle) + \gamma V_{\mathrm{MM}}(s_{t+1})\Big)$$

where $V(s_{t+1})$ is computed by using the maxmin strategy

$$V_{\mathrm{MM}}(s_{t+1}) = \max_{\pi_1} \min_{a_2} \sum_{a_1} \pi_1(a_1) Q(s_{t+1}, \langle a_1, a_2\rangle)$$

MiniMax Q-Learning converges to a Nash Equilibrium under the same assumptions than regular Q-learning [Littman1994]

Minimax Q-Learning

Pros:

- Solution concept can be efficiently implemented (linear program)
- Lower bound for agent performance

Cons:

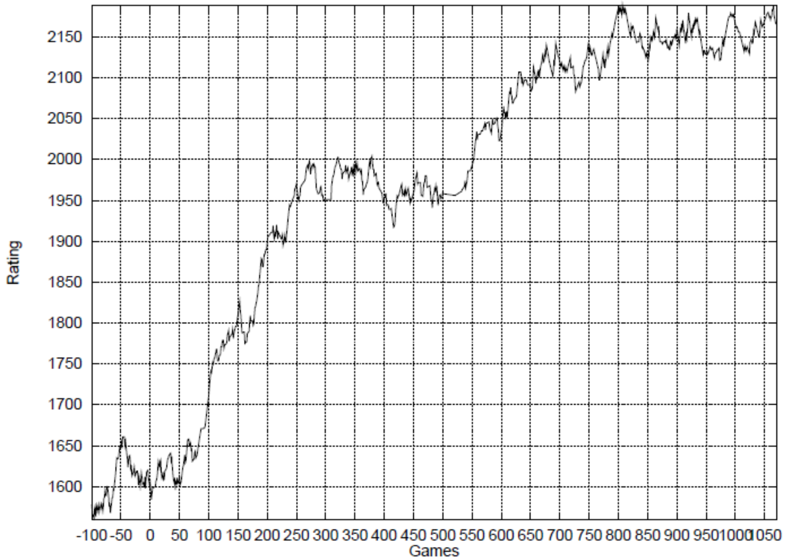
- Large actions spaces lead to big linear programs
- If the opponent plays sub-optimal, the minimax strategy is also sub-optimal

Knightcap: Playing Chess

Modified Minimax-Q learning algorithm

- Learn V-Function instead of Q-Function
- Model-Based: Use mini-max search tree of certain depth for action selection and Q-function update
- Use neural network to represent V-function

Score of the algorithm on an online platform:



Nash Q-Learning

Algorithm for general-sum markov games:

- Use the Nash-Equilibrium as solution concept for each stage game
- Each individual agent has to estimate the Q-Values of the other agents as well
- Assumption: We receive the rewards of the other agents
- Optimal Nash-Q Values: expected future reward if all agents play their optimal Nash strategy for each stage game

Nash Q-Learning

Update rule for agent *i*:

$$Q_i(s, \langle a_1, \ldots, a_n \rangle) = (1-\alpha) Q_i(s, \langle a_1, \ldots, a_n \rangle) \\ + \alpha\big(r_i(s, \langle a_1, \ldots, a_n \rangle) + \gamma V_{\mathrm{NE},i}(s')\big)$$

- Each stage game is solved by computing the Nash equilibrium for the matrix game defined by the Q-values $\langle Q_1(s', \cdot), \ldots, Q_n(s', \cdot) \rangle$
- $V_{\mathrm{NE},i}(s')$ is the value of this NE for player *i*
- Agent *i* uses the same update rules to learn the Q-tables of the other agents

Nash Q-Learning

Pros:

- Applicable to a wider range of problems

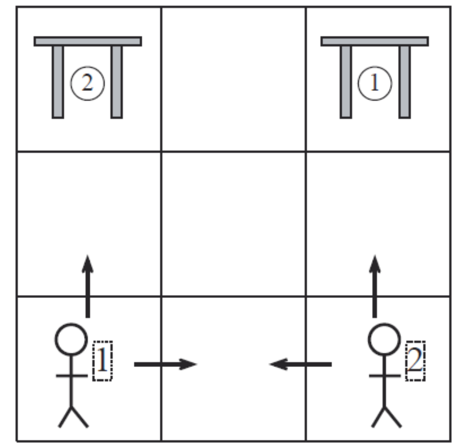
Cons:

- Convergence conditions are too strict and unrealistic
  - All intermediate games must have one equilibrium AND
  - It must be either a saddle point (like zero-sum games) or a global maximum (like team games).
- Nash Equilibria are very costly to find

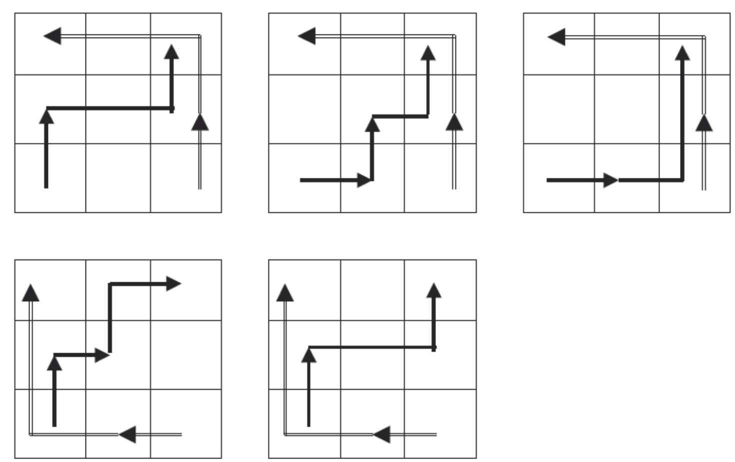
Nash Q-Learning

Results:

- 3x3 grid-world
- Agent 1 has to reach goal 1
- Agent 2 has to reach goal 2
- Collusions are not allowed



Found Nash Equilibria:



Outline

1. **Challenges of Multi-Agent Reinforcement Learning**
2. **Algorithms:**
   - Single-Agent Approaches
   - Opponent Modelling Approaches
   - Equilibrium Approaches
3. **Analysis**
   - Dynamics of Learning Algorithms

Deriving Dynamics of Learning Algorithms

- For most algorithms we can write the update rule of the policy as dynamical system

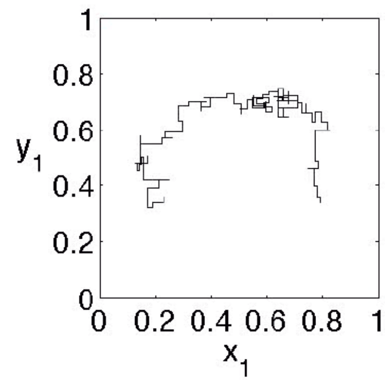
  Assuming the learning rate is 0 in the limit

$$\lim_{\alpha \to 0} \mathbb{E}[\Delta\pi] = \frac{d\pi}{dt}$$

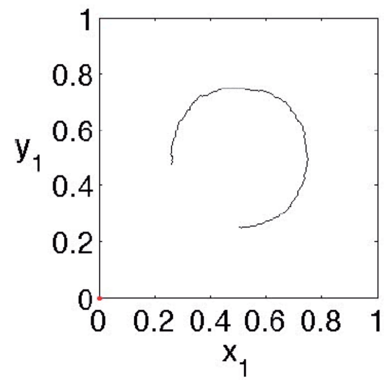
- Allows for stability analysis and intuitive illustrations
- We will do that in a much simpler setup (2 player matrix games)

Matching Pennies

	H	T
H	$1, 0$	$0, 1$
T	$0, 1$	$1, 0$



$\alpha = 0.1$



$\alpha = 0.001$

Infinite Gradient Ascent Algorithms (IGA)

Used for analysis of convergence in matrix games

- Compute expected reward given strategies of both agents

$$V_r(p,q) = p(qr_{11} + (1-q)r_{12}) + (1-p)(qr_{21} + (1-q)r_{22})$$

- Compute gradient

$$\frac{\partial V_r(p,q)}{\partial p} = q(r_{11} + r_{22} - r_{21} - r_{12}) - r_{22} + r_{12}$$

- Update with gradient ascent

$$p_{k+1} = p_k + \alpha \frac{\partial V_r(p,q)}{\partial p} \ldots \alpha \text{ learning rate}$$

Payoff-matrices

	q	1-q
p	r_{11}	r_{12}
1-p	r_{21}	r_{22}

	p	1-p
q	c_{11}	c_{12}
1-q	c_{21}	c_{22}

Gradient Ascent Algorithm

Convergence:

If in a two-person, two-action, iterated general-sum game, both players follow the IGA algorithm, their average payoff's will converge in the limit to the expected payoffs for some Nash equilibrium.

This will happen in one of two ways:

1) the strategy pair trajectory will itself converge to a Nash pair, or
2) the strategy pair trajectory will not converge, but the average payoffs of the two players will nevertheless converge to the expected payoffs of some Nash pair

IGA Dynamics

The gradient update can be seen as force field :

Prisoners' Dilemma

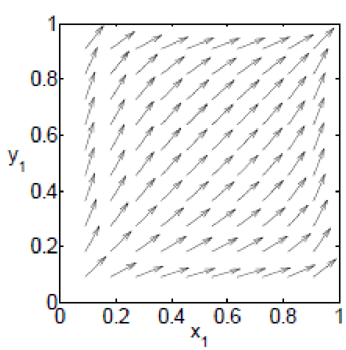
	D	C
D	$1, 1$	$5, 0$
C	$0, 5$	$3, 3$
	y_1	$1 - y_1$

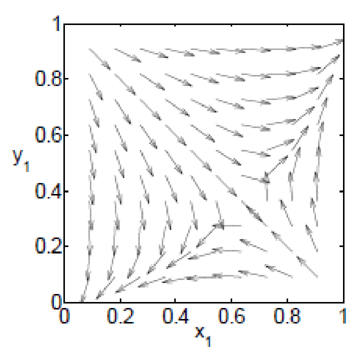
Battle of Sexes

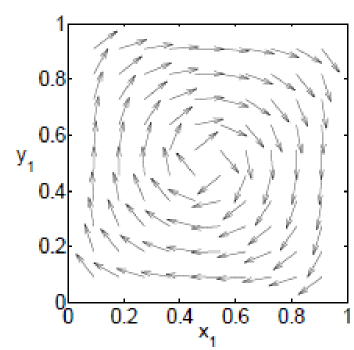
	B	S
B	$2, 1$	$0, 0$
S	$0, 0$	$1, 2$
	y_1	$1 - y_1$

Matching Pennies

	H	T	
H	$1, -1$	$-1, 1$	x_1
T	$-1, 1$	$1, -1$	$1 - x_1$
	y_1	$1 - y_1$	







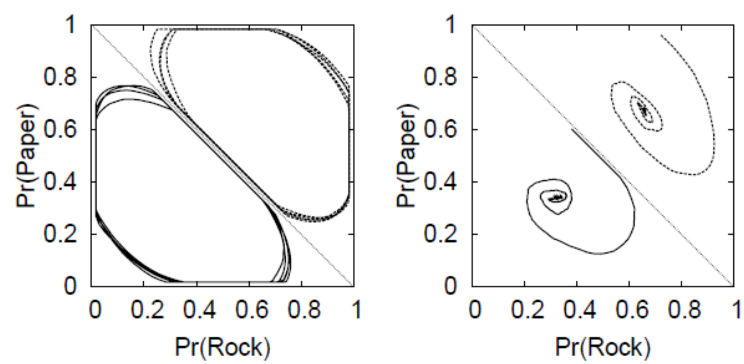
WoLF-IGA Algorithm

Win or Learn Fast (WoLF) Principle

- Low learning rate if winning
  - Give other players chance to adapt
- High learning rate if loosing

$$\alpha = \begin{cases} \alpha_{\min}, & \text{if } V(p,q) > V(p^*,q), \text{WINNING} \\ \alpha_{\max}, & \text{else } , \text{LOSING} \end{cases}$$

- p* is an equilibrium strategy selected by player 1
  - Rock-Paper-Scissors Wolf vs standard learning rate



(a) GIGA (b) GIGA-WoLF

Learning Algorithms as Dynamical Systems

Dynamics have also been derived for Q-Learning and variations

[see Kaisers and Tuyls, AAMAS 2010]

Q-learning [Watkins92]

- x_i probability of playing action i
- α learning rate
- r reward
- τ temperature

Update rule

$$Q_i(t+1) \leftarrow Q_i(t) + \alpha \left(r_i(t) + \gamma \max_j Q_j(t) - Q_i(t) \right)$$

Policy generation function

$$x_i(Q, \tau) = \frac{e^{\tau^{-1} Q_i}}{\sum_j e^{\tau^{-1} Q_j}}$$

Learning Algorithms as Dynamical Systems

Frequency Adjusted Q-learning (FAQ-learning) [Kaisers2010]

- x_i probability of playing action i
- α learning rate
- r reward
- τ temperature

Update rule

$$Q_i(t+1) \leftarrow Q_i(t) + \alpha \frac{1}{x_i} \left(r_i(t) + \gamma \max_j Q_j(t) - Q_i(t) \right)$$

Policy generation function

$$x_i(Q, \tau) = \frac{e^{\tau^{-1} Q_i}}{\sum_j e^{\tau^{-1} Q_j}}$$

[see Kaisers and Tuyls, AAMAS 2010]

Learn fast if action is hardly used!

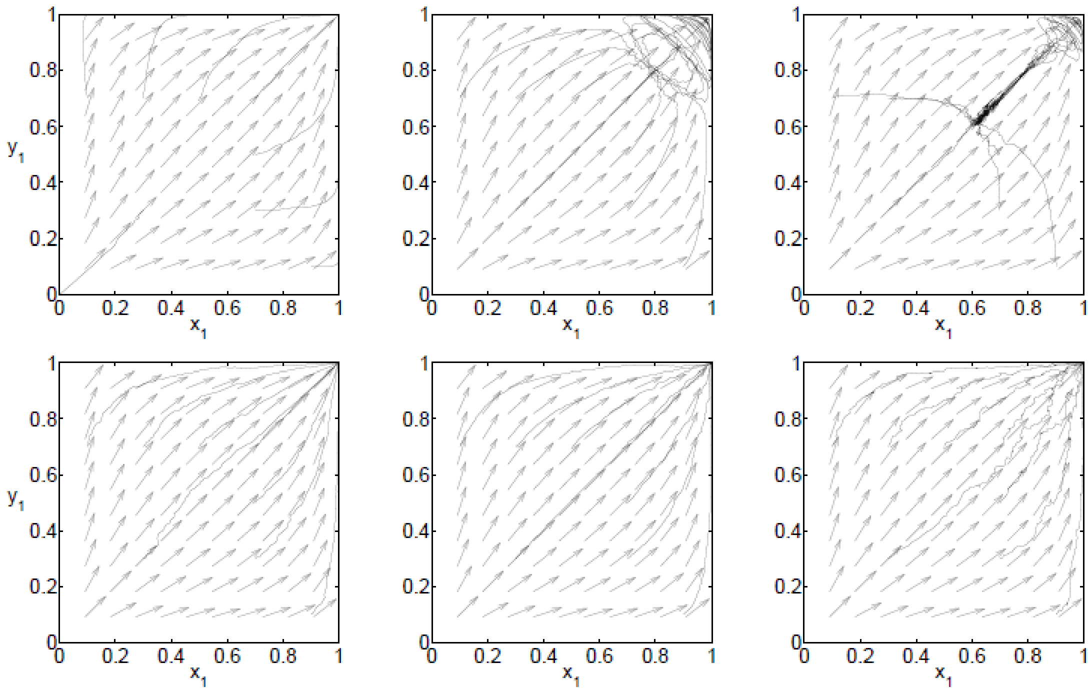
Exploration dependent update rule

Q-Learning Dynamics

Prisoners' Dilemma

Q

FAQ



Q_{init} pessimistic neutral optimistic

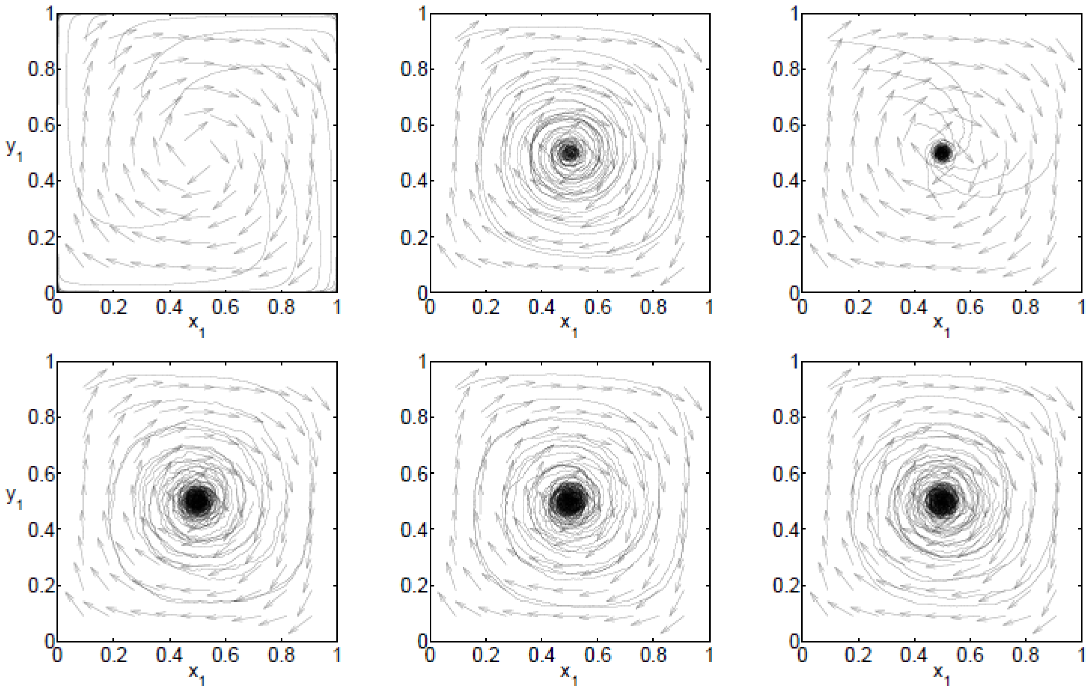
Convergence of Standard Q-Learning heavily depends on initialization!

Q-Learning Dynamics

Matching Pennies

Q

FAQ



Q_{init} pessimistic neutral optimistic

Convergence of Standard Q-Learning heavily depends on initialization!

Conclusion

- MARL has many challenges
  - Exploration is one of the key challenges
- We have seen several approaches for multi-agent learning
  - Most of them can only be applied to toy tasks
- Algorithms have many assumptions and are computationally heavy
- Deriving the Dynamics of learning algorithms helps with the theoretical analysis and gives good insights